DATA MINING TECHNIQUES AND ALGORITHMS
Partial Draft of forthcoming book from Prentice Hall

Margaret H. Dunham
Southern Methodist University

© Prentice Hall

August 23, 2000

Contents

PREFACE

Data doubles about every year, but useful information seems to be decreasing. The area of data mining has arisen over the last decade to address this problem. It has become not only an important research area, but also one with large potential in the real world. Current business users of data mining products achieve millions of dollars a year in savings by using data mining techniques to reduce the cost of day to day business operations.

The purpose of this book is to introduce the reader to various Data Mining concepts and algorithms. A database perspective is used throughout. This means that we examine algorithms, data structures, data types, and complexity of algorithms and space. The emphasis is on the use of data mining concepts in real world applications with large database components.

Data Mining reseach and practice is in a state similar to that of databases in the 1960s. At that time application programmers had to create an entire database environment each time they wrote a program. With the development of the relational data model, query processing and optimization techniques, transaction management strategies, and ad hoc query languages (SQL) and interfaces the current environment is drastically different. The evolution of data mining techniques may take a similar path over the next few decades making data mining techniques easier to use and develop. The objective of this book is to help in this evolution process.

The intended audience of this book is either the experienced database professional who wishes to learn more about data mining or graduate level Computer Science students who have completed at least an introductory database course. The book is meant to be used as the basis of a one semester graduate level course covering the basic Data Mining concepts. It may also be used as a reference book for computer professionals and researchers.

The book is divided into four major parts: Introduction, Core Topics, Advanced Topics, and Case Studies. The Introduction covers background information needed to understand the later material. In addition, it examines topics related to data mining such as OLAP, Data Warehousing, Information Retrieval, and Artificial Intelligence. The second chapter in this portion surveys some techniques used to implement data mining algorithms. These include statistical techniques, neural networks, and decision trees. The Core Topics covered are classification, clustering, and association rules. We view these as the major data mining functions. Other data mining concepts (such as prediction, regression, and pattern matching) can be viewed as special cases of these three. The advanced topics part looks as various concepts which complicate data mining applications. We concentrate on temporal data, spatial data, scalability, and parallelism. The case studies portion of the book examines some ongoing research prototypes as well as production data mining systems.

Part I

INTRODUCTION

Chapter 1

INTRODUCTION

"It's like having a genie in your database."
Lou Gerstner, CEO of IBM

The amount of data kept in computer files and databases is growing at a phenomenal rate. At the same time the users of these data are expecting more sophisticated information from them. A marketing manager is no longer satisfied with a simple listing of marketing contexts, but wants detailed information about customers' past purchases as well as predictions of future ones. Simple SQL queries are not adequate to support these increased demands on information. Data mining steps in to solve these needs. *Data Mining* is often defined as finding hidden information in a database. Alternatively it has been called exploratory data analysis, data driven discovery, and deductive learning.

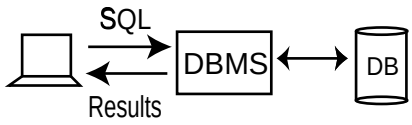


Figure 1.1: Database Access

Traditional database queries, see Figure 1.1, access a database using a well defined query stated in a language such as SQL. The output of the query is the data from the database which satisfies the query. The output is usually a subset of the database, but may also be an extracted view or contain aggregations. Data mining access of a database conflicts with this traditional access in several ways:

- Query - The query may not be well formed or precisely stated. The data miner may not even be exactly sure of what he wants to see.
- Data - The data accessed is usually a different version than that of the original operational database. The data has been cleansed, and modified to better support the mining process.
- Output - The output of the data mining query is probably not a subset of the database. Instead it is the output of some analysis of the contents of the database.

The current state of the art of data mining is similar to that of database query processing in the late sixties and early seventies. Over the next decade there will undoubtedly be great strides in extending the state of the art with respect to data mining. We will probably see the development of "query processing" models, standards, and algorithms targeting the data mining applications. We will probably also see new data structures designed for the storage of databases being used for data mining applications. Although data mining is currently in its infancy, over the last decade we have seen a proliferation of mining algorithms, applications, and algorithmic approaches. Example 1.1, illustrates one such application.

Example 1.1 *Credit card companies must determine whether or not to authorize credit card purchases. Suppose that each purchase is placed into four classes: 1) authorize, 2) ask for further identification prior to authorization, 3) do not authorize, and 4) do not authorize but contact police. Based upon past historical information about purchases, each credit card purchase is placed into one of these four categories. The data mining functions here are two fold. First the historical data must be examined to determine how the data fits into the four classes. Then the problem is to apply this model to each new purchase. While the second part may indeed be stated as a simple database query, the first part can not be.*

Data mining involves many different algorithms to accomplish different tasks. All of these algorithms attempt to fit a model to the data. The algorithms examine the data and determine a model that is closest to the characteristics of the data being examined. Data mining algorithms can be characterized as consisting of three parts [28]:

- Model - The purpose of the algorithm is to fit a model to the data.
- Preference - Some criteria must be used to fit one model over another.
- Search - All algorithms require some technique to search the data.

In Example 1.1 the data is modeled as divided into four classes. The search requires examining past data about credit card purchases and their outcome to determine what criteria should be used to define the class structure. The preference will be given to criteria which seem to best fit the data. For example, we would probably not want to classify a credit card purchase for a small amount of money to a credit card belonging to a long standing customer as a non authorized purchase. Likewise we would not want to authorize the use of a credit card to purchase anything if the card has been reported as stolen. The search process requires that the criteria needed to fit the data to the classes be properly defined.

The model which is created can be either predictive or descriptive in nature. A *predictive model* makes a prediction about values of data using known results found from different data. This prediction may be made based on the use of other historical data. For example, a credit card use may be refused not because of the user's own credit history, but instead based on the fact that the current purchase is similar to earlier ones which were subsequently found to be made with stolen cards. Thus this purchase is predicted to be a bad credit risk. Example 1.1 uses predictive modeling to predict the credit risk. Predictive model data mining tasks include classification, time series analysis, and regression. A *descriptive model* identifies patterns or relationships in data. Clustering, association rules, and sequence discovery are usually viewed as descriptive in nature.

1.1 Basic Data Mining Tasks

In the following paragraphs we briefly explore some of the data mining functions. This list is not intended to be exhaustive, but rather illustrative. Of course these individual tasks may be combined to obtain more sophisticated data mining applications. They are discussed separately here and in later chapters for ease of presentation

1.1.1 Classification

Classification maps data into predefined groups. It is often referred to as supervised learning as the classes are determined prior to examining the data. Classification applications include: determining whether or not to make a bank loan and identifying credit risks. Classification algorithms usually require that the classes be defined based on data attribute values. They often describe these classes by looking at the characteristics of data already known to belong to the classes. *Pattern Recognition* is a type of classification where an input pattern is classified into one of several classes based on its similarity to these predefined classes.

Regression is a type of classification used to map a real valued predicate variable into data values. Standard linear regression, as illustrated in Example 1.2, is a simple example of regression.

Example 1.2 *A college professor wishes to reach a certain level of savings prior to her retirement. Periodically, she predicts what her retirement savings will be based on its current value and several past values. She uses a simple linear regression formula to predict this value.*

1.1.2 Clustering

Clustering is similar to classification except that the groups are not predefined, but rather defined by the data itself. It is alternatively referred to as unsupervised learning or segmentation. It can be thought of as partitioning or segmenting the database into groups which may or may not be disjoint. The clustering is usually accomplished by determining the similarity among the data on predefined attributes. Clustering groups database tuples by determining the similarities among different tuples. The most similar data are grouped into clusters. Example 1.3 provides a simple clustering example. Since the clusters are not predefined, a domain expert is often required to interpret the meaning of the created clusters.

Example 1.3 *A certain national department store creates multiple special catalogs targeted to various demographic groups based on income, location, and physical characteristics of potential customers (age, height, weight, etc.). To determine the target mailings of the various catalogs and to assist in the creation of new more specific catalogs, the company performs a clustering of potential customers based on the determined attribute values. The results of the cluster exercise are then used by management to create special catalogs and distribute them to the correct target population based on the cluster for that catalog.*

A special type of clustering is called *segmentation*. With segmentation a database is patitioned into disjoint groupings of similar tuples, *segments*. Segmentation is often viewed to be identical to clustering. In other circles segmentation is viewed to be a specific type of clustering applied to a database itself. In this text we will use the two terms, clustering and segmentaiton, interchangeably.

Summarization maps data into subsets with associated simple descriptions. It is a special type of clustering problem. This new simpler description of the data is then used in place of the entire data set. Example 1.4 illustrates this process.

Example 1.4 *One of the many criteria used to compare universities in the the U.S. News & World Report ranking of universities is the average SAT or ACT score [40]. This is a summarization used to estimate the type and intellectural level of the student body.*

1.1.3 Prediction

Many real world data mining applications can be seen as predicting future data states based on past and current data. *Prediction* can be viewed as a type of clustering or classification. The difference is that prediction is predicting a future state rather than a current one. Prediction applications include: flooding, speech recognition, machine learning, and pattern recognition. Example 1.5 illustrates the process.

Example 1.5 *Predicting flooding is a difficult problem. One approach which has been used is to place monitors at various points in the river. These monitors collect data relevant to flood prediction: water level, rain amount, time, humidity, etc. Then the water level at a potential flooding point in the river can be predicted based on the data collected by the sensors upriver from this point. This prediction must be made with respect to the time that the data was collected.*

1.1.4 Link Analysis

An analysis may be done to determine the relationships among data. *Link Analysis*, alternatively referred to as *Affinity Analysis* or *Association*, refers to the data mining task of uncovering relationships among data. The best example of this type of application is to determine association rules. Example 1.6 illustrates the use of association rules in market basket analysis. Associations are also used in many other applications such as predicting failure of telecommunication switches,

Example 1.6 *A grocery store retailer is trying to decide whether to put bread on sale. To help determine the impact of this decision, they generate association rules which show what other products are frequently purchased with bread. They find that 60% of the time that bread is sold so are pretzels and that 70% of the time jelly is. Based on these facts they try to capitalize on the association between bread, and pretzels and jelly by placing some pretzels and jelly on the end of the aisle where the bread is placed. In addition, they decide not to place either of these items on sale at the same time.*

Users of association rules must be cautioned that these are not causal relationships. They do not represent any relationship inherent in the data itself (as is true with functional dependencies). There is probably no relationship between bread and pretzels that causes them to be purchased together. And there is no guarantee that this association will apply in the future. However association rules can be used to assist retail store management in effective advertising, marketing, and inventory control and placement.

Sequential Analysis is used to determine sequential patterns in data. These sequential patterns may be based upon a time sequence of actions. These patterns are like associations in that data

(or events) are found to be related, but the relationship is based on time. Unlike a market basket analysis which requires the data to be purchased at the same time, in a sequence the data would be purchased over time in some order. For example, most people who purchase CD players may be found to purchase CDs within one week. As we will see, temporal association rules really fall into this category.

1.1.5 Time Series Analysis

With *Time Series Analysis*, the value of an attribute is examined as it varies over time. Usually the values are obtained as evenly spaced time points (daily, weekly, hourly, etc.). A time series plot, see Figure 1.2, is used to visualize the time series. In this figure you can easily see that the plots for Y and Z have similar behavior, while X appears to have less volatility. These are two basic functions performed in time series analysis. In one case distance measures are used to determine the similarity between different time series. In the second case the structure of the line is examined to determine (and perhaps classify) its behavior. A third application would be to use the historical time series plot to predict future values. A time series example is given in Example 1.7.

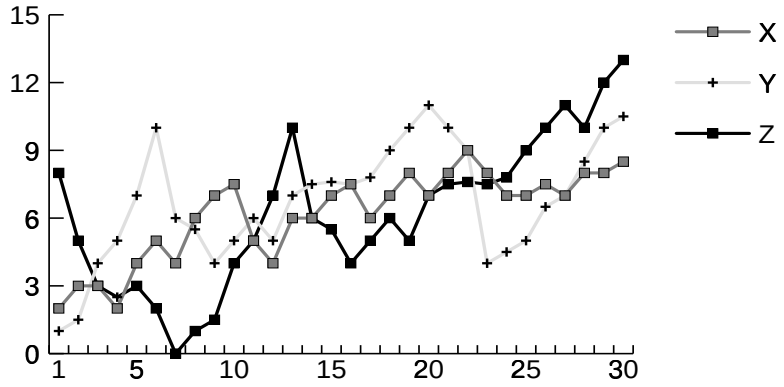


Figure 1.2: Time Series Plots

Example 1.7 *Mr. Smith is trying to determine whether to purchase stock from Companies X, Y, or Z. For a period of one month he charts the daily stock price for each company. Figure 1.2 shows the time series plot which Mr. Smith has generated. Using this and similar information available from his stock broker, Mr. Smith decides to purchase stock X as it is less volatile while ovreall showing a slightly larger relative amount of growth than either of the others. As a matter of fact, the stocks for Y and Z have a similar behavior. The behavior of Y between days 6 and 20 is identical to that for Z between days 13 and 27.*

1.2 Data Mining versus Knowledge Discovery in Databases

The terms *Knowledge Discovery in Databases (KDD)* and Data Mining are often used interchangeable. In fact, there have been many other names given to this process of discovering useful (hidden) patterns in data: knowledge extraction, information discovery, exploratory data analysis, information harvesting, and unsupervised pattern recognition. Over the last few years KDD has been used

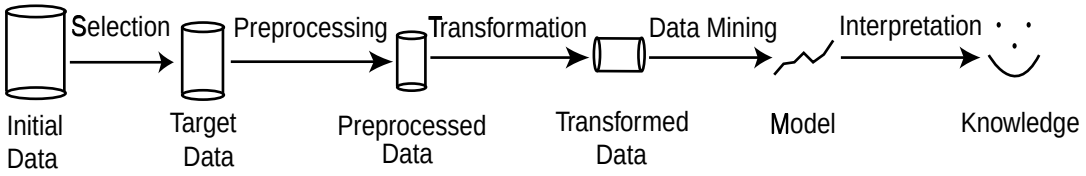


Figure 1.3: KDD Process

to refer to a process consisting of many steps, while data mining is only one of these steps [28, 26]. This is the approach taken in this book. The following definitions are modified from those found in [28, 26].

Definition 1.1 Knowledge Discovery in Databases (KDD) *is the process of finding useful information and patterns in data.*

Definition 1.2 Data Mining *is the step in the KDD process which actually accesses the data itself.*

The KDD process is often said to be non-trivial, however, we take the larger view that KDD is an all encompassing concept. A traditional SQL database query can be viewed as the data mining part of a KDD process. Indeed, this may be viewed as somewhat simple and trivial. However, thirty years ago this was not the case. If we were to advance in the future thirty years, we may find that processes thought of today as non-trivial and complex will be viewed as equally simple. The definition of KDD includes the keyword: useful. Although some definitions have included "potentially useful", we feel that if the information found in the process is not useful then it really isn't information. Of course the idea of being useful is relative and depends on the individuals involved.

KDD is a process involving many different steps. The input to this process is the data and the output is the useful information desired by the users. However, the objective may be unclear or not exact. The process itself is interactive and may require much elapsed time. To ensure the usefulness and accuracy of the results of the process, interaction throughout the process with both domain experts and technical experts may be needed. A complete discussion of the process is found in [11]. Figure 1.3 modified from [28] illustrates the overall KDD process.

The KDD process consists of the following five steps [28]:

- Selection: The data needed for the data mining process may be obtained from many different and heterogenous data sources. This first step obtains the data from these various databases, files, and nonelectronic sources.

- Preprocessing: The data to be used by the process may have incorrect or missing data. There may be anomalous data from multiple sources involving different data types and metrics. There may be many different activities performed at this time. Erroneous data may be corrected or removed while missing data must be supplied or predicted (often using data mining tools).

- Transformation: Data from different sources need to be converted in a common format for processing. Some data may be encoded or transformed into more useable formats. Data reduction may be used to reduce the number of possible data values being considered.

- Data Mining: Based on the data mining task being performed, this step processes algorithms against the transformed data to generate the desired results.
- Interpretation/Evaluation: How the data mining results are presented to the users is extremely important as the usefulness of the results is dependent on it. Various visualization and GUI strategies are used at this last step.

1.3 Related Concepts

Data Mining applications have existed for thousands of years. For example, the classification of plants into edible and nonedible is a data mining task. The development of the data mining discipline has its roots in many other areas. This section briefly examines some of these and how they relate to data mining.

1.3.1 Database/OLTP Systems

Users' expectations for queries have increased as have the amount and sohpistication of the associated data. In the early days of *Database (DB)* and *OnLine Transaction Processing (OLTP)* systems, simple selects were enough. Now queries are complex involving data distributed over many sites and use complicated functions such as joins, aggregates, and views. Traditional database queries usually involve retrieving data from a database based on a well defined query. For example, a user may request to find all employees who earn over $50,000. This could be viewed as a type of classfication application as we segment the database into two classes: those who have salaries satisfying the predicate and those who don't. A simple database application is not thought of as a data mining task, however, because the queries are well defined with precise results. Data mining applications, on the other hand, are often vaguely defined with imprecise results. Users may not even be able to precisely define what they want - let alone be able to tell if the results of their request are accurate. A database user can often tell if the results of his query are not correct. Figure 1.1 illustrated a typical database/OLTP query

1.3.2 Fuzzy Sets and Fuzzy Logic

A *fuzzy set* is a set, F, in which the set membership function, f, is a real valued (as opposed to boolean) function with output in the range [0,1]. An element x is said to belong to F with probability $f(x)$. Fuzzy sets have been used in many computer science and database areas. In the classification problem, all records in a database are assigned to belong to one of the predefined classification areas. A common approach to solving the classification problem is to assign a set membership function to each record for each class. The record is then assigned to the class which has the highest membership function value. Similarly, fuzzy sets may be used to describe other data mining functions. Association rules are generated given a confidence value which indicates the degree to which it holds in the entire database. This can be thought of as a membership function.

Fuzzy logic is reasoning with uncertainty. That is, instead of a two valued logic (True and False) there are multiple values (True, False, Maybe). Fuzzy logic has been used in database systems to retrieve data with imprecise or missing values. In this case, the membership of records in the query result set is fuzzy. Indeed conventional IR systems and web based browsers use fuzzy logic to retrieve documents.

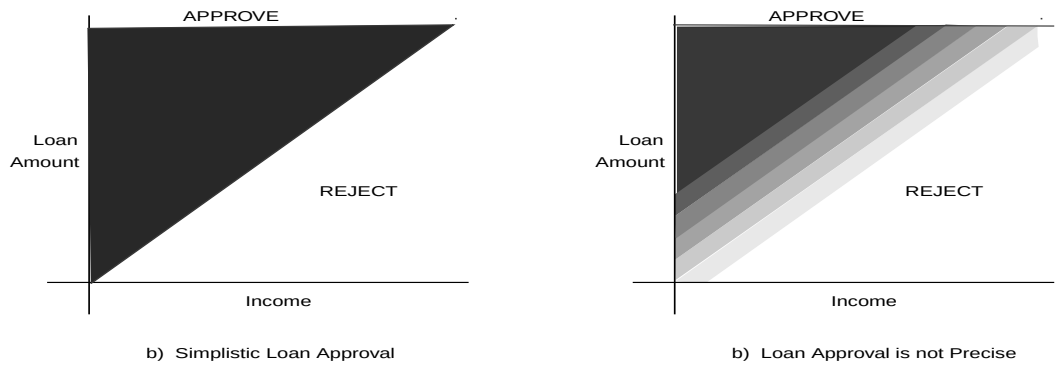


Figure 1.4: Fuzzy Classification

Most real world classification problems are fuzzy. This is illustrated by Figure 1.4. In this figure we graphically show the threshold for approving a loan based on the income of the individual and the loan amount requested. A loan officer may make the loan decision by simply approving any loan requests on or above the line and rejecting any that fall below the line, Figure 1.4 a). This type of decision would not be fuzzy. However, this type of decision could lead to erroneous and perhaps costly decisions by the loan officer. From a data mining perspective, this application is a classification problem. That is, classify a loan application into the approval or reject class. There are many other factors (other than income) which should be used to predict the classification problem (such as net worth and credit rating). Even if all the associated predictors could be identified, the classfication problem is not a black and white issue. It is possible that two individuals with exactly the same predictor values should be placed in two different classes. This is due to the fuzzy nature of this classification. In Figure 1.4 b), this is shown by the shading around the line. We could, perhaps, classify the individuals into multiple classes: Approve, Reject, Unknown, Probably Approve, Probably Reject. This approach attacks the fuzzy nature of the classification by flagging the applications requiring more analysis into the three new fuzzy classes. Procedural policies could then be used to determine the ultimate classification of these cases into the final Approve or Reject classes. This is one of many possible approaches to defuzzify the classification problem.

1.3.3 Information Retrieval

Information Retrieval (IR) (and more recently digital libraries indexinformation retrieval and internet searching) involves retrieving desired information from textual data. The historical development of IR was based on effective use of libraries. So a typical IR request would be to find all library documents related to a particular subject, for example "data mining". This is in fact a classification task as the set of documents in the library is divided into classes based on the keywords involved. In IR systems, documents are represented by document surrogates consisting of data such as identifiers, title, authors, dates, abstracts, extracts, revews, and keywords. As can be seen, this data consists of both formatted and unformatted (text) data. The retrieval of documents is based upon calculation of a similarity measure showing how close each document is to the desired results (ie. the stated query). Similarity measures are also used in classification and clustering problems.

An IR system consists of a set of documents $D = \{D_1, ..., D_n\}$. The input is a query, q, often

stated as a list of keywords. The similarity between the query and each document is then calculated: $sim(q, D_i)$. This similarity measure is a set membership function describing the likelihood that the document is of interest (relevant) to the use based on the user's interest as stated by the query. The effectiveness of the system in processing the query is often measured by looking at *precision* and *recall*:

$$Precision = \frac{\mid Relevant\ and\ Retrieved \mid}{\mid Retrieved \mid} \tag{1.1}$$

$$Recall = \frac{\mid Relevant\ and\ Retrieved \mid}{\mid Relevant \mid} \tag{1.2}$$

Precision is used to answer the question: "Are all documents retrieved ones that I'm interested in?" While recall answers: "Have all relevant documents been retrieved?" Here a document is relevant if it should have been retrieved by the query. Figure 1.5 illustrates the concepts of a conventional information retrieval query.

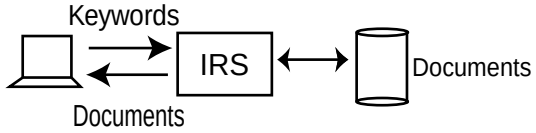


Figure 1.5: Information Retrieval Query

There are many similarity measures which have been proposed for use in Information Retrieval. As stated earlier, $sim(q, D_i)$ is used to determine the results of a query q applied against a set of documents $D = \{D_1, ..., D_n\}$. Similarity measures may also be used to cluster or classify documents by calculating $sim(D_i, D_j)$ for all documents in the databaset. Thus similarity is used for document-document, query-query, and query-document measurements. The *Inverse Document Frequency (IDF)* is often used by similarity measures. This assumes that the importance of a keyword in performing similarity measures is inversely proportional to the total number of documents. Given a keyword, k, and n documents IDF can be defined as follows [92]:

$$IDF_k = lg\frac{n}{\mid documents\ containing\ k \mid} + 1 \tag{1.3}$$

Concept hierarchies are often used in information retrieval to show the relationships between various keywords (concepts) as related to documents. Suppose you wish to find all documents about cats. Figure 1.6 illustrates a *concept hierarchy* which could be used to answer this query. This figure shows that feline and cat are similar terms. In addition, a cat may be domestic or tiger or lion or cheetah. In turn, a tiger may be a siberian, white, indochinese, sumatra, or south china. Lower levels in the tree represent more specific groups of tigers. When a user requests a book on tigers, this query could be modified by replacing the keyword "tiger" with a keyword at a higher level in the tree, such as "cat". Although this would result in a higher recall, the precision would decrease. A concept hierarchy may actually be a DAG (rather than a tree).

Information retrieval has had a major impact on the development of data mining. Much of the data mining classification and clustering approaches had their origins in the document retrieval problems of library science and information retrieval. Many of the similarity measures developed

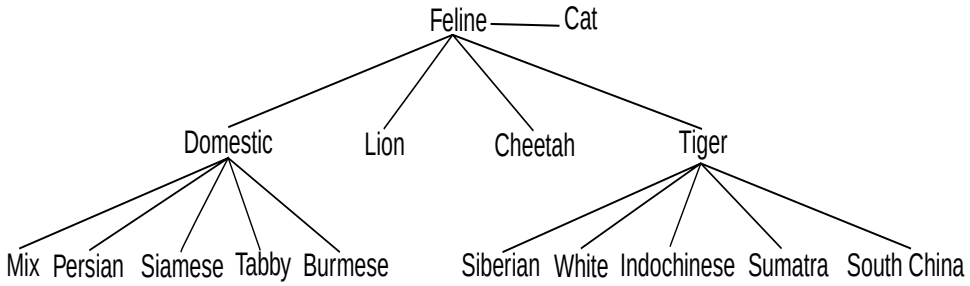


Figure 1.6: Concept Hierarchy

for information retrieval have been applied to more general data mining applications. Similarly, the precision and recall measures are often applied to data mining applications as illustrated in Example 1.8. Concept hierarchies are frequently used in spatial data mining applications. Data mining consists of many more types of applications that found in traditional information retrieval. The linking and predictive tasks have no real counterpart in information retrieval, for example.

Example 1.8 *The accuracy of a predictive modeling technique can be described based on precision and recall. Suppose college students are to be classified based on height. We could filter out students younger than 17 because the classification scheme does not properly classify children. Thus the model may not have a high recall as not all students fit into it. On the other hand, it could have a high precision if it correctly classified the majority of the students it did examine (ie. those over 17).*

1.3.4 Decision Support Systems

Decision Support Systems (DSS) are comprehensive computer systems and related tools which assist managers in making decisions and solving problems. The goal is to improve the decision making process by providing specific information needed by management. These differ from traditional database management systems in that more ad hoc queries and customized information may be provided. Recently the terms *Executive Information Systems (EIS)* and *Executive Support systems (ESS)* have evolved as well. These indexexecutive information systems all aim at developing the business structure and computer techniques to better provide information needed by management to make effective business decisions. Data mining can be thought of as a suite of tools which assist in the overall DSS process. That is, DSS may use data mining tools.

1.3.5 Dimensional Modeling

Dimensional modeling is a different way to view and interrogate data in a database. This view may be used in a DSS in conjunction with data mining tasks. Although not required, for efficiency purposes the data may be stored using different data structures as well. Decision support applications often require that information be obtained along many dimensions. For example, a sales manager may want to obtain information about the amount of sales in a geographic region, particular time frame, and by product type. This query requires three dimensions. A *dimension* is a collection of logically related attributes and is viewed as an axis for modeling the data. The time dimension could be divided into many different granularities: millenium, century, decade, year, month, day,

ProdID	LocID	Date	Quantity	UnitPrice
123	Dallas	022900	5	25
123	Houston	020100	10	20
150	Dallas	031500	1	100
150	Dallas	031500	5	95
150	Fort Worth	021000	5	80
150	Chicago	012000	20	75
200	Seattle	030100	5	50
300	Rochester	021500	200	5
500	Bradenton	022000	15	20
500	Chicago	012000	10	25

Table 1.1: Relational View of Multidimensional Data

hour, minute, second, etc. Within each dimension, these entities form levels on which various DSS questions may be asked. The specific data stored are called the *facts* and are usually numeric data. Facts consist of measures and context data. The measures are the numeric attribues about the facts which are queried. DSS queries may access the facts from many different dimensions and different levels. The levels in each dimension facilitate the retrieval of facts at different levels. For example, the sales information could be obtained for the year 1999, or for the month of February in the year 2000, or between the times of 10 and 11 am on March 1, 2000. The same query could be formulated for a more general level, *drill up*, or a more specific level, *drill down*.

Table 1.1 shows a relation with three dimensions: Products, Location, and Date. Determining a key for this relation could be difficult as it is possible for the same same product to be sold multiple times on the same day. In this case product 150 was sold two different times in Dallas on the same day. A finer granularity on the time (perhaps down to the minute rather than date as is here) could make a key. However, this illustrates that choice of key may be difficult. This same multidimensional data may also be viewed as a cube. Figure 1.7 shows a view of the data from Figure 1.1 as a cube. Each dimension is seen as an axis for the cube. This cube has one fact for each unique combination of dimension values. In this case we could have $8 * 7 * 5 = 280$ facts stored (even though the relation only showed ten tuples). Obviously, this sparse amount of data would need to be stored efficiently to reduce the amount of space required.

The levels of a dimension may support a partial or total order, and can be viewed via a directed path, hierarchy or lattice. To be consistent with earlier uses of the term, we use *Aggregation Hierarchy* , even though it may be a lattice, to refer to the order relationship among different levels in a dimension. We use $<$ to represent this order relation. $X < Y$ if X and Y are levels in the same dimension and X is contained in Y. Figure 1.8 a) shows a total order relationship among levels in the Product dimension from Figure 1.7. Here Product $<$ Type $<$ Company. The two facts that we are using in this example are Quantity and UnitPrice. When this order relationship is satisfied between two levels, there is an aggregate type of relationship among the facts. Here the Quanity of products sold of a particular type is the sum of the quantities for all products within that type. Similarly the quantity of products sold for a company is the sum of all products sold across all product types. The aggregate operation is not always sum, however. When looking at the Unit Price, it would be reasonable to look at such aggregate operations as average, maximum, and

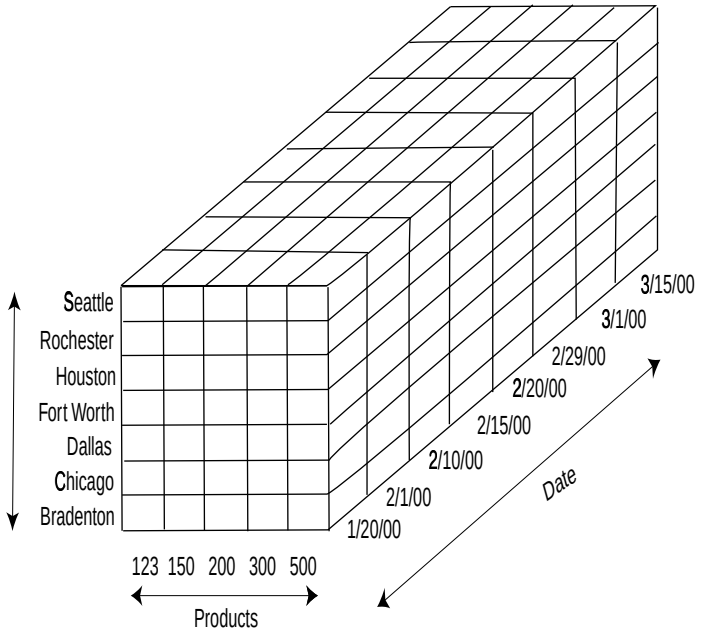


Figure 1.7: Cube

minimum prices. Figure 1.8 b) shows a hierarchical relationship among levels in the time dimension and Figure 1.8 c) shows a lattice for the location dimension. Here Day $<$ Month but Day $\not<$ Season. The aggregation can only be applied to levels which can be found in the same path as defined by the $<$ relationship. When levels for a dimension satisfy this structure, then the facts along these dimensions are said to be *additive*. If we add the sales data for all 24 hours in a day, then we get the sales data for that day. This is not always the case. Looking at the location dimension, if we were to sum up the sales data for all zip codes in a given county, however, we would not get the sales data for the county. Thus these dimensions are not additive. This is due to the fact that zip codes may span different counties. The use of non-additive dimensions complicates the drill up and drill down applications.

Multidimensional Schemas

Specialized schemas have been developed to portray multidimensional data. These include star schema, snowflake schema, and fact constellation schema.

A *star schema* shows data as a collection of two types: facts and dimensions. Unlike a relational schema which is flat, a star schema is a graphical view of the data. At the center of the star the data being examined, the facts, are shown in *fact tables* (sometimes called major tables). On the outside of the facts, each dimension is shown separately in *dimension tables* (sometimes called minor tables). The simplest star schema has one fact table with multiple dimension tables. In this case each fact points to one tuple in each of the dimensions. The actual data being accessed is stored in the fact tables and thus these tend to be quite large. Descriptive information about the dimensions is stored in the dimensions tables which tend to be smaller. Figure 1.9 shows a star schema based on the data in Figure 1.7. Here one extra dimension, division, is shown. The facts include the

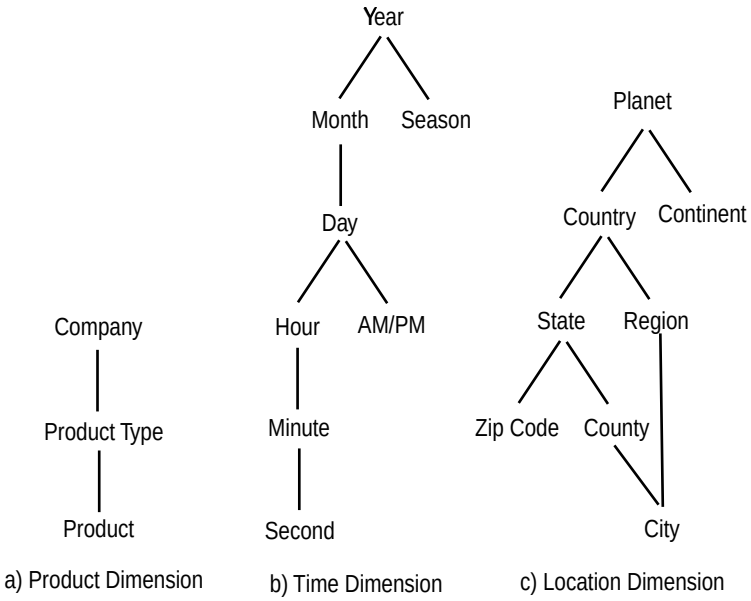


Figure 1.8: Aggregation Hierarchies

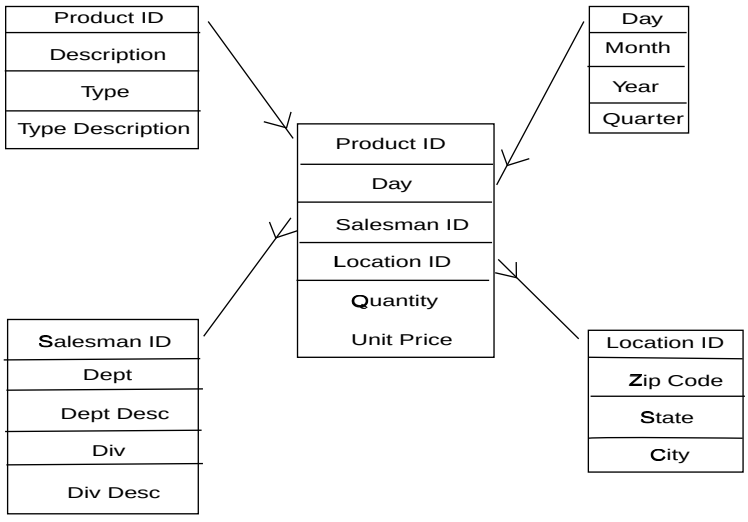


Figure 1.9: Star Schema

quantiy and price, while the dimensions are the product, time, location, and division. Descriptive information about a product includes the description, type and type description. Access to the fact table from a dimension table can be accomplished via a join between a dimension table and the fact table on particular dimension values. For example we could access all locations in Dallas by doing the following SQL query:

```
SELECT Quantity, Price
FROM Facts, Location
Where (Facts.LocationID = Location.LocationID)
and
(Location.City = "Dallas")
```

Here the LocationID is a foreign key from the facts table to the Location dimension table. The primary key for the fact table is a collection of foreign keys which point to the dimension tables. Although this example shows only one fact table, there may be several. In addition, a dimension table may itself point to another dimension table.

A star schema view can be obtained via a relational system where each dimension is a table and the facts are stored in a fact table. Through the creation of indices for the dimensions, the facts can be accessed relatively efficiently. However, the sheer volume of the data involved, exacerbated by the need to summarize the fact information at different levels across all the dimensions, complicates the process. For example, we may wish to see all sales in all regions for a particular day. Or we may want to see all sales in May 2000 for the salesmen in a specific department. To ensure efficiency of access, facts may be stored for all possible aggregation levels. The fact data would then be extended to include a level indicator.

Note that we assume that the data is stored as both fact and dimension tables. Data in the fact table can be viewed as a regular relation with an attribute for each fact to be stored and the key being the values for each dimension. There are four basic approaches to the storage of dimensional data [84]. The first technique, *Flattened*, stores the data just as it would be stored in a traditional relation. One table is used to store all information about one dimension. The dimensional data is shown as regular attribute values. The key to the data is the value for the dimension attributes at the lowet level. One tuple exists for each lowest level value in that dimension. One attribute exists for each possible level in each dimension. In addition, more attributes could be added to describe the dimension data. Figure 1.9 shows the dimension tables using a flattened approach. The Location dimension consists of levels Location ID, Zip Code, City, and State. In addition to these attributes, the flattened view shows any descriptive information about a location. We shown the Department Description for each department in the Division dimension table. With the flattened approach, a drill up is accomplished by a SUM aggregation operation over the appropriate tuples. Notice that even though this approach suffers from space problems as the number of attributes grows with the number of levels, it does facilitate the simple implementation of many DSS applications via the use of a traditional SQL aggregate operations.

The second technique to store a dimension table is called *Normalized* where a dimension table exists for each level in each dimension Each table has one tuple for every occurence at that level. As with traditional normalization, duplication is removed at the expense of creating more tables and potentially more expensive access to factual data due to the requirement of more joins. Each lower level dimension table has a foreign key pointing to the next higher level table.

Using *Expanded* dimension tables achieves the operational advantages of both the flattened and normalized views, while actually increasing the space requirements beyond that of the Flattened. The number of dimension tables is identical to that for the normalized approach and the structure of the lowest level dimension table is identical to that in the flattened technique. Each higher level dimension table has, in addition to the attributes existing for the normalized structure, attributes from all higher level dimensions.

The *Levelized* approach has one dimension table as was found with the flattened. However, the aggregations have already been performed. There is one tuple for each instance of each level in the dimension, the same number existing in all normalized tables combined. In addition, attributes are added to show the level number and key.

An extension of the star schema, *snowflake schema* facilitates more complex data views. In this case the aggregation hierarchy is shown explicity in the schema itself.

Indexing

With multidimensional data, indexes help to reduce the overhead of scanning the extremely large tables. Although the indexes used are not defined specifically to support multidimentional data, they do have inherent advantages in their use for these types of data.

With *Bitmap Indexes* each tuple in the table (fact or dimension) is represented by a predefined bit. So that a table with n tuples would be represented by a vector of n bits. The first tuple in the table is associated with the first bit, the second by the second bit, etc. There is a unique bit vector for each value in the domain. This vector indicates which associated tuples in the table have that domain value. To find the precise tuples, an address or pointer to each tuple would also have to be associated with each bit position not each vector. Bitmap indexes facilitate easy functions such as join and aggregation through the use of bit arithmetic operations. Bitmap indexes also save space over more traditional indexes where pointers to records are maintained.

Join Indexes support joins by precomputing tuples from tables which join together and pointing to the tuples in those tables. When used for multidimensional data, a common approach is to create a join index between a dimension and fact table. This facilitates the efficient identification of facts for a specific dimention level and/or value. Join indexes can be created for multiway joins across multiple dimension tables. Join indexes can be constructed using bit maps as opposed to pointers.

Tradition *B-tree* indexes may be constructed to access each entry in the fact table. Here the key would be the combination of the foreign keys to the dimension tables.

1.3.6 Data Warehousing

The term data warehouse was first used by William Inmon in 1990. He defined *data warehouse* to be a set of data which supports DSS and is "subject-oriented, integrated, time-variant, nonvolatile" [55].

With data warehousing, corporate wide data (current and historical) are merged together into a single repository. Traditional databases contain *operational data* which represents the day to day needs of a company. Traditional business data processing (such as billing, inventory control, payroll, and manufacturing support) support online transaction processing and batch reporting applications. A data warehouse, on the other hand, contains *informational data* which is used to support other functions such as planning and forecasting. Although much of the content is similar between the operational and informational data, much is different. As a matter of fact the

operational data is transformed into the informational data. Example 1.9 illustrates the difference between the two.

Example 1.9 *The ACME Manufacturing Company maintains several operational databases: sales, billing, employee, manufacturing, and warehousing. These are used to support the day to day functions such as writing paychecks, placing orders for supplies needed in the manufacturing process, billing customers, etc. The president of ACME, Stephanie Eich, wishes to streamline manufacturing in order to concentrate production on the most profitable products. To perform this task she asks several "What if" questions, does projection of current sales into the future, and examines data at different geographic and time dimensions. All the data that she needs to do this task can be found in one or more of the existing databases. However, it is not easily retrieved in the exact format that she desires. A data warehouse is created with exactly the sales information she needs by location and time. OLAP retrieval tools are provided to facilitate quick response to her questions at any and all dimension granularities.*

The data warehouse market has grown to an over $8B business supporting such diverse industries as manufacturing, retail, telecommunications, and healthcare. Think of a personnel database for a company which is continually modified as personnel are added and deleted. A personnel database which contains information about the current set of employees is sufficient. However, if management wishes to analyse trends with respect to employment history more data is needed. They may wish to determine if there is a problem with too many employees quitting. To analyze this problem they would need to know which employees have left, when they left, why they left, and other information about their employment. For management to make these typles of high level business analysis, more historical data (not just the current snapshot which is typically stored) and data from other sources (perhaps employment applications and results of exit interviews) is required. In addition, some of the data in the personnel database, such as address, may not be needed. A data warehouse provides just this information. In a nutshell, a *data warehouse* is a data repository used to support descision support systems.

The basic motivation for this shift to the strategic use of data is to increase business profitability. Traditional data processing applications support the day to day clerical and administration decisions, while data warehousing supports long term strategic decisions. A report by International Data Corporation (IDC) in 1996 stated that an average ROI in data warehousing reached 401% [10, p. 5]

Figure 1.10, adapted from [10, Figure 6.1] shows a simple view of a data warehouse. The basic components of a data warehousing system include data migration, the warehouse, and access tools. The data is extracted from operational systems, but must be reformatted, cleansed, integrated, and summarized prior to being placed in the warehouse. Much of the operational data is not needed in the warehouse (such as Employee addresses in Example 1.9) and is removed during this conversion process. This migration process is similar to that needed for data mining applications, except that data mining applications need not necessarily be performed on summarized or business wide data. The applications that are shown in Figure 1.10 to access a warehouse include traditional querying, OLAP, and data mining. Since the warehouse is stored as a database, it can be accessed by traditional query languages.

Table 1.2 summarizes the differences between operational data stored in traditional databases and the data stored in a data warehouse. The traditional database applications are related to OLTP where the users' requests are stated in a high level query language (such as SQL) and the results

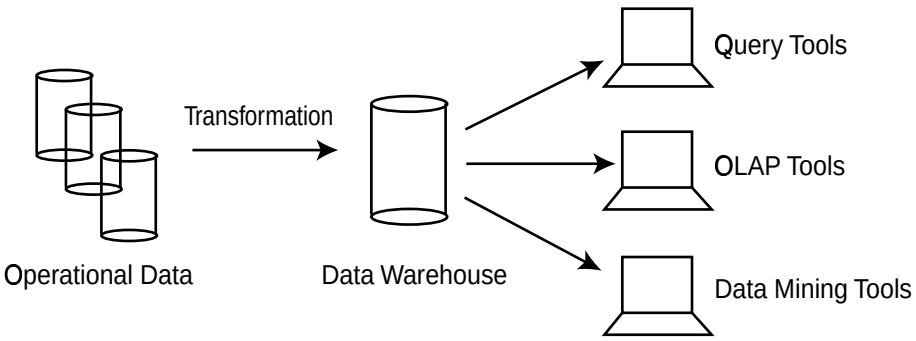


Figure 1.10: Data Warehouse

	Operational Data	Data Warehouse
Application	OLTP	OLAP
Use	Precise Queries	Ad Hoc
Temporal	Snapshot	Historical
Modification	Dynamic	Static
Orientation	Application	Business
Data	Operational Values	Integrated
Size	Gigabits	Terabits
Level	Detailed	Summarized
Access	Often	Less Often
Response	Few Seconds	Minutes
Data Schema	Relational	Star/Snowflake

Table 1.2: Comparing Operational Database to Data Warehouse

are subsets of the relations. Data warehouse applications are directly related to business decisions and analysis of the data, OLAP. While operational data usually represents facts concerning a snapshot in time (the current time), a warehouse as historical data as well. Data in the warehouse is not modified as frequently as that in a conventional database. Updates are batched and merged into the warehouse at weekly or monthly intervals. While this means that the warehouse data is not completely up to date, this is usually not a problem with the normal decision support applications. A conventional database is usually related to one application type. This is a fall out of the normalization design process used. The warehouse is associated with the business enterprise, not an application. Traditional databases may be in the order of megabytes or gigabytes, while a warehouse could be terabytes in size. The fact that conventional data is stored in many diverse formats and locations makes it inefficient to support decision support applications. OLTP users expect to get a response in a few seconds. Due to the complexity of OLAP application, their users may have to wait minutes for a response to their query.

The data transformation process required to convert operational data to informational involves many functions [10]:

- Unwanted data must be removed.
- Converting heterogeneous sources into one common schema. This problem is the same as that found when accessing data from multiple heterogenous sources. Each operational database may contain the same data with different attribute names. For example one system may use "Employee ID" while another uses "EID" for the same attribute. In addition there may be multiple data types for the same attribute.
- As the operational data is probaly a snapshot of the data, multiple snapshots may need to be merged to create the historical view.
- Summarizing data is performed to provide a highler level view of the data. This summarization may be done at multiple granularities and for different dimensions.
- New derived data (for example using age rather than birth date) may be added to better facilitate decision support functions.
- Handling missing and eroneous data needs to be performed. This could entail replacing them with predicted or default values or simply removing these entries.

The portions of the transformation which deal with ensuring valid and consistent data is sometimes referred to as *Data scrubbing* or *Data Staging*.

There are many benefits to the use of a data warehouse. Since it provides an integration of data from multiple sources, its use can provide more efficient access of the data. The data that is stored often provides different levels of summarization. For example, sales data may be found at a low level (purchase order), at a city level (total of sales for a city), or at higher levels (county, state, country, world). The summary can be provided for different types of granularity. The sales data could be summarized by salesman and department as well. These summarizations are provided by the conversion process instead of being calculated when the data is accessed. Thus this also speeds up the processing of the data for decision support applications.

The data warehouse may appear to increase the complexity of database mangagement due to the fact that it is a replica of the operational data. But keep in mind, much of the data in the

warehouse is not simply a replication but an extension to or aggregation of that data. In addition, due to the fact that the data warehouse contains historical data data stored there will probably have a longer life span than the snapshot data found in the operational databases. The fact that the data in the warehouse need not be kept consistent with the current operational data also simplifies its maintenance. The benefits obtained by the capabilities (eg. DSS support) provided are usually deemed to outweigh any disadvantages.

A subset of the complete data warehouse, *data mart*, may be stored and accessed separately. The level is at a departmental, regional, or functional level. These separate data marts are much smaller, and more efficiently support narrower analytical types of applications.

A *virtual warehouse* is a warehouse implemented as a view from the operational data. While some of this view may actually be materialized for efficiency, it need not all be.

There are several ways to improve the performance of data warehouse applications [96].

- Summarization: Since many applications require summary type information, data which is known to be needed for consolidation queries should be presummarized before storage. To improve performance, different levels of summarization should be included. With a 20-100 % increase in storage space, an increase in performance of two to ten times can be acheived [96, p 80].

- Denormalization: Traditional normalization reduces such problems as redundancy, and insert, update, and deletion anomalies. However, these improvements are achieved at the cost of increase processing time due to joins. With a data warehouse, improved performance can be achieved by storing denormalized data. Since data warehouses are not usually updated as frequently as operational data, the negatives associated with update operations are not really an issue.

- Partitioning: Dividing the data warehouse into smaller fragments may reduce processing time by allowing queries to acces small data sets.

The relationship between data mining and data warehousing can be viewed as a symbiotic one [56]. Data used in data mining applications is often slightly modified from that in the databases where it permanently resides. The same is true for data in a data warehouse. When data is placed in a warehouse, it is extracted from the database, cleansed, and reformatted. The fact that the data is derived from multiple sources, with heterogenous formate complicates the problem. In addition, the fact that the source databases are updated requires that the warehouse be updated periodically or work with stale data. These issues are identical to many of those associated with data mining and KDD. (see Figure 1.3). While data mining and data warehousing are actually orthogonal issues, they are complementary. Due to the types of applications and massive amount of data in a data warehouse, data mining applications can be used to provide meaningful information needed for decision support systems. For example, management may use the results of classification or association rule applications to help determine the target population for an advertising campaign. In addition, data mining activities can benefit from the use of data in a data warehouse. However its use is not required. Data warehousing and data mining are sometimes thought of as the same thing. Even though they are related, they are different and each can be used without the other.

1.3.7 OLAP

OnLine Analytic Processing (OLAP) systems are targeted to provide more complex query results

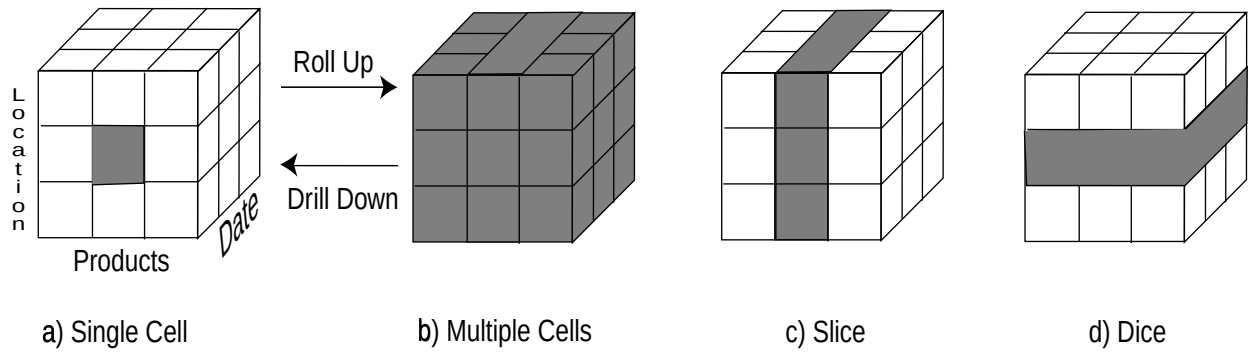


Figure 1.11: OLAP Operations

than traditional OLTP or database systems. Unlike database queries, however, OLAP applications usually involve analysis of the data itself. They can be thought of as an extension of some of the basic aggregation functions available in SQL. This extra analysis of the data as well as the more imprecise nature of the OLAP queries is what really differentiates OLAP applications from traditional database and OLTP applications. OLAP tools may also be used in DSS systems.

OLAP is performed on data warehouses or data marts. The primary goal of OLAP is to support ad hoc querying needed to support DSS. The multidimentional view of data is fundamental to OLAP applications. OLAP is an application view - not a data structure or schema. The complex nature of OLAP applications requires a multidimensional view of the data. The type of data accessed is often (although not a requirement) a data warehouse.

OLAP tools can be classified as ROLAP or MOLAP. With *MOLAP (Multidimensional OLAP)*, data is modeled, viewed, and physically stored in a *Multidimensional Database (MDD)*. MOLAP tools are supported by specialized DBMS and software systems capable of supporting the multidimensional data directly. With MOLAP, data is stored as an n-dimensional array (assuming there are n dimensions), thus the cube view is stored directly. Although MOLAP has extremely high storage requirements, indexes are used to speed up proceesing. With *ROLAP (Relational OLAP)*, on the other hand, data is stored in a relational database and a ROLAP server (middleware) creates the multidimensional view for the user. As one would think, the ROLAP tools tend to be less complex, but also less efficient. MDD systems may presummarize along all dimensions. A third approach, *Hybrid OLAP (HOLAP)*, combines the best features of ROLAP and MOLAP together. Queries are stated in multidimensional terms. Data which is not updated frequently will be stored as MDD, while data which is updated frequently will be stored as RDB.

As seen in Figure 1.11, there are several types of OLAP operations supported by OLAP tools:

- A simple query may look at a single cell within the cube (Figure 1.11 a)).
- Slice: Look at a subcube to get more specific information. As seen in Figure 1.11 c), this is looking at a portion of the cube.
- Dice: Rotate cube to look at another dimension. In Figure 1.11 d) a dice is made since the view in c) is rotated from all cells for one product to all cells for one location.
- Roll Up (Dimension Reduction, Aggregation): These allow the user to ask questions that move up an aggregation hierarchy. Figure 1.11 b) represents a roll up from a). Instead of

Area	Query	Data	Results	Output
DB/OLTP	Precise	Database	Precise	DB Objects or Aggregation
IR	Precise	Documents	Vague	DB Objects
OLAP	Analysis	Multidimensional	Precise	DB Objects or Aggregation
DM	Vague	Preprocessed	Vague	KDD Objects

Table 1.3: Relationship Between Topics

looking at one single fact, we look at all facts. Thus could for example look at the overall total sales for the company.

- Drill Down: Figure 1.11 a) represents a drill down from b). These functions allow a user to get more detailed fact information by navigating lower in the aggregation hierarchy. We could perhaps look at quantities sold within a specific area of each of the cities.
- Visualization: These operations allow the OLAP users to actually "see" results of an operation.

To assist with Roll Up and Drill Down operations, frequently used aggregations can be precomputed and stored in the warehouse.

1.3.8 Web Search Engines

Web search engines can be viewed as a type of query system much like IR systems. As with IR queries, search engine queries can be stated as keyword, boolean, weighted, etc. The difference is primarily in the data being searched, pages with heterogeneous data and extensive hyperlinks, and the architecture involved.

1.3.9 Data Mining

When viewed as a query system, data mining queries extend both database and IR concepts. Data mining problems are often ill-posed with many different solutions. Judging the effectiveness of the result of a data mining request is often difficult. A major difference between data mining queries and those of earlier types is the output. Basic database queries always output either a subset of the database or aggregates of the data. A data mining query outputs a KDD object. A KDD object is either a rule, classifier, or clustering [53]. These objects do not exist prior to executing the query nor or they part of the database being queried. Table 1.3 compares the different query systems. Aggregation operators have existed in SQL for years. They do not return objects existing in the database, but return a model of the data. For example an average operator returns the average of a set of attribute values rather than the values themselves. This is really a simple type of data mining operator.

1.3.10 Statistics

Such simple statistical concepts as determining a data distribution, and calculating a mean and variance can be viewed as data mining techniques. Each of these is, in its own right, a descriptive model for the data under consideration.

Part of the data mining modeling process requires searching the data itself. An equally important part requires inferencing from the results of the search to a general model. A current database state may be thought of as a sample (albeit large) of the real data which may not be stored electronically. When a model is generated, the goal is to fit it to the entire data not just that which was searched. Any model derived should be statistically significant, meaningful, and valid. This problem may be compounded by the preprocessing step of the KDD process which may actually remove some of the data itself. Outliers compound this problem. The fact that most database practitioners and users are not probability experts also complicates the issues. The tools need to make the computationally difficult problems tractable may actually invalid the results. Assumptions often made about independence of data may be incorrect, thus leading to errors in the resulting model.

An often used tool in data mining and machine learning is that of sampling. Here a subset of the total population is examined and from this subset a generalization (model) about the entire population is made. In statistics this approach is refered to as *statistical inference*. Of course this generalization can lead to errors in the final model caused by the sampling process.

The term *Exploratory Data Analysis* was actually coined by statisticians to describe the fact that the data can actually drive the creation of the model and any statistical characteristics. This seems contradictory to the traditional statistical view that one should not be corrupted or influenced by looking at the data prior to creating the model. This would unnecesarily bias any resulting hypothesis. Of course database practitioners would never think of creating a model of the data without looking at the data in detail first and then creating a schema to describe it.

Some data mining applications determine correlations among data. These relationships, however, are not causal in nature. A discovered association rule may show that 60buy beer. Care must be taken when assigning any significance to this relationship. We do not know why these items were purchased together. Perhaps the beer was on sale or it was an extremely hot day. There may be no relationship between these two data items - except that they were often purchased together. There need not be any probabalistic inference which can be deduced.

Statistics research has produced many of the proposed data mining algorithms. The difference lies in the goals, the fact that statisticians may deal with smaller more formatted data sets, and the emphasis of data mining on the use of machine learning techqnies. However, it is often the case that the term data mining is used in a derogatory manner by statisticians as data mining is "analysis without any clearly formulated hypotheses" [71]. Indeed this may be the case as data itself, not a predefined hypothesis, is the guide.

Probability distributions can be used to describe the domains found for different data attributes. Statistical inference techniques can be viewed as special estimators and prediction methods. Use of these approaches may not always be applicable as the precise distribution of real data values may not actually follow any specific probability distribution, assumptions on the independence of attributes may be invalid, and some heuristic based estimator techniques may never actually converge.

It has been stated that "The main difference between data mining and statistics is that data mining is meant to be used by the business user - not the statistician" [10, p336]. As such, data mining (particulary from a database pespective) involves not only modeling, but the development of effective and efficient algorithms (and data structures) to perform the modeling on large datasets.

1.3.11 Artifical Intelligence

Artificial Intelligence (AI) includes many DM techniques such as neural networks and classification. However, AI is more general and involves areas outside traditional data mining. AI applications may also not be concerned with scalability as datasets may be small.

Machine Learning is the area of Artifical Intelligence that examines how to write programs that can learn. In data mining, machine learning is often used for prediction or classification. With machine learning the computer makes a prediction and then based on feedback as to whether it is correct or not "learns" from this feedback. It learns through examples, domain knowledge, and feedback. When a similar situation arises in the future this feedback is used to make the same prediction or to make a completely different one. Statistics are very important in machine learning programs as the results of the predictions must be statistically significant and perform better than a naive prediction. Applications typically using machine learning techniques include speech recognition, training moving robots, classification of astronomical structures, and game playing [78].

When machine learning is applied to data mining tasks, a model is used to represent the data (such as a graphical structure like a neural network or a decision tree). During the learning process, a sample of the database is used to train the system to properly perform the desired task. Then the system is applied to the general database to actually perform the task. This predictive modeling approach is divided into two phases. During the training phase, historical or sampled data is used to create a model which represents that data. It is assumed that this model is representative not only for this sample data, but for the database as a whole and future data as well. The testing phase then applies this model to the remaining and future data.

A basic machine learning application includes several major aspects [78]. First an appropriate training set must be chosen. The quality of the training data determines how well the program learns. In addition, the type of feedback available is important. Direct feedback entails specific information about the results and impact of each possible move or database state). Indirect feedback is at a higher level with no specific information about individual moves or predictions. An important aspect is whether or not the learning program can actually propose new moves or database states. Another major feature that impacts the quality of learning is how representative the training set is of the overall final system to be examined. If a program is to be designed to perform speech recognition, it is hoped that the system is allowed to learn with a large sample of the speech patterns it will encounter during its actually processing.

There are two different types of machine learning: *supervised learning* and *unsupervised learning*. A supervised approach learns by example. Given a training set of data plus correct answers, the computational model successively applies each entry in the training set. Based on its ability to correctly handle each of these, the model is changed to ensure that it works better with this entry if it were applied again. Given enough input values, the model will learn the correct behavior for any potential entry. With unsupervised data, data exists but there is no knowledge of the correct answer of applying the model to the data.

Although machine learning is the basis for many of the core data mining research topics, there is a major difference between the approaches taken by the AI and database disciplines. Much of the machine learning research has focused on the learning portion rather than the creation of useful information (prediction) for the user. Also, machine learning looks at things that may be difficult for humans to do or concentrates on how to develop learning techniques that can mimic human

Database Management	Machine Learning
Database is an active evolving entity	Database is static
Records may contain erroneous or missing data	Databases are complete and noise free
Typical field is numeric	Typical feature is binary
Database contains millions of records	Database contains hundreds of instances
AI should get down to reality	All databas problems have been solved

Table 1.4: Relationship Between Topics

behavior. The objective for data mining is to uncover information which can be used to provide information to humans (not take their place). These two conflicting views are summarized in Table 1.4 originally appearing in [34]. The items listed in the first column indicate the concerns and views that are taken in this book. Many of the algorithms introduced in this text were created in the AI community and are now being exported into more realistic data mining activities. When applying these machine learning concepts to databases, additional concerns and problems are raised: size, complex data types, complex data relationships, noisy and missy data, and databases which are frequently updated.

1.3.12 Pattern Matching

Pattern Matching finds occurences of a predefined pattern in data. Pattern Matching is used in many diverse applications. A text editor uses pattern matching to find occurences of a string in the text being edited. Information retrieval and web search engines may use pattern matching to find documents containing a predefined pattern (perhaps a keyword). Time series analyses examine the patterns of behavior in data obtained from two different time series to determine similarity. Pattern matching can be viewed as a type of classification where the predefined patterns are the classes under consideration. The data is then placed in the correct class based on a similarity between the data and the classes.

1.4 The Development of Data Mining

The current evolution of data mining functions and products is the result of years of influence from many disciplines including databases, information retrieval, statistics, algorithms, and artificial intelligence (see Figure 1.12). Unlike previous research in these disparate areas, a major interest from the database community is to combine these seemingly different disciplines into one unifying data/algorithmic approach. Although in its infancy, the ultimate goal of this evolution is to develop a "big picture" view of the area which will facilitate integration of the various types of applications into real world user domains.

Table 1.5 shows developments in the areas of AI, IR, databases, and statistics leading to the current view of data mining. These different historical influences which have led to the development of the total data mining area, have given rise to different views of what data mining functions actually are[87]:

- **Induction** is used to proceed from very specifc knowledge to more general information. These

Time	Area	Contribution	Reference
Late 1700s	Stat	Bayes Theorem of Probability	[9]
1900	Stat	Regression Analysis	
Early 1920s	Stat	Maximum Likelihood Estimate	[30]
Early 1940s	AI	Neural Networks	[75]
Early 1950s		Nearest Neighbor	[31]
Early 1950s		Sinle Link	[32]
Late 1950s	AI	Perceptron	[89]
Late 1950s	Stat	Resampling, bias reduction, jackknife estimator	
Early 1960s	AI	ML started	[29]
Early 1960s	DB	Batch reports	
Mid 1960s		Decision Trees	[52]
Mid 1960s	Stat	Linear Models for Classification	[81]
	IR	Similarity Measures	
	IR	Clustering	
	Stat	Exploratory Data Analysis (EDA)	
Lte 1960s	DB	Relational data model	[21]
Early 1970s	IR	SMART IR systems	[91]
Mid 1970s	IR	Genetic Algorithms	[49]
Late 1970s	Stat	Estimation with Incomplete Data (EM algorithm)	[23]
Late 1970s	Stat	K-Means Clustering	
Early 1980s	AI	Kohonen Self Organizing Map	[63]
Mid 1980s	AI	Decision Tree Algorithms	[85]
Early 1990s	DB	Association Rule Algorithms	
		Web and search engines	
1990s	DB	Data warehousing	
1990s	DB	OLAP	

Table 1.5: Time Line of Data Mining Development

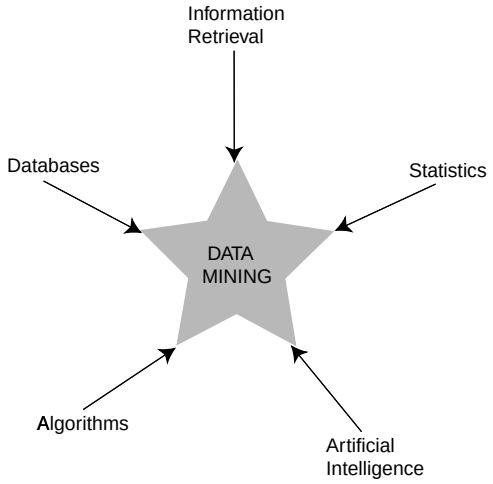


Figure 1.12: Historical Perspective of Data Mining

types of techniques are often found in AI applications.

- As the primary objective of data mining is one of describing some characterics of a set of data by a general model, this approach can be viewed as a type of **Compression**. Here the detailed data within the database is abstracted and compressed to a smaller description of the data characteristics that are found in the model.

- As stated earlier, the data mining process itself can be viewed as a type of **Querying** the underlying database. Indeed an ongoing direction of data mining research is into how to define a data mining query and whether or not a query language (like SQL) can be developed to capture the many different types of data mining queries.

- Describing a large database can be viewed as using **Approximation** to help in uncovering hidden information about the data.

- When dealing with large databases, the impact of size and efficiency of developing an abstract model can be thought of as a type of **Search** problem.

It is interesting to think about the various data mining problems and how each may be viewed in several different perspectives based on the viewpoint and background of the researcher/developer. We mention these different perspectives only to give the reader the "full picture" of data mining. Often, due to the varied backgrounds of the data mining participants, we find the same problem (and perhaps even the same solution) are described differently. Indeed different terminologies can lead to misunderstandings and apprehension among the different players. You can see statisticians voice concern over the compounded use of estimates (approximation) with results being generalized when they shouldn't be. Database researchers often voice concern about the inefficiency of many proposed AI algorithms particularly when used on very large databases. IR and those interested in data mining of textual databases may be concerned about the fact that many algorithms are only targeted to numeric data. The approach taken in this book is to look at Data Mining contributions from all these different disciplines together.

There are at least two issues which distinguish a database perspective of examining data mining concepts: efficiency and scalability. Any solutions to data mining problems must be able to perform well against real world databases. As part of the efficiency we are concerned about both the algorithms and data structures used. To improve efficiency, parallelization may be used. In addition, how the proposed algorithms behave as the associated database is updated is also important. Many proposed data mining algorithms may work well against a static database, but may be extremely inefficient as changes are made to the database. As database practitioners we are interested in how algorithms perform against very large databases, not "toy" problems. We also usually assume that the data is stored on disk and may even be distributed.

1.5 Data Mining Issues

There are many important implementation issues associated with data mining:

- **Human Interaction:** Since data mining problems are often not precisely stated, interface may be needed with both domain and technical experts. The technical expert is used to formulate the queries and assist in interpreting the results. The users are needed to identify training data and results desired.
- **Overfitting:** When a model is generated that is associated with a given database state, it is desirable that the model also fit future database states. Overfitting occurs when the model does not fit future states. This may be caused by assumptions that are made about the data or may simply be caused by the small size of the training database. For example, a classification model for an employee database may be developed to classify employees as short, medium, or tall. If the database is quite small the model might erroneously indicate that a short person is anyone under five feet eight inches because there is only one entry in the database under five feet eight. In this case many future employees would be erroneously classified as short. Overfitting can arise under other circumstances as well even though the data is not changing.
- **Outliers:** There are often many data entries that do not fit nicely into the derived model. With very large databases this becomes even more of an issue. If a model is developed which includes these outliers, then the model may not behave well for data which are not outliers.
- **Interpretation of Results:** Currently, data mining output may require the use of experts to correctly interpret the results. They might be otherwise meaningless to the average database user.
- **Visulization of Results:** To easily view and understand the output of data mining algorithms, visualization of the results is helpful.
- **Large Datasets:** The massive datasets associated with data mining problems create problems when applying algorithms designed for small datasets. Many modeling applications grow exponentially on the dataset size and are thus too inefficient for larger datasets. Sampling and parallelization have been shown to be effective tools to attack this problem.
- **High Dimensionality:** An issue similar to that of the dataset size is the large number of attributes needed to model the data. Some of the same approaches may be used to attack this

problem. In addition, this high dimensionality problem may be addressed by dimensionality reduction techniques. However, as with sampling, reduction may produce invalid results.

- **Multimedia Data:** Most previous data mining algorithms are targeted to traditional data types (numeric, character, text, etc.). The use of multimedia data such as is found in GIS databases complicates or invalidates many proposed algorithms.
- **Missing Data:** During the preprocessing phase of KDD, data which is missing may be replaced with estimates. This and other approaches to handle missing data can lead to invalid results in the data mining step.
- **Irrelevant Data:** Some attributes in the database may not be of interest to the data mining task being developed.
- **Noisy Data:** Some attribute values may be invalid or incorrect. These are often corrected prior to running data mining applications.
- **Changing Data:** Databases can not be assumed to be static. However, most data mining algorithms do assume a static database. This requires that the algorithm be completely rerun anytime the database changes.
- **Integration:** The KDD process is not currently integrated into normal data processing activities. KDD requests may be treated as special unusual one time needs. This makes them inefficient, ineffective, and not general enough to be used in an ongoing basis. Integration of data mining functions into traditional DBMS systems is certainly a desirable goal.
- **Application:** The intended use for the information obtained form the data mining function is a challenge to determine. Indeed, how business executives can effectively use the output is sometimes considered the more difficult part, not the running of the algorithms themselves. Because the data is of a type that has not been previously known, business practices may have to be modified to determine how to effectively use the information uncovered.

1.6 Data Mining Metrics

How does one measure the effectiveness or usefulness of a data mining approach? There are different levels that one could look at. From an overall business or usefulness perspective a measure such as ROI could be used. Of course this would be difficult to measure as the return is hard to quantify. It could be measured as increased sales, reduced advertising expenditure, or both. In a specific advertising campaign implemented via targeted catalog mailings, the percentage of catalog recipients and the amount of purchase per recipient would provide one means to measure the effectiveness of the mailings.

In this text, however, we will use a more computer science/database perspective to measuring various data mining approaches. We assume that the business management has determined that a particular data mining application be made. They will subsequently determine the overall effectiveness of the approach using some ROI (or related) strategy. Our objective is to compare different alternatives to implementing a specific data mining task. The metrics used include the traditional ones of space and time based on complexity analysis. In some cases more specific metrics may be used.

1.7 The Future

The advent of the relational data model and SQL were major milestones in the evolution of database systems. Currently data mining is little more than a set of tools which can be used to uncover previously hidden information in a database. While there are many tools to aid in this process there is no all encompassing model or approach. Over the next few years not only will there be more efficient algorithms with better interface techniques, but steps will be taken to develop an all encompassing model for data mining. While not looking like the relational model, it will probably include similar items: algorithms, data model, metrics for goodness (like normal forms). Current data mining tools require much human interaction to not only define the request, but also to interpret the results. As the tools become better and more intergrated, this is likely to decrease. The various data mining applications are of many diverse types, so the development of a comlete data mining model is desirable. A major development will be the creation of a sophisticated "query language" which includes traditional SQL functions but also more complicated requests such as those found in OLAP and data mining applications.

The term *Knowledge and Data Discovery Management System (KDDMS)* has been coined [53] to describe the future generation of data mining systems which include not only data mining tools but techniques to manage the underlying data, ensure its consistency, and provide concurrency and recovery features. A KDDMS will provide access via ad hoc data mining queries which have been optimized for efficient access.

1.8 Bibliographic Notes

There have been several recent surveys and overviews of data mining including special issues of *The Communications of the ACM* in November 1996 and November 1999, and *Computer* in August 1999. Other survey articles can be found: [16]. [28], [27], [71], [72], [22], [36], and [87]. There have also been several tutorials surveying data mining concepts: [3], [4], and [88]. The idea of developing an approach unifying all data mining activities has been proposed in [27], [71], and [72].

Chapter 2

DATA MINING TECHNIQUES

2.1 Introduction

There are many different methods used to peform data mining tasks. These techniques require not only specific types of data structures, but imply certain types of algorithmic approaches. In this chapter we briefly introduce some of the common data mining techniques. These will all be examined in more detail in later chapters of the book as they are used by specific approaches.

2.2 Parametric vs. Nonparametric Models

Parametric Models describe the relationship between input and output through the use of algebraic equations where some parameters are not specified. These unspecific parameters are determined by providing input examples. One of the most common forms of a parametric model is linear regression. *Linear Regression* assumes that a linear relationship between the input and output data exists. The common formula for a linear relationship is used in this model:

$$y = c_0 + c_1 x_1 + ... + c_n x_n \tag{2.1}$$

Here there are n input variables, called *predictors* or *regressors*; one output variable (the one being predicted), called the *response*; and $n + 1$ constants which are chosen during the modeling process to match the input examples (or sample). This is sometimes called *multiple linear regression* as there is more than one predictor.

Even though parametric modeling is a nice theoretical topic and can sometimes be used, often it is either too simplistic or requires more knowledge about the data involved than is available. Thus for real world problems, these parametric models may not be used for large prediction applications. Example 2.1 is an example of the use of linear regression.

Example 2.1 *It is known that a state has a fixed sales tax, but it is not known what the amount happens to be. The problem is to derive the equation that derives the amount of sales tax given an input purchase amount. We can state the desired linear equation to be* $y = c_0 + c_1 x_1$. *So we really only need to have two samples of actual data to determine the values of* c_0 *and* c_1. *Suppose that we know* $< 10, 0.5 >$ *and* $< 25, 1.25 >$ *are actual purchase amount, tax amount pairs. using these data points we easily determine that* $c_0 = 0$ *and* $c_1 = 0.05$.

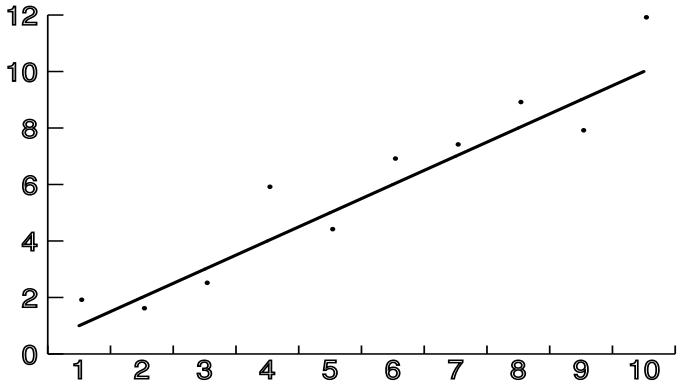


Figure 2.1: Simple Linear Regression

Admittedly, Example 2.1 is an extremely simple problem. However, it illustrates how we all use the basic classification/prediction techniques frequently. Figure 2.1 illustrates the more general use of linear regression with one input value. Here we have a sample of data that we wish to model using a linear model. The line generated by the linear regression technique is shown in the figure. Notice, however, that the actual data points do not usually fit the linear model exactly. Thus this model is an estimate of what the actual input/output relationship is. We can use the generated linear model to predict an output value given an input value, but unlike that for Example 2.1 the prediction would be an estimate rather than the actual output value.

Nonparametric techniques are more appropriate for data mining applications. A *nonparametric model* is one that is data driven. No explicit equations are used to determine the model. This means that the modeling process adapts to the data at hand. Unlike parametric modeling where a specific model is assumed ahead of time, the nonparametric techniques create a model based on the input. While the parametric methods require more knowledge about the data prior to the modeling process, the nonparametric technique requires a large amount of data as input to the modeling process itself. The modeling process then creates the model by sifting through the data. Recent nonparametric techniques have employed machine learning techniques to be able to learn dynamically as data is added to the input. Thus, the more data the better the model created. Also, this dynamic learning process allows the model to be created continuously as the the data is input. These features make nonparametric techniques particularly suitable to database applications with large amounts of dynamically changing data. Nonparametric techniques include neural networks, decision trees, and genetic algorithms.

2.3 A Statistical Perspective on Data Mining

There have been many statistical concepts which are the basis for data mining techniques. We briefly review some of these.

2.3.1 Point Estimation

Point estimation refers to the process of estimating a population parameter, Θ, by an estimate of the parameter, $\hat{\Theta}$. This can be done to estimate mean, variance, standard deviation, or any

other statistical paramater. Often the estimate of the parameter for a general population may be made by actually calculating the parameter value for a population sample. An estimator technique may also be used to estimate (predict) the value of missing data. The *bias* of an estimator is the difference between the expected value of the estimator and the actual value:

$$Bias = E(\hat{\Theta}) - \Theta \tag{2.2}$$

An *unbiased* estimator is one whose bias is 0. While point estimators for small datasets may actually be unbiased, for larger database applications we would expect that most estimators are biased. One measure of the effectiveness of an estimate is the *mean squared error (MSE)* defined as the expected value of the squared difference between the estimate and the actual value:

$$MSE(\hat{\Theta}) = E(\hat{\Theta} - \Theta)^2 \tag{2.3}$$

The *squared error* is often examined for specific prediction to measure accuracy rather than looking at the average. For example, if the true value for an attribute were 10 and the prediction was 5, the squared error would be $(5-10)^2 = 25$. The squaring is performed to ensure that the measure is always positive and to also give a higher weighting to the esimates that are grossly inaccurate. As we will see, the mean squared error is commonly used in evaluating the effectivenes of data mining prediction techniques. It is also important in machine learning. At times, instead of predicting a simple point estimate for a parameter one may determine a range of values within which the true parameter value should fall. This range is called a *confidence interval.*

A popular estimating technique is the *Jackknife Estimate.* With this approach, the estimate of a parameter, $\hat{\theta}$, is obtained by omitting one value from the set of observed values. Suppose that there is a set of n values $X = \{x_1, ..., x_n\}$. An estimate for the mean would be:

$$\hat{\mu}_{(i)} = \frac{\sum_{j=1}^{i-1} x_j + \sum_{j=i+1}^{n} x_j}{n-1} \tag{2.4}$$

Here the subscript of (i) indicates that this estimate is obtained by omitting the i^{th} value. Given a set of jackknife estimates,$\hat{\theta}_{(i)}$, these can in turn be used to obtain an overall estimate:

$$\hat{\theta}_{(.)} = \frac{\sum_{j=1}^{n} \hat{\theta}_{(j)}}{n} \tag{2.5}$$

This can then be used to estimate the bias:

$$\widehat{Bias}_{jack} = (n-1)\ (\hat{\theta}_{(.)} - \hat{\theta}) \tag{2.6}$$

The jackknife estimate of the standard deviation is:

$$\hat{\sigma}_{jack} = (\frac{n-1}{n}\ \sum_{j=1}^{n} (\hat{\mu}_{(j)} - \hat{\mu}_{(.)})^2)^{1/2} \tag{2.7}$$

Example 2.2 *Suppose that a coin is tossed in the air five times with the following results (H indicates a head and T indicates a tail): { H, H, H, H, T }. If we assume that the coin toss follows the Bernoulli distribution where H is represented as a 1 and T as a 0, then we know that*

$$f(x_i \mid p) = p^{x_i}\ (1-p)^{1-x_i}. \tag{2.8}$$

Assuming a perfect coin when the probability of H and T are both 1/2 the likelihood is then:

$$L(p \mid H, H, H, H, T) = \prod_{i=1}^{5} 0.5 = 0.03. \tag{2.9}$$

However if the coin is not perfect but has a bias towards heads such that the probability of getting a Head is 0.8 then the likelihood is:

$$L(p \mid H, H, H, H, T) = 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.2 = 0.08. \tag{2.10}$$

Here it is more likely that the coin is biased towards getting a head than it is that it is not biased. The general formula for likelihood is:

$$L(p \mid x_1, ..., x_5) = \prod_{i=1}^{5} p^{x_i} \ (1-p)^{1-x_i} = p^{\sum_{i=1}^{5} x_i} \ (1-p)^{5-\sum_{i=1}^{5} x_i}. \tag{2.11}$$

By taking the log we get

$$l(p) = logL(p) = \sum_{i=1}^{5} x_i log(p) + (5 - \sum_{i=1}^{5} x_i) \ log(1-p) \tag{2.12}$$

and then we take the derivative with respect to p:

$$\frac{\partial l(p)}{\partial p} = \sum_{i=1}^{5} \frac{x_i}{p} - \frac{5 - \sum_{i=1}^{5} x_i}{1-p}. \tag{2.13}$$

Setting equal to zero we finally obtain:

$$p = \frac{\sum_{i=1}^{5} x_i}{5} \tag{2.14}$$

For this example the estimate for p is then $\hat{p} = \frac{4}{5} = 0.8$.

Another technique for point estimation is called the *Maximum Likelihood Estimate (MLE). Likelihood* can be defined as a value proportional to the actual probability that with a specific distribution the given sample exists. So the sample gives us an estimate for a parameter from the distribution. The higher the likelihood value, the more likely the underlying distribution will produce the results observed. Given a sample set of values $X = \{x_1, ..., x_n\}$ from a known distribution function $f(x_i \mid \Theta)$, the MLE can estimate parameters for the population from which the sample is drawn. The approach obtains parameter estimates which maximize the probability that the sample data occurs for the specific model. It looks at the joint probability for observing the sample data by multiplying the individual probabilitites. The likelihood function, L, is thus defined as:

$$L(\Theta \mid x_1, ..., x_n) = \prod_{i=1}^{n} f(x_i \mid \Theta) \tag{2.15}$$

The value of Θ which maximizes L is the estimate chosen. This can be found by taking the derivative (perhaps after finding the log of each side to simplify the formula) with respect to Θ. Example 2.2 illustrates the use of MLE.

Algorithm 2.1
Input:
$\Theta = \{\theta_1, ..., \theta_p\}$ *//Parameters to be Estimated*
$X_{obs} = \{x_1, ..., x_k\}$ *//Input Database Values Observed*
$X_{miss} = \{x_{k+1}, ..., x_n\}$ *//Input Database Values Missing*
Output:
$\hat{\Theta}$ *//Estimates for* Θ
EM Algorithm:
i := 0;
Obtain initial parameter MLE estimate,$\hat{\Theta}^i$*;*
repeat
Estimate missing data,$\hat{X}^i_{miss}$*;*
i++;
Obtain next parameter estimate,$\hat{\theta}^i$ *to maximize data;*
until *estimate converges;*

The *Expectation-Maximization (EM)* Algorithm is an approach which solves the estimation problem with incomplete data [23]. The EM algorithm finds an MLE for a parameter (such as mean) using a two step process: Estimation and Maximization. The basic EM algorithm is shown in Algorithm 2.1. An initial set of estimates for the parameters is obtained. Given these estimates and the training data as input, the algorithm then calculates a value for the missing data. For example it might use the estimated mean to predict a missing value. This data (with the new value added) is then used to determine an estimate for the mean which maximizes this set of data. These steps are applied iteratively until successive parameter estimates converge. Any approach can be used to find the initial parameter estimates. In Algorithm 2.1 it is assumed that the input database has actual observed values $X_{obs} = \{x_1, ..., x_k\}$ as well as values that are missing $X_{miss} = \{x_{k+1}, ..., x_n\}$. We assume that the entire database is actually $X = X_{obs} \cup X_{miss}$. The parameters to be estimated are $\Theta = \{\theta_1, ..., \theta_p\}$. The likelihood function is defined by:

$$L(\Theta \mid X) = \prod_{i=1}^{n} f(x_i \mid \Theta) \tag{2.16}$$

We are looking for the Θ which maximized L. The MLE of Θ are the estimates which satisfy:

$$\frac{\partial ln L(\Theta \mid X)}{\partial \theta_i} = 0 \tag{2.17}$$

The Expectation part of the algorithm estimates the missing values using the current estimates of Θ. Notice that this can initially be done by finding a weighted average of the observed data. The Maximization step then finds the new estimates for the Θ parameters which maximize the likelihood by using those estimates of the missing data. An illustrative example of the EM algorithm is shown in Example 2.3.

Example 2.3 *We wish to find the mean,* μ*, for data which follows the normal distribution where the known data is* $\{1, 5, 10, 4\}$ *with two data items missing. Here* $n = 6$ *and* $k = 4$*. Suppose that*

we initially guess $\hat{\mu}^0 = 3$. *We then use this value for the two missing values. Using this we obtain MLE estimate for the mean as:*

$$\hat{\mu}^1 = \frac{\sum_{i=1}^{k} x_i}{n} + \frac{\sum_{i=k+1}^{n} x_i}{n} = 3.33 + \frac{3+3}{6} = 4.33 \tag{2.18}$$

We now repeat using this as the new value for the missing items, then estimate the mean as:

$$\hat{\mu}^2 = \frac{\sum_{i=1}^{k} x_i}{n} + \frac{\sum_{i=k+1}^{n} x_i}{n} = 3.33 + \frac{4.33+4.33}{6} = 4.77 \tag{2.19}$$

Repeating we obtain:

$$\hat{\mu}^3 = \frac{\sum_{i=1}^{k} x_i}{n} + \frac{\sum_{i=k+1}^{n} x_i}{n} = 3.33 + \frac{4.77+4.77}{6} = 4.92 \tag{2.20}$$

and then

$$\hat{\mu}^4 = \frac{\sum_{i=1}^{k} x_i}{n} + \frac{\sum_{i=k+1}^{n} x_i}{n} = 3.33 + \frac{4.92+4.92}{6} = 4.97 \tag{2.21}$$

We decide to stop here as the last two estimates are only 0.05 apart. Thus our estimate is $\hat{\mu} = 4.97$.

One of the basic guidelines in estimating is *Ockham's Razor* [1] which basically states that simpler models generally yield the best results [54].

2.3.2 Models Based on Summarization

There are many basic concepts which provide an abstraction and summarization of the data as a whole. The basic well known statistical concepts such as *mean*, *variance*, *standard deviation*, *median*, and *mode* are simple models of the underlying population. Fitting a population to a specific *frequency distribution* provides an even better model of the data. Of course doing this with large databases having mutliple attributes, complex/multimedia attributes, and which are constantly changing is not practical (let alone always possible).

There are also many well known techniques to graphically display the structure of the data. For example, a *histogram* shows the distribution of the data. A *box plot* is a more sophisticated technique which illustrates several different features of the population at once. Figure 2.2 shows a sample box plot. The total *range* of the data values is divided into four equal parts called *quartiles*. The box in the center of the figure shows the range between the first, second, and third quartiles. The line in the box shows the median. The lines extending from either end of the box are the values that are a distance of 1.5 of the interquartile range from the 1^{st} and 3^{rd} quartiles respectively. Outliers are shown as points beyond these values.

Another visual technique to displaying data is called a *scatter diagram*. This is a graph on a two-dimensional axes of points representing the relationships between x and y values. By plotting the actually observable (x,y) points as seen in a sample, a visual image of some derivable functional relationship between the x and y values in the total population may be seen. Figure 2.3 shows a scatter diagram which plots some observed values. Notice that even though the points do not lie on

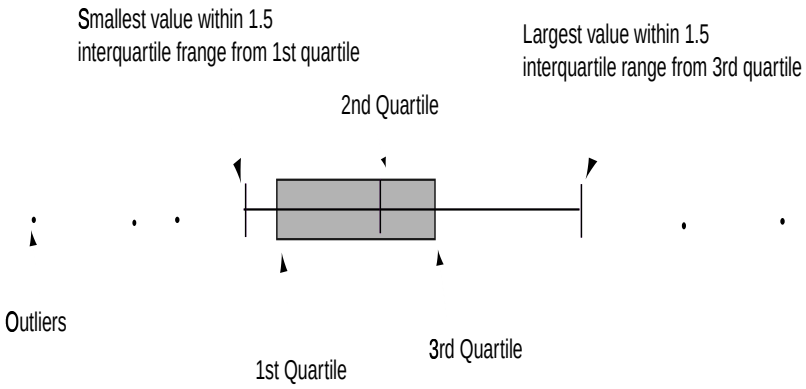


Figure 2.2: Box Plot Example

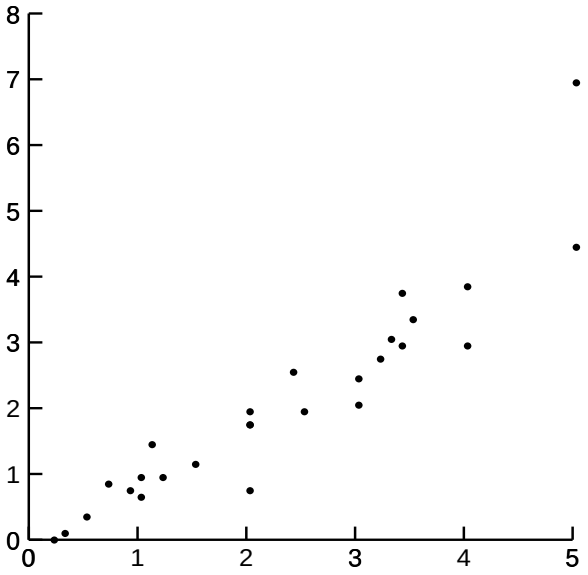


Figure 2.3: Scatter Diagram Example

a precisely linear line, they do hint that this may be a good predictor of the relationship between x and y.

Figure 2.1 illustrates the more general use of linear regression with one input value. Here we have a sample of data that we wish to model using a linear model. The line generated by the linear regression technique is shown in the figure. Notice, however, that the actual data points do not usually fit the linear model exactly. Thus this model is an estimate of what the actual input/output relationship is. We can use the generated linear model to predict an output value given an input value, but unlike that for Example 2.1 the prediction would be an estimate rather than the actual output value.

[1] Sometimes this is spelled Occum or Occam. This is named after William Ockham who was a monk in the late 13th and early 14th centuries. However it was first used by Durand de Saint-Pourcain an earlier French theologian.

2.3.3 Bayes Theorem

With statistical inference, information about a data distribution are inferred by examining data which follows that distribution. Given a set of data $X = \{x_1, ..., x_n\}$, the data mining problem can be viewed as to uncover properties of the distribution from which this set comes. Bayes Rule, defined in Definition 2.1, presents a technique to estimate the likelihood of a property given the set of data as evidence or input. Suppose that either hypothesis h_1 or h_2 must occur but not both. Also suppose that x_i is an observable event.

Definition 2.1 Bayes Rule *is:*

$$P(h_1 \mid x_i) = \frac{P(x_i \mid h_1)\ P(h_1)}{P(x_i \mid h_1)\ P(h_1) + P(x_i \mid h_2)\ P(h_2)} \tag{2.22}$$

Here $P(h_1 \mid x_i)$ is called the posterior probability while $P(h_1)$ is the prior probability associated with hypothesis h_1. $P(x_i)$ is the probability of the occurence of data value x_i and $P(x_i \mid h_1)$ is the conditional probability that given a hypothesis the tuple satisfies it.

Where there are m different hypotheses we have:

$$P(x_i) = \sum_{j=1}^{m} P(x_i \mid h_j)\ P(h_j). \tag{2.23}$$

Thus we have:

$$P(h_1 \mid x_i) = \frac{P(x_i \mid h_1)\ P(h_1)}{P(x_i)}. \tag{2.24}$$

Bayes rule allows us to assign probabilities of hyptheses given a data value, $P(h_j \mid x_i)$. Notice that here we discuss tuples when in actuality each x_i may be an attribute value or other data label. Each h_i may be an attribute value, set of attribute values (such as a range), or even a combination of attribute values.

Example 2.4 illustrates the use of Bayes rule. Example 2.4 also illustratrates that we may take advantage of other probability laws to determine combinations of probabilities. For example we may find $P(h_1) = P(I < \$10,000 \wedge good)$ by instead finding $P(I < \$10,000)P(good \mid I < \$10,000)$.

Example 2.4 *Suppose that a credit loan authorization problem can be associated with four hypotheses: $H = \{h_1, h_2, h_3, h_4\}$ where h_1 = authorize purchase , h_2 = authorize after further identification, h_3 = do not authorize, and h_4 = do not authorize but contact police. Further suppose that from training data we find that $P(h_1) = 65\%$, $P(h_2) = 20\%$, $P(h_3) = 10\%$, and $P(h_4) = 5\%$. To make our predictions, a domain expert has determined that the attributes we should be looking at are income and credit category. Assume that income, I, has been categorized by ranges: $[0, \$10,000)$, $[\$10,000, \$50,000)$, $[\$50,000, \$100,000)$, and $[\$100,000, \infty)$. Suppose that credit category is categorized as excellent, good, and bad. By combining these we then have twelve values in the data space: $D = \{x_1, ..., x_{12}\}$. After all tuples in the training set have been examined we can then calculate $P(x_i \mid h_j)$ and $P(x_i)$.*

Hypothesis Testing attempts to find a model which explains the observed data by first creating a hypothesis and then testing the hypothesis against the data. This is in contrast with most data

mining approaches which create the model from the data itself - without guessing what it is first. The data itself drives the model creation. The hypothesis is usually verified by examining a data sample. If the hypothesis holds for the sample it is assumed to hold for the population in general. Given a populationm, the hypothesis to be tested, H_0, is called the *null hypothesis*. Rejection of the null hypothesis causes another hypothesis, H_1, called the *alternative hypothesis* to be made.

2.4 Similarity Measures

Certainly the use of similarity measures is well known to anyone who has performed internet searches using a search engine. In these cases the set of all web pages represents the whole database and these are divided into two classes. Those which answer your query and those which don't. Those which answer your query should be more like each other than those which don't answer your query. The similarity in this case is defined by the query you state - usually based upon a keyword list. Thus the retrieved pages are similar because they all contain (to some degree) the keyword list you have specified.

The idea of similarity measures can be abstracted and applied to more general classification problems. The difficulty lies in how the similarity measures are defined and applied to the items in the database. Since most similarity measures assume numeric (and often discrete) values they may be difficult to use for more general data types. A mapping from the attribute domain to a subset of the integers may be used.

Definition 2.2 *The* **Similarity** *between two tuples t_i and t_j is a mapping from $D \times D$ to the range $[0, 1]$. Thus $sim(t_i, t_j) \in [0, 1]$.*

The objective is to define the similarity mapping such that documents which are more alike have a higher similarity value. Thus the following are desirable characteristics of a good similarity measure:

- $\forall t_i \in D, sim(t_i, t_i) = 1$
- $\forall t_i, t_j \in D, sim(t_i, t_j) = 0$ if t_i and t_j are not alike at all.
- $\forall t_i, t_j, t_k \in D, sim(t_i, t_j) < sim(t_i, t_k)$ if t_i is more like t_k than it is like t_j.

So how does one define such a similarity mapping? This, of course, is the difficult part. Often the concept of "alikeness" is itself not well defined. When the idea of similarity measure is used in classification where the classes are predefined, this problem is somewhat simpler than when it is used for clustering where the classes are not known in advance. Again, think of the IR example. Each IR query provides the class definition in the form of the IR query itself. So the classification problem becomes one, then, of determining similarity not among all tuples in the database but between each tuple and the query. This makes the problem an $O(n)$ problem rather than an $O(n^2)$ problem.

Listed below are some of the more common similarity measures used in traditional IR systems and more recent Internet search engines [92]:

- Dice: $sim(t_i, t_j) = \frac{2\sum_{h=1}^{k} t_{ih} t_{jh}}{\sum_{h=1}^{k} t_{ih} + \sum_{h=1}^{k} t_{jh}}$

- Jaccard: $sim(t_i, t_j) = \frac{\sum_{h=1}^{k} t_{ih} t_{jh}}{\sum_{h=1}^{k} t_{ih} + \sum_{h=1}^{k} t_{jh} - \sum_{h=1}^{k} t_{ih} t_{jh}}$
- Cosine: $sim(t_i, t_j) = \frac{\sum_{h=1}^{k} t_{ih} t_{jh}}{\sqrt{\sum_{h=1}^{k} t_{ih}^2 \sum_{h=1}^{k} t_{jh}^2}}$
- Overlap: $sim(t_i, C_j) = \frac{\sum_{h=1}^{k} t_{ih} t_{jh}}{min(\sum_{h=1}^{k} t_{ih}, \sum_{h=1}^{k} t_{jh})}$

In these formulas it is assumed that similarity is being evaluated between two vectors $t_i =< t_{i1}, ..., t_{ik} >$ and $t_j =< t_{j1}, ..., t_{jk} >$ and vector entries are usually assumed to be nonnegative numeric values. They could, for example, be a count of the number of times an associated keyword appears in the document.

Distance or dissimilarity measures are often used instead of similarity measures. As implied, these measure how "unlike" items are. In a two-dimensional space traditional distance measures may be used. These include:

- Euclidean: $dis(t_i, t_j) = \sqrt{\sum_{h=1}^{k} (t_{ih} - t_{jh})^2}$
- Manhattan: $dis(t_i, t_j) = \sum_{h=1}^{k} \mid (t_{ih} - t_{jh}) \mid$

To compensate for the different scales between different attribute values, the attribute values may be normalized to be in the range $[0, 1]$. If nominal rather than numeric values are used, some approach to determining the difference is needed. One method is to assign a difference of 0 if the values are identical and 1 is they are different.

2.5 Decision Trees

A Decision Tree is a predictive modeling technique used in classification, clustering, and prediction tasks. Decision trees use a divide and conquer technique to split the problem search space into subsets. It is based on the "Twenty Questions" game which children play as illustrated by Example 2.5. Figure 2.4 graphically shows the steps in the game. This tree has as the root, the first question asked. Each subsequent level in the tree consists of questions at that stage in the game. Nodes at the third level, show questions asked at the third level in the game. Leaf nodes represent a successful guess as to the object. This represents a correct prediction. Notice that each question successively divides the search space much as a binary search does. As with a binary search, questions should be posed so that the remaining space is divided into two equal parts. Often young children tend to ask poor questions by being too specific, such as initially asking "Is it my Mother?". This is a poor approach as the search space is not divided into two equal parts.

Example 2.5 *Stephanie and Shannon are playing a game of "Twenty Questions". Shannon has in mind some object that Stephanie tries to guess with no more than twenty questions. Stephanie's first question is: "Is this object alive"? Based on Shannon's answer, Stephanie then asks a second question. Her second question is based on the answer which Shannon provides to the first question. Suppose that Shannon says "Yes" as her first answer. Stephanie's second question is: "Is this a person?". When Shannon responds "Yes", Stephanie asks "Is it a friend?". When Shannon says "No", Stephanie then asks "Is it someone in my family?". When Shannon responds "Yes",*

Stephanie then begins asking the names of family members and can immediately narrows down the search space to identify the target individual. This game is illustrated in Figure 2.4.

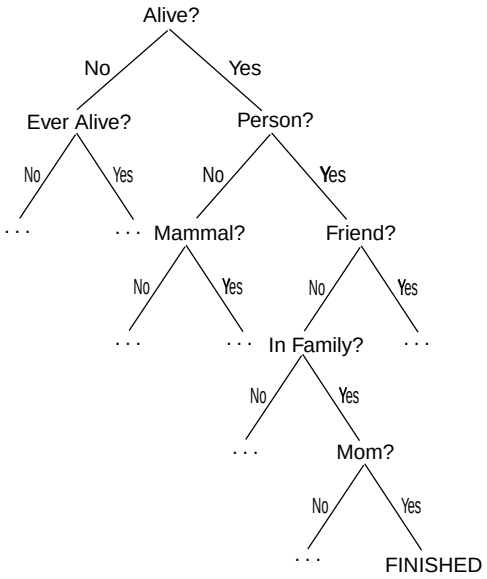


Figure 2.4: Decision Tree for Example 2.5

Definition 2.3 *A* **Decision Tree** *is a tree where the root and each internal node is labeled with a question. The arcs emanating from each node represent each possible answer to the associated question. Each leaf node represents a prediction of a solution to the problem under consideration.*

Definition 2.4 *A* **Decision Tree (DT) Model** [2] *is a computational model consisting of three parts:*

1. *A decision tree as defined in Definition2.3.*
2. *An algorithm to create the tree.*
3. *An algorithm that applies the tree to data and solves the problem under consideration.*

The building of the tree may be accomplished via an algorithm which examines data from a training sample or could be created by a domain expert. Most decision tree techniques differ in how the tree is created. We will examine several in later chapters of the book. Algorithm 2.2 shows the basic steps in applying a tuple to the DT, step three in Definition 2.4. We assume here that the problem to be performed is one of prediction, so the last step is to make the prediction as dictated by the final leaf node in the tree.

[2]Notice that we have two separate definitions: one for the tree itself and one for the model. Although we differentiate between the two here, the more common approach is to use the term decision tree for either.

Algorithm 2.2
Input:
T *//Decision Tree*
D *//Input Database*
Output:
M *//Model Prediction*
DTProc Algorithm:
//Simplistic algorithm to illustrate Prediction Technique using DT
for each $t \in D$ **do**
n = root node of T;
while *n not leaf node* **do**
Obtain answer to question on n applied t;
Identify arc from t which contains correct answer;
n = node at end of this arc;
Make prediction for t based on labeling of n;

To further illustrate the use of decision trees we use Example 2.6.

Example 2.6 *Suppose that students in a particular university are to be classified as short, tall, and medium based on their height. Assume that the database schema is {name, address, gender, height, age, year, major }. To construct a decision tree we need to identify the attributes which are important to the classification problem at hand. Suppose that height, age, and gender are chosen. Certainly a female who is 5'10" is considered as tall, while a male of the same height may not be. Also, a child 10 years of age may be tall if only 5'. Since this is a set of university students, we would expect most to be over 17 years of age. We thus decide to filter out those under this age and perform their classification separately. We may consider these to be outliers as their ages (and more importantly height classifications) are not typical of most university students. Thus for classification we only have gender and height. Using these two attributes, a decision tree building algorithm will construct a tree using a sample of the database with known classification values. This training sample forms the basis of how the tree is constructed. The resulting tree after training is shown in Figure 2.5.*

2.6 Neural Networks

The first proposal to use an artificul neuron appeared in 1943 [75]. Computer usage of neural networks, however, didn't actually begin until the 1980s. *Neural networks (NN)*, often referred to as *Artificial Neural Networks (ANN)* to distinguish them from biological neural networks, are modeled after the working of the human brain. A neural network is actually an information processing system which consists of a graphical representation of the model represented as well as various algorithms which access that graph. As with the human brain, a NN consists of many processing elements with connections between them. A NN, then, is structured as a graph with many nodes (processing elements) and arcs (interconnections) between them. The nodes in the

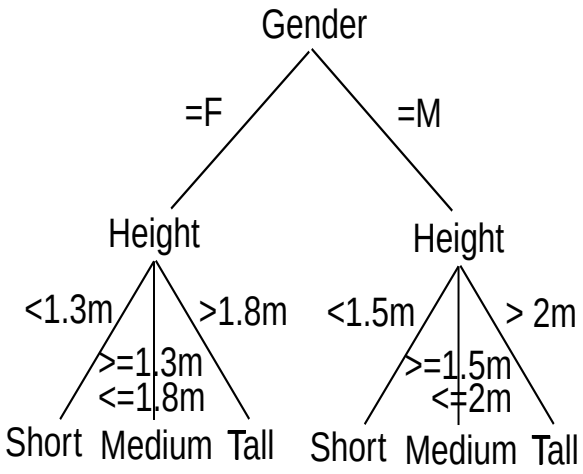


Figure 2.5: Decision Tree for Example 2.6

graph are like individual neurons while the arcs are their interconnections. Each of these processing elements functions independently from the others and uses only local data (input and output to the node) to direct its processing. This feature facilitates the use of neural networks in a distributed and/or parallel evironment.

The neural networks approach, like decision trees, requires that a graphical structure be built to represent the model and then that the structure be applied to the data. A neural network can be viewed as a directed graph with source (input), sink(output), and internal (hidden) nodes. The hidden nodes may exist over one or more hidden layers. To perform the data mining task, a tuple is input through the input nodes and the output node determines what the prediction is. Unlike decision trees which have only one input node (the root of the tree), a neural net has one input node for each attribute value to be examined to solve the data mining function. Unlike decision trees, after a tuple is classified, the neural network may be changed to improved future classification applications. Although the structure of the graph does not change, the labeling of the nodes and edges may change.

In addition to solving complex problems, neural networks can "learn" from prior applications. That is, if a poor solution to the problem is made, then the network is modified to produce a better solution to this problem the next time. A major drawback to the use of neural networks is the fact that they are difficult to explain to the end users (unlike decision trees which are easy to understand). Also, unlike decision trees, neural networks usually work only with numeric data.

To better illustrate the process, we reexamine Example 2.6 and show a simple neural network for this problem in Figure 2.6. We first must determine the basic structure of the tree. Since there are two important attributes, we assume that there are two input nodes. Since we are to classify into three classes, we use three output nodes. This graph has three layers. Each node is labeled with a function that describes it effect on the data coming into that node. At the input layer, functions f_1 and f_2 simply take the corresponding attribute value in and replicate it as output on each of the arcs coming out of the node. The functions at the hidden layer, f_3 and f_4, and those at the output layer, f_5, f_6, and f_7 perform more complicated functions which will be investigated later in this section. The arcs are all labeled with weights, w_{ij} is the weight between nodes i and

j. In processing, the functions at each node are applied to the input data producing the output. For example, the output of node 3 is:

$$f_3(w_{13}\ h + w_{23}\ g) \tag{2.25}$$

where h and g are the input height and gender values. Notice that to determine the output of a node we have to know: the values input on each arc, the weights on all input arcs, the technique used to combine the input values (a weighted sum is used here), and the function f_3 definition.

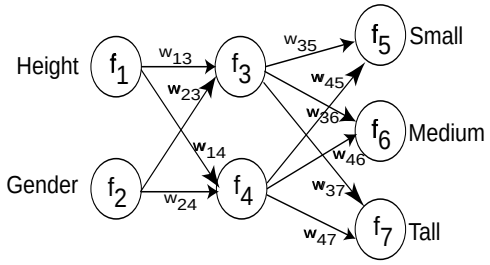


Figure 2.6: Neural Network for Example 2.6

As with decision trees, we define a neural network in two parts: one for the data structure and one for the general approach used including the data structure and the algorithms needed to use it.

Definition 2.5 *A* **Neural Network** *is a directed graph, $F =< V, A >$ with vertices $V = \{1, 2, ..., n\}$ and arcs $A = \{< i, j > |\ 1 \leq i, j \leq n\}$, with the following restrictions:*

1. *A is partitioned into a set of input nodes, A_I, hidden nodes, A_H, and output nodes, A_O.*
2. *The arcs in A are also partitioned into layers $\{1, ..., k\}$ with all input nodes in layer 1 and output nodes in layer k. All hidden nodes are in layers 2 to $k-1$ which are called the* **hidden layers**.
3. *Any arc $< i, j >$ must have node i in layer $h-1$ and node j in layer h.*
4. *Arc $< i, j >$ is labeled with a numeric value w_{ij}.*
5. *Node i is labeled with a function f_i.*

Definition 2.5 defines the most common type of neural network and which will be used throughout this text. There are more general structures including some with arcs which may be between any two nodes. Any approaches which use a more generalized view of a graph for neural networks will be adequately defined prior to usage.

Definition 2.6 *A* **Neural Network (NN) Model** *is a computational model consisting of three parts:*

1. *Neural Network Graph which defines the data structure of the neural network.*

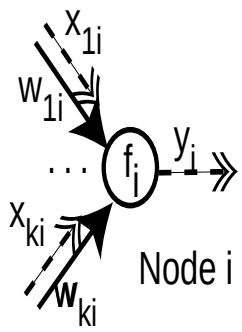


Figure 2.7: Neural Network Node

2. *Learning algorithm which indicates how learning takes place.*

3. *Recall techniques which determine how information is obtained from the network. We discuss propagation in this text.*

Neural networks have been used in pattern recognition, speech recognition and synthesis, medical applications (diagnosis, drug design), fault detection, problem diagnosis, robot control, and computer vision. In business, neural networks have been used to "advise" booking of airline seats to increase profitability. As a matter of fact, NNs can be used to compute any function. Although NNs can solve problems that seem more elusive to other AI techniques, they have a long training time (time during which the learning takes place) and thus are not appropriate for real time applications. Neural networks may contain many processing elements and can thus be used in massively parallel systems.

Artificial neural networks can be classified based on the type of connectivity and learning. The basic type of connectivity discussed in this text is *feedforward* where connections are only to layers later in the structure. Alternatively, a NN may be *feedback* where some links are back to earlier layers. Learning can be either supervised or unsupervised as is discussed in a subsequent section.

Figure 2.6 shows a sample node, i, in a neural network. Here there are k input arcs coming from nodes $1, 2, ..., k$. with weights of $w_{1i}, ..., w_{ki}$ and input values of $x_{1i}, ..., x_{ki}$. There is one output value y_i produced. During propagation this value is output on all output arcs of the node. The activation function, f_i, is applied to the inputs which are scaled by applying the corresponding weights. The weight in a neural network may be determined in two ways. In simple cases where much is known about the problem, the weights may be predetermined by a domain expert. The more common approach is to have them determined via a learning process.

The structure of a neural net may also be viewed from the perspective of matrices. Input and weight on the arcs into node i are:

$$[x_{1i}...x_{ki}]^T, [w_{1i}...w_{ki}] \tag{2.26}$$

There is one output value from node i, y_i, which is propagated to all output arcs during the

propagation process. Using summazation to combine the inputs, then, the output of a node is:

$$y_i = f_i(\sum_{j=1}^{k} w_{ji}\ x_{ji}) = f_i([w_{1i}...w_{ki}] \begin{bmatrix} x_{1i} \\ ... \\ x_{ki} \end{bmatrix}) \tag{2.27}$$

Here f_i is the activation function.

There are several advantages to the use of neural networks:

- A neural network can continue the learning process even after the training set has been applied.
- Neural networks can be easily parallelized to obtain improved performance.

There are several disadvantages to the use of neural networks:

- Neural networks are difficult to understand. Non technical users may have difficulty understanding how they work.
- Neural networks may suffer from overfitting.
- The structure of the NN graph must be determined apriori.
- Input attribute values must be numeric.
- Verification of the correct functioning of a neural network may be difficult to perform.

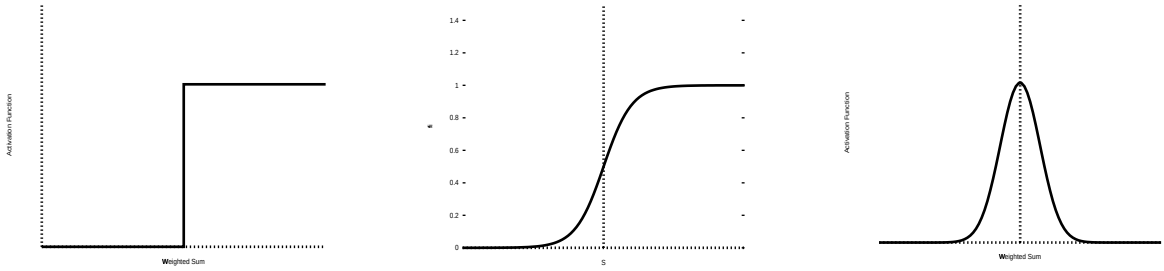
Due to the fact that neural networks are complicated, domain experts and data mining experts are often advised to assist in their use. This in term complicates the process.

2.6.1 Activation Function

The output of each node i in the NN is based on the definition of a function f_i, *Activation Function*, associated with it. An activation function is sometimes called a *Processing Element Function* or *Squashing Function*. The function is applied to the set of inputs coming in on the input arcs. Figure 2.6 illustrates the process. There have been many proposals for activation functions including threshold, sigmoid, symmetric sigmoid, and Guassian.

An activation function may also be called a *Firing Rule* relating it back to the workings of the human brain. When the input to a neuron is large enough, it *fires* sending an electrical signal out on its axon (output link). Likewise, in an artificial neural network the output may only be generated if the input is above a certain level, thus the idea of a firing rule. When dealing only with binary input, the output is either 0 or 1 depending on whether the neuron should fire. Some activation functions use -1 and 1 instead, while still other output a range of values. Based on these ideas, the input and output values are often considered to be either 0 or 1. The model used in this text is more general allowing any numeric values for weights and input/output values. In addition, the functions associated with each node may be more complicated than a simple threshold function. The desirable properties for an activation function are:

- Output between [0,1] or [-1,1]



a) Threshold b) Sigmoid d) Gaussian

Figure 2.8: Sample Activation Functions

- Since many learning algorithms use the derivative of the activation function, in these cases the function should have an easy to find derivative.

An activation function, f_i is applied to the set of input values $x_{1i}, ..., x_{ki}\}$ and weights $w_{1i}, ..., wki$. These inputs are usually conbined in a sum of producs form $S = (\sum_{j=1}^{k}(w_{hi}\ x_{hi}))$. If a bias input exists, then this formula becomes $S = w_{0i} + (\sum_{j=1}^{k}(w_{hi}\ x_{hi}))$. The following are alternative definitions for activation functions, $f_i(S)$ at node i. Activation functions may be *unipolar* which values in $[0, 1]$ or *bipolar* with values in $[-1, 1]$. The functions are also shown in Figure 2.8.

- **Linear**: A linear activation function produces a linear output value based on the input. The following is a typical activation function:

$$f_i(S) = c\ S \tag{2.28}$$

  Here c is a constant positive value. With the linear function, the output value has no limits in terms of maximum or minimum values.

- **Threshold or Step**: The output value is either a 1 or 0 depending on the sum of the products of the input values and their associated weights. As seen in Figure 2.8 a), values above a threshold, T, will be 1 else 0:

$$f_i(S) = \left\{ \begin{array}{ll} 1 & if S > T \\ 0 & otherwise \end{array} \right\} \tag{2.29}$$

  The binary output values may also be 1 or -1. Alternatively, the 1 value may be replaced by any constant. A variation of this "hard limit" threshold function is a linear threshold function. With the *linear threshold function*, also called a *ramp function* or *piecewise linear function*, the value of the activation function increases gradually from the low value to the high value. One such function is:

$$f_i(S) = \left\{ \begin{array}{ll} 1 & if S > T_2 \\ \frac{S-T_1}{T_2-T_1} & if T_1 \leq S \leq T_2 \\ 0 & if S < T_1 \end{array} \right\} \tag{2.30}$$

  Here the linear increase is between T_1 and T_2. As with the regular threshold function, the value may be between -1 and 1 or 0 and 1.

- **Sigmoid Activation Function**: As seen in Figure 2.8 b), this is an "S" shaped curve with output values between -1 and 1 (or 0 and 1), and which is monotonically increasing. Although there are several types of sigmoid functions, they all have this characteristic "S" shape. A common one is the *logistic function*:

$$f_i(S) = \frac{1}{(1 + e^{-c\,S})} \tag{2.31}$$

  Here c is a constant positive value which changes the slope of the function. This function possesses a simple derivative: $\frac{\partial f_i}{\partial S} = f_i\,(1 - f_i)$.

- **Hyperbolic Tangent Function**: A variation of the Sigmoid function is the hyperbolic tangent function shown below:

$$f_i(S) = \frac{(1 - e^{-S})}{(1 + e^{-c\,S})} \tag{2.32}$$

  This function has an output centered at zero which may help with learning.

- **Gaussian Activation Function**: The gaussian function, Figure 2.8 c), is a bell shaped curve with output values in the range [0,1]. A typical guassian function is:

$$f_i(S) = e^{\frac{-S^2}{v}} \tag{2.33}$$

  Here S is the mean and V the predefined positive variance of the function.

These are only a representative subset of the possible set of activation functions which could be and have been used.

Nodes in neural networks often have an extra input called a bias. This bias value of 1 is input on an arc with a weight of $-\theta$. The summation with bias input thus becomes:

$$S_i = \sum_{j=1}^{k} w_{ji}\,x_{ji} - \theta \tag{2.34}$$

The effect of the bias input is to move the activation function on the X-axis by a value of θ. Thus a weight of $-\theta$ becomes a threshold of θ.

2.6.2 Propagation

The normal approach used for processing is called *propagation*. Given a tuple of values input to the NN, $X =< x_1, ..., x_h >$, one value is input at each node in the input layer. Then the summation and activation function are applied at each node with an output value created for each output arc from that node. These values are in turn sent to the subsequent nodes. This process continues until a tuple of output values, $Y =< y_1, ..., y_m >$, is produced from the nodes in the output layer. The process of propagation is shown in Algorithm 2.3 using a NN with one hidden layer. Here a hyperbolic tangent activation function is used for the nodes in the hidden layer while a sigmoid function is used for nodes in the output layer.

Algorithm 2.3
Input:
N *//Neural Network*
$X =< x_1, ..., x_h >$ *//Input tuple consisting of values for input attributes only*
Output:
$Y =< y_1, ..., y_m >$ *//Tuple consisting of output values from NN*
Propagation Algorithm:
//Algorithm illustrates propagation of a tuple through a NN
foreach *node i in the input layer* **do**
Output x_i *on each output arc from i;*
foreach *hidden layer* **do**
foreach *node i* **do**
$S_i = (\sum_{j=1}^{k}(w_{hi}\ x_{hi}))$;
foreach *output arc from i* **do**
Output $\frac{(1-e^{-S_i})}{(1+e^{-c\ S_i})}$;
foreach *node i in the output layer* **do**
$S_i = (\sum_{j=1}^{k}(w_{ji}\ x_{ji}))$;
Output $y_i = \frac{1}{(1+e^{-c\ S_i})}$;

2.6.3 NN Learning

The NN starting state is modified based on feedback of its performance with the data in the training set. This type of learning is referred to as *supervised* as it is known apriori what the desired output should be. *Unsupervised* learning can also be performed if the output is not known. With unsupervised approaches, no external teacher set is used. A training set may be provided but no labeling of the desired outcome is included. In this case, similarities and differences between different tuples in the training set are uncovered The techniques discussed here are all supervised.

Supervised Learning

Supervised learning in a neural network is the process of adjusting the arc weights based on its performance with a tuple from the training set. The behavior of the training data is known apriori and thus can be used to fine tune the network for better behavior in future similar situations. Thus the training set can be used as a "teacher" during the training process. The output from the network is compared to this known desired behavior. Algorithm 2.4 outlines the steps required. One potential problem with supervised learning is that the error may not be continually reduced. It would of course be hoped that each iteration in the learing process reduces the error so that it is ultimately below an exceptable level. However, this is not always the case. This may be due to the error calculation technique or to the approach used for modifying the weights.

Algorithm 2.4
Input:
N *//Starting Neural Network*
X *//Input tuple from Training Set*

D *//Output tuple desired*

Output:

N *//Improved Neural Network*

SupLearn Algorithm:

//Simplistic algorithm to illustrate approach to NN learning

Propagate X through N producing output Y;

Calculate error by comparing D to Y;

Update weights on arcs in N to reduce error;

Notice that this algorithm needs to be associated with a means to calculate the error as well as some technique to adjust the weights. There are many techniques which have been proposed to calculate the error. Assuming that the output from node i is y_i but should be d_i the error produced from a node in any layer can be found by:

$$| y_i - d_i | \tag{2.35}$$

The $Mean - SquaredError(MSE)$ is found by:

$$\frac{(y_i - d_i)^2}{2} \tag{2.36}$$

This MSE can then be used to find a total error over all nodes in the network or over only the output nodes. In the discussion below, the assumption is made that only the final output of the NN is known for a tuple in the training data. Thus the *Total MSE* error over all m output nodes in the NN is:

$$\sum_{i=1}^{m} \frac{(y_i - d_i)^2}{m} \tag{2.37}$$

This formula could be expanded over all tuples in the training set to see the total error over all of them. Thus an error can be calculated for a specific test tuple or for the total set of all entries.

The Hebb and Delta rules are approaches to change the weight on an input arc to a node based on the knowledge that the output value from that node is incorrect. With both techniques, a *learning rule* is used to modify the input weights. Suppose for a given node, j, the input weights are represented as a tuple $< w_{1j}, ..., w_{kj} >$, while the input and output values are $< x_{1j}, ..., x_{kj} >$ and y_j, respectively. The objective of a learning technique is to change the weights based on the output obtained for a specific input tuple. The change in weights using the *Hebb Rule* are represented by the following rule:

$$\Delta w_{ij} = c\ x_{ij}\ y_j \tag{2.38}$$

Here c is a constant often called the *learning rate*. A rule of thumb is that $c = \frac{1}{|\#entriesintrainingset|}$.

A variation of this approach called the *Delta Rule* examines not only the output value y_j but also the desired value d_j for output. In this case the change in weight is found by the rule:

$$\Delta w_{ij} = c\ x_{ij}\ (d_j - y_j) \tag{2.39}$$

The nice feature of the delta rule is that it minimizes the error $d_j - y_j$ at each node.

BackPropagation is a learning technique which adjusts weights in the NN by propagating weight changes backwards from the sink to the source nodes. Backpropagation is the most well known

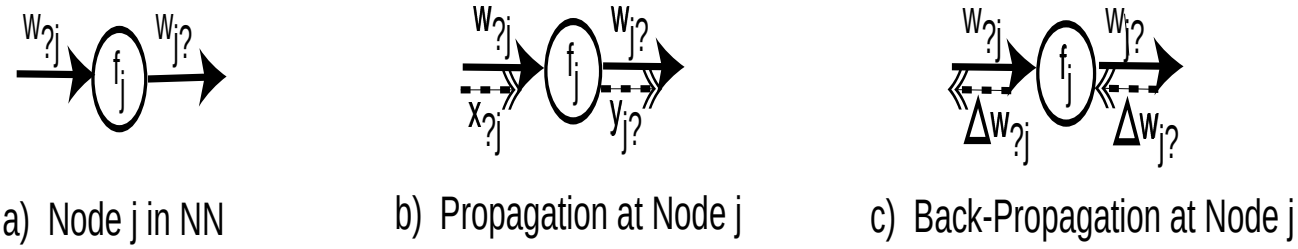


Figure 2.9: Neural Network Usage

form of learning as it is easy to understand and generally applicable. Backpropagation can be thought of as a generalized delta rule approach.

Figure 2.9 shows the structure and usage of one node, j in a neural network graph. In part a) the basic node structure is shown. Here the representative input arc has a weight of $w_{?j}$ where ? is used to show that the input to node i is coming from another node shown here as ?. Of course there are probably multiple input arcs to a node. The output is similarly labeled $w_{j?}$. During *propagation* data values input at the input layer flow through the network with final values coming out of the network at the output layer. In part b) of this figure the propagation technique is shown. Here the smaller dashed arrow underneath the regular graph arc shows the input value $x_{?j}$ flowing into node j. The activation function f_j is applied to all the input values and weights with output values resulting. There is an associated input function that is applied to the input values and weights prior to applying the activation function. Thi input function is typically a weighted sum of the input values. Here $y_{j?}$ shows the output value flowing (propagating) to next node from node j. Thus propagation occurs by applying the activation function at each node which then places the output value on the arc to be sent as input to the next nodes. In most cases, the activation function only produces one output value that is propagated to the set of connected nodes. The neural network can be used for classification and/or learning. During the classification process, only propagation occurs. However, when learning is used after the output of the classification occurs a comparison to the known classification is used to determine how to change the weights in the graph. In the simplest types of learning, learning progresses from the output layer backwards to the input layer. Weights are changed based on the changes that were made in weights in subsequent arcs. This backward learning process is called backpropagation and is illustrated in Figure 2.9 c). Weight $w_{j?}$ is modified to become $w_{j?} + \triangle w_{j?}$. A learning rule is applied to this $\triangle w_{j?}$ to determine the change at the next higher level $\triangle w_{?}j$.

Algorithm 2.5
Input:
N *//Starting Neural Network*
$X =< x_1, ..., x_h >$ *//Input tuple from Training Set*
$D =< d_1, ..., d_m >$ *//Output tuple desired*
Output:
N *//Improved Neural Network*
BackPropagation Algorithm:

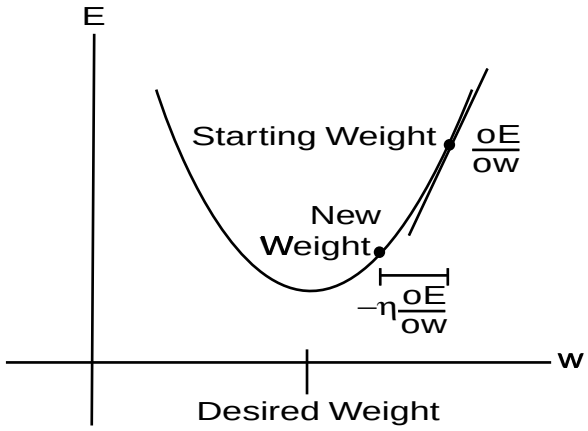


Figure 2.10: Gradient Descent

//Illustrate backpropagation
Propagation(N, X);
$E = 1/2 \sum_{i=1}^{m} (d_i - y_i)^2$*;*
Gradient(N,E);

A simple version of the backpropagation algorithm is shown in Algorithm 2.5. The MSE is used to calculate the error. Each tuple in the training set is applied to this algorithm. The last step of the algorithm uses *Gradient Descent* as the technique to modify the weights in the graph. The basic idea of gradient descent is to find the set of weights which minimize mean squared error. $\frac{\partial E}{\partial w_{ji}}$ gives the slope (or gradient) of the error function for one weight. We thus wish to find the weight where this slope is zero. Figure 2.10 and Algorithm 2.6 illustrate the concept.

Algorithm 2.6
Input:
N *//Starting Neural Network*
E *//Error found from Back algorithm*
Output:
N *//Improved Neural Network*
Gradient Algorithm:
//Illustrates incremental gradient descent
foreach *node i in output layer* **do**
foreach *node j input to i* **do**
$\Delta w_{ji} = \eta\ (d_i - y_i)\ x_j\ (1 - \frac{1}{1+e^{S_j}})\ \frac{1}{1+e^{S_j}}$;
$w_{ji} = w_{ji} + \Delta w_{ji}$;
layer = previous layer;
repeat
foreach *node i in this layer* **do**
$\Delta w_{ji} = \eta\ \sum_{l=1}^{n} (d_i - y_i)\ w_{il}\ x_j\ \frac{1-(\frac{1-e^{-S_i}}{1+e^{-S_i}})^2}{2}$;

$w_{ji} = w_{ji} + \Delta w_{ji}$;
layer = previous layer;
until *layer = input layer;*

This algorithm changes weights by working backwards from the output layer to the input layer. There are two basic versions of this algorithm With the *batch* or *offline* approach the weights are changed once after all tuples in the training set are applied and a total MSE found. With the *incremental* or *online* approach, the weights are changed after each tuple in the training set is applied. The incremental technique is usually preferred as it requires less space, and may actually examine more potential solutions (weights) thus leading to a better solution. In this equation η is referred to as the *learning parameter*. It is typically found in the range $(0, 1]$ although may be larger. This value deterines how fast the algorithm learns.

Applying a learning rule back through multiple layers in the network may be difficult. How to do this for the hidden layers is not as easy as with the output layer. Overall, however, we are trying to minimize the error at the output nodes, not at each node in the network. Thus the approach that is used is to propagate the output errors backwards through the network. The learning function in the gradient descent technique is based on using the following value for delta at the output layer:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial S_i} \frac{\partial S_i}{\partial w_{ji}} \tag{2.40}$$

Here the weight w_{ji} is that at the layer being examined. Assuming a sigmoidal activation function in the output layer, for the output layer we have:

$$\frac{\partial y_i}{\partial S_i} = \frac{\partial}{\partial S_i}(\frac{1}{(1+e^{-S_i})}) = (1 - (\frac{1}{1+e^{-S_i}}))\ (\frac{1}{1+e^{-S_i}}) \tag{2.41}$$

For hidden layers, where a hyperbolic tangent activation function is assumed, this becomes:

$$\frac{\partial y_i}{\partial S_i} = \frac{\partial}{\partial S_i}(\frac{(1-e^{-S_i})}{(1+e^{-S_i})}) = \frac{(1 + (\frac{1-e^{-S}}{1+e^{-S}}))\ (1 - (\frac{1-e^{-S}}{1+e^{-S}}))}{2} \tag{2.42}$$

Also,

$$\frac{\partial S_i}{\partial w_{ji}} = x_j \tag{2.43}$$

where x_j is the output of node j. For nodes in the output layer the third partial is:

$$\frac{\partial E}{\partial y_i} = \frac{\partial}{\partial y_i}(1/2 \sum_{i=1}^{m}(d_i - y_i)^2) = -\sum_{i=1}^{m}(d_i - y_i) \tag{2.44}$$

For nodes in hidden layers this partial is more complicated:

$$\frac{\partial E}{\partial y_i} = -\sum_{k=1}^{m}(d_i - y_i)w_{ik} \tag{2.45}$$

Another common formula for the change in weight is:

$$\Delta w_{ji}(t+1) = -\eta \frac{\partial E}{\partial w_{ji}} + \alpha\ \Delta w_{ji}(t) \tag{2.46}$$

Here the change in weight at time $t+1$ is based not only on the same partial derivative as earlier, but also on the last change in weight. Here α is called the *momentum* and is used to prevent oscillation problems which may occur without it.

Unsupervised Learning

Neural networks which use unsupervised learning attempt to find features in the data which characterize the desired output. They look for clusters of like data. These types of neural networks are often called *self organizing neural networks*. There are two basic types of unsupervised learning: noncompetitive and competitive.

With the *noncompetitive* or *Hebbian* learning, the weight between two nodes is changed to be proportional to both output values. That is:

$$\Delta w_{ji} = \eta \, y_j \, y_i \tag{2.47}$$

With competitive learning, nodes are allowed to compete and the winner takes all. This approach usually assumes a two layer neural network in which all nodes from one layer are connected to all nodes in the other layer. As training occurs, nodes in the output layer become associated with certain tuples in the input data set. Thus this provides a grouping of these tuples together into a cluster. Imagine each input tuple with one attribute value input to each input node. We can thus associate each weight input to each output node with one of the attributes from the input tuple. When a tuple is input to the neural net, all output nodes produce an output value. The node with the weights more like those of the input tuple is declared the winner. Its weights are then adjusted. This process continues with each tuple input from the training set. With a large and varied enough training set, over time each output node should become associated with a set of tuples. The input weights to the node are then close to an average of the tuples in this cluster.

2.6.4 Self Organizing Feature Maps

A *Self Organizing Feature Map (SOFM)* or *Self Organizing Map (SOM)* is a NN approach which uses competitive unsupervised learning. Learning is based on the concept that the behavior of a node should only impact those nodes and arcs near it. Weights are initially asigned randomly and adjusted during the learing process to produce better results. During this learning process, hidden features or patterns in the data are uncovered and the weights adjusted accordingly. SOFMs were developed by observing how neurons work in the brain and also in ANNs. That is [10]:

- The firing of neurons impact the firing of others which are near it.
- Neurons which are far apart seem to inhibit each other.
- Neurons seem to have specific nonoverlapping tasks.

The term *self organizing* indicates the ability of these NNs to

organize the nodes into clusters based upon the similarity between them. Those closer together are more similar than those far apart. This hints at how the actual clustering is performed. Over time, nodes in the output layer become matched to input nodes and patterns of nodes in the output layer emerge.

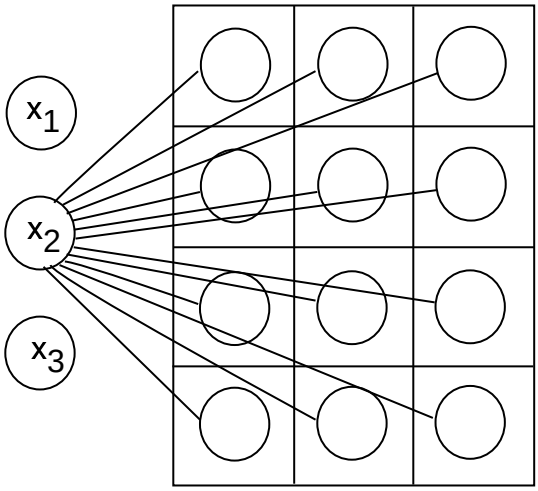


Figure 2.11: Kohonen Network

Perhaps the most common example of a SOFM is the *Kohonen Self Organizing Map* [63] which is used extensively in commercial data mining products to perform clustering. There is one input layer and one special layer which produces output values that compete. In effect, multiple outputs are created and the best one chosen. This extra layer is not technically either a hidden or output layer, thus we refer to it here as the *competitive layer.* indexcompetitive layer Nodes in this layer are viewed as a two-dimensional grid of nodes as seen in Figure 2.11. Each input node is connected to each node in this grid. Propagation occurs by sending the input value for each input node to each node in the competitive layer. As with regular NNs, each arc has an associated weight and each node in the competitive layer has an activation function. Thus each node in the competitive layer produces an output value, and the node with the best output wins the competition and is determined to be the output for that input. A nice feature of Kohonen nets is that the data can be fed into the multiple competitive nodes in parallel. Training occurs by adjusting weights so that the best output is even better the next time this input is used. Best is determined by computing a distance measure.

A common approach is to initialize the weights on the input arcs to the competitive layer with normalized values. The similarity between output nodes and input vectors is then determined by the dot product of the two vectors. Given an input tuple $X =< x_1, ..., x_h >$ and weights on arcs input to a competitive node i as $w_{1i}, ..., w_{hi}$, the similarity between X and i can be calculated by:

$$Sim(X, i) = \sum_{j=1}^{h} x_j \; w_{ji} \tag{2.48}$$

The competitive node most similar to the input node wins the competitive. Based on this, the weights coming into i as well as those for the nodes immediately surrounding it in the matrx are increased. This is the learning phase. Given a node i, we use the notation N_i to represent the union of i and the nodes near it in the matrix. Thus the learning process uses:

$$\Delta w_{kj} = \left\{ \begin{array}{ll} c\,(x_k - w_{kj}) & if j \in N_i \\ 0 & otherwise \end{array} \right\} \tag{2.49}$$

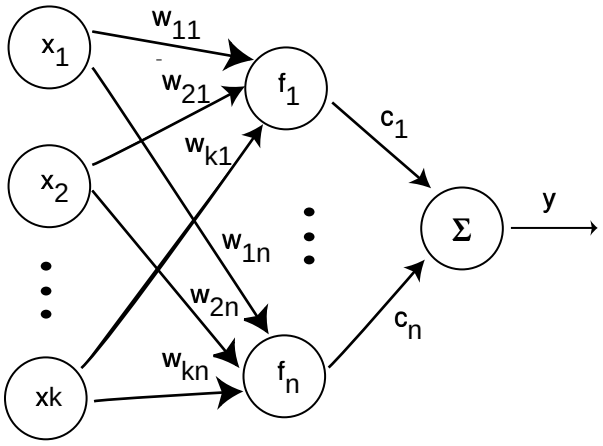


Figure 2.12: Radial Basis Function Network

In this formular, c, indicates the learning rate and may actually vary based on the node rather than being a constant. The basic idea of SOM learning is that, after each input tuple in the training set the winner and its neighbors have their weights changed to be closer to that of the tuple. Over time, a pattern on the output nodes emerges which is close to that of the training data. At the beginning of the training process, the neighborhood of a node may be defined to be large. However, during the processing the neighborhood can decrease.

2.6.5 Radial Basis Function Networks

A Radial Basis Function (RBF) has a Gaussian shape and an *RBF Network* is is typically a NN with three layers. A Gaussian activation function is used at the hidden layer while a linear one is used at the output layer. The objective is to have the hidden nodes learn to respond to only a subset of the input, namely that where the Gaussian function is centered. This is usually accomplished via supervised learning. When RBF functions are used as the activation functions on the hidden layer, the nodes can be sensitive to a subset fo the input values. Figure 2.12 shows the basic structure of an RBF unit with one output node.

2.7 Genetic Algorithms

Genetic algorithms are examples of *evolutionary computing* methods, and are optimization type algorithms. Given a population of potential problem solutions (individuals), evolutionary computing expands this population with new potentially better solutions. The basis for evolutionary computing algorithms is biological evolution where over time evolution produces the best or "fittest" individuals.

Genetic algorithms were first proposed in 1975 by John Holland [49]. In data mining, genetic algorithms may be used for clustering, prediction, and even association rules. You can think of

these techniques as finding the "fittest" models from a set of models to represent the data. In this approach a starting model is assumed and through many iterations, models are combined to create new models. The best of these, as defined by a fitness function, are then input into the next iteration. Algorithms differ in how the model is represented, how different individuals in the model are combined, and in the fitness function used.

Definition 2.7 *A* **Genetic Algorithm (GA)** *is a computational model consisting of three parts:*

1. *A starting set of individuals, P.*
2. *Crossover technique*
3. *Mutation algorithm*
4. *Fitness function*
5. *Algorithm which applies the crossover and mutation techniques to P iteratively using the fitness function to determine the best individuals in P to keep. The algorithm replaces a predefined number of individuals from the population with each iteration and terminates when some threshold is met.*

The problem to be solved is modeled by a set of individuals. These individuals are like a DNA encoding in that the structure for each individual represents an encoding of the major features needed to model the problem. In genetic algorithms, reproduction is defined by precise algorithms that indicate how to combine the given set of individuals to produce new ones. These are called *crossover* algorithms. As in nature, however, mutations sometimes appear and these may also be present in genetic algorithms Since genetic algorithms attempt to model nature, only the strong survive. When new individuals are created, a choice must be made about which individuals survive. This may be the new ones, the old ones, or more likely a combination of the two. The third major component of genetic algorithms, then, is the part that determines the best (or fittest) individuals to survive.

When using genetic algorithms to solve a problem, the first thing that must be determined is how to model the problem as a set of individuals. This may be extremely difficult as the abstraction of the problem to this representation is not always straightforward. Indeed, this may be the most difficult part of using genetic algorithms. In the real world, individuals may be identified by a complete encoding of the DNA structure. Typically an individual is viewed as an array or tuple of values. Based on the recombination (crossover) algorithms, the values are usually numeric and may be binary strings.

Suppose that each solution to the problem to be solved is represented as one of these individuals. A complete search of all possible individuals would yield the best individual or solution to the problem using the predefined fittness function. Since the search space is quite large (perhaps infinite), what a genetic algorithm does is to prune from the search space individuals which will not solve the problem. In addition it only creates new individuals which will probably be much different from those previously examined. Since genetic algorithms do not search the entire space they may not yield the best result. However, they can provide approximate solutions to difficult problems.

Genetic algorithms have been used to solve most data mining problems including classification, clustering, and generating association rultes. Typlical applications of genetic algorithms include scheduling, robotics and

The major advantage to the use of genetic algorithms is that they are easily parallelized. There are, however, many disadvantages to their use:

- Genetic algorithms are difficult to understand and explain to end users.
- The abstraction of the problem and method to represent individuals is quite difficult. In fact, it may not always be possible to determine any representation method yet alone the best.
- Determining the best fitness function is difficult.
- Determining how to do crossover and mutation is difficult.

Algorithm 2.7
Input:
P *//Initial Population*
Output:
P' *//Improved Population*
Genetic Algorithm:
//Algorithm to illustrate Genetic Algorithm
repeat
repeat
$N = \mid P \mid$;
$P' = \emptyset$;
i_1, i_2 = *select(P)*;
o_1, o_2 = *cross*(i_1, i_2);
o_1 = *mutate*(o_1);
o_2 = *mutate*(o_2);
$P' = P' \cup \{o_1, o_2\}$;
until $\mid P' \mid = N$;
until *termination criteria satisfied;*

Algorithm 2.7 outlines the steps performed by a genetic algorithm. Initially a population of individuals, P is created. Although different approaches can be used to perform this, they are typically generated randomly. From this population, a new population, P', of the same size is created. Repeatedly the algorithm selects individuals from which to create new ones. These parents, i_1, i_2, are then used to produce two offspring, o_1, o_2, using a crossover process. Then mutants may be generated. The process continues until the new population satisfies the termination condition. We assume here that the entire population is replaced with each iteration. Although this algorithm is quite general, it is representative of all genetic algorithms.

Perhaps the most difficult part of defining a genetic algorithm is determining how to represent the population. Each individual is represented as a string of characters from the given alphabet. Often the strings are boolean.

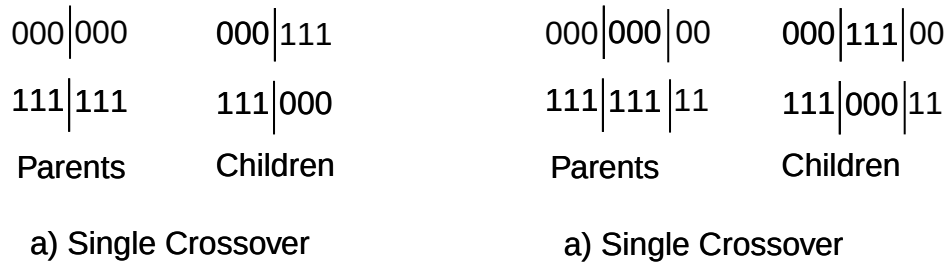


Figure 2.13: Crossover

Definition 2.8 *Given an alphabet A, an* **individual** *is a string $I = I_1, I_2, ..., I_n$ where $I_j \in A$. Each character in the string, I_j, is called a* **gene**. *The values that each character can have are called the* **alleles**. *A* **population**, *P, is a set of individuals.*

The first step in the GA algorithm is to select individuals. A fitness function, f, is used to determine the best individuals in a population. This is then used in the selection process to choose parents. Given an objective by which the population can be measured, the fitness function indicates how well the goodness objection is being met by an individual.

Definition 2.9 *Given a population P, a* **fitness algorithm**, *f, is a mapping $f : P \rightarrow \mathbf{R}$.*

The simplest selection process is to select individuals based on their fitness:

$$p_{I_i} = \frac{f(I_i)}{\sum_{I_j \in P} f(I_j)} \tag{2.50}$$

Here p_{I_i} is the probability of selecting individual I_i. This type of selection is called *roulette wheel selection*. One problem with this approach is that it is still possible to select individuals with a very low fitness value. In addition when the distribution is quite skewed with a small number of extremely fit individuals, these may be chosen repeatedly. In addition, as the search continues, the population becomes less diverse so that the selection process has little effect.

Given two individuals (*parents* from the population, the crossover technique generates new individuals (*offsprings* or *children*) by switching subsequences of the strings. Figure 2.13 illustrates the process of crossover. The locations indicating the crossover points are shown in the Figure with the vertical lines. In figure 2.13 a) crossover is achieved by interchanging the last three bits of the two strings. In part b) the center three bits are interchanaged. Figure 2.13 shows single and multiple crossover points. There are many variations of the crossover approach including determining crossover points randomly A crossover probability is used to determine how many new offspring are created via crossover.

The mutation operation randomly changes characters in the offspring. A very small probability of mutation is set to determine whether or not a character should change.

2.8 Exercises

1. Use linear regression with one predictor to determine the formula for the output given the following samples: (1,3,3) and (2,5,10).

2. Given the following set of values {1, 3, 9, 15, 20 } determine the jacknife estimate for both the mean and standard deviation.

3. Find the similarity between $< 0, 1, 0.5, 0.3, 1 >$ and $< 1, 0, 0.5, 0, 0 >$ using the Dice, Jaccard, and Cosine similarity measures.

4. Given the decision tree in Figure 2.5 classify each of the following students:
$< Mary, 20, F, 2m, Senior, Math >$, $< Dave, 19, M, 1.7m, A, Sophomore, ComputerScience >$, and $< Martha, 18, F, 1.2m, A, Freshman, English >$.

5. Using the neural network shown in Figure 2.6 classify the same students as those used in Exercise 4. Assume that the hidden nodes use a hyperbolic tangent activation function, the output layer uses a sigmoidal function, and a weighted sum is used to calculate input to each node in the hidden and output layers. Suppose all the weights are 0.5.

6. Suppose that the output of Mary in Exercise 5 should have been 0 for Small, 0 for Medium, and 1 for Tall. Use the Gradient Descent algorithm to modify the weights in the neural network.

7. Given an initial population $\{< 101010 >, < 001100 >, < 010101 >, < 000010 >\}$ apply the Genetic Algorithm to find a better population. Suppose the fitness function is defined as the sum of the bit values for each individual and that mutation always occurs by negating the second bit. The termination condition is that the average fitness value for the entire population must be greater than 5. Also, pairs are chosen for crossover only if their fitness is greater than 3.

2.9 Bibliographic Notes

Data mining owes much of its development to previous work in machine learning, statistics, and databases [71]. The contribution of statistics to data mining can be traced by to the seminal work by Bayes in 1763. An examination of the impact statistics has played on the development of data mining techniques can be found in [57], [38], and [79]. Many data mining techniques find their birth in the machine learning field. Excellent studies of the relationship between machine learning and data mining can be found in two recent books [77] and [107].

Excellent overviews of neural networks can be found in [47]. Discussions of perceptrons are available in [89]. For an introduction to Kohonen's self organizing maps the reader is referred to [63].

A great survey of genetic algorithms is available in [39].

Part II

CORE TOPICS

Chapter 3

CLASSIFICATION

3.1 Introduction

Classification is perhaps the most familiar and most popular data mining technique. Examples of classification applications include image and pattern recognition, medical diagnosis, loan approval, detecting faults in industry applications, and classifying financial market trends. Estimation and prediction may be viewed as types of classification. When someone estimates your age or guesses the number of marbles in a jar, these are actually classification problems. Prediction can be thought of classifying an attribute value into one of a set of possible classes. Example 1.1 in Chapter 1 illustrates the use of classification for credit card purchases. Also in Chapter 2 the use of decision trees and neural networks to classify people as to their height was illustrated. Prior to current data mining techniques, classification was frequently performed by simply applying knowledge of the data. This is illustrated in Example 3.1.

Example 3.1 *Teachers classify students as A, B, C, D, or F based on their grades. By using simple boundaries (60,70,80,90) the following classification is possible:*

$$\begin{array}{rl} 90 \leq grade & A \\ 80 \leq grade \leq 90 & B \\ 70 \leq grade \leq 80 & C \\ 60 \leq grade \leq 70 & D \\ grade < 60 & F \end{array}$$

All approaches to performing classification assume some knowledge of the data. Often a training set is used to develop the specific parameters required by the technique. *Training Data* consists of sample input data as well as the classification assignment for the data. Domain experts may also be used to assist in the process.

The classification problem is stated as shown in Definition 3.1:

Definition 3.1 *Given a database $D = \{t_1, t_2, ..., t_n\}$ of tuples (items, records, variables) and a set of classes $C = \{C_1, ..., C_m\}$, the* **Classification Problem** *is to define a mapping $f : D \rightarrow C$ where each t_i is assigned to one class. A* **Class**, *C_j, contains precisely those tuples mapped to it. That is $C_j = \{t_i \mid f(t_i) = C_j, 1 \leq i \leq n, and\ t_i \in D\}$.*

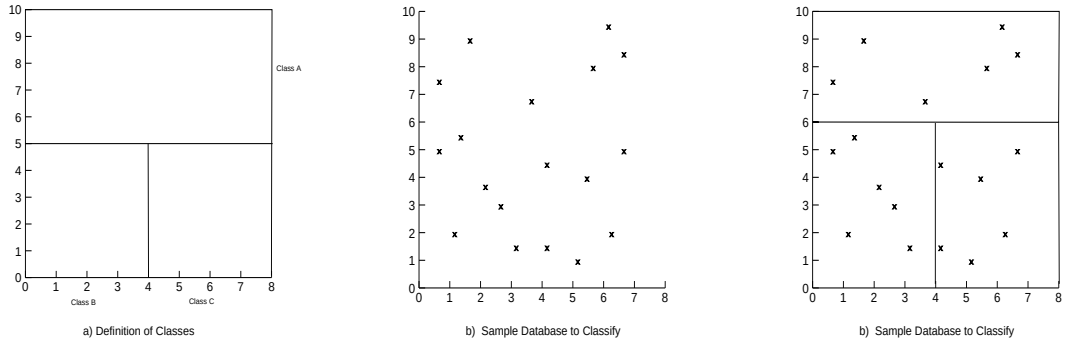


Figure 3.1: Classification Problem

Our definition views classification as a mapping from the database to the set of classes. Notice that the classes are predefined, nonoverlapping and partition the entire database. Each tuple in the database is assigned to exactly one class. The classes which exist for a classification problem are indeed *equivalence classes*. In actuality, the problem is usually implemented in two phases:

1. Create specific model by evaluating training data. This step has as input the training data (including defined classification for each tuple) and as output a definition of the model developed. The model created accurately classifies the training data to as high a degree as possible.

2. Apply the model developed in step one by classifying tuples from the target database.

Although the second step is the one that actually does the classification (according to the definition in Definition 3.1), most research has been applied to step one. Step two is often straightforward.

As discussed in [62] there are three basic methods used to solve the classification problem:

- Specifying Boundaries. Here classification is performed by dividing the input space of potential database tuples into regions where each region is associated with one class.

- Using Probability Distributions. For any given class, C_j, $P(t_i \mid C_j)$ is the PDF for the class evaluated at one point, t_i. [1] If a probability of occurrence for each class, $P(C_j)$ is known (perhaps determined by a domain expert) then $P(C_j)P(t_i \mid C_j)$ is used to estimate the probability that t_i is in class C_j.

- Using Posterior Probabilities. Given a data value t_i, we would like to determine the probability that t_i is in a class C_j. This is denoted by $P(C_j \mid t_i)$ and is called the *Posterior Probability*. One classification approach would be to determine the posterior probability for each class and then assign t_i to the class with the highest proability.

The naive divisions used in Example 3.1 as well as decision tree techniques are examples of the first modeling approach. Neural networks fall into the third category.

[1] In this discussion each tuple in the database is assumed to consist of a single value rather than a set of values.

Name	Gender	Height	Naive	DT
Kristina	F	1.6 m	Short	Medium
Jim	M	2 m	Tall	Medium
Maggie	F	1.9 m	Medium	Tall
Martha	F	1.88 m	Medium	Tall
Stephanie	F	1.7 m	Short	Medium
Bob	M	1.85 m	Medium	Medium
Kathy	F	1.6 m	Short	Medium
Dave	M	1.7 m	Short	Medium
Worth	M	2.2 m	Tall	Tall
Steven	M	2.1 m	Tall	Tall
Debbie	F	1.8 m	Medium	Medium
Todd	M	1.95 m	Medium	Medium
Kim	F	1.9 m	Medium	Tall
Amy	F	1.8 m	Medium	Medium
Wynette	F	1.75 m	Medium	Medium

Table 3.1: Data for Height Classification

Suppose that we are given that a database consists of tuples of the form $t =< x, y >$ Where $0 \leq x \leq 8$ and $0 \leq y \leq 10$. Figure 3.1 illustrates the classification problem. Figure 3.1 a) shows the predefined classes by dividing the reference space, Figure 3.1 b) provides sample input data, and Figure 3.1 c) shows the classification of the data based on the defined classes.

A major issue associated with classification is that of overfitting. If the classification strategy fits the training data exactly, it may be applicable to a more broader population of data. For example, suppose that the training data has erroneous or noisy data. Certainly in this case, fitting the data exactly is not desired.

In the following sections various approaches to performing classification are examined. Table 3.1 contains data to be used throughout this chapter to illustrate the various techniques. This example assumes that the problem is to classify adults as short, medium, or tall. The table lists height in meters. The last two columns of this table show two classifications which could be made. With the naive approach, the simple divisions used are:

$$\begin{array}{cc} 2m \leq Height & \text{Tall} \\ 1.7m < Height < 2m & \text{Medium} \\ Height \leq 1.7m & \text{Short} \end{array}$$

The DT results use the decision tree shown in Figure 3.6.

3.1.1 Issues in Classification

Missing Data

Missing data values cause problems during both the training phase and the classification process itself. Missing values in the training data have to be handled and may produce an inaccurate result.

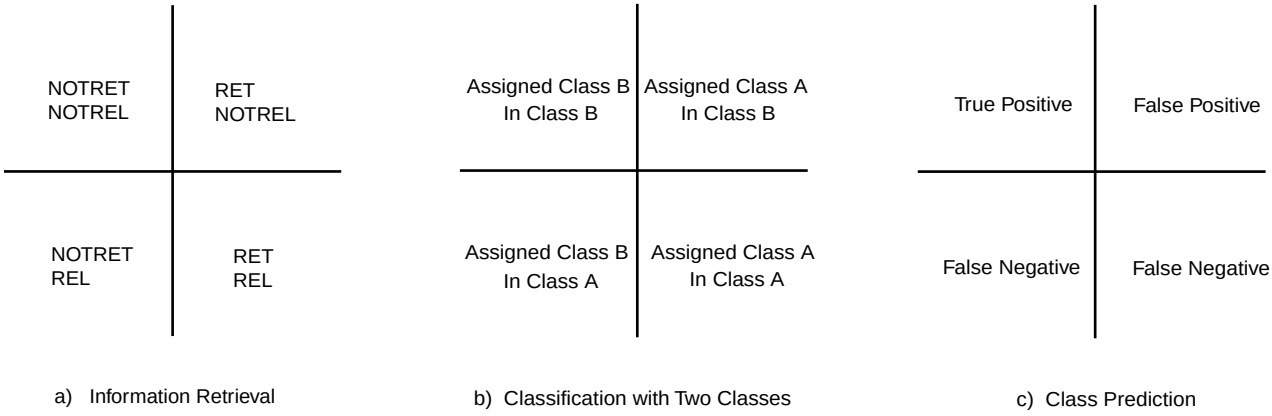


Figure 3.2: Comparing Classification Performance to Information Retrieval

Missing data in a tuple to be classified must be able to be handled by the resulting classification scheme. There are many approaches to handling missing data:

- Ignore the missing data.
- Assume a missing value for the data. This may be determined by using some method to predict what the value could be.
- Assume a special value for the missing data. This means that the missing data is taken to be a specific value all of its own.

Notice the similarity between missing data in the classification problem and that of *nulls* in traditional databases.

Measuring Performance

Table 3.1 shows two different classification results using two different classification tools. Determining which is best depends on the interpretation of the problem by users. The performance of classification algorithms is usually examined by evaluating the accuracy of the classification. However, since classification is often a fuzzy problem, the correct answer may depend on the user. Traditional algorithm evaluation approaches such as determining the space and time overhead can be used, but these are usually secondary.

Classification accuracy is usually calculated by determining the percentage of tuples placed in the correct class. This ignores the fact that there may also be a cost associated with an incorrect assignment to the wrong class. This should perhaps also be determined.

We can examine the performance of classification much as is done with Information Retrieval systems. With only two classes, there are four possible outcomes with the classification as is shown in Figure 3.2. The upper left and lower right quadrants (for both Figure 3.2 a) and b)) represent correct actions. The remaining two quadrants are incorrect actions. The performance of a classification could be determined by associating costs with each of the quadrants. However this would be difficult as the total number of costs needed is: m^2, where m is the number of classes.

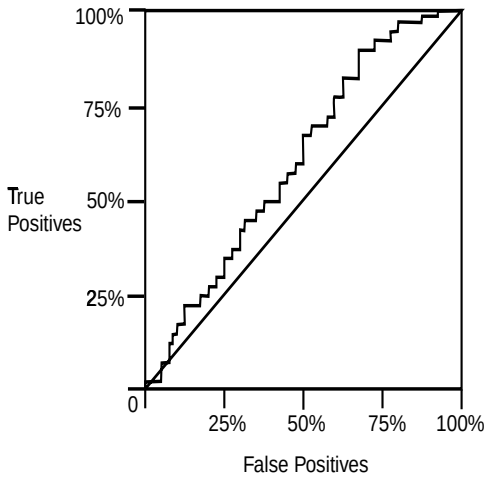


Figure 3.3: Operating Characteristic Curve

Given a specific class, C_j, and a database tuple, t_i, that tuple may or may not be assigned to that class while it's actual membership may nor may not be in that class. This again gives us the four quadrants shown in Figure 3.2 which can be described in the following ways:

- True Positive (TP): t_i predicted to be in C_j and is actually in it.
- False Positive (FP): t_i predicted to be in C_j, but is not actually in it.
- True Negative (TN): t_i not predicted to be in C_j, and is not actually in it.
- False Negative (FN): t_i not predicted to be in C_j, but is actually in it.

An *OC (Operating Characteristic) Curve* or *ROC (Receiver Operating Characteristic) Curve* or *ROC (Relative Operating Characteristic) Curve* shows the relationship between FPs and TPs. A OC curve was originally used in the communications area to examine false alarm rates. It has also been used in Information Retrieval to examine Fallout (percentage of retrieved which are not relevant) versus Recall (percentage of retrieved which are relevant). In the OC curve the horizontal axis has the percentage of False Positives and the vertical axis has the percentage of True Positives for a database sample. At the beginning of evaluating a sample, there are none of either catgory, while at the end there are 100% of each. When evaluating the results for a specific sample, the curve looks like a jagged stair-step, as seen in Figure 3.3, as each new tuple is either a FP or a TP. A more smoothed version of the OC curve can also be obtained.

A confusion matrix illustrates the accuracy of the solution to a classification problem. Given m classes, a *confusion matrix* is an mxm matrix where entry $c_{i,j}$ indicates the number of tuples from D which were assigned to class C_j but where the correct class is C_i. Obviously the best solutions will have only zero values outside the diagonal. Table 3.2 shows a confusion matrix for the height data in Table 3.1 where the Naive assignment is assumed to be correct and the DT assignment is what is actually made.

Actual Membership	Assigment		
	Short	Medium	Tall
Short	0	4	0
Medium	0	5	3
Tall	0	1	2

Table 3.2: Confusion Matrix

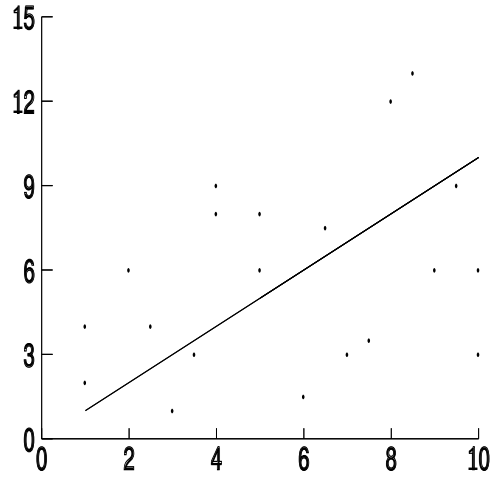


Figure 3.4: Example of Poor Fit for Linear Regression

3.2 Linear Regression

Linear regression problems deal with estimation of an output value based on input values. Here the input values are values from the database D and the output values represent the classes. Looking at Figure 2.1 from Chapter 2, we see that a simple linear regression problem can be thought of as estimating the formula for a straight line (in a two-dimensional space). This can be equated to partitioning the data into two classes. With the banking example, these would be to approve or reject a loan application. The straight line is the break even point or division between the two classes.

In Chapter 2, we briefly introduced linear regression using the formula:

$$y = c_0 + c_1 x_1 + ... + c_n x_n \tag{3.1}$$

By determining the *regression coefficients* $c_0, c_1, ..., c_n$ the relationship between the output parameter, y, and the input parameters, $x_1, , , , , x_n$ can be estimated. All high school algebra students are familiar with determining the formula for a straight line, $y = mx + b$, given two points in the $x - y$ plane. They are determining the regression coefficients m and b. Here the two points represent the training data.

Admittedly, Example 2.1 is an extremely simple problem. However, it illustrates how we all use the basic classification/prediction techniques frequently. Figure 3.4 illustrates the more general

use of linear regression with one input value. Here there is a sample of data that we wish to model (shown by the scatter dots) using a linear model. The line generated by the linear regression technique is shown in the figure. Notice, however, that the actual data points do not usually fit the linear model exactly. Thus this model is an estimate of what the actual input/output relationship is. We can use the generated linear model to predict an output value given an input value, but unlike that for Example 2.1 the prediction is an estimate rather than the actual output value. If we attempt to fit data that is not linear to a linear model, the results will be a poor model of the data as illustrated by Figure 3.4.

There are many reasons why the linear regression model may not be used to estimate output data. One is that the data does not fit a linear model. It is possible, however, that the data generally does actually represent a linear model, but the linear model generated is poor due to the fact that noise or outliers exist in the data. *Noise* is data that is erroneous. *Outliers* are data values which are exceptions to the usual and expected data. Example 3.2 illustrates outliers. In these cases the observable data may actually be described by the following:

$$y = c_0 + c_1 x_1 + ... + c_n x_n + \epsilon \tag{3.2}$$

Here ϵ is a random error with mean of 0. As with point estimation, we can estimate the accuracy of the fit of a linear regression model to the actual data using a mean squared error function.

Example 3.2 *Suppose that a graduate level Abstract Algebra class has 100 students. Kristina consistently outperforms the other students on exams. On the final exam, Kristina gets a grade of 99. The next highest grade is 75 with the range of grades being between 5 and 99. Kristina clearly is hated by the other students in the class as she does not perform at the same level that they do. She "ruins the curve". If we were to try to fit a model to the grades, this one outlier grade would cause problems as any model which attempted to include it would then not accurately model the remaining data.*

We illustrate the process using a simple linear regression formula and assuming k points in our training sample. We thus have the following k formulas:

$$y_i = c_{0i} + c_{1i} x_{1i} + \epsilon_i, i = 1, ..., k \tag{3.3}$$

With a simple linear regression, given an observable value (x_{1i}, y_i), ϵ_i estimates the error and thus the squared error technique introduced in Chapter 1 can be used to indicate the error. To minimize the error a *Method of Least Squares* is used to minimize the least squared error. This approach finds coefficients c_0, c_1 so that the squared error is minimized for the set of observable values. The sum of the squares of the errors is:

$$L = \sum_{i=1}^{k} \epsilon_i^2 = \sum_{i=1}^{k} (y_i - c_{0i} - c_{1i} x_{1i})^2 \tag{3.4}$$

Taking the partial derivatives (with respect to the coefficients) and setting equal to zero we can obtain the *Least Squares Estimates* for the coefficients, $\hat{c_{0i}}$ and $\hat{c_{1i}}$. Example 3.3 illustrates the process using the data from Table 3.1 Likewise, with multiple linear regression we can derive estimates for the coefficients that minimize the sum of the square errors.

Example 3.3 *By looking at the data in Table 3.1 and the basic understanding that the class to which a person is assigned is based on the numeric value of his/her height, in this example we apply the linear regression concept to determine how to distinguish between the short and medium classes. In looking at this data we first need to decide whether the classification is based on only the height value or both the gender and height values. The naive classification shown in Table 3.1 assumes that only the height value is pertinent while the DT and NN columns assume that both attributes are important. Here we assume only the height attribute value is needed. We thus have the linear regression formula of* $y = c_0 + c_1 x_1 + \epsilon$. *Notice that this implies we are actually going to be finding the values for* c_0 *and* c_1 *which best partitions the height numeric values into those that are short and those that are medium. Had we assumed both attributes were needed, the formula to be used would be* $y = c_0 + c_1 x_1 * c_1 x_2 + \epsilon$ *and the best values of* c_0, c_1, *and* c_1 *would be found to derive a formula for a plan in three-dimensional space. Looking at the data in Table 3.1, we see that only 12 of the 15 entries can be used to differentiate between short and medium persons. We thus obtain the following values for* y_i *in our training data:* $\{1.6, 1.9, 1.88, 1.7, 1.85, 1.6, 1.7, 1.8, 1.95, 1.9, 1.8, 1.75\}$. *We wish to minimize:*

$$L = \sum_{i=1}^{12} \epsilon_i^2 = \sum_{i=1}^{12} (y_i - c_0)^2 \tag{3.5}$$

Taking the partial derivative with respect to c_0 *and setting equal to zero we get:*
Solving for c_0 *we find that* $\hat{c_0} = 1.786$. *Notice that when we look at the data, we see that all but one of the items in the training set will be correctly classified using the value for* c_0.

If the predictors in the linear regression function are modified by some function (square, square root, etc.) then the model looks like:

$$y = c_0 + f_1(x_1) + ... + f_n(x_n) \tag{3.6}$$

where f_i is the function being used to transform the predictor. In this case the regression is called *nonlinear regression.* Linear Regression techniques, while easy to understand, are not applicable to most complex data mining applications. They don't work well with nonnumeric data. They also make the assumption that the relationship between the input and output values is linear, which of course may not be the case.

3.3 Similarity Measures

Each item which is mapped to the same class may be thought of as more alike the other items in that class than it is to the items found in other classes. Therefore similarity measures may be used to identify the "alikeness" of different items in the database. The concept of similarity measure was introduced in Chapter 2 with respect to IR retrieval. Certainly the concept is well known to anyone who has performed internet searches uses a search engine. In these cases the set of web pages represents the whole database and these are divided into two classes. Those which answer your query and those which don't. Those which answer your query should be more alike each other than those which don't answer your query. The similarity in this case is defined by the query you state - usually a keyword list. Thus the retrieved pages are similar because they all contain (to some degree) the keyword list you have specified.

The idea of similarity measures can be abstracted and applied to more general classification problems. The difficulty lies in how the similarity measures are defined and applied to the items in the database. Since most similarity measures assume numeric (and often discrete) values they may be difficult to use for more general or abstract data types. A mapping from the attribute domain to a subset of the integers may be used.

When the idea of similarity measure is used in classification where the classes are predefined, this problem is somewhat simpler than when it is used for clustering where the classes are not known in advance. Again, think of the IR example. Each IR query provides the class definition in the form of the IR query itself. So the classification problem becomes one, then, of determining similarity not among all tuples in the database but between each tuple and the query. This makes the problem an $O(n)$ problem rather than an $O(n^2)$ problem.

Using the IR approach, if we have a representative of each class we can perform classification by assigning each tuple to the class to which it is most similar. We assume here that each tuple, t_i, in the database is defined as a vector $< t_i, t_{i2}, ..., t_{ik} >$ of numeric values. Likewise we assume that each class C_j is defined by a tuple $< C_{j1}, C_{j2}, ..., C_{jk} >$ of numeric values. The classification problem is then restated as seen in Definition 3.2.

Definition 3.2 *Given a database $D = \{t_1, t_2, ..., t_n\}$ of tuples where each tuple $t_i =< t_{i1}, t_{i2}, ..., t_{ik} >$ contains numeric values and a set of classes $C = \{C_1, ..., C_m\}$ where each class $C_j =< C_{j1}, C_{j2}, ..., C_{jk} >$ has numeric values, the Classification Problem is to assign each t_i to the class C_j such that $sim(t_i, C_j) \geq sim(t_i, C_l)$ $\forall C_l \in C$ where $C_l <> C_j$.*

To calculate these similarity measures, the representative vector for each class must be determined. Referring to the three classes in Figure 3.1 a) we can determine a representative for each class by calculating the center of each region. Thus class A is represented by $< 4, 6 >$, class B by $< 2, 3 >$, and class C by $< 6, 3 >$. A simple classification technique, then, would be to place each item in the class where it is most similar to that class's center. The representative for the class may be found in other ways. For example, in pattern recognition problems, a predefined pattern can be used to represent each class. Once a similarity measure is defined, each item to be classified will be compared to each predefined pattern. The item will be placed in the class with the largest similarity value.

Figure 3.3 illustrates the use of distance measures to perform classification using the data found in Figure 3.1. The three large dark circles are the class representatives for the three classes. The dashed lines show the distance from each item to the closest center.

Algorithm 3.1
Input:
D *//Training data*
K *//Number of neighbors*
t *//Input tuple to classify*
Output:
c *//Class to which t is assigned*
KNN Algorithm:
//Algorithm to classify tuple using KNN
$N = \emptyset$;

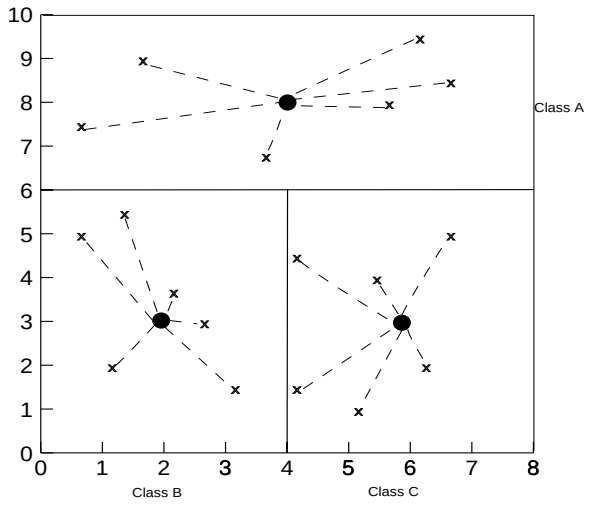


Figure 3.5: Classification Using Distrance Measures

//Find set of neighbors, N, for t
foreach $d \in D$ **do**
  if $\mid N \mid \leq K$ **then**
    $N = N \cup d$;
  else
    if $\exists\ u \in N$ *such that* $sim(t,u) \geq sim(t,d)$ **then**
      $N = N - u$;
      $N = N \cup d$;
      //Find class for classification
c = class to which the most $u \in N$ *are classified.*

One common classification scheme based on the use of distance measures is that of the *K Nearest Neighbors (KNN)*. The KNN technique assumes that the entire training set includes not only the data in the set but the desired classification for each item. In effect, the training data is the model. When a classification is to be made for a new item, its distance to each item in the training set must be determined. Only the K closest entries in the training set are considered further. The new item is then placed in the class which contains the most items from this set of K closest items. Example 3.4 illustrates the use of the KNN algorithm.

Example 3.4 *Using the sample data from Table 3.1 and the naive classification as the training set output value, we classify the tuple* $< Pat, F, 1.6 >$. *Only the height is used for distance calculation so that both the Euclidean and Manhattan distance measures yield the same results, that is the distance is simply the absolute value of the difference between the values. Suppose that K=5 is given. We then have that the K nearest neighbors to the input tuple are :* $\{< Mary, F, 1.6 >, < Kathy, F, 1.6 >, < Stephanie, F, 1.7 >, < Dave, M, 1.7 >, < Wynette, F, 1.75 >\}$. *Of these 5 items, 4 are classified as short and 1 as medium. Thus the KNN will classify Pat as medium.*

Example 3.4 illustrates this technique using the sample data from Table 3.1. The KNN technique is extremely sensitive to the value of K. A rule of thumb is that $K \leq \sqrt{number of training items}$ [62]. For this example that is 3.46.

3.4 Bayesian Classification

Assuming that the contribution by all attributes are independent and that each contributes equally to the classification problem, a simple classification scheme called *Naive Bayes* has been proposed which is based on Bayes's Rule of conditional probability as stated in Definition 2.1 in Chapter 2. The approach is called naive as it assumes the independence between the various attribute values. Given a data value x_i the probability that the related tuple is in class C_j is described by: $P(C_j \mid t_i)$. Training data can be used to determine $P(t_i)$, $P(t_i \mid C_j)$, and $P(C_j)$. From these, Bayes theorem allows us to estimate $P(C_j \mid t_i)$. Given a training set we can calculate the probability of being placed in each class given all combinations of attribute values. We illustrate the technique with the height data in Table 3.1 in Example 3.5

Example 3.5 *Using the Naive classification results for Tabel 3.1 there are 4 tuples classified as Short, 8 as Medium, and 3 as Tall. To facilitate classification, we divide the height data into six ranges:* $(0, 1.6], (1.6, 1.7], (1.7, 1.8], (1.8, 1.9], (1.9, 2.0], (2.0, \infty)$
There are Table 3.3 shows the counts and subsequent probabilities associated with the attributes. With this training data we estimate: $P(Short) = 4/15$, $P(Medium) = 8/15$, *and* $P(Tall) = 3/15$. *We use these values to classify a new tuple. For example, suppose we wish to classify* $< Adam, M, 1.95m >$. *By using these values and the associated probabilities of gender and height we obtain the following estimates:* $P(t \mid Short) = 1/4 \times 0$, $P(t \mid Medium) = 2/8 \times 1/2$, *and* $P(t \mid Tall) = 3/3 \times 1/2$. *Combining these we get:*

$$LikelihoodOfBeingShort = 1/4 \times 0 \times 4/15 = 0 \tag{3.7}$$

$$LikelihoodOfBeingMedium = 2/8 \times 1/2 \times 8/15 = 0.067 \tag{3.8}$$

$$LikelihoodOfBeingTall = 3/3 \times 1/2 \times 3/15 = 0.1 \tag{3.9}$$

We estimate $P(t)$ *by summing up these individual likelihood vlaues since t will be either Short or Medium or Tall. Finally we obtain the actual probabilities of each event:*

$$P(Short) = \frac{0}{0 + 0.067 + 0.1} = 0 \tag{3.10}$$

$$P(Medium) = \frac{0.067}{0 + 0.067 + 0.1} = 40.12 \tag{3.11}$$

$$P(Tall) = \frac{0.1}{0 + 0.067 + 0.1} = 59.88 \tag{3.12}$$

Therefore based on these probabilities, we classify the new tuple as Tall.

Attribute	Value	Count			Probabilities		
		Short	Medium	Tall	Short	Medium	Tall
Gender	M	1	2	3	1/4	2/8	3/3
	F	3	6	0	3/4	6/8	0/3
Height	$(0, 1.6]$	2	0	0	2/2	0	0
Height	$(1.6, 1.7]$	2	0	0	2/2	0	0
Height	$(1.7, 1.8]$	0	3	0	0	3/3	0
Height	$(1.8, 1.9]$	0	4	0	0	4/4	0
Height	$(1.9, 2]$	0	1	1	0	1/2	1/2
Height	$(2, \infty)$	0	0	2	0	0	2/2

Table 3.3: Probabilities Associated with Attributes

The Naive Bayes approach has several advantages. First of all, it is easy to use. Secondly, unlike other classification approaches only one scan of the training data is required. The Naive Bayes approach can easily handle missing values by simply omitting that probability when calculating the likelihoods of membership in each class.

There are, however, many disadvantages. Although the Naive Bayes is straightforward to use, it doesn't always yield satisfactory results. First of all, the attributes are not usually independent. We could use a subset of the attributes by ignoring any that are dependent on others. The techniques does not handle continous data. If numeric attributes are used the division of the domain into ranges is not an easy task and how this is done can certainly impact the results.

3.5 Decision Trees

The Decision Tree approach is most useful in classification problems. With this technique, a tree is constructed to model the classificaiton process. Once the tree is built, it is applied to each tuple in the datbase resulting in a classification for that tuple. So there are two basic steps in the technique: building the tree and applying the tree to the database. Most research has focused on how to build effective trees as the applciation process is straightforward.

The decision tree approach to classification is to divide the search space into rectangular regions. A tuple is classified based on the region into which it falls. Using Decision Trees is a very popular technique to solving the classification problems due to their simplicity, accuracy, ease of use and understanding, and speed of the algorithms. A definition for a decision tree used in classification is contained in Definition 3.3. Alternative definitions may be found. For example in a binary DT the nodes could be labeled with the predicates themselves and each arc would be labeled with Yes or No (like in the Twenty Questions Game).

Definition 3.3 *Given a database $D = \{t_1, ..., t_n\}$ where $t_i =< t_{i1}, ..., t_{ih} >$ and the database schema contains the following attributes $\{A_1, A_2, ..., A_h\}$. Also given is a set of classes $C = \{C_1, .., C_m\}$. A* **Decision Tree (DT)** *or* **Classification Tree** *is a tree associated with D that has the following properties:*

- *Each internal node is labeled with an attribute,A_i.*

- *Each arc is labeled with a predicate which can be applied to the attribute associated with the parent.*
- *Each leaf node is labeled with a class, C_j.*

Solving the classification problem using decision trees is a two step process:

1. **Decision Tree Induction**: Construct a DT using training data.
2. For each $t_i \in D$ apply the DT to determine its class.

Based on our definition of the Classification Problem, Definition 3.1, the constructed DT represents the logic needed to perform the mapping. Thus it implicitly defines the mapping. Using the DT shown in Figure 3.6 from Chapter 2, the classification of the sample data found in Table 3.1 is that shown in the column labeled DT. A different DT could yield a different classification. Since the application of a given tuple to a DT is relatively straightforward, we do not consider the second part of the problem further. Instead we focus on algorithms to contruct decision trees. In the following subsections, several algorithms are surveyed

There are many advantages to the use of DT for classification. They are certainly easy to use and efficient. Rules can be generated which are easy to interpret and understand. They scale well for large databases as the tree size is independent of the database size. Once a tree is built, the application of the tree to a database is O(n). Each tuple in the database must be filter through the tree. This takes time proportional to the height of the tree which is fixed. Trees can be constructed for data with many attributes.

Disadvantages also exist for the DT algorithms. First they do not easily handle continous data. These attribute domains will need to be divided into categories to be handled. The approach used is that the domain space is divided into rectangular regions (such as seen in Figure 3.1 a). Not all classification problems are of this type. The division shown by the simple loan classification problem in Figure 1.4 a) in Chapter 1 can not be handled by decision trees. Handling missing data is difficult as correct branches in the tree could not be taken. Since the DT is constructed from the training data, overfitting may occur. This is an be overcome via tree pruning, but of course pruning causes other problems as outlined above. Finally, correlations among attributes in the database are ignored by the DT process.

Algorithm 3.2
Input:
D *//Training data*
Output:
T *//Decision Tree*
DTBuild Algorithm:
//Simplistic algorithm to illustrate naive approach to building DT
$T = \emptyset$;
Determine best splitting criterion;
T = Create root node node and label with splitting attribute;
T = Add arc to root node for each split predicate and label;
for each arc **do**
D = Database created by applying splitting predicate to D;

if *stopping point reached for this path* **then**
  T' = *Create leaf node and label with appropriate class;*
else
  $T' = DTBuild(D)$;
T = *Add* T' *to arc;*

There have been many decision tree algorithms. We illustrate the tree building phase in the simplistic DTBuild Algorithm 3.2. Attributes in the database schema which will be used to label nodes in the tree and around which the divisions will take place are called the *splitting attributes*. The predicates by which the arcs in the tree are labeled are called the *splitting predicates*. In the descision tree shown in Figure 3.6, the splitting attributes are $\{Gender, Height\}$. The splitting predicates for Sex are $\{= Female, = Male\}$, while those for Height are $\{< 1.3m, > 1.8m, < 1.5m, > 2m\}$. Notice that the splitting predicates for height differ based on whether the tuple is for a Male or Female. This recursive algorithm builds the tree in a top town fashion by examining the training data. Using the initial training data, the "best" splitting attribute is first chosen. Algorithms differ in how they determine the "best attribute" and its "best predicates" to use for splitting. Once this has been determined the node and its arcs are created and added to the created tree. The algorithm continues recursively by adding new subtrees to each branching arc. The algorithm terminates when some "stopping criteria" is reached. Again, each algorithm determines when to stop the tree differently. One simple approach would be to stop when the tuples in the reduced training set all belong to the same class. This class is then used to label the leaf node created.

Notice that the major factors in the performance of the DTBuild algorithm is the size of the training set and how the best splitting attribute is chosen. The following issues are faced by most DT algorithms:

- **Choosing Spliting Attributes:** Which attributes to use for splitting attributes impacts the performance applying the built DT. Some attributes are better to use than others. In the data shown in Table 3.1 the Name attribute should definitely not be used and the Gender may or may not be used. The choice of attribute involves not only an examination of the data in the training set, but also the informed input of domain experts.

- **Ordering of Splitting Attributes:** The order in which the attributes are chosen is also important. In Figure 3.6 a) the Gender attribute is chosen first. Alternatively the height attribute could be chosen first. As is seen in Figure 3.6 b), in this case the height attribute must be examined a second time requiring unnecessary comparisons.

- **Splits:** Associated with the ordering of hte attributes is the number of splits to take. With some attributes, the domain is small and so the number of splits is obvious based on the domain (as with the Gender attribute). However, if the domain is contious or has a large number of values the number of splits to use is not easily answered.

- **Tree Structure:** To improve the performance of applying the tree for classification, a balanced tree with the fewest levels is desirable. However in this case, more complicated comparisons with multiway branching (see Figure 3.6 c)) may be needed. Some algorithms only build binary trees.

- **Stopping Criteria:** The creation of the tree usually stops when the training data is perfectly clasified. There may be situations when earlier stopping would be desirable to prevent the

creation of larger trees. This is a tradeoff between accuracy of classification and performance. In addition, stopping earlier may be performed to prevent overfitting. It is even conceivable that more levels than needed would be created in a tree if it is known that there are data distributions not represented in the training data itself.

- **Training Data:** The structure of the DT created depends on the training data. If the training data set is too small then the generated tree may not be specific enough to work properly with the more general data. If the training data is too large then the created tree may overfit.
- **Pruning:** Once a tree is constructed, some modifications to the tree may be need to improve the performance of the tree during the classification phase. The pruning phase may remove redundant comparisons or remove subtrees to achieve better performance.

To illustrate some of these design decisions, Figure 3.6 shows four different decision trees which can be used to classify persons as to height. The first tree is a duplicate of that from Chapter 1. The first three trees of this figure all perform the same classification. However, they all perform it differently. Underneath each tree is a graph showing the logical divisions used by the associated tree for classification. A nice feature of Figure 3.6 a) is that it is balanced. The tree is of the same depth for any path from root to leaf. Figures b) and c), however, are not balanced. In addition the height of the tree in figure b) is greater than any of the others, implying a slightly worse behavior when used for classification. However, all of these factors impact the time required to do the actual classification. These may not be crucial performance issues unless the database is extremely large. In that case a balanced shorter tree would be desirable. The tree shown in Figure 3.6 d) does not logically represent the same classification logic as the others.

The training data itself as well as the tree induciton algorithm determines the tree shape. Thus the best shaped tree which performs perfectly on the training set is desirable. Some algorithms create only binary trees. Binary trees are easily created, however they tend to be deeper. The performance results when applying these types of trees for classification may be worse due to the fact that more comparisons are usually needed. However, since these comparisons are simpler than those requiring multiple way branches, the ultimate behaviors may be comparable.

The DT building algorithms may initially build the tree and then prune it for more effective classification. With pruning techniques, portions of the tree may be removed or combined to reduce the overall size of the tree. Portions of the tree relating to classification using an attribute which is not important can be removed. This sort of change with a node close to the root couls ripple down to create major changes in the lower parts of the tree. For example, with the data in 3.1 if a tree were constructed by looking at values of the Name attribute, all nodes labeled with that attribute would be removed. Lower level nodes would move up or be combined in some way. The approach to doing this could become quite complicated. In a case of overfitting, lower level subtrees may be removed completely. Pruning may be performed while the tree is being created thus preventing a tree from become too large. A second approach prunes the tree after it is built.

The time and space complexity of DT algorithms depends on the size of the training data, n, the number of attributes, h, and the shape of the resulting tree. In the worst case, the DT which is built may be quite deep and not bushy. However if a bushy tree is built, the tree is of size $O(nlogn)$. As the tree is built, for each of these nodes, each attribute will be examined to determine if it is the best. This gives a time complexity to build the tree of $O(h\ n\ logn)$

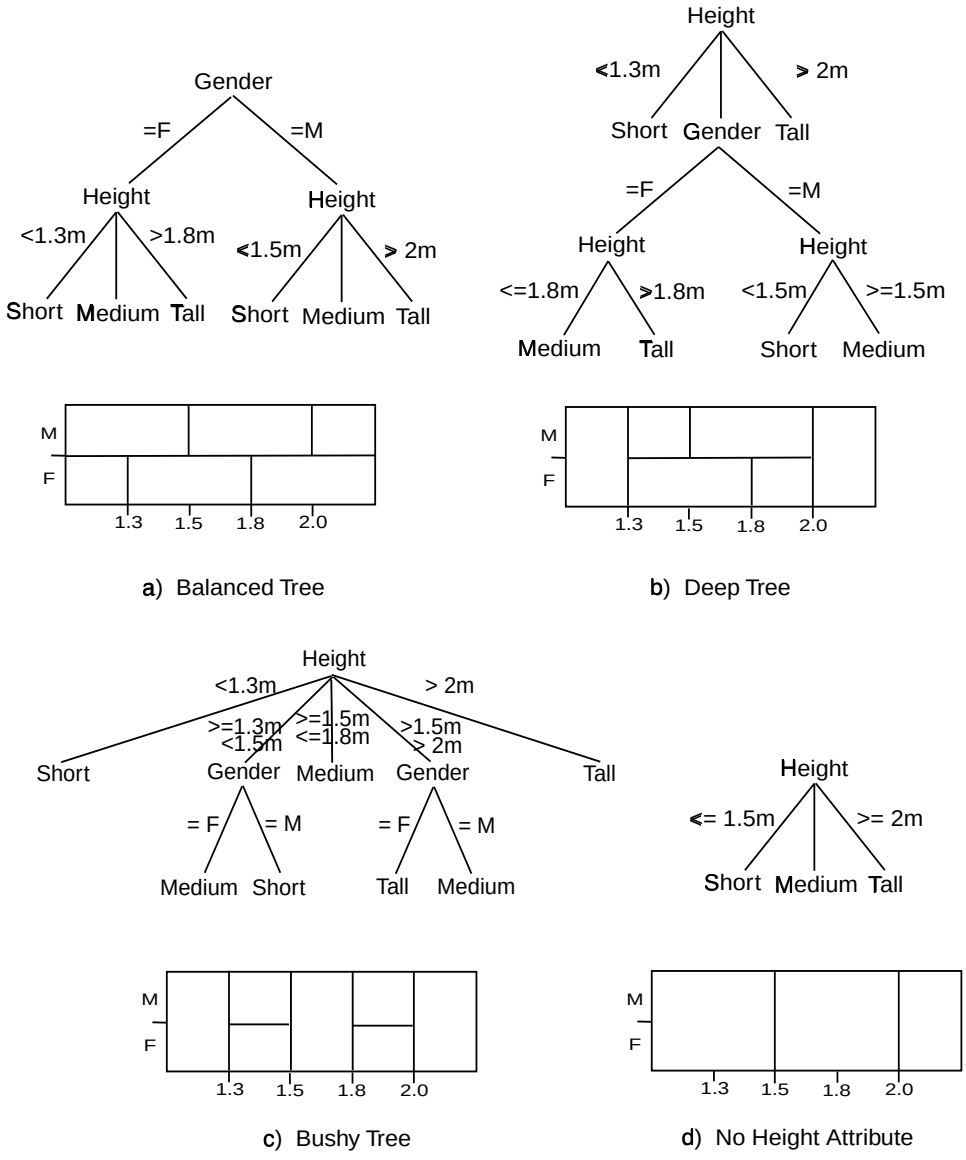
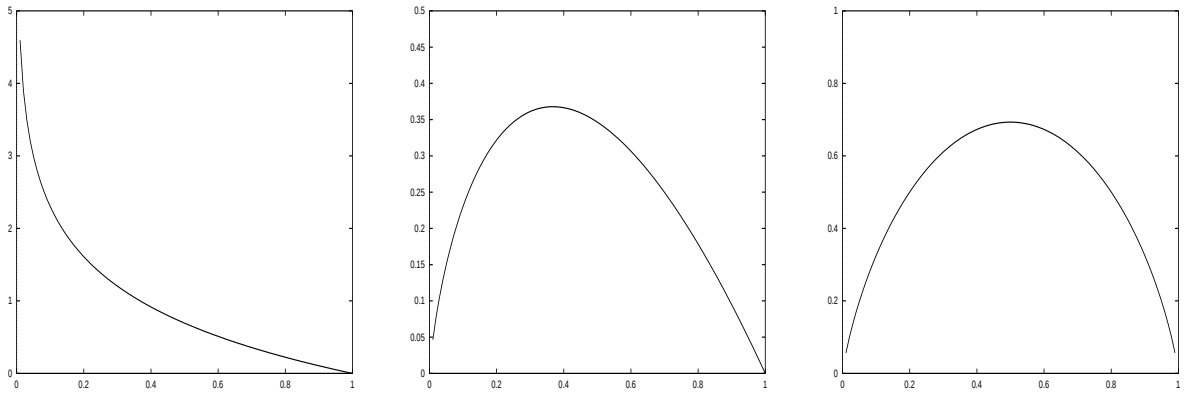


Figure 3.6: Comparing Decision Trees

In the following subsections we examine several popular DT approaches.

3.5.1 ID3

The ID3 technique to building a decision tree is based on information theory and attempts to minimize the expected number of comparisons. This technique was first proposed in the lates 1970s by J. Ross Quinlan [85]. Using information theory, the basic idea of the induciton algorithm is to ask questions whose answers provide the most information. This is similar to the intuitive approach taken by adults when playing the Twenty Questions game. The first question an adult might ask could be "Is the thing alive?", while a child might ask "Is it my Daddy?" Notice that the first question divides the search space into two large search domains while the second performs little division of the space. The basic strategy used by ID3 is to choose splitting attributes with the highest information gain first. The amount of information associated with an attribute value is related to the probability of occurrence. Looking at the Twenty Questions example, the childs question divides the search space into two sets. One (Daddy) has an infinitesimal probability associated with it and the other is almost certain. While the question the adult makes divides the search space into two subsets with almost equal probability of occuring.



a) log(1/p) b) p log(1/p) c) H(p,1-p)

Figure 3.7: Entropy

The concept used to quantify information is called entropy. Entropy is used to measure the amount of information. Information is maximized when entropy is minimized. It measure the amount of uncertainty or surprise or randomness in a set of data. The figures in Figure 3.7 will help to explain the concept. Figure 3.7 a) shows $log(1/p)$ as a probability p ranges from 0 to 1. This intuitively shows the amount of information based on the probability. When $p = 1$ there is no information. This means that is an event has a probability of 1, if you were told that the event occured you would not be surprised. There is no information in this statement. As $p \rightarrow 0$ the information increases. You are provided more information when you are told that a person is of height 2m than you are if you are told the person is female. When we deal with a divide and conquer approach like that used with decision trees, the division results in multiple probabilities whose sum is 1. In the Twenty Questions game, the $P(Daddy) < P(\neg Daddy)$ and $P(Daddy) + P(\neg Daddy) = 1$. To measure the information associated with this division we need to be able to combine the information associated with both events. That is we need to be able to

calculate the average information associated with the division. This can be performed by adding the two values together and taking into account the probability that each occurs. Figure 3.7 b) shows the function $p\ log(1/p)$ which is the expected information based on probability of an event. To determine the expected information associated with two events we add the individual values together. This function $p\ log(1/p) + (1-p)\ log(1/(1-p))$ is plotted in Figure 3.7 c). Notice that the maximum occurs when the two probabilitites are equal. This supports our intuitive idea that the more sophisticated questions posed by the adult are better than those posed by the child.

Disorder to order?

The formal definition of entropy is shown in Definition 3.4. The value for entropy is between 0 and 1 and reaches a maximum when the probabilities are all the same.

Definition 3.4 *Given probabilities* $p_1, p_2, ..., p_s$ *where* $\sum_{i=1}^{s} p_i = 1$ **Entropy** *is defined as:*

$$H(p_1, p_2, ..., p_s) = \sum_{i=1}^{s} (p_i log(1/p_i)) \tag{3.13}$$

Given a database state, D, $H(D)$ finds the amount of order (or lack thereof) in that state. When that state is split into s new states $S = \{D_1, D_2, ..., D_s\}$, we can again look at the entropy of those states. Each step in ID3 choses the state that orders the splitting the most. A database state is completly ordered if all tuples in it are in the same class. ID3 chooses the splitting attribute with the highest gain in information where gain is defined as the difference between how much information is needed to make a correct classification before the split versus how much is needed after the split. Certainly the split should reduce the information needed and reduce it by the largest amount. This is calculated by determining the differences between the entropies of the original dataset and the weighted sum of the entropies from each of the subdivided datasets. The entropies of the split datasets are weighted by the fraction of the dataset being placed in that division. the ID3 algorithm calculates the *Gain* of a particular by the following formula:

$$Gain(D, S) = H(D) - \sum_{i=1}^{s} P(D_i) H(D_i) \tag{3.14}$$

Example 3.6 and associated Figure 3.8 illustrate this process using the height data. Notice that in this example, six divisions of the possible ranges of heights is used. This division into ranges is needed when the domain of an attribute is continuous or (as in this case) consists of many possible values. While the choice of these divisions is somewhat arbitrary, a domain expert should be able to perform the task.

Example 3.6 *The beginning state of the training data in Table 3.1 (with the naive classification) is that* (4/15) *are Short,* (8/15) *are Medium, and* (3/15) *are Tall. Thus the entropy of the starting set is:* $4/15log(15/4) + 8/15log(15/8) + 3/15log(15/3) = 0.4384$
Choosing the Gender as the splitting attribute, there are 9 tuples that are F and 6 which are M. The entropy of the subset which are F is:

$$3/9log(9/3) + 6/9log(9/6) = 0.2764 \tag{3.15}$$

while that for the M subset is:

$$1/6log(6/1) + 2/6log(6/2) + 3/6log(6/3) = 0.4392 \tag{3.16}$$

The ID3 algorithm must determine what the gain in information is by using this split. To do this we calculate the weighted sum of these last two entropies to get:

$$9/150.2764) + (6/150.4392) = 0.34152 \tag{3.17}$$

The gain in entropy by using the Gender attribute is thus

$$0.4384 - 0.34152 = 0.09688 \tag{3.18}$$

Looking at the Height attribute, we have 2 tuples that are 1.6, 2 are 1.7, 1 is 1.75, 2 are 1.8, 1 is 1.85, 1 is 1.88, 2 are 1.9, 1 is 1.95, 1 is 2, 1 is 2.1, and 1 is 2.2. Determining the split values for Height is not easy. Even though the training data has these 11 values, we know that there will be many more. Just as with continuous data, we divide into ranges:

$$(0, 1.6], (1.6, 1.7], (1.7, 1.8], (1.8, 1.9], (1.9, 2.0], (2.0, \infty)$$

There are 2 tuples in the first division with entropy $(2/2(0) + 0 + 0) = 0$*, 2 in (1.6,1.7] with entropy* $(2/2(0) + 0 + 0) = 0$*. 3 in (1.7,1.8] with entropy* $(0 + 3/3(0) + 0) = 0$*, 4 in (1.8,1.9] with entropy* $(0 + 4/4(0) + 0) = 0$*, 2 in (1.9,2.0] with entropy* $(0 + 1/2(1) + 1/2(1)) = 1$*, and 2 in the last with entropy* $(0 + 0 + 2/2(0)) = 0$*. Notice that all of these states are completely ordered and thus an entropy of 0 except for the* $(1.9, 2.0]$ *state. The gain in entropy by using the Height attribute is thus*

$$0.4384 - 2/15(1) = 0.3051 \tag{3.19}$$

Thus this has the greater gain and we choose this over Gender as the first splitting attribute. Within this division there are two males - one medium and one tall. This has occurred because this grouping was too large. A further subdivision on height is needed and this generates the DT seen in Figure 3.8 a).

Figure 3.8 illustrates a problem in that the tree has multiple splits with identical results. In addition, there is a subdivision of range $[1.9, 2.0]$. Figure 3.8 b) shows an optimized version of the tree.

The ID3 approach favors attributes with a large number of divisions and thus may lead to overfitting. In the extreme, an attribute that as a unique value for each tuple in the training set would be the best as there would only be one tuple (and thus one class) for each division. An improvement can be made by taking into account the cardinality of each division. This approach uses the Gain Ratio as opposed to Gain. The *Gain Ratio* is defined as

$$GainRatio(D, S) = \frac{Gain(D, S)}{H(\mid D_1/D \mid, ..., \mid D_s/D \mid)} \tag{3.20}$$

3.5.2 C4.5 and C5.0

The decision tree algorithm C4.5 is an improvement to ID3 also proposed by Quinlan [86]. C4.5 handles missing data, handles continuous data, proposes a technique to prune the DT, and indicates

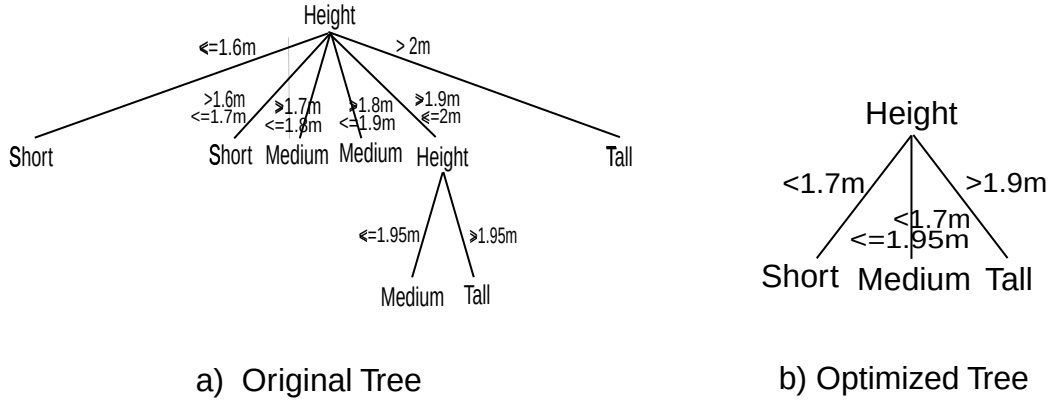


Figure 3.8: Classification Problem

how rules can be derived from the tree. For splitting purposes, C4.5 uses the largest gain ratio which ensures a larger than average information gain.

There are several pruning approaches proposed in C4.5. A subtree can be replaced by a leaf node if this replacement results in an error rate close to that of the original tree itself. Another pruning strategy is to replace a subtree by its most used subtree. Here, a subtree is raised from its current location to a node higher up in the tree. Again we must determine the increase in error rate for this replacement.

C5.0 is a commercial version of C4.5 now widely used in many data mining packages such as Clementine. The DT induction is close to that of C4.5, but the rule generation is different. Unlike C4.5, the precise algorithms used for C5.0 have not be divulged.

3.5.3 CART

Classification and Regression Trees (CART) is a technique which generates a binary decision tree [13]. As with ID3, entropy is used as a measure to choose the best splitting attribute and criterion. At each step an exhaustive search is used to determine the best split where best is defined by:

$$\Phi(s/t) = 2P_L P_R \sum_{j=1}^{m} \mid P(C_j \mid t_L) - P(C_j \mid t_R) \mid \tag{3.21}$$

This formula is evaluated at the current node, t, and for each possible splitting attribute and criterion, s. Here L and R are used to indicate the left and right subtree of the current node in the tree. P_L, P_R are the probability that a tuple in the training set will be on the left or right side of the tree. This is defined as $\frac{|tuples in subtree|}{|tuples in training set|}$ We assume that the right branch is taken on equality. $P(C_j \mid t_L), P(C_j \mid t_R)$ is the probability that a tuple is in this class, C_j, and in the left or right subtree. This is defined as the $\frac{|tuples of class j in subtree|}{|tuples at the target node|}$ At each step only one criterion is chosen as the best over all possible criteria. Example 3.7 shows its use with the height data.

Example 3.7 *The first step is to determine the split attribute and criterion for the first split. We again assume that there are six subranges to consider with the height attribute. Using these ranges*

we have the potential split values of $1.6, 1.7, 1.8, 1.9, 2.0$. *We thus have a choice of six split points which yield the following goodness measures:*

$$\Phi(Gender) = 2(6/15)(9/15)(2/15 + 4/15 + 3/15) = 0.224 \tag{3.22}$$

$$\Phi(1.6) = 0 \tag{3.23}$$

$$\Phi(1.7) = 2(2/15)(13/15)(0 + 8/15 + 3/15) = 0.169 \tag{3.24}$$

$$\Phi(1.8) = 2(5/15)(10/15)(4/15 + 6/15 + 3/15) = 0.038 \tag{3.25}$$

$$\Phi(1.9) = 2(9/15)(6/15)(4/15 + 2/15 + 3/15) = 0.256 \tag{3.26}$$

$$\Phi(2.0) = 2(12/15)(3/15)(4/15 + 8/15 + 3/15) = 0.32 \tag{3.27}$$

The largest of these is the last split. Notice that this split ends up with the right subtree only containing Tall persons while on the left there are no Tall persons. The process continues by looking at splitting the left subtree using the remaining five split points. This is left as an exercise.

Since Gender is really unordered, we use the lexicographic ordering and assume $M < F$.

As was illustrated with the Gender attribute, CART forces that an ordering of the attributes be used. CART handles missing data by simpling ignoring that record in calculating the goodness of a split on that attribute. The CART algorithm also contains a pruning strategy which we will not discuss here but can be found in [62].

3.5.4 SPRINT

Most DT algorithms assume that all or a portion of the training set is memory resident. The SPRINT technique removes all memory restrictions. In addition, it can be easily parallelized. With SPRINT, a gini index is used to find the best split. Here *gini* The *SPRINT (Scalable PaRallelizable INduction of decision Trees)* algorithm addresses the scalability issue by ensuring that the CART technique can be applied regardless of availability of main memory [94]. All memory restrictions are removed. With SPRINT, a gini index is used to find the best split. Here *gini* for a database D is defined as:

$$gini(D) = 1 - \sum {p_j}^2 \tag{3.28}$$

where p_j is the frequency of class C_j in D. The goodnes of a split of D into subsets D_1 and D_2 is defined by:

$$gini_{split}(D) = \frac{n_1}{n}(D_1) + \frac{n_1}{n}(D_1). \tag{3.29}$$

The split with the best gini value is chosen. Unlike the earlier approaches, SPRINT need not sort the data by goodness value at each node during the DT induction process. With continuous data, the split point is chosen to be the midpoint of every pair of consecutive values from the training set. range.

3.5.5 RainForest

By maintaining aggregate meta-data concerning database attributes, the *RainForest* approach allows a choice of split attribute without needing a training set [37]. For each node of a DT, a table called the *Attribute-Value Class (AVC) label group* The table summarizes for an attribute, the count of entries per class/attribute value grouping. Thus the AVC table summarizes the information needed to determine splitting attribute. The size of the table is not proportional to the size of the database or training set, but rather to the product of the number of classes, unique attribute values, and potential splitting attributes. This reduction in size (for large training sets) facilitates the scaling of decision tree induction algorithms to extremely large training sets. During the tree building phase, the training data is scanned, the AVC built, and the best splitting attribute chosen. The algorithm continues by splitting the training data and constructing the AVC for the next node.

3.6 Rules

One straightforward way to perform classification is to generate *if-then* rules which cover all cases. For example, we could have the following rules to determine classification of grades:

$$\begin{aligned} \text{If } 90 \leq grade \text{ then class} &= \text{A}. \\ \text{If } 80 \leq gradeandgrade \leq 90 \text{ then class} &= \text{B}. \\ \text{If } 70 \leq gradeandgrade \leq 80 \text{ then class} &= \text{C}. \\ \text{If } 60 \leq gradeandgrade \leq 70 \text{ then class} &= \text{D}. \\ \text{If } grade < 60 \text{ then class} &= \text{F}. \end{aligned}$$

A *Classification Rule*, $r =< a, c >$, consists of the *if* or *antecedent*, a, part and the *then* or *consequent* portion, c. The antecedent contains a predicate which can be evaluated as True or False against each tuple in the database (and obviously the training data). These rules relate directly to the corresponding DT which could be created. A decision tree can always be used to generate rules, but they are not equivalent. There are differences between rules and trees.

- The tree has an implied order to which the splitting is performed. Rules have no order.
- A tree is created based upon looking at all classes. When generating rules, only one class need be examined at a time.

There are algorithms which generate rules from trees as well as algorithms which generate rules without first creating DTs.

3.6.1 Generating Rules from a DT

The process to generate a rule from a DT is straightforward and is outlined in Algorithm Gen 3.3. This algorithm will generate a rule for each leaf node in the decision tree. All rules with the same consequent could be combined together by ORing the antecedents of the simpler rules.

Algorithm 3.3
Input:
T *//Decision Tree*
Output:

Option	Attribute	Rules	Errors	Total Errors
1	Gender	F $\rightarrow$ Medium	3/9	6/15
		M $\rightarrow$ Tall	3/6	
2	Height	(0,1.6] $\rightarrow$ Short	0/2	1/15
		(1.6,1.7] $\rightarrow$ Short	0/2	
		(1.7,1.8] $\rightarrow$ Medium	0/3	
		(1.8,1.9] $\rightarrow$ Medium	0/4	
		(1.9,2.0] $\rightarrow$ Medium	1/2	
		(2.0,∞) $\rightarrow$ Tall	0/2	

Table 3.4: 1R Classification

R *//Rules*
Gen Algorithm:
//Illustrate simple approach to generating classification rules from a DT
$R = \emptyset$
for each *path from root to a leaf in* T **do**
$a = True$
for each *internal node* **do**
$a = a \wedge (label\ of\ parent\ node\ in\ path\ combined\ with label\ of\ incident\ arc)$
$c = label of leaf node$
$R = R \cup r =< a, c >$

3.6.2 Generating Rules without a DT

Sometimes these techniques are called *Covering* algorithms as they attempt to generate rules exactly covering a specific class [107]. Tree algorithms work in a top-down divide and conquer approach, this need not be the case for covering algorithms. They generate the best rule possible by optimizing the desired classification probability. Usually the "best" attribute-value pair is chosen as opposed to the best attribute with the tree based algorithms. Suppose that we wished to generate a rule to classify persons as Tall. The basic format for the rule is then:

If ? then Class = Tall.

The objective for the covering algorithms is to replace the "?" in this statement with predicates which can be used to obtain the "best" probability of being Tall.

One simple approach is called *1R* as it generates a simple set of rules which are equivalent to a DT with only one level [50, 107]. The basic idea is to choose the best attribute to perform the classification based on the training data. Best here is defined by counting the number of errors. In Table 3.4 this approach is illustrated using the height data. If we only use the Gender attribute there are a total of 6/15 errors while if we use the Height there are only 1/15. Thus the height would be chosen and the five rules stated in the table would be used. As with ID3, 1R tends to choose attributes with a large number of values leading to overfitting. 1R can handle missing data by adding an additional attribute value for the value of *missing*. Algorithm 3.4, adapted from [107],

shows the outline for this algorithm.

Algorithm 3.4
Input:
D *//Training data*
R *//Attributes to consider for rules*
C *//Classes*
Output:
R *//Rules*
1R Algorithm:
//1R algorithm generates rules based on one attribute
$R = \emptyset$;
for each $A \in R$ **do**
for each possible value, v*, of* A **do**
//v may be a range rather than a specific value
for each $C_j \in C$ *find* $count(C_j)$;
// Here count is the number of occurences of this class for this attribute
let C_m *be the class with the largest count;*
$R = R \cup ((A = v) \rightarrow (class = C_m))$;

Another approach to generating rules without first having a DT is called *PRISM* [15]. PRISM generates rules for each class by looking at the training data and adding rules that completely describe all tuples in that class. Its accuracy is 100%. Example 3.8 illustrates the use of PRISM. Algorithm 3.5 adapted from [107] shows the process.

Example 3.8 *Using the data in Table 3.1 and the naive classification, the following shows the basic probability of putting a tuple in the Tall class based on the given attribute-value pair:*

$$\begin{array}{cc} Gender = F & 0/9 \\ Gender = M & 3/6 \end{array}$$

Based on this analysis we could generate the rule:

If Gender = M then Class = Tall.

Of course the accuracy of this rule is not high. We could add the Height attribute to this rule to make it more accurate. The rule would then become:

If Gender = M and Height in range ? then Class = Tall.

Since Height not a categorical attribute, though, we need to have ranges for it. Using the ranges from before we can develop the following:

$$\begin{array}{cc} Height <= 1.6 & 0/0 \\ 1.6 < Height <= 1.7 & 0/1 \\ 1.7 < Height <= 1.8 & 0/0 \\ 1.8 < Height <= 1.9 & 0/1 \\ 1.9 < Height <= 2.0 & 1/2 \\ 2.0 < Height & 2/2 \end{array}$$

Using these results we can expand the rule to become:

If Gender = M and Height > 2 then Class = Tall.

Algorithm 3.5
Input:
D //*Training data*
C //*Classes*
Output:
R //*Rules*
PRISM Algorithm:
//*PRISM algorithm generates rules based on best attribute-value pairs*
$R = \emptyset$;
for each $C_j \in C$ **do**
$T = D$;
//*All instances of class* C_j *will be systematically removed from* T
$p = \emptyset$;
//*Create new rule with empty left hand side*
r = (*If* p *then* C_j);
while *there are tuples in* T *belonging to class* C_j **do**
repeat
for each *attribute* A *value* v *pair found in* T **do**
calculate $(\frac{|(tuples \in T\ with\ A=v) \wedge p \wedge (\in C_j)|}{|(tuples \in T with\ A=v) \wedge p|})$ *for* $If p \wedge A = v then C_j$
Find $A = v$ *that maximizes this value*
$p = p \wedge A = v$
$T = T - \{tuples in T that satisfy A = v\}$
until *no other attributes to use or r assigns all tuples (and only) those tuples in* C_j *to* $C - j$
$D = D - tuples satisfying predicate in r$

The approach illustrated in Example 3.8 is that defined in *PRISM* [15]. It generates rules for each class by looking at the training data and adding rules that completely describe all tuples in that class. Its accuracy is 100%. Algorithm 3.5 adapted from [107] illustrates the process.

3.7 Neural Networks

With neural networks), just as with decision trees, a model representing how to classify any given database tuple is constructed. Typically the activation functions are sigmoidal. When a tuple needs to be classified, certain attribute values from that tuple are input into the directed graph at the corresponding source nodes. There is often one sink node for each class. The output value that is generated indicates the probability that the corresponding input tuple belongs to that class. The tuple will then be assigned to the class with the highest probability of membership. The learning process modifies the labeling of the arcs to better classify tuples.. Given a starting structure and value for all the labels in the graph, as each tuple in the training set is sent through the network the projected classification made by the graph can be compared to the actual classification. Based

on the accuracy of the prediction, various labelings in the graph can change. This learning process continues with all the training data or until the classification accuracy is good enough.

Solving a classification problem using NNs involves several steps:

1. as well as what attributes should be used has input. The number of hidden layers (between the source and sink nodes) must also be decided. This step is performed by a domain expert.
2. Determine weights(labels) and functions to be used for the graph.
3. For each tuple in the training set, propagate it through the network and evaluate the output prediciton to the actual result. If the prediction is accurate adjust labels to ensure this prediction has a higher output weight the next time. If the prediction is not correct, adjust the weights to provide a lower output value for this class.
4. For each tuple $t_i \in D$ propagate t_i through the network and make appropriate classification.

There are many issues to be examined:

- **Attributes (Number of Source Nodes):** This is the same issue as determining which attributes to use as splitting attributes.
- **Number of Hidden Layers:** In the simplist case there is only one hidden layer. **Number of Hidden Nodes:** A rule of thumb is that the number of hidden layers is the square root of the number of tuples in th training data set.
- **Training Data:** As with DTs, too much data and the NN may suffer from overfitting, while too little and it may not be able to classify accurately enough.
- **Number of Sinks:** Although it is usually assumed that the number of output nodes is the same as the number of classes, this is not always the case. For example with two classes there could only be one output node with the resulting value being the probability of being in the associated class. Subtracting this value from one would give the probability of being in the second class.
- **Interconnections:** In the simplest case, each node is connected to all nodes in the next level.
- **Weights:** The weight assigned to an arc indicates the relative weight between those two arcs. Initial weights are usually assumed to be small positive numbers and are assigned randomly.
- **Activation Functions:** There are many different types of activation functions which can be used. A subsection below elaborates on some of the most popular ones.
- **Learning Technique:** The technique for adjusting the weights is called the learning. Although there are many approaches which can be used, the most common is some form of back propagation which is discussed in a subsequent subsection.
- **Stop:** The learning may stop when all the training tuples have propagated through the network, or may be based on time or error rate.

There are many advantages to the use of neural networks for classification:

- NNs are more robust than DTs because of the weights.
- The NN improves its performance by learning. This can continue even after the training set has been applied.
- The use of NNs can be parallelised for better performance.
- There is a low error rate and htus a high degree of accuracy once the appropriate training has been performed.
- Neural networks are more robust than DTs in noisy environments.

On the other hand, NNs have many disadvantages:

- NNs are difficult to understand. Non technical users may have difficulting understanding how NNs work. While it is easy to explain decision trees, NNs are much more difficult.
- As with DTs, NNs may suffer from overfitting.
- Generating rules from them is not straightforward.
- Input attribute values must be numeric.
- Scalability
- Testing
- Verification
- As with DTs, overfitting may result.
- The learning phase may fail to converge.
- Neural networks are difficult to understand and explain to end users.
- Generating rules to describe the process may be difficult to understand. The learning phase can be quite expensive due to the
- They may be quite expensive to use.

3.7.1 Perceptrons

A simple perceptron can be used to classify into two classes. Using a unipolar activation function, and output of 1 would be used to classify into one class while one of 0 would be used to plass in the other class. Example 3.9 illustrates this.

Example 3.9 *Figure 3.9 a) shows a perceptron with two inputs and a Bias input. The three weights are* $3, 2, and -6$ *respsectively. The activation function* f_4 *is thus applied to the value* $S = 3\ x_1 + 2\ x_2 - 6$. *Using a simple unipolar step activation function we get:*

$$y_1 = \left\{ \begin{array}{ll} 1 & if S > 0 \\ 0 & otherwise \end{array} \right\} \tag{3.30}$$

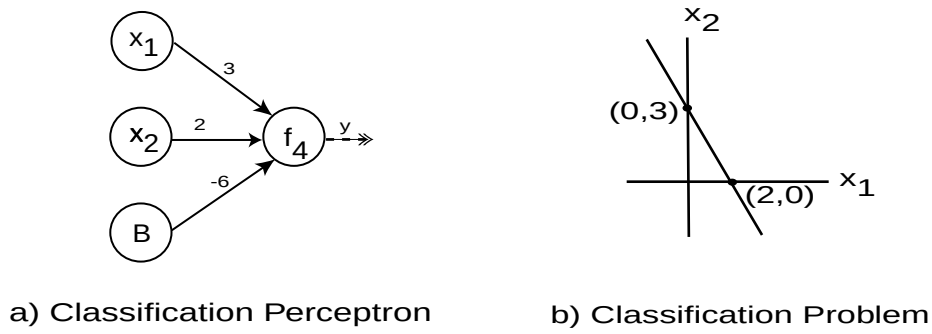


Figure 3.9: Perceptron Classification Example

An alternative way to view this classification problem is shown in Figure 3.9 b). Here x_1 is shown on the horizontal axis and x_2 on the vertical axis. The area of the plane to the right of the line $x_2 = 3 - 3/2\ x_1$ represents one class and the rest of the plane the other class.

In general, a perceptron used for classification will have one output node for each class and one input node for each attribute being used for classification. Figure 2.6 showed a perceptron used for classifying the height data given in Table 3.1.

3.7.2 Generating Rules from a NN

To increase the understanding of a neural net, classification rules may be derived from one. While the source neural network may still be used for classification, the derived rules can be used to verify or interpret the network [69, 70]. The problem is that the rules don't explicitly exist. They are buried in the structure of the graph itself. In addition, if learning is still occuring, the rules themselves are dynamic. The rules generated tend to be both more concise and have a lower error rate than rules used with decision trees [69, 70]. The basic idea of the RX algorithm is to cluster output values with the associated hidden nodes and input. A major problem existing with rule extraction is the potential size that these rules should be. For example, if you have a node with n inputs each having 5 values, there are 5^n different input combinations to this one node alone. These patterns would all have to be accounted for when constructing rules. To overcome this problem and that of having continuous ranges of output values from nodes, the output values for both the hidden and output layers are first discretized. This is accomplished by clustering the values and dividing continuous values into disjoint ranges. The rule extraction algorithm, RX, shown in Algorithm 3.6 is derived from [69].

Algorithm 3.6
Input:
D *//Training data*
N *//Initial Neural Network*
Output:
R *//Derived rules*
RX Algorithm:
//Rule Extraction algorithm to extract rules from NN

cluster output node activation values;
cluster hidden node activation values;
generate rules that describe the output values in terms of the hidden activation values;
generate rules that describe hidden output values in terms of inputs;
Combine the two sets of rules.

3.8 Exercises

1. Explain the differences between the definition of the classification problem found in Definition 3.1 and an alternative one with the mapping from C to D.

2. Apply the method of least squares technique to determine the division between medium and tall persons using the training data to be that in Table 3.1 and the classification shown in the Naive column. (See Example 3.3).

3. Apply the method of least squares technique to determine the division between short and medium persons using the training data to be that in Table 3.1 and the classification shown in the DT column. As discussed in Example 3.3, this requires finding values for c_0 and c_1. In addition, this assumes that both gender and height data values are needed to perform the classification.

4. Redo Exercise 3 using the training data to be the classification shown in the NN column.

5. Generate a DT for the height data in Table 3.1 using the ID3 algorithm and the training classifications shown in the DT column of that table. Repeat using the classificaitons in the NN column.

6. Repeat Exercise 5 using the Gain Ratio instead of the Gain.

7. Using 1R generate rules for the Height data using both the DT and NN columns in Table 3.1.

8. Complete Exercise 3.7 by generating the DT for the height data (using the naive classification) using the CART algorithm.

3.9 Bibliographic Notes

Extracting rules from neural networks has been investigated since the early 1990s [68], [104], [69].

Chapter 4

CLUSTERING

August 23, 2000

4.1 Introduction

Clustering is similar to classification in that data is grouped. However, unlike classification, the groups are not predefined. Instead, the grouping is accomplished by finding similarities between data according to characterics found in the data itself. Thus clustering is viewed to be driven by the data itself and is often based on the similarity between attribute values. The groups are called *clusters*. Some authors view clustering to be a special type of classification [59]. In this text, however, we follow a more conventional view in that the two are different. A simple example of clustering is found in Example 4.1.

Example 4.1 *An international online catalog company wishes to group its customers based on common features. Company management doesn't have any predefined labels for these groups. Based on the outcome of the grouping they will target marketing and advertising campaigns to the different groups. The information they have on the customers includes country, income, age, number of children, marital status, and education. Table 4.1 shows some tuples from this database for customers in the United States. Depending on what the type of advertising, not all attributes will be important.*

Income	Age	Children	Marital Status	Education
\$25,000	35	3	Single	High School
\$15,000	25	1	Married	High School
\$20,000	40	0	Single	High School
\$30,000	20	0	Divorced	High School
\$20,000	25	3	Divorced	College
\$70,000	60	0	Married	College
\$50,000	30	0	Married	Graduate School
\$200,000	45	5	Married	Graduate School
\$100,000	50	2	Divorced	College

Table 4.1: Sample Data for Example 4.1

For example, suppose the advertising is for a special sale on children's clothes. We could target the advertising only to the persons with children. One possible clustering is that shown by the divisions of the table. However, this example points out the fact that determining how to do the clustering is not easy.

There have been many definitions for clusters [59, 90]:

- Set of like elements. Elements from different clusters are not alike.
- Distance between points in a cluster is less than the distance between a point in the cluster and any point outside it.

A term similar to clustering is *database segmentation* where like tuples (records) in a database are grouped together. This is done to partition or segment the database into components that then give the user a more general view of the data itself. In this text we do not differentiate between segmentation and clustering.

Clustering has been used in many application domains including biology, medicine, anthropology, marketing, and economics. Clustering applications include plant and animal classification, disease classification, image processing, pattern recognition, and document retrieval. One of the first domains in which clustering was used was biological taxonomy. Recent uses also include examining Web log data to detect usage patterns.

When clustering is applied to a real world database, many interesting problems occur:

- Outlier handling is difficult. Here the elements do not naturally fall into any cluster. They can be viewed as solitary clusters. However, if a clustering algorithm attempts to find larger clusters, these outliers will be forced to be placed in some cluster. This process may end up creating poor clusters by combining two existing clusters and leaving the outlier in its own cluster. This problem is illustrated in Figure 4.1. Here if three clusters are found the outlier will occur in a cluster by itself. However, if two clusters are found the two (obviously) different sets of data will be placed in one cluster as they are closer together than the outlier. This problem is complicated by the fact that many clustering algorithms actually have as input the number of desired clusters to be found.
- Dynamic data in the database implies that cluster membership may change over time.
- Interpreting the semantic meaning of each cluster may be difficult. With classification, the labeling of the classes is known ahead of time. However, with clustering, this may not be the case. Thus when the clustering process finishes creating a set of cluster, the exact meaning of each cluster may not be obvious. Here is where a domain expert is needed to assign a label or interpretation for each cluster.
- There is not one correct answer to a clustering problem. In fact many answers may be found. Given the same data one could find 1 or 2 or 3 or more clusters. The exact number of clusters required is not easy to determine. Again a domain expert may be required. For example, suppose we have a set of data about plants that have been collected during a field trip. Without any prior knowledge of plant classification, if we attempt to divide this set of data into similar groupings, it would not be clear how many groups should be created.

- Another related issue is that of what data should be used for clustering. Unlike learning during a classification process where there is some apriori knowledge concerning what the attributes of each classification should be, in clustering we have no supervised learning to aid the process. Indeed, clustering can be viewed as similar to unsupervised learning.

We can then summarize some basic features of clustering (as opposed to classification):

- The (best) number of clusters is not known.
- There may not be any apriori knowledge concerning the clusters.
- Cluster results are dynamic.

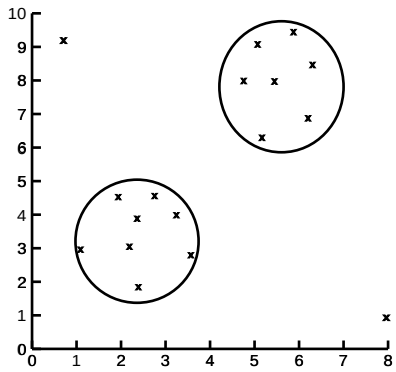


Figure 4.1: Clustering Problem

The clustering problem is stated as shown in Definition 4.1. Here, we assume that the number of clusters to be created is an input value, k. The actual content (and interpretation) of each cluster, $K_j, 1 \leq j \leq k$, is determined as a result of the function definition. Without loss of generality we will view that the result of solving a clustering problem is that a set of clusters is created: $K = \{K_1, K_2, ..., K - k\}$.

Definition 4.1 *Given a database $D = \{t_1, t_2, ..., t_n\}$ of tuples and an integer value k, the* **Clustering Problem** *is to define a mapping $f : D \rightarrow \{1, ..., k\}$ where each t_i is assigned to one cluster $K_j, 1 \leq j \leq k$. A* **Cluster**, *K_j, contains precisely those tuples mapped to it. That is $K_j = \{t_i \mid f(t_i) = K_j, 1 \leq i \leq n, \text{and } t_i \in D\}$.*

A classification of the different types of clustering algorithms has been given in [59, Figure 3.1, p. 56] and is shown in Figure 4.2. Here they view clustering as a special type of classification, which is a common view even though we treat them separately in this text. At the highest level, clusters can be either overlapping or non-overlapping. Even though we only consider non-overlapping, it is possible to place an item in multiple clusters. In turn, non-overlapping can be viewed as extrinsic or intrinsic. *Extrinsic* techniques use labeling of the items to assist in the classification process. These algorithms are the traditional classification supervised learning algorithms where a special input training set is used. *Intrinsic* algorithms do not use any apriori category labels, but only depend on

the adjacency matrix containing the distance between objects. All algorithms we examine in this chapter fall into the instrinsic class. Clustering algorithms, then, may be viewed as hierarchical or partitional. With *hierarchical* clustering, a nested set of clusters is created. Each level in the hierarchy has a separate set of clusters. At the lowest level each item is in its own unique cluster. At the highest level, all items belong to the same cluster. With hierarchical clustering, the desired number of clusters is not input. With *partitional* clustering, the algorithm creates one set of clusters only. These use the desired number of clusters to drive how the final set is created.

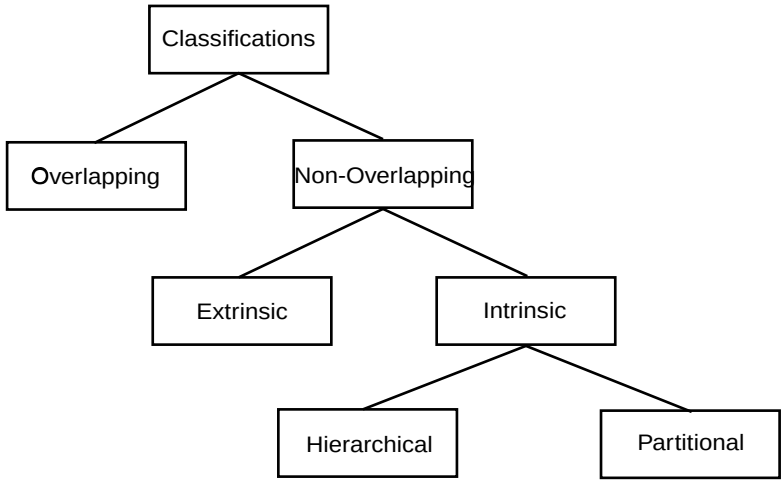


Figure 4.2: Classification of Clustering Algorithms, from [59, Figure 3.1, p. 56]

The types of clustering algorithms can be furthered classified based on the implementation technique used. Again looking at [59] we find several classification criteria. Hierarchical algorithms can be categorized as agglomerative or divisive. *Agglomerative* implies that the clusters are created in a bottom up fashion, while *divisive* algorithms work in a top down mode. Although both hierarchical and partitional algorithms could be described using the agglomerative vs. divisive label, it is more typically associated with hierarchical algorithms. Another descriptive tag indicates whether each individual element is handled one by one, *serial* (sometimes called *incremental*), or whether all items are examined together, *simultaneous*. If a specific tuple is viewed as having attribute values for all attributes in the schema, then clustering algorithms could differ as to how the attribute values are examined. As is usually done with decision tree classification techniques, some algorithms examine attribute values one at a time, *monothetic*. *Polythetic* algorithms consider all attribute values at one time. Finally, clustering algorithms can be labeled based on the mathematical formulation given to the algorithm: graph theoretic or matrix algebra. In this chapter we generally use the graph approach and describe the input to the clustering algorithm as an adjacency matrix labeled with distance measures.

In the following sections we will discuss many representative clustering algorithms. This is only a representative subset of the many algorithms that have been proposed in the literature.

4.2 Similarity/Distance Measures

There are many desirable properties for the clusters created by a solution to a specific clustering problem. The most important one is that a tuple within one cluster is more like tuples within that cluster than it is to tuples outside it. As with classification, then, we assume the definition of a similarity measure, $sim(t_i, t_l)$, defined between any two tuples, $t_i, t_l \in D$. This provides a more strict (and alternative clustering definition) as found in Definition 4.2. Unless otherwise stated, we will use the first definition rather than the second. Although keep in mind, the similarity relationship stated within the second definition is a desirable, although not always obtainable, property.

Definition 4.2 *Given a database $D = \{t_1, t_2, ..., t_n\}$ of tuples, a similarity measure, $sim(t_i, t_l)$, defined between any two tuples, $t_i, t_l \in D$, and an integer value k, the* **Clustering Problem** *is to define a mapping $f : D \rightarrow \{1, ..., k\}$ where each t_i is assigned to one cluster $K_j, 1 \leq j \leq k$. Given a cluster, K_j, $\forall t_{jl}, t_{jm} \in K_j$ and $t_i \notin K_j, sim(t_{jl}, t_{jm}) > sim(t_{jl}, t_i)$.*

As was stated earlier, clustering is impacted by the existence of outliers. As a matter of fact, clustering may be performed not so much to create groups of like records but to identify records which don't seem to fit any grouping.

The distance measure, $dis(t_i, j)$, as opposed to similarity is often used in clustering. The clustering problem then has the desirable property that given a cluster, K_j, $\forall t_{jl}, t_{jm} \in K_j$ *and* $t_i \notin K_j, dis(t_{jl}, t_{jm}) \leq dis(t_{jl}, t_i)$.

Some clustering algorithms look at numeric data only, usually assuming metric data points. *Metric* attributes satisfy the triangular inequality. Given a cluster, K_m of N points $\{t_{m1}, t_{m2}, ..., t_{mN}\}$ we make the following definitions [113]:

$$centroid = C_m = \frac{\sum_{i=1}^{N}(t_{mi})}{N} \tag{4.1}$$

$$radius = R_m = \sqrt{\frac{\sum_{i=1}^{N}(t_{mi} - C_m)^2}{N}} \tag{4.2}$$

$$diameter = D_m = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}(t_{mi} - t_{mj})^2}{(N)(N-1)}} \tag{4.3}$$

Many clustering algorithms require that the distance between clusters (rather than elements) be determined. This is not an easy task given that there are many interpretations for distance between clusters. Given clusters K_i and K_j, there are several standard alternatives to calculate the distance between clusters [114, p.18]. A representative list is:

- *Single Link*: Smallest distance between an element in one cluster and an element in the other. We thus have: $dis(K_i, K_J) = min(dis(t_{il}, t_{jm}))\forall t_{il} \in K_i \notin K_j$ *and* $\forall t_{jm} \in K_j \notin K_i$.
- *Complete Link*: Largest distance between an element in one cluster and an element in the other. We thus have: $dis(K_i, K_J) = max(dis(t_{il}, t_{jm}))\forall t_{il} \in K_i \notin K_j$ *and* $\forall t_{jm} \in K_j \notin K_i$.
- *Average*: Average distance between an element in one cluster and an element in the other. We thus have: $dis(K_i, K_J) = mean(dis(t_{il}, t_{jm}))\forall t_{il} \in K_i \notin K_j$ *and* $\forall t_{jm} \in K_j \notin K_i$.

- *Centroid*: If clusters have a representative centroid, then the centroid distance is defined as the distance between the centroids. We thus have: $dis(K_i, K_J) = dis(C_i, C_j)$ where C_i is the centroid for K_i and similarly for C_j.

Some clustering algorithms alternatively assume that the cluster is represented by one centrally located object in the cluster called a *medoid*

4.3 Hierarchical

As was mentioned earlier, hierarchical clustering algorithms actually create a set of clusters. Example 4.2 illustrates the concept. Hierarchical algorithms differ in how the sets are created. A tree data structure, called a *dendrogram*, can be used to illustrate the hierarchical clustering technique and the sets of different clusters. The root in a dendrogram tree contains one cluster where all elements are together. The leaves in the dendrogram each consist of a single element cluster. Internal nodes in the dendrogram represent new clusters formed by merging the clusters which appear as its children in the tree. Each level in the tree is associated with the distance measure which was used to merge the clusters together. All clusters created at a particular level were combined because the children clusters had a distance between them less than the distance value associated with this level in the tree. A dendrogram for Example 4.2 is seen in Figure 4.4.

Example 4.2 *Figure 4.3 shows six elements, { A,B,C,D,E,F }, to be clustered. Portions a) - e) of this figure show five different sets of clusters. In part a) of this figure each cluster is viewed to consist of a single element. Part b) illustrates four clusters. Here there are two sets of two-element clusters. These clusters are formed at this level because these two elements are closer to each other than any of the other elements. Part c) shows a new cluster formed by adding a close element to one of the two-element clusters. In part d) the two-element and three-element clusters are merged together to give a five-element cluster. This is done because these two clusters are closer to each other than to the remote element cluster, { F }. Finally at the last stage, part e), all six elements are merged together.*

The space complexity for hierarchical algorithms is $O(n^2)$ as this is the space required for the adjacency matrix. The space required for the dendrogram is $O(kn)$, but this is much less than $O(n^2)$. The time complexity for hierarchical algorithms is $O(kn^2 maxd^2)$ there is one iteration for each distance from 0 up to the maximum distance $maxd$.

Hierarchical techniques are well suited for many clustering applications which naturally exhibit a nesting relationship between clusters. For example, in biology developing plant and animal taxonomies could easily be viewed as a hierarchy of clusters.

4.3.1 Agglomerative

Agglomerative algorithms start with each individual item in its own cluster and iteratively merge clusters together until all items belong in one cluster. Different agglomerative algorithms differ in how the clusters are actually merged together at each level. Algorithm 4.1 illustrates the typical agglomerative clustering algorithm. It assumes that a set of elements and distances between them

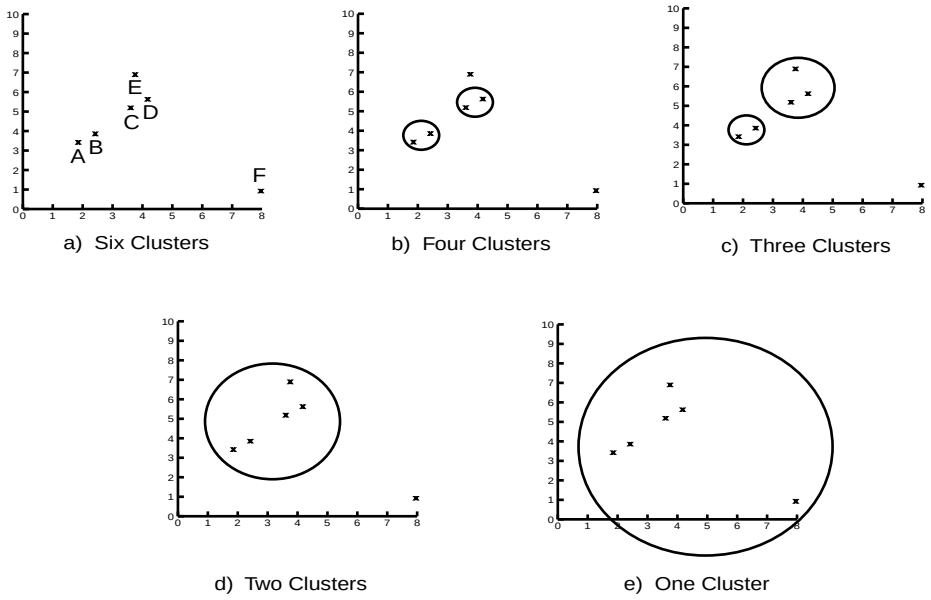


Figure 4.3: Five Levels of Clustering for Example 4.2

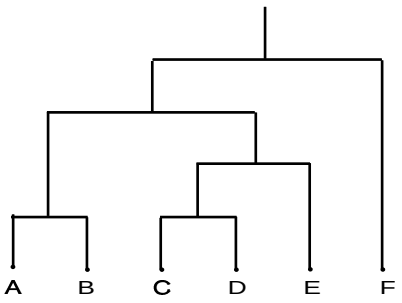


Figure 4.4: Dendrogram for Example 4.2

is given as input. We use an $n \times n$ vertex adjacency matrix, A, as input of the distances. Here the adjacenty matrix, A, contains a distance value rather than a simple boolean value: $A[i,j] dis(t_i, t_j)$. The output of the algorithm is a dendrogram, DE, which we represent as a set of ordered triples $< d, k, K >$ where d is the threshold distance, k is the number of clusters, and K is the set of clusters. The dendrogram in Figure **??** a) would be represented by the following:

$$\{< 0, 5, \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\} >, < 1, 3, \{\{A, B\}, \{C, D\}, \{E\}\} >,$$

$$< 2, 2, \{\{A, B, C, D\}, \{E\}\} >, < 3, 1, \{\{A, B, C, D, E\}\} >\}$$

Outputing the dendrogram actually produces a set of clusters rather than just one clustering. The user can determine which of the clusters (based on distance threshold) she wishes to use.

Algorithm 4.1
Input:
$D = \{t_1, t_2, ..., t_n\}$ // *Set of elements*
A // *Adjacency matrix showing distance between elements.*
Output:
DE // *Dendrogram represented as a set of ordered pairs.*
Agglomerative Algorithm:
$d = 0$;
$k = n$;
$K = \{\{t_1\}, ..., \{t_n\}\}$;
$DE =< d, k, K >$; // *Initially dendrogram contains each element in its own cluster.*
repeat
$oldk = k$;
$d = d + 1$;
$A_d =$ *Vertex adjacency matrix for graph with threshold distance of* d;
$< k, K >= NewClusters(A_d, D)$;
if $oldk \neq k$ **then**
$DE = DE \cup < d, k, K >$; // *New set of clusters added to dendrogram.*
until $k = 1$

This algorithm uses a procedure $NewClusters$ to determine how to create the next level of clusters from the previous level. This is where the different types of agglomerative algorithms differ. It is possible that only two clusters from the prior level are merged or that multiple clusters are merged. Algorithms also differ in which clusters are merged when there are several with identical distances. In addition the technique used to determine the distance between clusters may vary. *Single Link*, *Complete Link*, and *Average Link* techniques are perhaps the most well known agglomerative techniques based on well known graph theory concepts.

All agglomerative approaches suffer from excessive time and space constraints. The space required for the adjacency matrix os $O(n^2)$ where there are n items to cluster. Due to the iterative nature of the algorithm, the matrix (or a subset of it) must be accessed multiple times. The simplistic algorithm provided in Algorithm **??** performs at most $maxdist$ examinations of this matrix where $maxdist$ is the largest distance between any two points. In addition, the complexity of the $NewClusters$ procedure could also be expensive. This is a potentially severe problem in large databases. Another issue which exists with the agglomerative approach is that it is not incremental. Thus when new elements are added or old ones removed/changed the entire algorithm must be rerun. More recent incremental variations as discussed later in this text address this problem.

Single Link

The Single Link technique is based on the idea of finding maximal connected components in a graph. A *connected component* is a graph in which there exists a path between any two vertices. With the single link approach two clusters are merged together if there is at least one edge which connects the two clusters. That is, if the minimum distance between any two points is less than

or equal to the threshold distance being considered. For this reason it is often called the *nearest neighbor* clustering technique. Example 4.3 illustrates the process.

Example 4.3 *The table below contains five sample data items with distance between the elements indicated in the table entries:*

Item	*A*	*B*	*C*	*D*	*E*
A	*0*	*1*	*2*	*2*	*3*
B	*1*	*0*	*2*	*4*	*3*
C	*2*	*2*	*0*	*1*	*5*
D	*2*	*4*	*1*	*0*	*3*
E	*3*	*3*	*5*	*3*	*0*

When viewed as a graph problem, Figure 4.5 a) shows the general graph with all edges labeled with the respective distances. To understand the idea behind the hierarchical approaches, we show several graph variations in figures Figure 4.5 b), c), d), and e). Figure 4.5 b) shows only those edges with a distance of 1 or less. There are only two edges. The first level of single link clustering, then will combine the connected clusters (single elements from the first phase) giving 3 clusters: {A,B}, {C,D}, and {E}. During the next level of clustering, we look at edges with a length of 2 or less. The graph representing this threshold distance is shown in Figure 4.5 c). Notice that we now have an edge (actually three) between the two clusters {A,B} and {C,D}. Thus at this level of the single link clustering algorithm we merge these two clusters to obtain a total of two clusters: {A,B,C,D} and {E}. The graph which is created with a threshold distance of 3 is shown in Figure 4.5 d). Here the graph is connected, thus the two clusters from the last level are merged into one large cluster containing all elements. The dendrogram for this single link example is shown in Figure 4.6 a). The labeling on the right hand side shows the threshold distance used to merge the clusters at each level.

The Single Link Algorithm is obtained by replacing the *NewClusters* procedure in the Agglomerative algorithm, with a procedure to find connected components of a graph. As this is a well known graph technique we do not present an algorithm for this here, but instead refer the reader to any graph theory or data structures text [45]. We assume that this connected components procedure has as input a graph (actually represented by a vertex adjacency matrix and set of vertices) and outputs a set of connected components defined by a number (indicating the number of components) and an array containing the membership of each component. Notice that this is exactly what the last two entries in the ordered triple are used by the dendrogram data structure.

The single link approach is quite simple, but suffers from several problems. This algorithm is not very efficient as at each iteration the connected components procedure, which is an $O(n^2)$ space and time algorithm, is called. A more efficient algorithm could be developed by looking at which clusters from an earlier level can be merged at each step. Another problem is that the clustering creates clusters with long chains.

An alternative view to merging clusters in the Single Link approach is that two clusters are merged at a stage where the threshold distance is d if the minimum distance between any vertex in one and any vertex in the other is at least d.

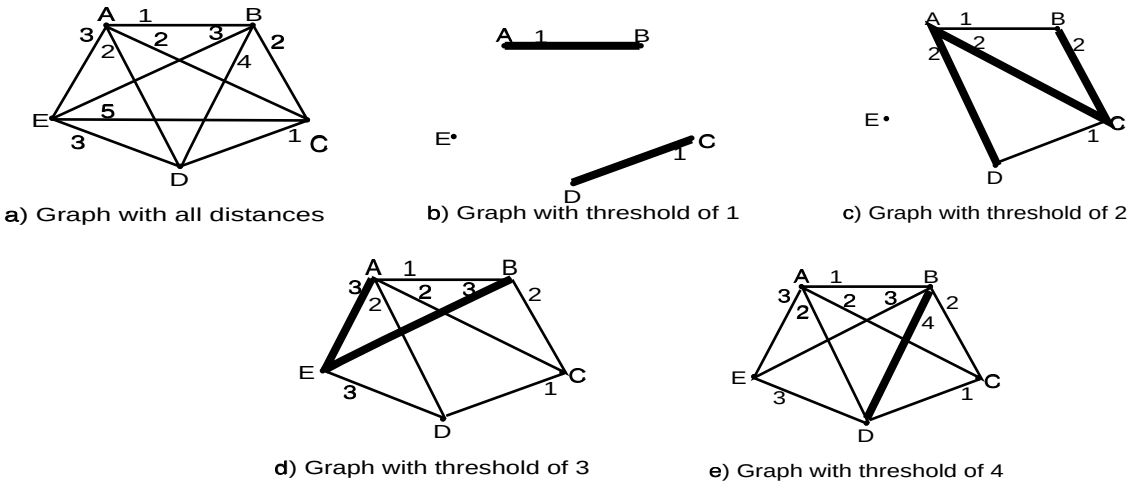


Figure 4.5: Graphs for Example 4.3

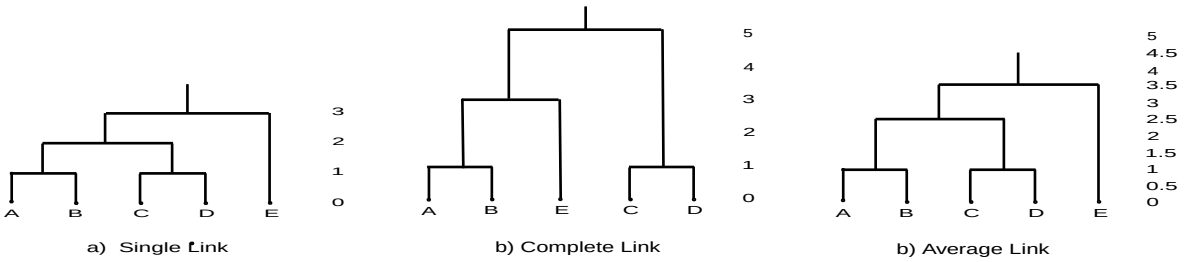


Figure 4.6: Dendrograms for Example 4.3

There have been other variations of the Single Link algorithm. One, based on the use of a *Minimum Spanning Tree (MST)* is shown in Algorithm 4.2. Here we asume that a procedure, MST, produces a minimum spanning tree given an adjacency matrix as input. The clusters are merged together in increasing order of the distance found in the MST. In the algorithm we show that once two clusters are merged the distance between them in the tree becomes ∞. We could have replaced the two nodes and edge with one node instead.

Algorithm 4.2
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*

A // *Adjacency matrix showing distance between elements.*

Output:

DE // *Dendrogram represented as a set of ordered pairs.*

MST Single Link Algorithm:

$d = 0$;
$k = n$;
$K = \{\{t_1\}, ..., \{t_n\}\}$;
$DE =< d, k, K >$; // *Initially dendrogram contains each element in its own cluster.*
$M = MST(A)$;
repeat
  $oldk = k$;
  K_i, K_j = *two clusters closest together in MST;*
  $K = K - \{K_i\} - \{K_j\} \cup \{K_i \cup K_j\}$;
  $k = oldk - 1$;
  $d = dis(K_i, K_j)$;
  $DE = DE \cup < d, k, K >$; // *New set of clusters added to dendrogram.*
  $dis(K_i, K_j) = \infty$;
until $k = 1$

We illustrate this algorithm using the data in Example 4.3. Figure 4.7 shows one MST for the example. The algorithm will merge A and B then C and D (or the other way). These two clusters will then be merged at a threashold of 2. Finally E will be merged in at 3. Notice we get exactly the same dendrogram as in Figure 4.6.

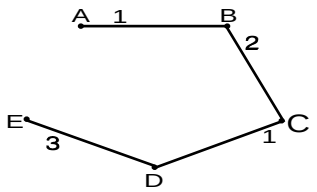


Figure 4.7: MST for Example 1

The time complexity of this algorithm is $O(n^2)$ as the the procedure to create the minimum spanning tree is $O(n^2)$ and it dominates the time of the algorithm. Once it is created having $n-1$ edges, the **repeat** loop will only be repeated $n-1$ times.

Complete Link

Although the Complete Link algorithm is similar to the Single Link, it looks for cliques rather than connected components. A *clique* is a graph in which there is an edge between any two vertices. Here a procedure is used to find the maximum distance between any clusters. So that two clusters are merged if the maximum distance is less than or equal to the distance threshold. In this algorithm

we assume the existence of a procedure, Clique, which finds all cliques in a graph. As with the Single Link algorithm, this is expensive as it is an $O(n^2)$ algorithm.

Clusters found with the complete link method tend to be more compact than those found using the Single Link technique. Using the data found in Example 4.3, Figure 4.6 b) shows the dendrogram created. A variation of the Complete Link algorithm is called the *Farthest Neighbor*. Here clusters are merged if the distance between them is the smallest (measured by looking at the maximum distance between any two points).

Average Link

The average link technique merges two clusters if the average distance between any two points in the two target clusters is below the distance threshold. The algorithm used here is slightly different from that found in single and complete link as we must examine the complete graph (not just the threshold graph) at each stage. Thus we restate this algorithm in Algorithm 4.3.

Algorithm 4.3
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*
A *// Adjacency matrix showing distance between elements.*
Output:
DE *// Dendrogram represented as a set of ordered pairs.*
Average Link Algorithm:

```
d = 0;
k = n;
K = {{t1}, ..., {tn}};
DE =< d, k, K >; // Initially dendrogram contains each element in its own cluster.
repeat
    oldk = k;
    d = d + 0.5;
    for each pair of Ki, Kj ∈ K do
        ave = average distance between all ti ∈ Ki and tj ∈ Kj;
        if ave ≤ 1 then
            K = K − {Ki} − {Kj} ∪ {Ki ∪ Kj};
            k = oldk − 1;
            DE = DE∪ < d, k, K >; // New set of clusters added to dendrogram.
    until k = 1
```

Notice that in this algorithm we increment d by 0.5 rather than 1. This is a rather arbitrary decision based on understanding of the data. Certainly we could have used a increment of 1, but we would have had a different dendrogram than that seen in Figure 4.6 c).

4.3.2 Divisive

With divisive clustering, all items are initially placed in one cluster and clusters are repeatedly split in two until all items are in their own cluster. The idea is to split up clusters where some elements

are not sufficiently close enough to other elements.

One simple example of a divisive algorithm is based on the MST version of Single Link. Here, however, we cut out edges from the MST from the largest to the smallest. Looking at Figure 4.7 we would start off with a cluster containing all items: $\{A, B, C, D, E\}$. Looking at the MST we see the largest edge is between D and E. Cutting this out of the MST we then split the one cluster into two: $\{E\}$ and $\{A, B, C, D\}$. Next we remove the edge between B and C. This splits the one large cluster into two: $\{A, B\}$ and $\{C, D\}$. These will then be split at the next step. The order depends on how a specific implementation would treat identical values. Looking at the dendrogram in Figure 4.6 a), we see that we have created the same set of clusters as the agglomerative approach just in reverse order.

4.4 Partitional

Nonhierarchical or Partitional clustering creates the clusters in one step as opposed to several steps. Only one set of clusters is created, although internally within the various algorithms several different sets of clusters may be created. Since only one set of clusters is output, the user must input the desired number, k, of clusters. In addition, some metric or criterion function is used to determine the goodness of any proposed solution. This measure of quality could be the average distance between clusters or some other metric. The solution with the best value for the criterion function is the clustering solution used.

A problem with partitional algorithms is that they suffer from a combinatorial explosion due to the number of possible solutions. Clearly searching all possible clustering alternatives would not usually be feasible. For example, given a measurement criteria a naive approach could look at all possible sets of k clusters. There are $S(n, k)$ possible combinations to examine [59, p.91]. Here:

$$S(n,k) = \frac{1}{k!}\sum_{i=1}^{k}(-1)^{k-i}\left(\begin{array}{c} k \\ i \end{array}\right)(i)^n. \tag{4.4}$$

There are 11,259,666,000 different ways to cluster 19 items into 4 clusters. Thus most algorithms only look at a small subset of all the clusters using some strategy to identify sensible clusters. Due to the plethora of partitional algorithms we will look at only a representative few. We've chosen some of the most well known ones as well as some others which have appeared recently in the literature.

4.4.1 Minimum Spanning Tree

As we have seen agglomerative and divisive algorithms based on the use of an MST, we also present a partitional MST algorithm. This is a very simplistic approach, but it illustrates how partitional algorithms work. The algorithm (originally proposed by Zahn [112]) is shown in Algorithm 4.4. Since the clustering problem is to define a mapping, the output of this algorithm shows the clusters as a set of ordered pairs $< t_i, j >$ where $f(t_i) = K_j$.

Algorithm 4.4
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*

A // *Adjacency matrix showing distance between elements.*
k // *Number of desired clusters.*

Output:

f // *Mapping represented as a set of ordered pairs.*

Partitional MST Algorithm:

$M = MST(A)$;
identify inconsistent edges in M;
remove $k-1$ *inconsistent edges;*
create output representation;

The problem is how to define inconsistent. It could be defined as in the earlier division MST algorithm based on distance. This would remove the largest $k-1$ edges from the starting completely connected graph and yield the same results as this corresponding level in the dendrogram. Zahn proposes more reasonable inconsistent measures based on the weight (distance) of an edge as compared to those close to it. For example an inconsistent edge would be one whose weight is much larger than the average of the adjacent edges.

The time complexity of this algorithm is again dominated by the MST procedure which is $O(n^2)$. At most $k-1$ edges will be removed so the last three steps of the algorithm, assuming each step takes a constant time, is only $O(k-1)$. Although determining the inconsistent edges in M may be quite complicated, it will not require a time greater than the number of edges in M. When looking at edges adjacent to one edge, there are at most $k-2$ of these. In this case, then, the last three steps are $O(k^2)$, and the total algorithm is still $O(n^2)$.

4.4.2 Squared Error Clustering

The *Squared Error* clustering algorithm minimizes the squared error. The *squared error for a cluster* is the sum of the squared Euclidean distances between each element in the cluster and the cluster centroid,C_k. Given a cluster K_i let the set of items mapped to that be $\{t_{i1}, t_{i2}, ..., t_{im}\}$. The squared error is defined as:

$$se_{K_i} = \sum_{j=1}^{im} \|t_{ij} - C_k\|^2 \tag{4.5}$$

Given a set of clusters $K = \{K_1, K_2, ..., K_k\}$, the *squared error for* K is defined as:

$$se_K = \sum_{j=1}^{k} se_{K_j} \tag{4.6}$$

In actuality, there are many different examples of squared error clustering algorithms. They all follow the basic algorithm structure shown in Algorithm 4.5.

Algorithm 4.5

Input:

$D = \{t_1, t_2, ..., t_n\}$ // *Set of elements*
A // *Adjacency matrix showing distance between elements.*

k // *Number of desired clusters.*

Output:

K // *Set of clusters.*

Squared Error Algorithm:

assign each item t_i *to a cluster;*
calculate center for each cluster;
repeat
assign each item t_i *to the cluster which has the closest center ;*
calculate new center for each cluster;
until *the difference between successive squared errors is below a threshold;*

4.4.3 K-Means Clustering

K-Means is an iterative clustering algorithm where items are moved among sets of clusters until the desired set is reached. As such it may be viewed as a type of squared error algorithm although the convergence criteria need not be defined based on the squared error. A high degree of similarity among elements in clusters is obtained while simultaneously a high degree of dissimilarity among elements in different clusters is also achieved. The *cluster mean* of $K_i = \{t_{i1}, t_{i2}, ..., t_{im}\}$ is defined as:

$$m_i = \frac{1}{m} \sum_{j=1}^{m} t_{ij} \tag{4.7}$$

Here it is assumed that each tuple has only one numeric value as opposed to a tuple with many attribute values. The K-Means algorithm assumes that some definition of cluster mean exists, but it does not have to be this particular one. This algorithm assumes that the desired number of clusters, k, is an input parameter. Algorithm 4.6 shows the K-Means algorithm. Notice that the initial values for the means are arbitrarily assigned. These could be assigned randomly or perhaps use the values from the first k input items themselves. The convergence criteria could be based on the squared error, but need not be. For example the algorithm could stop when no (or a very small) number of tuples are assigned to different clusters. Other termination techniques have looked at simply a fixed number of iterations.

Algorithm 4.6

Input:

$D = \{t_1, t_2, ..., t_n\}$ // *Set of elements*
A // *Adjacency matrix showing distance between elements.*
k // *Number of desired clusters.*

Output:

K // *Set of clusters.*

K-Means Algorithm:

assign initial values for means $m_1, m_2, ..., m_k$ *;*
repeat
assign each item t_i *to the cluster which has the closest mean ;*
calculate new mean for each cluster;

until *convergence criteria is met;*

The K-Means algorithm is illustrated in Example 4.4.

Example 4.4 *Suppose that we are given the following items to cluster:*

$$\{2, 4, 10, 12, 3, 20, 30, 11, 25\} \tag{4.8}$$

and supposed that $k = 2$. We initially assign the means to the first two values: $m_1 = 2$ and $m_2 = 4$. Using Euclidean distance we find that initially $K_1 = \{2, 3\}$ and $K_2 = \{4, 10, 12, 20, 30, 11, 25\}$. The value 3 is equally close to both means, so we arbitrarily choose K_1. Any desired assignment in the case of ties could be used. We then recalculate the means to get $m_1 = 2.5$ and $m_2 = 16$. We again make assignments to clusters to get: $K_1 = \{2, 3, 4\}$ and $K_2 = \{10, 12, 20, 30, 11, 25\}$ Continuing in this fashion we obtain the following:

m_1	m_2	K_1	K_2
3	*18*	$\{2, 3, 4, 10\}$	$\{12, 20, 30, 11, 25\}$
4.75	*19.6*	$\{2, 3, 4, 10, 11, 12\}$	$\{20, 30, 25\}$
7	*25*	$\{2, 3, 4, 10, 11, 12\}$	$\{20, 30, 25\}$

Notice that the clusters in the last two steps are identical. This will yield identical means and thus the means have converged. Our answer is thus: $K_1 = \{2, 3, 4, 10, 11, 12\}$ and $K_2 = \{20, 30, 25\}$.

The time complexity of K-Means is $O(tkn)$ where t is the number of iterations. K-Means finds a local optimum and may actually miss the global optimum. Some research has been performed which examine ways to improve the chances of finding the global optimum. K-Means does not work on categorical data as the mean must be defined on the attribute type. It also does not handle ouliers well.

4.4.4 Nearest Neighbor

An algorithm similar to the Single Link is called the *Nearest Neighbor*. With this serial algorithm, items are iteratively merged into the existing clusters which are closest.

Algorithm 4.7
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*
A *// Adjacency matrix showing distance between elements.*
Output:
K *// Set of clusters.*
Nearest Neighbor Algorithm:
$K_1 = \{t_1\}$;
$K = \{K_1\}$;
$k = 1$;
for $i = 1$ *to* n **do**

find the t_m in some cluster K_m in K such that $dis(t_i, t_m)$ is the smallest;
if $dis(t_i, t_m) \leq t$ **then**
    $K_m = K_m \cup t_i$
else
    $k = k + 1$;
    $K_k = \{t_i\}$;

In this algorithm a threshold, t, is used to determine if items will be added to existing clusters or if a new cluster is created.

4.4.5 PAM

The *PAM (Partitioning Around Medoids)*, also called *K-Medoids* algorithm represents a cluster by a medoid [61]. Using a medoid is an approach which handles outliers well, ordering of input does not impact the results, and they are the basis for handling very large databases [80]

The PAM algorithm is shown in Algorithm 4.8. Initially a random set of k items is taken to be the set of medoids. Then at each step, all items from the input dataset which are not currently medoids are examined one by one to see if they should be. That is, the algorithm finds out if there is an item that should replace one of the existing medoids. By looking at all pairs of medoid, non-medoid objects, the algorithm chooses the pair that improves the overall quality of the clustering and swaps these. Quality here is measured by the sum of all distances from a non-medoid object to the medoid for the cluster it is in. Although not explicitly shown in the algorithm, an item is assigned to the cluster represented by the medoid to which it is closest (minimum distance). In addition, we assume that K_i is the cluster represented by t_i. Suppose t_i is a current medoid and we wish to determine if it should be swapped with a non-medoid t_h. We only wish to do this swap if the overall impact to the quality (sum of the distances to cluster medoids) is an improvement. Following the lead in [80] we use C_{jih} to be the cost change for an item t_j associated with swapping medoid t_i with non-medoid t_h. There are four cases which need to be examined when calculating this cost: 1) $t_j \in K_i$, but $\exists$ another medoid t_m *where* $dis(t_j, t_m) \leq dis(t_j, t_i)$, and $dis(t_j, t_m) \leq dis(t_j, t_h)$; 2) $t_j \in K_i$, but $dis(t_j, t_h) \leq dis(t_j, t_m) \forall$ *other medoids* t_m; 3) $t_j \in K_m$, $\notin K_i$, and $dis(t_j, t_m) \leq dis(t_j, t_h)$; and 4) $t_j \in K_m$, $\notin K_i$, but $dis(t_j, t_h) \leq dis(t_j, t_m)$. We leave it as an exercise to determine what the cost of each of these cases is. The total impact to quality by a medoid change TC_{ih} then is given by:

$$TC_{ih} = \sum_{j=1}^{n} C_{jih} \tag{4.9}$$

Algorithm 4.8
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*
A *// Adjacency matrix showing distance between elements.*
k *// Number of desired clusters.*
Output:

```
K     // Set of clusters.
PAM Algorithm:
  arbitrarily select k medoids from D;
  repeat
    for each t_h not a medoid do
      for each medoid t_i do
        calculate TC_ih;
    find i, h where TC_ih is the smallest;
    if TC_ih < 0 then
      replace medoid t_i with t_h;
  until TC_ih ≥ 0;
  for each t_i ∈ D do
    assign t_i to K_j where dis(t_i, t_j is the smallest over all medoids;
```

PAM does not scale well to large datasets due to the computational complexity. For each iteration we have $k(n-k)$ pairs of objects i, h for which a cost, TC_{ih}, should be determined. Calculating the cost during each iteration requires that the cost be calculated for all other non-medoids t_j. There are $n-k$ of these. Thus the total complexity per iteration is $n(n-k)^2$. The total number of iterations can be quite large. Thus PAM is not an alternative for large databases. However, there are some clustering algorithms based on PAM that we will discuss which are targeted to large datasets.

4.4.6 Bond Energy Algorithm

The *Bond Energy Algorithm (BEA)* was developed and has been used in the database design area to determine how to group data and how to physically place data on disk [106, 82, 101]. It can be used to cluster attributes together based on usage and then perform logical/physical design accordingly. With BEA, the *affinity* (bond) between database attributes is based on common usage. This bond is used by the clustering algorithm to represent a similarity measure. The idea is that attributes that are used together for a cluster and should be stored together. The entries in the similarity matrix are based on the frequency of common usage of attribute pairs. The BEA then converts this similarity matrix into a BOND matrix where the entries represent a type of nearest neighbor bonding based on probability of coaccess. The BEA algorithm rearranges rows/columns so that similar attributes appear close together in the matrix. Finally the designer draws boxes around regions in the matrix with high similarity.

4.4.7 Clustering with Genetic Algorithms

There have been clustering techniques based on the use of genetic algorithms. To determine how to perform clustering with genetic algorithms, we first need to determine how to represent each cluster. One simple approach would be to use a bit map representation for each possible cluster. So, given a database with four items, $\{A, B, C, D\}$, we would represent one solution to creating two clusters as 1001 and 0110. This represents the two clusters $\{A, D\}$ and $\{B, C\}$.

Algorithm 4.9 shows one possible iterative refinement technique for clustering which uses a genetic algorithm. Notice that the approach is similar to that in the Squared Error approach in that an initial random solution is given and successive changes to this converge on a local optimum. A new solution is generated to the previous one using crossover and mutation operations. Our algorithm only shows crossover. The use of crossover to create a new solution from a previous one is shown in Example 4.5. The new "solution" must be created in such a way that it is a valid k clusterings. A fitness function must be used and may be defined based on an inverse of the squared error. Due to the manner in which crossover works, genetic clustering algorithms perform a global rather than local search of potential solutions.

Algorithm 4.9
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*
A *// Adjacency matrix showing distance between elements.*
k *// Number of desired clusters.*
Output:
K *// Set of clusters.*
GA Clustering Algorithm:
randomly create an initial solution;
repeat
use crossover to create a new solution;
until *termination criteria is met;*

Example 4.5 *Suppose a database contains the following eight items* $\{A, B, C, D, E, F, G, H\}$ *which are to be placed into three clusters. We could initially place the items into the three clusters* $\{A, C, E\}$, $\{B, F\}$, *and* $\{D, G, H\}$ *which are represented by* 10101000, 01000100, *and* 00010011 *respectively. Suppose we choose the 1st and 3rd individuals as parents and do a simple crossover at point four. This yields the new solution:* 01001000, 01000100, *and* 10100011.

4.5 Clustering Large Databases

The clustering algorithms presented in the preceding sections are some of the classical clustering techniques. When clustering is used with dynamic databases, these algorithms may not be appropriate. First of all, they all assume that (because they are $O(n^2)$) sufficient main memory exists to hold the data to be clustered and the data structures needed to support them. With large databases containing thousands (or more) of items these assumptions are not realistic. In addition, performing I/Os continuously through the multiple iterations of an algorithm is too expensive. Due to these main memory restrictions, the algorithms do not scale up to large databases. Another issue is that some assume that the data is present all at once. Certainly the simultaneous algorithms assume this. These techniques are not appropriate for dynamic databases. Clustering techniques should be able to adapt as the database changes.

The algorithms discussed in the following subsections each examine some issue associated with performing clustering in a database environment. It has been argued that to perform effectively on large databases, a clustering algorithm should [12]:

1. Require no more (preferrably less) than one scan of the database.
2. Have the ability to provide status and "best" answer so far during the algorithm execution. This may be referred to as the ability to be online.
3. Be suspendable, stoppable, and resumable.
4. Be able to incrementally update the results as data is added/removed from the database.
5. Work with limited main memory.
6. Be capable of performing different techniques for scanning the database. This may include sampling.
7. Process each tuple only once.

Recent research at Microsoft has examined how to efficiently perform the clustering algorithms with large databases [12]. The basic idea of this scaling approach is as follows:

1. Read a subset of the database into main memory.
2. Apply clustering technique to data in memory.
3. Combine results with those from prior samples.
4. This in-memory data is then divided into three different types: those items that will always be needed even when the next sample is brought in, those that can be discarded with appropriate updates to data being kept in order answer to the problem, and those that will be saved in a compressed format. Based on the type, each data item is then kept, deleted, or compressed in memory.
5. If termination criteria is not met then repeat from step 1.

This approach has been applied to K-Means and shown to be effective [12].

4.5.1 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering) is designed for clustering a large amount of metric data [113]. It assumes that there may be a limited amount of main memory and achieves a linear I/O time requiring only one database scan. It is incremental and hierarchical. The basic idea of the algorithm is that a tree is built which captures needed information to perform clustering. The clustering is then performed on the tree itself where labelings of nodes in the tree contain the needed information to calculate distance values. A major characteristic of the BIRCH algorithm is the use of the *Clustering Feature* which is a triple that contains information about a cluster, see Definition 4.3 The clustering feature provides a summary of the information about one cluster. Notice by this definition it is clear that BIRCH only applies to numeric data. This algorithm uses a tree called a *CF tree* as defined in Definition 4.4. The size of the tree is determined by a threshold value, T, associated with each leaf node. This is the maximum diameter allowed for any leaf. Here *diameter* is the average of the pairwise distance between all points in the cluster. Each internal node corresponds to a cluster which is composed of the subclusters represented by its children. Specifics concerning CF tree maintenance can be found in the literature [113].

Definition 4.3 *A* **Clustering Feature (CF)** *is a triple* $(N, \vec{LS}, SS)$*, where the number of the points in the cluster is N,* $\vec{LS}$ *is the sum of the points in the cluster, and SS is the sum of the squares of the points in the cluster.*

Definition 4.4 *A* **CF Tree** *is a balanced tree with a branching factor (maximum number of children a node may have), B. Each internal node contains a CF triple for each of its children. Each leaf node also represents a cluster and contains a CF entry for each subcluster in it. A subcluster in a leaf node must have a diameter no greater than a given threshold valued, T.*

Unlike a dendrogram, a CF tree is searched in a top down fashion. Each node in the CF tree contains clustering feature information about its subclusters. As points are added to the clustering problem, the CF tree is built. Points are inserted into the cluster (represented by a leaf node) to which it is closest. If the diameter for the leaf node is greater than T, then a splitting and balancing of the tree is performed (similar to that used in a B-tree). The algorithm adapts to main memory size by changing the threshold value. A larger threshold, T, yields a smaller CF tree. This process can be performed without rereading the data. The clustering feature data provides enough information to perform this condensation. The complexity of the algorithm is $O(n)$.

Algorithm 4.10
Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*
A *// Adjacency matrix showing distance between elements.*
k *// Number of desired clusters.*
T *// Threshold for CF tree construction.*
Output:
K *// Set of clusters.*
BIRCH Clustering Algorithm:

```
for each t_i ∈ D do
    determine correct leaf node for t_i insertion;
    if threshold condition is not violated then
        add t_i to cluster and update CF triples;
    else
        if room to insert t_i then
            insert t_i as single cluster and update CF triples;
        else
            split leaf node and redistribute tuples;
        reduce T;
        construct new CF tree from old one;
    insert t_i in CF tree;
```

Algorithm 4.10 outlines the steps performed in BIRCH. The first step creates the CF tree in memory. The threshold value can be modified if necessary to ensure that the tree fits into the available memory space. Insertion into the CF tree requires scanning the tree from the root down choosing the node closest to the new point at each level. The distance here is calculated by looking

at the distance between the new point and the centroid of the cluster. Note that this can be easily calculated with most distance measures (Euclidean or Manhattan for example) using the CF triple. When the new item is inserted, the CF triple is appropriately updated as is each triple on the path from the root down to the leaf. It is then added to the closest leaf node found by adjusting the CF value for that node. When an item is inserted into a cluster at the leaf node of the tree, the cluster must satisfy the threshold value. If it does, then the CF entry for that cluster is modified. If it does not, then that item is added to that as a single item cluster.

Node splits occur if no space exists in a given node. This is based on the size of the physical page as each node size is determined by the page size. The nice feature about the CF values is that they are additive. That is, if two clusters are merged, the resulting CF is the addition of the CF values for the starting clusters. Once the tree is built, then the leaf nodes of the CF tree represent the current clusters The second step applies a clustering scheme to the CF tree. Although the original work proposes a a centroid based agglomerative hierarchical clustering algorihtm to cluster the subclusters, other clustering algorithms could be used. [113].

BIRCH is linear in both space and time, as well as being insensitive to order of inputing data items [113]. The authors claim that the proposed CF tree could be used with other algorithms, such as CLARANS. The choice of threshold value is imperative to an efficient execution of the algorithm. Otherwise, the tree may have to be rebuilt many times to ensure it can be memory resident.

4.5.2 CLARA and CLARANS

CLARA (Clustering LARge Applications) improves on the time complexity of PAM by using samples of the dataset [61]. The basic idea is that it applies PAM to a sample of the underlying database and then uses the medoids found as the medoids for the complete clustering. Each item from the complete database is then assigned to the cluster with the medoid to which it is closest. To improve the CLARA accuracy, several samples can be drawn with PAM applied to each. The one resulting is the best clustering, as found by looking at the entire dataset. Five samples of size $40 + 2k$ seem to give good results [61]. Because of the sampling, CLARA is better than PAM for large databases.

CLARANS (Clustering Large Applications Based upon Randomized Search) improves on CLARA by using multiple different samples. In addition to the normal input to PAM, CLARANS requires two additional parameters: maxneighbor and numlocal. Maxneighbor is the number of neighbors of a node to which any specific node can be compared. As maxneighbor increases CLARANS looks more and more like PAM as all nodes will be examined. Numlocal indicates the number of samples to be taken. Since a new clustering is performed on each sample, this also indicates the number of clusterings to be made. Performance studies indicate that numlocal $= 2$ *and maxneighbor* $= max((0.0125 \times k(n-k)), 250)$ *are good choices [80]. CLARANS is shown to be more efficient than either PAM or CLARA for any size dataset.*

CLARANS assumes that all data is in main memory. This certainly is not a valid assumption for large databases.

4.5.3 DBSCAN

The approach used by DBSCAN (Density Based Spatial Clustering of Applications wiht Noise) is to create clusters with a minimum size and density [25]. Here density is defined to be a minimum

number of points within a certain density of each other. This handles the outlier problem by ensuring that an outlier (or a small set of outliers) will not create a cluster. One input parameter, $MinPts$, indicates the minimum number of points in any cluster. In addition, for each point in a cluster there must be another point in the cluster whose distance from it is less than a threshold input value, Eps. The desired number of clusters, k, is not input but rather determined by the algorithm itself.

DBSCAN uses a new concept of density. We first need to look at some definitions from [25]. Definiiton 4.5 defines directly density-reachable. The first part of the definition ensures that one point is "close enough" to the second. The second portion of the definition ensures that thare are enough core points close enough to each other. These core points form the main portion of a cluster in that they are all close to each other. A directly density-reachable point must be close to one of these core points, but need not be a core point itelf. In that case it is called a border point. A point is said to be density-reachable from another point if there is a path form one to the other which contains only directly density-reachable points. This guarantees that any cluster will have a core set of points very close to a large number of other points (core points) and then some other points (border points) which are sufficiently close to at least one core point.

Definition 4.5 *Given values Eps and $MinPts$, a point p is* **directly density-reachable** *from q if:*

- $dis(p,q) \leq Eps$ *and*
- $\mid \{r \mid dis(r,q) \leq Eps\} \mid \geq MinPts$.

Algorithm 4.11 outlines the DBSCAN algorithm. Due to the restrictions on what constitutes a cluster, when the algorithm finishes, there will be points not assigned to a cluster. These are defined as noise.

Algorithm 4.11
Input:
$D = \{t_1, t_2, ..., t_n\}$ *//Set of elements.*
A *// Adjacency matrix showing distance between elements.*
$MinPts$ *// Number of points in cluster.*
Eps *// Maximum distance for density measure.*
Output:
$K = \{K_1, K_2, ..., K_k\}$ *//Set of clusters.*
DBSCAN Algorithm:
$k = 0$; *// Initially there are no clusters.*
for $i = 1$ *to* n **do**
  if t_i *is not in a cluster* **then**
    $X = \{t_j \mid t_j$ *is density-reachable from* $t_i\}$;
    $k = k + 1$;
    $K_k = X$;

The authors propose that an R-tree be used to store the points to be clustered. In this case the average time complexity of DBSCAN is $O(n \lg n)$ [25]. It is possible that a border point could belong to two clusters, the stated algorithm will place this point in which ever cluster is first generated. DBSCAN was compared to CLARANS and found to be more efficient by a factor of 250 to 1900 [25]. In addition, if successfully found all clusters and noise from the test dataset, where CLARANS did not.*

4.5.4 CURE

One objective for the CURE (Clustering Using REpresentatives) clustering algorithm is to handle outliers well [41]. It has both a hierarchical and a partitioning component. First, a constant number of points are chosen from each cluster. These well scattered points are then shrunk towards the cluster's centroid by applying a shrinkage factor, α. When α is 1, all points are shrunk to just one - the centroid. These points then represent the cluster better than a single point (such as medoid or centroid) could. With multiple representative points, clusters of unusual shapes (not just a sphere) can be better represented. CURE then uses a hierarchical clustering algorithm. At each step in the agglomerative algorithm, clusters with the closest pair of representative points are chosen to be merged. The distance between them is defined to be the minimum distance between any pair of points in the representative sets from the two clusters.

CURE handles limited main memory by obtaining a random sample to find the initial clusters. The random sample is partitioned and each partition is then partially clustered. These resulting clusters are then completely clustered in a second pass. The sampling and partitioning is done solely to ensure that the data (regardless of database size) can fit into available main memory. When the clustering of the sample is complete, the labeling of data on disk is performed. A data item is assigned to the cluster with the closest representative point.

The time complexity of CURE is $O(n^2 lg\ n)$ while space is $O(n)$. A heap and k-d tree data structure are used to ensure this performance. One entry in the heap exists for each cluster. Each cluster has not only its representative points, but also the cluster that is closest to it. Entries in the heap are stored in increasing order of the distances between clusters. We assume that each entry u in the heap contains the set of representative points, u.rep, and the cluster closest to it, u.close. We use the heap operations: heapify to create the heap, min to extract the minimum entry in the heap, insert to add a new entry, and delete to delete an entry. A merge procedure is used to merge two clusters together. It determines the new representative points for the new cluster. The basic idea of this process is to first find the point that is farthest from the mean. Subsequent points are then chosen based on being the farthese from those that were previously chosen. A predefined number of points is picked. A k-d tree is a balanced binary tree can be thought of as a generalization of a binary search tree. It is used to index data of k dimensions where the i^{th} level of the tree indexes the i^{th} dimension. In CURE, a k-d tree is used to assist in the merging of clusters. Operations peformed on the tree are: delete to delete an entry form the tree, insert to insert an entry into it, and build to initially create it. The hierarchical clustering algorithm itself, from [41], is shown in Algorithm 4.12. We do not include here either the sampling or the merging algorithms.

Algorithm 4.12
Input:
$D = \{t_1, t_2, ..., t_n\}$ *//Set of elements.*
A *// Adjacency matrix showing distance between elements.*

k // *Desired number of clusters.*
Output:
Q //*Heap containing one entry for each cluster.*
CURE Algorithm:
$T = build(D);$
$Q = heapify(D)$;/ *Initially build heap with one entry per item;*
repeat
$u = min(Q);$
$delete(Q, u.close);$
$w = merge(u, v);$
$delete(T, u);$
$delete(T, v);$
$insert(T, w);$
for each $x \in Q$ **do**
$x.close$ = *find closest cluster to* x;
if x *is closest to* w **then**
$w.close = x;$
$insert(Q, w);$
until *number of nodes in* Q *is* k;

Performance experiments compared CURE to BIRCH and the MST approach [41]. The quality of the clusters found by CURE is better. While the value of the shrinking factor α *does impact results, with a value between* 0.2 *and* 0.7 *the correct clusters are still found. When the number of representative points per cluster is greater than 5, the correct clusters are still always found. A random sample size of aobut* 2.5% *and the number of partitions is freater than 1 or 2 times k seem to work well. The results with large data sets indicate that CURE scales well and outperformed BIRCH.*

4.6 Clusterig with Categorical Attributes

Traditional algorithms do not always work with categorical data. Example 4.6 illustrates some problems which exist when clustering categorical data. This example uses a hierarchical based centroid algorithm to illustrate the problems. The problem illustrated here is that the centroid tends to weaken the relationship between the associated cluster and others. The problems gets worse as more and more clusters are merged. The number of attributes appearing in the mean increases while the individual values actually decreases. This makes the centroid representations become very similar and makes distinguishing between clusters difficult.

Example 4.6 *Consider an information retrieval system where documents can contain keyworks* { *book, water, sun, sand, swim, read* }. *Suppose there are four documents where the first contains the words* { *book* }, *the second contains* { *water, sun, sand, swim*}, *the third contains* { *water, sun, swim, read* }, *and the fourth* { *read, sun*}. *We can represent the four books using the following boolean points:* $(1,0,0,0,0,0), (0,1,1,1,1,0), (0,1,1,0,1,1), (0,0,0,1,0,1)$. *We can use the Euclidean distance to develop the following adjacency matrix of distances:*

	1	2	3	4
1	0	2.24	2.24	1.73
2	2.24	0	1.41	2
3	2.24	1.41	0	1.73
4	1.73	2	1.73	0

The distance between points 2 and 3 is the smallest (1.41) and thus they are merged. When they are merged we get a cluster containing $\{(0,1,1,1,1,0),(0,1,1,0,1,1)\}$ *with a centroid of* $(0,1,1,0.5,1,0.5)$. *Notice that at this point we have a distance from this new cluster centroid to the original points 1 and 4 being 2.24 and 2 respectively, while the distance between original points 1 and 4 is 1.73. Thus we next merge these points even though they have no keywords in common. So with k=2 we have the following clusters:* $\{\{1,4\},\{2,3\}\}$.

Instead of using a Euclidean distance, a different distance, such as jaccard's coefficient has been proposed [59]. In the following subsections we briefly review some recent algorithms specifically targeted at categorical data.

4.6.1 CACTUS

A recent work has looked at extending the definition of clustering to be more applicable to categorical data. The definition of similarity between tuples is given by looking at the support of two attribute values within the database D. Given two categorical attribute A_i, A_j *with domains* D_i, D_j *respectively, the support of the attribute pair* (a_i, a_j) *is defined as follows:*

$$\sigma_D(a_i, a_j) = \mid \{t \in D : t.A_i = a_i \wedge t.A_j = a_j\} \mid \tag{4.10}$$

The similarity of attributes used for clustering is based on the support [35].

4.6.2 ROCK

The ROCK (RObust Clustering using linKs) clustering algorithms is targeted to both boolean and categorical data [42]. A novel approach to identifying similarity is based on the number of links between items. A pair of items is said to be neighbors if their similarity exceeds some threshold. This need not be defined based on a precise metric, but rather could be a more intuitive approach using domain experts. The number of links between two items is defined to be the number of common neighbors that they have. The objective of the clustering algorithm is to group together points that have more links. The algorithm is a hierarchical agglomerative algorithm using the number of links as the similarity measure rather than one based on distance.

On proposed similarity measure based on the Jaccard coefficient is defined as:

$$sim(t_i, t_j) = \frac{\mid t_i \cap t_j \mid}{\mid t_i \cup t_j \mid} \tag{4.11}$$

If the tuples are viewed to be sets of items purchased (ie. basket market data), then we look at the number of items they have in common divided by the total number in both. The denominator is used to normalize the value to be between 0 *and* 1.

The number of links between a pair of points can be viewed as the number of unique paths of length 2 betweeen them. The authors argue that the use of links rather than similarity (distance)

measures provides a more global approach as the similarity between points is impacted by other points as well [42]. value to be between 0 *and* 1. *Example 4.7 illustrates the use of links by the ROCK algorithm using the data from Example 4.6 using the Jaccard coefficient. Notice that difference threshold values for neighbors could be used to get different results. Also notice, that a hierarchical approach could be used with different threshold values for each level in the dendrogram.*

Example 4.7 *Using the data from Example 4.6 we have the following table of similarities (as opposed to the distances given in the example):*

	1	*2*	*3*	*4*
1	*1*	*0*	*0*	*0*
2	*0*	*1*	*0.6*	*0.2*
3	*0*	*0.6*	*1*	*0.2*
4	*0*	*0.2*	*0.2*	*1*

Suppose we say that the threshold for a neighbor is 0.2*, then we have the following are the neighbors:* $\{(2,3),(2,4),(3,4)\}$. *Note that in the following we add to the that a point is a neighbor of itself so that we have the additional neighbors:* $\{(1,1),(2,2),(3,3),(4,4)\}$. *The following table shows the number of links (common neighbors between points) assuming the threshold for a neighbor is* 0.2*:*

	1	*2*	*3*	*4*
1	*1*	*0*	*0*	*0*
2	*0*	*3*	*3*	*3*
3	*0*	*3*	*3*	*3*
4	*0*	*3*	*3*	*3*

In this case then, we have the following clusters:$\{\{1\},\{2,3,4\}\}$. *comparing this to the set of clustering found with a traditional Euclidean distance, we see that a "better" set of clusters has been created.*

The ROCK algorithm is divided into three general parts: obtain a random sample, perform a hierarchical agglomerative clustering algorithm, and label the data on disk. Sampling is used to ensure scalability to very large datasets as (with most clustering algorithms) the data is assumed to be memory resident during processing. The sample is used to determine the clusters, then using these clusters the remaining data on disk is assigned to them. A goodness measure is used to determine which pair of points is merged at each step.

The first step in the algorithm converts the adjacency matrix into a boolean one where an entry is 1 if the two corresponding points are neighbors. As the adjacency matrix is of size n^2*, this is an* $O(n^2)$ *step. The next step converts this into a matrix indicating the links. This can be found by calculating* $S \times S$ *which can be done in* $O(n^{2.37})$ *[42]. The hierarchical clustering portion of the algorithm then starts by placing each point in the sample in a separate cluster. It then successively merges clusters until k clusters are found. To facilitate this processing, both local and global heaps are used. A local heap, q, is created to represent each cluster. Here q contains every cluster which has a nonzero link to the cluster which corresponds to this one. Initially a cluster is created for each point,* t_i*. The heap for* t_i*,* $q[t_i]$*, contains eavery cluster which has a nonzero link to* $\{t_i\}$ *The global heap contains information about each cluster. All information in the heap is ordered based on a goodness measure which is described in a subsequent paragraph.*

4.7 Exercises

1. *Show the dendrogram created by the Single, Complete, and Average Link clustering algorithms using the following adjacency matrix:*

Item	*A*	*B*	*C*	*D*
A	*0*	*1*	*4*	*5*
B	*1*	*0*	*2*	*6*
C	*4*	*2*	*0*	*3*
D	*5*	*6*	*3*	*0*

2. *Convert Algorithm 4.1 into a generic divisive algorithm. What technique would be used to split clusters in the single link and complete link versions?*

3. *Use the K-Means algorithm to cluster the data in Example 4.4 into 3 clusters.*

4. *Determine the cost C_{jih} for each of the four cases given for the PAM Algorithm.*

4.8 Bibliographic Notes

There have been many excellent books examining the concept of clustering. In [59] a thorough treatment of clustering algorithms including application domains and statement of algorithms is provided. Earlier books include [105, 60, 97, 46]

For a thorough survey of clustering with a complete list of references look at [58].

The agglomerative clustering methods are among the oldest. Articles on single link clustering date back to 1951 [32].

There have been many variations of the K-Means clustering algorithm. The earliest reference is to a version by Forgy in 1965 [33], [76]. Another approach for partitional clustering is to allow splitting and merging of clusters. Here merging is performed based on the distance between the centroids of two clusters. A cluster is split if its variance is above a threshold. One proposed algorithm performing this is called ISODATA [8].

Although not discussed in this chapter, there have been some approaches to clustering based on the use of neural networks and genetic algorithms.

Chapter 5

ASSOCIATION RULES

August 23, 2000

5.1 Introduction

Association rules are used to show the relationships between data items. These uncovered relationships are not inherent in the data itself, as with functional dependencies, nor do they represent any sort of causality. Instead, association rules detect common usage of items. Example 5.1 illustrates.

Example 5.1 *A grocery store chain keeps a record of weekly transactions where each transaction represents the items bought during one cash register transaction. The executives of the chain get a summarized report of the transactions indicating what types of items have sold at what quantity. In addition, they periodically request information about what items are commonly purchased together. They find that* 100% *of the time that peanut butter is purchased, so is bread. In addition,* 33.3% *of the time peanut butter is purchased, jelly is also purchased. However, peanut butter is only obtained in about* 50% *of the overal transactions.*

The purchasing of one product when another one is purchased represents as association rule. Association rules are frequently used by retail stores to assist in marketin, advertising, floor placement, and inventory control. The development of association rules can be traced to one paper in 1993 [5]. Although they have direct applicability to retail businesses they have also been used for other purposes as well, including predicting faults in telecommunications networks.

A database in which an association rule is to be found is viewed as a set of tuples where each tuple contains a set of items. For example a tuple could be $\{peanutbutter, bread, jelly\}$ *which consists of the three items: peanut butter, bread, jelly. Keeping grocery story cash register transactions in mind, each item represents an item purchased while each tuple is the list of items purchased at one time. In the simplest cases we are not interested in quantity or cost, so these may be removed from the records prior to processing. Table 5.1 will be used throughout this chapter to illustrate different algorithms. Here there are five transactions and five items:* $\{Beer, Bread, Jelly, Milk, PeanutButter\}$. *Throughout this chapter we will list items in alphabetical order within a transaction. Although this isn't required, algorithms often assume this sorting is done in a preprocessing step.*

The support of an item (or set of items) is the percentage of transactions in which that item (or items) occurs. Table 5.2 shows the support for all subsets of items from our total set. As seen,

Transaction	Items
t_1	Bread,Jelly,PeanutButter
t_2	Bread,PeanutButter
t_3	Bread,Milk,PeanutButter
t_4	Beer,Bread
t_5	Beer,Milk

Table 5.1: Sample Data to Illustrate Association Rules

Set	Support
Beer	40
Bread	80
Jelly	20
Milk	40
PeanutButter	60
Beer,Bread	20
Beer,Jelly	0
Beer,Milk	20
Beer,PeanutButter	0
Bread,Jelly	20
Bread,Milk	20
Bread,PeanutButter	60
Jelly,Milk	0
Jelly,PeanutButter	20
Milk,PeanutButter	20
Beer,Bread,Jelly	0

Set	Support
Beer,Bread,Milk	0
Beer,Bread,PeanutButter	0
Beer,Jelly,Milk	0
Beer,Jelly,PeanutButter	0
Beer,Milk,PeanutButter	0
Bread,Jelly,Milk	0
Bread,Jelly,PeanutButter	20
Bread,Milk,PeanutButter	20
Jelly,Milk,PeanutButter	0
Beer,Bread,Jelly,Milk	0
Beer,Bread,Jelly,PeanutButter	0
Beer,Bread,Milk,PeanutButter	0
Beer,Jelly,Milk,PeanutButter	0
Bread,Jelly,Milk,PeanutButter	0
Beer,Bread,Jelly,Milk,PeanutButter	0

Table 5.2: Support of all Sets of Items Found in Table 5.1

there is an exponential growth in the sets of items. In this case we could have 31 sets of items from the original set of five items (ignoring the empty set). This explosive growth in potential sets of items is an issue that most association rule algorithms must contend with as the conventional approach to generating association rules is in actually counting the occurence of sets of items in the transaction database.

Notice that we are dealing with categorical data. Given a target domain, the underlying set of items is usually known so that an encoding of the transactions could be performed prior to processing. As we will see, however, association rules can be applied to data domains other than categorical.

Definition 5.1 *Given a set of items* $I = \{I_1, I_2, ..., I_m\}$ *and a database of transactions* $D = \{t_1, t_2, ..., t_n\}$ *where* $t_i = \{I_{i1}, I_{i2}, ...I_{ik}\}$ *and* $I_{ij} \in I$, *an* **Association Rule** *is an implication of the form* $X \Rightarrow Y$ *where* $X, Y \subset I$ *are sets of items called itemsets and* $X \cap Y = \emptyset$.

Definition 5.2 *The* **support (s)** *for an association rule* $X \Rightarrow Y$ *is the percentage of transactions*

$X \Rightarrow Y$	s	α
Bread $\Rightarrow$ PeanutButter	60%	75%
PeanutButter $\Rightarrow$ Bread	60%	100%
Beer $\Rightarrow$ Bread	20%	50%
PeanutButter $\Rightarrow$ Jelly	20%	33.3%
Jelly $\Rightarrow$ PeanutButter	20%	100%
Jelly $\Rightarrow$ Milk	0%	0%

Table 5.3: Support and Confidence for Some Association Rules

in the database that contain $X \cup Y$.

Definition 5.3 *The* **confidence (α)** *for an association rule $X \Rightarrow Y$ is the ratio of the number of transactions that contain $X \cup Y$ to the number that contain X.*

A formal definition, from [5], is found in Definition 5.1. We are generally not interested in all implications but only those that are important. Here importance is measured by two features called support and confidence as defined in Definitions 5.2 and 5.3. Table 5.3 shows the support and confidence for several association rules including those from Example 5.1.

The selection of association rules is based on these two values as described in the definition of the Association Rule Problem in Definition 5.4. Confidence measures the strength of the rule where as support measures how often it should occur in the database. Typically large confidence values and a smaller support are used. For example, look at Bread $\Rightarrow$ PeanutButter in Table 5.3. With $\alpha = 75\%$ this indicates that this rule holds 75% of the time that it could. That is, 3/4 times that Bread occurs, so does PeanutButter. This is a stronger rule than Jelly $\Rightarrow$ Milk since there are no times Milk is purchased when Jelly is. An advertising executive would probably not want to base an advertising campaign on the fact that when a person buys Jelly he also buys bread. Lower values for support may be allowed as support indicates the percent of time the rule occurs throughout the database. For example, with Jelly $\Rightarrow$ PeanutButter the confidence is 100% but the support is only 20%. It may be the case that this association rule exists in only 20% of the transactions, however when the antecepedent, Jelly, occurs the consequent always occurs. Here an advertising strategy targeted to people who purchase Jelly would be appropriate.

The discussion so far has centered around the use of association rules in the Market Basket area. Example 5.2 illustrates a use for association rules in another domain: telecommunications. This example, although quite simplified from the similar real world problem, illustrates the importance of association rules in other domains and the fact that support need not always be high.

Example 5.2 *A telephone company must ensure that a high percentage of all phone calls is made within a certain period of time. Since each phone call must be routed through many switches, it is imperative that each switch works correctly. The failure of any switch could result in a call not being completed or being completed in an unacceptably long period of time. In this environment, a potential data mining problem would be to predict a failure of a node. Then when the node is predicted to fail, measures can be taken by the phone company to route all calls around the node and replace the switch. To this end, the company keeps a history of calls through a switch. Each*

call history indicates the success or failure of the switch, associated timing, and error indication. The history actually contains results of the last and prior traffic through the switch. A transaction of the type $\langle Success, Failure \rangle$ indicates that the most recent call could not be handled successfully while the one before that was handled fine. Another transaction $\langle ERR1, Failure \rangle$ indicates that the previous call was handled but an error occured, ERR1. This error could be something like - excessive time. The data mining problem can then be stated as finding association rules of the type $X \Rightarrow Failure$. If these types of rules occur with a high confidence we could predict failure and immediately take the node offline. Even though the support might be low because the X condition does not frequently occur, most often when it occurs the node fails with the next traffic.

Definition 5.4 *Given a set of items $I = \{I_1, I_2, ..., I_m\}$ and a database of transactions $D = \{t_1, t_2, ..., t_n\}$ where $t_i = \{I_{i1}, I_{i2}, ...I_{ik}\}$ and $I_{ij} \in I$, the* **Association Rule Problem** *is to identify all association rules $X \Rightarrow Y$ with a minimum support and confidence. These values (s, α) are given as input to the problem.*

The efficiency of association rule algorithms is usually discussed with respect to the number of scans of the database that are required and the maximum number of itemsets which have to be counted.

A classification of association rule algorithms has recently been proposed [24]. As stated in [24] there are twelve dimensions along which the classification is made:

1. *Target: Indicates whether additional constraints beyond support and confidence are used to generate rules.*

2. *Type: There are generalizations of association rules, such as spatial, temporal, qualitative, and generalized.*

3. *Data Type: Conventional algorithms, as discussed in this chapter, assume a basket market type of data. However association rules could be found by looking at more complicated types of data such as text and multimedia.*

4. *Data Source: An indication of where the data is obtained from. The data could actually be obtained from a non-electronic source.*

5. *Technique: Most approaches to date find association rules by first generating large itemsets. However, this may not be the case in future algorithms. A recent technique based on the concept of strongly collective itemsets has been proposed [2].*

6. *Itemset Strategy: This categorizes when itemsets are generated and counted. One approach would be to generate all itemsets, complete, prior to counting. Another approach, Apriori, generates candidates level by level from large itemsets of the previous scan. There are combinations of these approaches, hybrid.*

7. *Transaction Strategy: This describes how the transactions are read. All could be examined or some sampling or partitioning technique used.*

8. *Itemset Data Structure: Different data structures have been used to track and count the itemsets. These include trie, hash tree, or lattice.*

Term	Description
D	Database of transactions
t_i	Transaction in D
s	Support
α	Confidence
X, Y	itemsets
$X \Rightarrow Y$	Association Rule
L	Set of large itemsets
l	Large itemset in L
C	Set of candidate itemsets
p	Number of partitions

Table 5.4: Association Rule Notation

9. *Transaction Data Structure: Transactions may be viewed as in a flat file or alternatively a TID list.*

10. *Optimization: These techniques look at how to improve on the performance of an algorithm given data distribution (skewness) or amount of main memory.*

11. *Architecture: Sequential, parallel, and distributed algorithms have been proposed.*

12. *Parallelism Strategy: Parallel strategies may use a data or task parallelism approach.*

5.2 Large Itemsets

The most common approach to finding association rules is to break up the problem into two parts [7]:

1. *Find large itemsets as defined in Definition 5.5.*
2. *Generate rules from frequent itemsets.*

An itemset is any subset of the set of all items, I.

Definition 5.5 *A* **Large (Frequent) Itemset** *is an itemset whose number of occurences is above a threshold, s. We use the notation L to indicate the complete set of large itemsets and l to indicate a specific large itemset.*

Once the large itemsets have been found we know that any interesting association rule, $X \Rightarrow Y$, must have $X \cup Y$ in this set of frequent itemsets. Notice that the subset of any large itemset is also large. Due to the large number of notations used in association rule algorithms, we summarize them in Table 5.4. When a specific term has a subscript, this indicates the size of the set being considered. For example, l_k is a large itemset of size k. Some algorithms divide the set transactions into partitions. In this case we use p to indicate the number of partitions and a superscript to indicate which partition. For example, D^i is the i^{th} partition of D.

Finding large itemsets is generally quite easy, but very costly. The naive approach would be to count all itemsets which appear in any transaction. Given a set of items of size m, there are 2^m *subsets. Since we are not interested in the empty set the potential number of large itemsets is then* $2^m - 1$. *Do to the explosive growth of this number, the challenge of solving the association rule problem is often viewed as how to efficiently determine all large itemsets. (When m=5, there are potentially 31 itemsets. When m=30 this becomes 1073741823.) Most association rule algorithms are based on smart ways to reduce the number of itemsets to be counted. These potentially large itemsets are called Candidates, and the set of all counted (potentially large itemsets) is the Candidate Itemset (c). One performance measure used for association rule algorithms is the size of* C. *Another problem to be solved by association rule algorithms is what data structure is to be used during the counting process. As we will see there have been several proposed. A trie or hash tree are common.*

When all large itemsets are found, generating the association rules is straightforward. Algorithm 5.1 modified from [7] outlines this technique. In this algorithm we use a function support which returns the support for the input itemset.

Algorithm 5.1

Input:

D *//Database of transactions*
I *//Items*
L *//Large itemsets*
s *//Support*
α *//Confidence*

Output:

R *//Association Rules satisfying s and* α

ARGen Algorithm:

$R = \emptyset$;
for each $l \in L$ **do**
  for each $x \subset l$ *such that* $x \neq \emptyset$ *and* $x \neq l$ **do**
    if $\frac{support(l)}{support(x)} \geq \alpha$ **then**
      $R = R \cup \{x \Rightarrow (l - x)\}$;

To illustrate this algorithm, again refer to the data in Table 5.1 with associated supports shown in Table 5.2. Suppose that the input support and confidence are $s = 30\%$ *and* $\alpha = 50\%$ *respectively. Using this value of s, we obtain the following set of large itemsets:*

$$L = \{\{Beer\}, \{Bread\}, \{Milk\}, \{PeanutButter\}\{Bread, PeanutButter\}\}.$$

We'll now look at what association rules are generated from the last large itemset. Here $l = \{Bread, PeanutButter\}$. *There are two nonempty subsets of* l: { *Bread* } *and* { *PeanutButter*}. *With the first one we see:*

$$\frac{support(\{Bread, PeanutButter\})}{support(\{Bread\})} = \frac{60}{80} = 0.75.$$

This means that the confidence of the association rule $Bread \Rightarrow PeanutButter$ *is 75% just as is seen in Table 5.3. Since this is about* α *this is a valid association rule and is added to* R. *Likewise with the second large itemset:*

Pass	Candidates	Large Itemsets
1	{Beer},{Bread},{Jelly}, {Milk},{PeanutButter}	{Beer},{Bread}, {Milk},{PeanutButter}
2	{Beer,Bread},{Beer,Milk}, {Beer,PeanutButter},{Bread,Milk}, {Bread,PeanutButter},{Milk,PeanutButter}	{Bread,PeanutButter}

Table 5.5: Using Apriori with Transactions in Table 5.1

$$\frac{support(\{Bread, PeanutButter\})}{support(\{PeanutButter\})} = \frac{60}{60} = 1.$$

This means that the confidence of the association rule PeanutButter ⇒ Bread is 100% and this is a valid association rule.

All of the algorithms discussed in subsequent sections look primarily at ways to efficiently discover large itemsets.

5.3 Basic Algorithms

5.3.1 Apriori

The Apriori algorithm is the most well know association rule algorithm and is used in most commercial products [7]. It uses the following property we call The Large Itemset Property:

Any subset of a large itemset must be large.

The large itemsets are also said to be downward closed since if an itemset satisfies the minimum support requirements, so do all of its subsets. Looking at the contrapositive of this, if we know that an itemset is small we need not generate any supersets of it as candidates as they must also be small. We use the lattice shown in Figure 5.1 a) to illustrate the concept of this important property. In this case there are four items $\{A, B, C, D\}$. The lines in the lattice represent the subset relationship. So the large itemset property says that any set in a path above an itemset must be large if the original one is. In Figure 5.1 b) the nonempty subsets of ACD [1] are seen as $\{AC, AD, CD, A, C, D\}$. If ACD is large so is each of these. If any one of these is small then so is ACD.

The basic idea of Apriori is to generate candidate itemsets of a particular size and then scan the database to count these to see if they are large. During scan i, candidates of size i, C_i are counted. Only those which are large are used to generate candidates for the next pass. That is L_i are used to generate C_{i+1}. An itemset is considered as a candidate only if all its subsets are also large. To generate candidates of size $i+1$ joins are made of large itemsets found in the previous pass. Table 5.5 shows the process using the data found in Table 5.1 with $s = 30\%$ and $\alpha = 50\%$. There are no candidates of size three as there is only one large itemset of size two.

An algorithm called Apriori-Gen is used to generate the candidate itemsets for each pass after the first. All singleton itemsets are used as candidates in the first pass. Here the set of large itemsets

[1] Following the usual convention with association rule discussions, we will simply list the items in the set rather than using the traditional set notation. So here we use ACD to mean $\{A, C, D\}$.

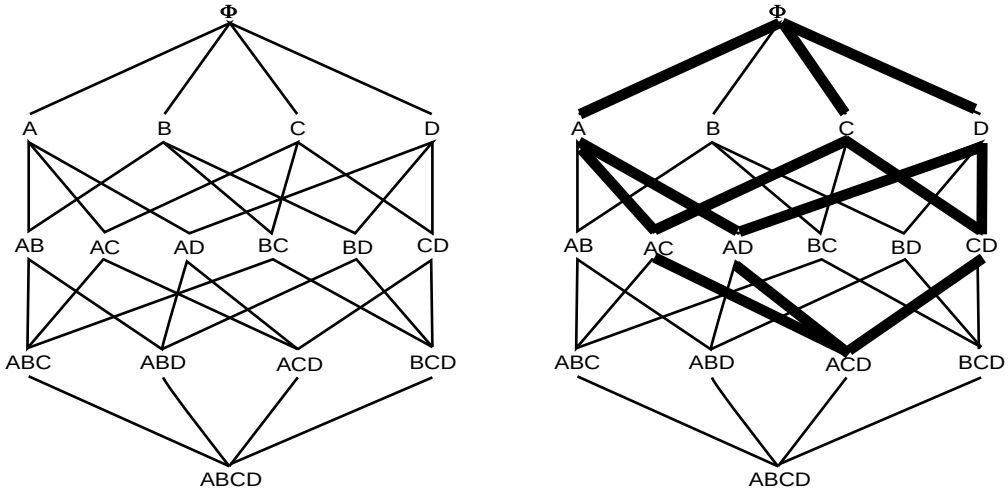


Figure 5.1: Downward Closure

of the previous pass, L_{i-1} is joined with itself to determine the candidates. Individual itemsets must have all but one item in common to be combined. Example 5.3 further illustrates the concept. After the first scan, every large itemset is combined with every other large itemset

Example 5.3 *A small lady's clothing store has ten cash register transactions during one day, shown in Table 5.6. When Apriori is applied to this data, during scan one, we have six candidate itemsets as seen in Table 5.7. Of these, five are large. When Apriori-Gen is applied to these five, we combine every one with all the other five. Thus we get a total of $4+3+2+1=10$ candidates during scan two. Of these, 7 are large. When we apply Apriori-Gen at this level, we join any set with another set which has one item in common. Thus {Jeans,Shoes} is joined with {Jeans,Shorts} but not with {Shorts,TShirt}. {Jeans,Shoes} will be joined with any other itemset containing either Jeans or Shoes. When it is joined, the new item is added to it. There are four large itemsets after scan four. When we go to join these we must match on two of the three attributes. For example {Jeans,Shoes,Shorts} After scan four, there is only one large itemset. So we obtain no new itemsets of size five to count in the next pass. joins with {Jeans,Shoes,TShirt} to yield new candidate {Jeans,Shoes, Shorts,TShirt}.*

The Apriori-Gen algorithm is shown in Algorithm 5.2. Apriori-Gen is guaranteed to generate a superset of the large itemsets of size i, $C_i \supset L_i$, when input L_{i-1} [7]. A pruning step, not shown, could be added at the end of this algorithm to prune away any candidates that have subsets of size $i-1$ which are not large.

Algorithm 5.2

Transaction	Items	Transaction	Items
t_1	Blouse	t_{11}	TShirt
t_2	Shoes,Skirt,TShirt	t_{12}	Blouse,Jeans,Shoes,Skirt,TShirt
t_3	Jeans,TShirt	t_{13}	Jeans,Shoes,Shorts,TShirt
t_4	Jeans,Shoes,TShirt	t_{14}	Shoes,Skirt,TShirt
t_5	Jeans,Shorts	t_{15}	Jeans,TShirt
t_6	Shoes,TShirt	t_{16}	Skirt,TShirt
t_7	Jeans,Skirt	t_{17}	Blouse,Jeans,Skirt
t_8	Jeans,Shoes,Shorts,TShirt	t_{18}	Jeans,Shoes,Shorts,TShirt
t_9	Jeans	t_{19}	Jeans
t_{10}	Jeans,Shoes,TShirt	t_{20}	Jeans,Shoes,Shorts,TShirt

Table 5.6: Sample Clothing Transactions

Scan	Candidates	Large Itemsets
1	{Blouse},{Jeans},{Shoes}, {Shorts},{Skirt},{TShirt}	{Jeans},{Shoes},{Shorts} {Skirt},{Tshirt}
2	{Jeans,Shoes},{Jeans,Shorts},{Jeans,Skirt}, {Jeans,TShirt},{Shoes,Shorts},{Shoes,Skirt}, {Shoes,TShirt},{Shorts,Skirt},{Shorts,TShirt}, {Skirt,TShirt}	{Jeans,Shoes},{Jeans,Shorts}, {Jeans,TShirt},{Shoes,Shorts}, {Shoes,TShirt},{Shorts,TShirt}, {Skirt,TShirt}
3	{Jeans,Shoes,Shorts},{Jeans,Shoes,TShirt}, {Jeans,Shorts,TShirt},{Jeans,Skirt,TShirt}, {Shoes,Shorts,TShirt},{Shoes,Skirt,TShirt}, {Shorts,Skirt,TShirt}	{Jeans,Shoes,Shorts}, {Jeans,Shoes,TShirt}, {Jeans,Shorts,TShirt}, {Shoes,Shorts,TShirt}
4	{Jeans,Shoes,Shorts,TShirt}	{Jeans,Shoes,Shorts,TShirt}
5	$\emptyset$	$\emptyset$

Table 5.7: Apriori-Gen Example

Input:
L_{i-1} *//Large itemsets of size i-1.*
Output:
C_i *//Candidates of size i.*
Apriori-Gen Algorithm:
$C_i = \emptyset$;
for each $I \in L_{i-1}$ **do**
for each $J \neq I \in L_{i-1}$ **do**
if $i-1$ *of the elements in* I *and* J *are equal* **then**
$C_k = C_k \cup \{I \cup J\}$;

Given the Large Itemset Property and Apriori-Gen, the Apriori algorithm itself, see Algorithm 5.3 is rather straightforward. In this algorithm we use c_i *to be the count for item* $I_i \in I$.

Algorithm 5.3
Input:
I *//Itemsets.*
D *//Database of Transactions.*
s *//Support.*
Output:
L *//Large Itemsets.*
Apriori Algorithm:
$k = 0$;*//k is used as the scan number.*
$L = \emptyset$;
$C_1 = I$; *//Initial candidates are set to be the items.*
repeat
$k = k + 1$;
$L_k = \emptyset$;
for each $I_i \in C_k$ **do**
$c_i = 0$; *// Initial counts for each itemset are 0.*
for each $t_j \in D$ **do**
for each $I_i \in C_K$ **do**
if $I_i \in t_j$ **then**
$c_i = c_i + 1$;
for each $I_i \in C_k$ **do**
if $c_i \geq (s\times \mid D \mid)$ **do**
$L_k = L_k \cup I_i$;
$L = L \cup L_k$;
$C_{k+1} = Apriori - Gen(L_k)$
until $C_{k+1} = \emptyset$;

The Apriori algorithm assumes that the database is memory resident. The maximum number of database scans is one more than the cardinality of the largest large itemset. This potentially large number of database scans is a weakness of the Apriori approach.

5.3.2 Sampling

To facilitate efficient counting of itemsets with large databases, sampling of the database may be used. The original sampling algorithm reduces the number of database scans to one in the best case and two in the worst [103]. The database sample is drawn such that it can be memory resident. Then any algorithm, such as Apriori, is used to find the large itemsets for the sample. These are viewed as Potentially Large (PL) itemsets and used as candidates to be counted using the entire database. Additional candidates are determined by applying the negative border function, BD^-, against the large itemsets from the sample. The entire set of candidates is then $C = BD^-(PL) \cup PL$. The negative border function is a generalization of the Apriori-Gen algorithm. It is defined to be the minimal set of itemsets which are not in PL, but whose subsets are all in PL. Example 5.4 illustrates the idea.

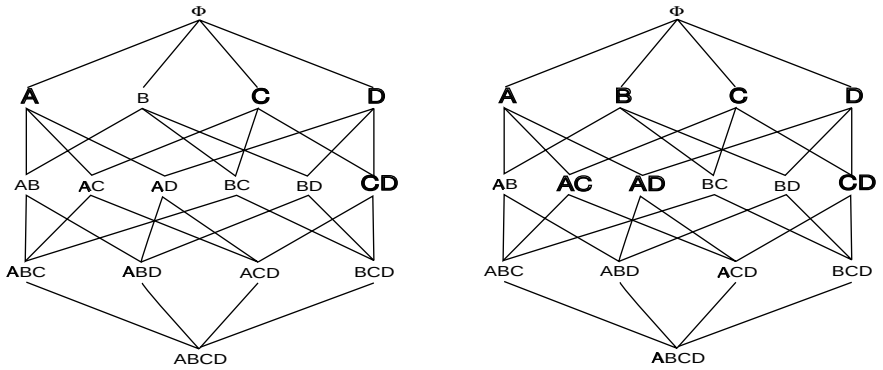


Figure 5.2: Negative Border

Example 5.4 *Suppose the set of items is $\{A, B, C, D\}$. The set of large itemsets found to exist in a sample of the database is $PL = \{A, C, D, CD\}$. The first scan of the entire database, then, generates the set of candidates as follows: $C = BD^-(PL) \cup PL = \{B, AC, AD\} \cup \{A, C, D, CD\}$. Here we add AC as both A and C are in PL. Likewise we add AD. We could not have added ACD as neither AC nor AD are in PL. When looking at the lattice, Figure 5.2, we only add sets where all subsets are already in PL. Notice that we add B as all its subsets are vacuously in PL.*

Algorithm 5.4 shows the Sampling algorithm. Here apriori is shown to find the large itemsets in the sample, but any large itemset algorithm could be used. Any algorithm to obtain a sample of the database could be used as well. During the first scan of the database, all candidates are counted. If all candidates which are large are in PL (none in $BD^-(PL)$) then all large itemsets are found. If however, some large itemsets are in the negative border, then a second scan is needed. During the second scan additional candidates are generated and counted. This is done to ensure that all large itemsets are found. Here ML, the missing large itemsets, are those in L which are not in PL. The algorithm repeatedly applies the negative border until no new itemsets are generated. This set contains all itemsets which could possibly be large. Thus when these are counted during the second scan all large itemsets are found.

Algorithm 5.4
Input:
I *//Itemsets.*
D *//Database of Transactions.*
s *//Support.*
Output:
L *//Large Itemsets.*
Sampling Algorithm:
D_S = *Sample drawn from* D;
$PL = Apriori(I, D_S, s)$;
$C = PL \cup BD^-(PL)$;
$L = \emptyset$;
for each $I_i \in C$ **do**
  $c_i = 0$; *// Initial counts for each itemset are 0.*
for each $t_j \in D$ **do** *// First scan count.*
  for each $I_i \in C$ **do**
    if $I_i \in t_j$ **then**
      $c_i = c_i + 1$;
for each $I_i \in C$ **do**
  if $c_i \geq (s \times \mid D \mid)$ **do**
    $L = L \cup I_i$;
$ML = \{x \mid x \in BD^-(PL) \wedge x \in L\}$; *//Missing large itemsets.*
if $ML \neq \emptyset$ **then**
  $C = L$; *// Set candidates to be the large itemsets.*
  repeat
    $C = C \cup BD^-(C)$; *// Expand candidate sets using negative border.*
  until *no new itemsets are added to* C;
  for each $I_i \in C$ **do**
    $c_i = 0$; *// Initial counts for each itemset are 0.*
  for each $t_j \in D$ **do** *// Second scan count.*
    for each $I_i \in C$ **do**
      if $I_i \in t_j$ **then**
        $c_i = c_i + 1$;
  if $c_i \geq (s \times \mid D \mid)$ **do**
    $L = L \cup I_i$;

5.3.3 Partitioning

Various approaches to generating large itemsets have been proposed based on a partitioning of the set of transactions. In this case D *is divided into* p *partitions* $D^1, D^2, ..., D^p$. *Partitioning may improve the performance of finding large itemsets in several ways;*

- *By taking advantage of the Large Itemset Property, we know that a large itemset must be large in at least one of the partitions. This idea can help to design algorithms more efficiently than those based on looking at the entire database.*

- *Partitioning algorithms may be able to better adapt to limited main memory. Each partition can be created such that it fits into main memory. In addition, it would be expected that the number of itemsets to be counted per partition would be smaller than those needed for the entire database.*
- *By using partitioning, parallel and/or distributed algorithms can be easily created where each partition could be handled by a separate machine.*
- *Incremental generation of association rules may be easier to perform by treating the current state of the database as one partition and the new entries as a second partition.*

The basic Partition algorithm reduces the number of database scans to two and divides the database into partitions such that each can be placed in main memory [93]. When it scans the database, it brings that partition of the database into main memory and counts the items in that partition alone. During the first database scan, the algorithm finds all large itemsets in each partition. Although any algorithm could be used for this purpose, the original proposal assumes that some level-wise approach, such as Apriori, is used. Here L^i represents the large itemsets from partition D^i. During the second scan, only those itemsets which are large in at least one partition are used as candidates and counted to determine if they are large across the entire database. Algorithm 5.5 shows this basic Partition algoritm.

Algorithm 5.5
Input:
I //*Itemsets.*
$D = \{D^1, D^2, ...D^p\}$ //*Database of Transactions Divided into Partitions.*
s //*Support.*
Output:
L //*Large Itemsets.*
Partition Algorithm:
$C = \emptyset$;
for $i = 1$ *to* p **do** //*Find large itemsets in each partition.*
  $L^i = Apriori(I, D^i, s)$;
  $C = C \cup L^i$;
$L = \emptyset$;
for each $I_i \in C$ **do**
  $c_i = 0$; // *Initial counts for each itemset are 0.*
for each $t_j \in D$ **do** // *Count candidates during second scan.*
  for each $I_i \in C$ **do**
    if $I_i \in t_j$ **then**
      $c_i = c_i + 1$;
for each $I_i \in C$ **do**
  if $c_i \geq (s \times \mid D \mid)$ **do**
    $L = L \cup I_i$;

If the items are uniformly distributed across the partitions, then a large fraction of the itemsets will turn out to be large. However, if the data is not, then there may be a large percentage of false

candidates. This problems was addressed in [66] where a set of algorithms were proposed which better prune away false candidates prior to the second scan.

5.4 Parallel and Distributed Algorithms

Most parallel or distributed association rule algorithms strive to parallelize either the data, data parallelism, or the candidates, task parallelism. . With data parallelism, each processor counts the same set of itemsets. With task parallelsim, the candidates are partitioned and counted separately at each processor. Obviously, the Partition algorithm would be easy to parallelize using the task parallelism approach. Other dimensions in differentiating different parallel association rule algorithms are the load-balancing approach used, and the architecture [109]. The data parallelism algorithms have reduced communication cost over the task, as only the initial candidates (the set of items) and the local counts at each iteration have to be distributed. With task parallism not only the candidates, but also the local set of transactions must be broadcast to all other sites. However, the data parallelism algorithms require that memory at each processor is large enough to store all candidates at eacy scan (otherwise the performance will degrade considerably due to the fact that I/O is required for both the database and the candidate set. The task parallelism approaches can avoid this as only the subset of the candidates which are assigned to a processor during each scan need fit in memory. Since not all partitions of the candidates need be the same size, the task parallel algorithms can adapt to the amount of memory at each site. The only restriction is that the total size of all candidates be small enough to fit into the total size of memory in all processors combined. Note that there are some variations of the basic algorithms discussed in this section which address these memory issues. Performance studies have shown that the data parallelism tasks scale linearly with the number of of processors and database size. Due to the reduced memory requirements, however, the task parallelism may work where task parallelism may not. A hybrid approach, Hybrid Distribution (HD), which combines the advantages of each technique has a speedup close to the data parallelism approach [44].

5.4.1 Data Parallelism

One data parallelism algorithm is the Count Distribution algorithm [6]. The database is divided into p partitions, one for each processor. Each processor counts the candidates for its data and then broadcasts its counts to all other processors. Each then determines the global counts. These then are used to determine the large itemsets and to generate the candidates for the next scan. The algorithm is shown in Algorithm 5.6.

Algorithm 5.6
Input:
I //*Itemsets.*
$P^1, P^2, ..., P^p$; //*Processors*
$D = D^1, D^2, ..., D^p$; //*Database divided into partitions.*
s //*Support.*
Output:
L //*Large Itemsets.*
Count Distribution Algorithm:

perform in parallel at each processor P^l; *//Count in parallel.*
  $k = 0$; *//k is used as the scan number.*
  $L = \emptyset$;
  $C_1 = I$; *//Initial candidates are set to be the items.*
  repeat
    $k = k + 1$;
    $L_k = \emptyset$;
    for each $I_i \in C_k$ **do**
      $c_i^l = 0$; *// Initial counts for each itemset are 0.*
    for each $t_j \in D^l$ **do**
      for each $I_i \in C_K$ **do**
        if $I_i \in t_j$ **then**
          $c_i^l = c_i^l + 1$;
    broadcast c_i^l *to all other processors;*
    for each $I_i \in C_k$ **do** *// Determine global counts.*
      $c_i = \sum_{l=1}^{p} c_i^l$;
    for each $I_i \in C_k$ **do**
      if $c_i \geq (s \times \mid D^1 \cup D^2 \cup ... \cup D^p \mid)$ **do**
        $L_k = L_k \cup I_i$;
    $L = L \cup L_k$;
    $C_{k+1} = Apriori - Gen(L_k)$
  until $C_{k+1} = \emptyset$;

5.4.2 Task Parallelism

The Data Distribution algorithm is one that demonstrates task parallelism [6]. Here the candidates as well as the database are partitioned among the processors. Each processor in parallel counts the candidates given to it using its local database partition. Following our convention, we use C_k^l *to indicate the candidates of size k examined at processor* P^l. *Also,* L_k^l *are the local large k-itemsets at processor l. Then each processor broadcasts its database partition to all other processors. Each processor uses this to then obtain a global count for its data and broadcasts this count to all other processors. Each processor can then determine globally large itemsets and generate the next candidates. These candidates are then divided among the processors for the next scan. Algorithm 5.7 shows this approach. Here we show that the candidates are actually sent to each processor. However some prearranged technique could be used locally by each processor to determine its own candidates. This algorithm suffers from high message traffic whose impact can be reduced by overlapping communication and processing.*

Algorithm 5.7
Input:
  I *//Itemsets.*
  $P^1, P^2, ..., P^p$; *//Processors*
  $D = D^1, D^2, ..., D^p$; *//Database divided into partitions.*
  s *//Support.*
Output:

L *//Large Itemsets.*

Data Distribution Algorithm:

$C_1 = I$;
for each $1 \leq l \leq p$ **do** *//Distribute size 1 candidates to each processor.*
determine C_1^l *and distribute to* P^l;
perform in parallel at each processor P^l; *//Count in parallel.*
$k = 0$; *//k is used as the scan number.*
$L = \emptyset$;
repeat
$k = k + 1$;
$L_k^l = \emptyset$;
for each $I_i \in C_k^l$ **do**
$c_i^l = 0$; *//Initial counts for each itemset are 0.*
for each $t_j \in D^l$ **do**
for each $I_i \in C_K^l$ **do**
if $I_i \in t_j$ **then**
$c_i^l = c_i^l + 1$; *//Determine local counts.*
broadcast D_l *to all other processors;*
for *every other processor* $m \neq l$ **do**
for each $t_j \in D^m$ **do**
for each $I_i \in C_K^l$ **do**
if $I_i \in t_j$ **then**
$c_i^l = c_i^l + 1$; *//Determine glocal counts.*
if $c_i \geq (s \times \mid D^1 \cup D^2 \cup ... \cup D^p \mid)$ **do**
$L_k^l = L_k^l \cup I_i$;
broadcast L_k^l *to all other processors;*
$L_k = L_k^1 \cup L_k^2 \cup ... \cup L_k^p$; *//Global large k-itemsets.*
$C_{k+1} = Apriori - Gen(L_k)$
$C_{k+1}^l \subset C_{k+1}$; *//Determine next set of local candidates.*
until $C_{k+1}^l = \emptyset$;

5.5 Incremental Rules

All algorithms discussed so far assume a static database. However, in reality we can not assume this. With these prior algorithms, generating association rules for a new database state requires a complete rerun of the algorithm. There have been some approaches which have been proposed to address the issue of how to maintain the association rules as the underlying database changes. Most of the approaches proposed have addressed the issue of how to modify the association rules as inserts are performed on the database. These incrementatl updating approaches concentrate on determining the large itemsets for $D \cup db$ *where* D *is a database state and* db *are updates to it and where the large itemsets for* D, L, *are known.*

One incremental approach, Fast Update (FUP) is based on the Apriori algorithm [17]. Each iteration, k, *scans both* db *and* D *with candidates generated from the prior iteration,* $k-1$, *based on the large itemsets at that scan. In addition, we use as part of the candidate set for scan* k *to be*

L_k found in D. The difference is that through pruning of the candidates, the number of candidates examined at each iteration is reduced. Although there are other pruning techniques used, primary pruning is based on the fact that we already know L from D. Remember that according to the large itemset property, an itemset must be large in at least one of these partitions of the new database. For each scan k of db, L_k plus the counts for each itemset in L_k are used as input. When the count for each item in L_k is found in db we automatically know if it will be large in the entire database without scanning D. We need not even count any items in L_k during the scan of db if they have a subset which is not large in the entire database.

5.5.1 Generalized Association Rules

Using a concept hierarchy which shows the set relationship between different items, Generalized Association Rules allow rules at different levels. Example 5.5 illustrates the usage of these generalized rules using the concept hierarchy in Figure 5.3. Association rules could be generated for any and all levels in the hierarchy. A generalized association rule, $X \Rightarrow Y$, is defined like a regular association rule with the restriction that no item in Y can be above any item in X. When generating generalized association rules, all possible rules are generated using one or more given hierarchies. There have been several algorithms proposed to generate generalized rules. The simplest would be to expand each transaction by adding (for each item in it) all items above it in any hierarchy [99].

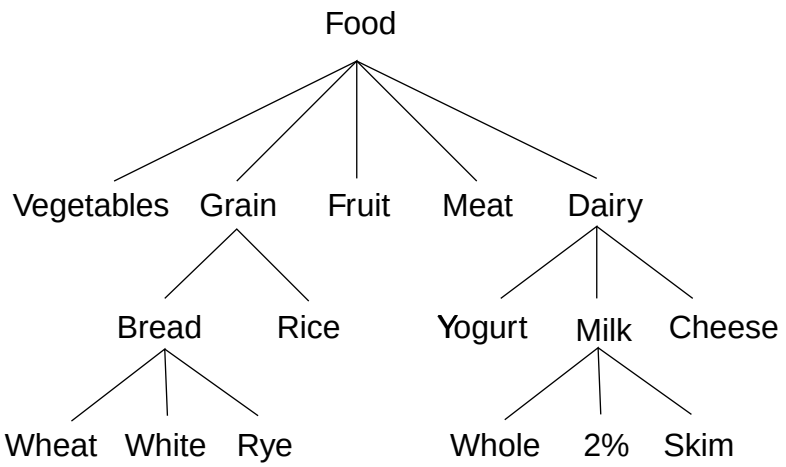


Figure 5.3: Concept Hierarchy

Example 5.5 *Figure 5.3 shows a partial concept hierarchy for food. This hierarchy shows that Wheat Bread is a type of Bread which is a type of grain. An association rule of the form $Bread \Rightarrow PeanutButter$ has a lower support and threshold than one of the form $Grain \Rightarrow PeanutButter$. There are obviously more transactions containing any type of grain than containing Bread. Likewise $WheatBread \Rightarrow Peanutbutter$ has a lower threshold and support then $Bread \Rightarrow PeanutButter$.*

5.5.2 Quantitative Association Rules

Association rule algorithms discussed so far assume that the data is categorical. A quantitative association rule is one involving categorical and quantitative data [98]. An example of a quantitative rule is:

A customer buys wine for between \$30 and \$50 a bottle ⇒ she also buys caviar

Notice that this differs from a traditional association rule such as:

A customer buys wine ⇒ she also buys caviar.

The cost quantity has been divided into an interval (much as was done when we looked at handling numeric data in clustering and classification). In these cases, the items are not simple literals. For example, instead of having the items $\{Bread, Jelly\}$*, we might have the items* $\{(Bread : [0..1]), (Bread : (1..2]), (Bread : (2..\infty)), (Jelly : [0..1.5]), (Jelly : (1.5..3]), (Jelly : (3..\infty))\}$

The basic approach to finding quantitative association rules is found in Algorithm 5.8. Here we show Apriori being used to generate the large itemsets, but any such algorithm could be used.

Algorithm 5.8
Input:
I *//Itemsets.*
$P^1, P^2, ..., P^p$; *//Processors*
$D = D^1, D^2, ..., D^p$; *//Database divided into partitions.*
s *//Support.*
Output:
L *//Large Itemsets.*
Quantitative Association Rule Algorithm:
for each $I_j \in I$ **do** *// Partition items.*
if I_j *is to be partitioned* **then**
determine number of partitions;
map atttribute values into new partitions creating new items;
replace I_j *in* I *with the new items* $I_{j1}, ..., I_{jm}$;
Apriori(I,D,s);

Due to the fact that we have divided what was one item into several, the minimum support and confidence used for quantitative rules may need to be lowered. The minimum support problem is obviously worse with a large number of intervals. Thus an alternative solution would be to combine adjacent intervals when calculating support. Similarly when there are a small number of intervals, the confidence threshold may need to be lowered. For example, look at $X \Rightarrow Y$*. Suppose there are only two intervals for* X*. Then the count for those transactions containing* X *will be quite high when compared to those containing* Y *(if this is a more typical item with many intervals).*

Algorithm 5.8 does not show how to determine if an item should be partitioned. One technique proposed to do this is a metric called partial-completeness [100].

5.5.3 Using Multiple Minimum Supports

When looking at large databases with many types of data, using one minimum support value can be a problem. Different items behave differently. Certainly it is easier to obtain a given support threshold with an attribute which has only two values than it is with one which has hundreds. It might be more meaningful to find a rule of the form:

$$SkimMilk \Rightarrow WheatBread$$

with a support of 3% *than it is to find:*

$$Milk \Rightarrow Bread$$

with a support of 6%*. Thus having only one support value for all association rules may not work well. Some useful rules could be missed. Note that this is particularly of interest when looking at generalized association rules, but may arise in other situations as well. Think of generating association rules from a non-market basket database. As was seen with quantitative rules, we may partition attribute values into ranges. Partitions which have a small number of values will obviously produce lower supports than those with a large number of values. If a larger support is used, we may miss out on generating meaningful association rules.*

This is called the rare item problem [73]. If the minimum support is too high then rules involving items which rarely occur will not be generated. If it is set to be too low, then too many rules may be generated, many of which (particularly for the frequently occuring items) are not important. Different approaches have been proposed to handle this. One is to partition the data based on support and generate association rules for each partition separately. Alternatively you could group rare items together and generate association rules for these groupings. A more recent approach to handling this problem is to combine clustering and association rules. First you cluster the data together based on some clustering criteria and then you generate rules for each cluster separately. This is a generalization of the partitioning of the data solution.

One approach, MSapriori, allows a different support threshold to be indicated for each item [67]. Here MIS stands for Minimum Item Support. The minimum support for a rule is the minimum of all the minimum supports for each item in the rule. An interesting problem occurs when multiple minimum supports are used: The minimum support requirements for an itemset may be met even though it isn't met for some of its subsets. Thus this seems to violate the large itemset property. Example 5.6 adapted from [67] illustrates this. A variation of the downward closure property, called the sorted downward closure property is satisfied and used for the MSapriori algorithm. First the items are sorted in ascending MIS value. Then the candidate generation at scan two only looks at adding to a large item any item following it (larger or equal MIS value) in the sorted order.

Example 5.6 *Suppose we have three items,* $\{A, B, C\}$*, with minimum supports* $MIS(A) = 20\%$*,* $MIS(B) = 3\%$*, and* $MIS(C) = 4\%$*. Because the support for* A *is so large, it may be small while both* AB *and* AC *may be large since the required support for* $AB = min(MIS(A), MIS(B)) = 3\%$ *and* $AC = min(MIS(A), MIS(C)) = 4\%$*.*

5.6 Exercises

1. *Trace the results of using Apriori to both the grocery and clothing store examples with* $s = 20\%$ *and* $\alpha = 40\%$*. Be sure to show the candidate and large itemsets for each database scan.*

2. *Prove that all potentially large itemsets are found by the repeated application of BD^- starting with PL in the Sampling algorithm.*

3. *Trace the results of using Sampling to both the grocery and clothing store examples with $s = 20\%$ and $\alpha = 40\%$. Be sure to show the usage of the negative border function as well as the candidates and large itemsets for each database scan.*

4. *Trace the results of using the Partition to both the grocery and clothing store examples with $s = 20\%$ and $\alpha = 40\%$. For the grocery store example use two partitions of size 2 and 3 respectively. For the clothing example, use three partitions (starting from the beginning of the database) of size 7, 7, and 6. You need not show all the steps involved in find the large itemsets for each partition. Simply show the resulting large itemsets found for each partition.*

5. *Trace the results of using the Count Distribution algorithm on the clothing data. Assume that there are three processors with partitions created from the beginning of the database of size 7, 7, and 6 respectively.*

6. *Trace the results of using the Data Distribution algorithm on the clothing data. Assume that there are three processors with partitions created from the beginning of the database of size 7, 7, and 6 respectively. Assume that candidates are distributed at each scan by dividing the total set into subsets of equal size.*

5.7 Bibliographic Notes

A survey of association rules has recently appeared [24]. A survey of parallel and distribution association rule algorithms has also been published [109].

Agrawal proposed the AIS algorithm prior to Apriori [5]. However, this algorithm another, SETM [51] do not take advantage of the Large Itemset Property, and thus generate too many candidate sets. The Apriori is still the major technique used by commercial products to detect large itemsets. Another algorithm proposed about the same time as Apriori, OCD, uses sampling [74]. It produces fewer candidates than AIS.

There have been many proposed algorithms which improve on Apriori. Apriori-TID does not use the database to count support [7]. Instead, it uses a special encoding fo the candidates from the previous pass. Apriori has better performance in early passes of the database while Apriori-TID has better performance in later. A combination of the two, Apriori-Hybrid, has been proposed [98]. The Dynamic Itemset Counting (DIC) algorithm divides the database into intervals (like the partitions in the Partition algorithm) [14]. The scan of first interval counts the 1-itemsets. Then candidates of size 1 are generated.The scan of the second interval,then, counts the 1-itemsets as well as those 2-itemsets. In this manner itemsets are counted earlier. However, more memory space may be required.

Other data parallel algorithms include PDM [83], DMA [19], and CCPD [110]. Additional task parallel algorithms include IDD [44], HPA [95], and PAR [111]. For a discussion of other parallel algorithms see either [24] or [109].

Many additional algorithms have been proposed. CARMA (Continous Association Rule Mining Algorithm) [48] proposes a technique which is dynamic in that it allows the user to change the support and confidence while the algorithm is running.

A variation of generalized rules are multiple-level association rules [43]. An approach to determining if an item should be partitioned when generating quantitative rules has been proposed [100]. Variations of quantitative rules include profile association rules where the left side of the rule represents some profile information about a custern while the right contains the purchase information [1]. Another variation on quantitative rules is a ratio rule [64]. These rules indicate the ratio between the quantitative values of individual items. When fuzzy regions instead of discrete partitions are used, you obtain fuzzy association rules [65]. Recent research has examined association rules for multimedia data [108].

Much additional work has examined association rules in an incremental environment. Several improvements on the original FUP have been proposed [18] [20]. Another technique aims at reducing the number of additional scans of the original database [102].

Bibliography

[1] *Charu C. Aggarwal, Zheng Sun, and Philip S. Yu. Online algorithms for finding profile association rules.* Proceedings of the ACM CIKM Conference, *86-95, 1998.*

[2] *Charu C. Aggarwal and Philip S. Yu. A new framework for itemset generation.* Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, *18-24, 1998.*

[3] *Rakesh Agrawal. Tutorial database mining.* Proceedings of the ACM International Conference on Management of Data, *pages 75–76, 1994.*

[4] *Rakesh Agrawal. Data mining: The quest perspective. Tutorial presented at EDBT Summer School Advances in Database Technology, September 1995.*

[5] *Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases.* Proceedings of the ACM International Conference on Management of Data, *pages 207–216, 1993.*

[6] *Rakesh Agrawal and John C. Shafer. Parallel mining of association rules.* IEEE Transactions on Knowledge and Data Engineering, *8(6):962–969, December 1996.*

[7] *Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases.* Proceedings of the International Very Large Databases Conference, *pages 487–499, 1994.*

[8] *G. H. Ball and D. J. Hall. Isodata, a novel method of data analysis and classification. Technical report, 1965.*

[9] *T. Bayes. An essay towards solving a problem in the doctrine of chances.* Philosophical Transactions of the Royal Society of London, *53:370–418, 1763.*

[10] *Alex Berson and Stephen J. Smith.* Data Warehousing, Data Mining, and OLAP. *McGraw-Hill, 1997.*

[11] *Ronald J. Brachman and Tej Anand. The process of knowledge discovery in databases. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors,* Advances in Knowledge Discovery and Data Mining, *pages 37–57. AAAI/MIT Press, 1996.*

[12] *Paul Bradley, Usama Fayyad, and Cory Reina. Scaling clustering algorithms to large databases.* Proceedings of the International Conference on Knowledge Discovery and Data Mining, *pages 9–15, 1998.*

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

[14] *Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data.* Proceedings of the ACM International Conference on Management of Data, *pages 255–264, 1977.*

[15] *J. Cendrowska. Prism: An algorithm for inducing modular rules.* International Journal of Man-Machine Studies, *27(4):349–370.*

[16] *Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: An overview from a database perspective.* IEEE Transactions on Knowledge and Data Engineering, *8(6):866–883, December 1996.*

[17] *D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered associaiton rules in large databases: An incremental updating technique.* Proceedings of the IEEE International Conference on Data Engineering, *pages 106–114, 1996.*

[18] *David W. Cheung, Vincent T. Ng, and Benjamin W. Tam. Maintenance of discovered knowledge: A case in multi-level association rules.* Proceedings of the Second International KDD Conference, *pages 307–310, 1996.*

[19] *David Wai-Lok Cheung, Jiawei Han, Vincent Ng, Ada Wai-Chee Fu, and Yongqian Fu. A fast distributed algorithm for mining association rules.* Proceedings of the Parallel and Distributed Information Systems Conference, *1996.*

[20] *Dvid Wai-Lok Cheung, Sau Dan Lee, and Benjamin C. M. Kao. A general incremental technique for maintaining discovered association rules.* Proceedings of the DASFAA, *1997.*

[21] *E. F. Codd. A relational model of data for large shared data banks.* Communications of the ACM, *13(6):377–387, June 1970.*

[22] *Two Crows Corporation.* Introduction to Data Mining and knowledge Discovery. *Two Crows Corporation, 1998.*

[23] *A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm.* Journal of the Royal Statistical Society, *B39:1–38, 1977.*

[24] *Margaret H. Dunham, Yongqiao Xiao, Le Gruenwald, and Zahid Hossain. A survey of association rules.* submitted to ACM Computing Surveys, *2000.*

[25] *Martin Ester, Hans-Peter Kriegel, J. Sander, and Xiaowei Xu. A density-base algorithm for discovering clusters in large spatial databases with noises.* Proceedings of the International Conference on Knowledge Discovery and Data Mining, *pages 226–231, 1996.*

[26] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. Communications of the ACM, *39(11):27–34, November 1996.*

[27] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Knowledge discovery and data mining: Towards a unifying framework. Proceedings of the International Conference on Knowledge Discovery and Data Mining, *pages 82–88, 1996.*

[28] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, *pages 1–34. AAAI/MIT Press, 1996.*

[29] E. A. Feigenbaum and J. Feldman, editors. Computers and Thought. *McGraw-Hill, 1963.*

[30] R. A. Fisher. On the probable error of a coefficient of correlation deduced from a small sample. Metron International Journal of Statistics, *1(4):3–32, 1921.*

[31] E. Fix and J. L. Hodges Jr. Discriminatory analysis; non-parametric discrimination: Consistency properties. Technical report, 1951.

[32] K. Florek, J. Lukaszewicz, J. Perkal, H. Steinhaus, and S. Zubrzycki. Taksonomia wroclawska. Przeglad Antropologiczny, *17(4):93–207, 1951.*

[33] E. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. Biometrics, *21:768, 1965.*

[34] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. In Knowledge Discovery in Databases. *AAAI Press, 1991.*

[35] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus - clustering categorical data using summaries. Proceedings of the International Conference on Knowledge Discovery and Data Mining, *pages 73–84, 1999.*

[36] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining very large databases. Computer, *32(8):38–45, August 1999.*

[37] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. Proceedings of the International Very Large Databases Conference, *pages 416–427, 1998.*

[38] Clark Glymour, David Madigan, Daryl Pregibon, and Padhraic Smyth. Statistical inference and data mining. Communications of the ACM, *39(11):35–34, November 1996.*

[39] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesley, 1989.*

[40] Amy E. Graham and Robert J. Morse. How u.s. news ranks colleges. U.S. News & World Report, *pages 84–87, August 1999.*

[41] *Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases.* Proceedings of the ACM International Conference on Management of Data, *pages 73–84, 1998.*

[42] *Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clusteirng algorithm for categorical attributes.* Proceedings of the IEEE International Conference on Data Engineering, *pages 512–521, 1999.*

[43] *Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules form large databases.* Proceedings of the International Very Large Databases Conference, *pages 420–431, 1995.*

[44] *Eui-Hong Hand, George Karypis, and Vipin Kumar. Scalable parallel data mining for association rules.* Proceedings of the ACM International Conference on Management of Data, *pages 277–288, 1997.*

[45] *Frank harary.* Graph Theory. *Addison-Wesley Publishing Company, 1972.*

[46] *John A. Hartigan.* Clustering Algorithms. *John Wiley & Sons, Inc., 1975.*

[47] *Simon Haykin.* Neural Networks - a Comprehensive Foundation, 2nd Edition. *Prentice Hall, 1999.*

[48] *Christian Hidber. Online association rule mining.* Proceedings of the ACM International Conference on Management of Data, *pages 145–156, 1999.*

[49] *J. H. Holland.* Adaptation in Natural and Artificial Systems. *University of Michigan Press, 1975.*

[50] *R. C. Holte. Very simple classification rules perform well on most commonly used datasets.* Machine Learning, *11:63–91, 1993.*

[51] *M. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases.* Proceedings of the IEEE International Conference on Data Engineering, *pages 25–34, 1995.*

[52] *E. B. Hunt, J. Martin, and P. J. Stone.* Experiments in Induction. *Academic Press, 1966.*

[53] *Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery.* Communications of the ACM, *39(11):58–64, November 1996.*

[54] *Britannica.com Inc.* Ockham's Razor. *http://www.britannica.com/bcom/eb/article, 2000.*

[55] *W. H. Inmon. What is a data warehouse?* http://www.cait.wustl.edu/papers/prism/vol1_no1/, *1995.*

[56] *W. H. Inmon. The data warehouse and data mining.* Communications of the ACM, *39(11):49–50, November 1996.*

[57] *John F. Elder IV and Daryl Pregibon. A statistical perspective on knowledge discovery in databases. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors,* Advances in Knowledge Discovery and Data Mining, *pages 83–113. AAAI/MIT Press, 1996.*

[58] *A. K. Jain, M. N. Murty, and P. J. Flynn. What is a data warehouse?* ACM Computing Surveys, *pages 264–323, September 1999.*

[59] *Anil K. Jain and Richard C. Dubes.* Algorithms for Clustering Data. *Prentice-Hall, 1988.*

[60] *N. Jardine and R. Sibson.* Mathematical Taxonomy. *John Wiley & Sons, Inc., 1971.*

[61] *L. Kaufman and P.J. Rousseeuw.* Finding Groups in Data: An Introduction to Cluster Analysis. *John Wiley & Sons, 1990.*

[62] *Ruby L. Kennedy, Yuchun Lee, Benjamin Van Roy, Christopher D. Reed, and Richard P. Lippman.* Solving Data Mining Problems Through Pattern Recognition. *Prentice Hall, 1998.*

[63] *T. Kohonen. Self-organized formation of topologically correct feature maps.* Biological Cybernetics, *43:59–69, 1982.*

[64] *Flip Korn, Alexandros Labrinidis, Yannis Kotidis, and Christos Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining.* Proceedings of the International Very Large Databases Conference, *1998.*

[65] *Chan Man Kuok, Ada Fu, and Man Hon Wong. Mining fuzzy association rules in databases.* Proceedings of the International Very Large Databases Conference, *1998.*

[66] *Jun-Lin Lin and Margaret H. Dunham. Mining association rules: Anti-skew algorithms.* Proceedings of the IEEE International Conference on Data Engineering, *pages 486–493, 1998.*

[67] *Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple supports.* Proceedings of KDD, *pages 337–341, 1999.*

[68] *H. Liu. X2r: A fast rule generator.* Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, *1995.*

[69] *Hongjun Lu, Rudy Setiono, and Huan Liu. Neurorule: A connectionist approach to data mining.* Proceedings of the International Very Large Databases Conference, *pages 478–489, 1995.*

[70] *Hongjun Lu, Rudy Setiono, and Huan Liu. Effective data minng using neural networks.* IEEE Transactions on Knowledge and Data Engineering, *8(6):957–961, 1996.*

[71] *Heikki Mannila. Data mining: Machine learning, statistics, and databases.* Proceedings of the Eighth International Conference on Scientific and Statistical Database Management, *pages 1–8, June 1996.*

[72] *Heikki Mannila. Methods and problems in data mining.* Proceedings of International Conference on Database Theory, *1997.*

[73] *Heikki Mannila. Database methods for data mining.* Proceedings of the KDD Conference, *1998.*

[74] *Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering associaton rules.* Proceedings of the AAAI Workshop on Knowledge Discovery in Databases (KDD-94), *pages 181–192, July 1994.*

[75] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, *5:115–133, 1943.*

[76] J. McQueen. Some methods for classificaiton and analysis of multivariate observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, *pages 281–297, 1967.*

[77] Ryszard S. Michalski, Ivan Bratko, and Miroslav Kubat. Machine Learning and Data Mining Methods and Applications. *John Wiley & Sons Ltd., 1999.*

[78] Tom M. Mitchell. Machine Learning. *McGraw-Hill, 1997.*

[79] Tom M. Mitchell. Machine learning and data mining. Communications of the ACM, *42(11):31–36, November 1999.*

[80] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. Proceedings of the International Very Large Databases Conference, *pages 144–155, 1994.*

[81] N. J. Nilsson. Learning Machines. *McGraw Hill, 1965.*

[82] Ozsu and Patrick Valduriez. Principles of Distributed Database Systems. *Prentice-Hall, 1999.*

[83] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash based algorithm for mining association rules. Proceedings of the ACM International Conference on Management of Data, *pages 175–186, 1995.*

[84] Gregor Purdy and Stephen Brobst. Perfect dimensions. Intelligent ENTERPRISE, *2(8):48–53, June 1999.*

[85] J. R. Quinlan. Induction of decision trees. Machine Learning, *11(1):81–106, 1986.*

[86] J. R. Quinlan. C4.5: Programs for Machine Learning. *Morgan Kaufmann, 1993.*

[87] Naren Ramakrishnan and Ananth Y. Grama. Data mining: From serendipity to science. Computer, *32(8):34–37, August 1999.*

[88] Rajeev Rastogi and Kyuseok Shim. Scalable Algorithms for Mining Large Databases. *http://www.bell-labs.com/projects/serendip, August 1999. Tutorial presented at the CIKM Conference.*

[89] M. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, *65:386–408, 1958.*

[90] B. s. Everitt. Cluster Analysis. *John Wiley & Sons, Inc., 1974.*

[91] G. Salton. The SMART Retrieval System - Experiments in Automatic Document Processing. *Prentice-Hall, 1971.*

[92] Gerald Salton and Michael J. McGill. Introduction to Modern Information Retrieval. *McGraw-Hill, 1983.*

[93] *Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases.* Proceedings of the International Very Large Databases Conference, *pages 432–444, 1995.*

[94] *J. Shafer, R. Agrawal, , and M. Meha. Sprint: A scalable parallel classifier for data mining.* Proceedings of the International Very Large Databases Conference, *pages 544–555, 1996.*

[95] *Takahiko Shintani and Masaru Kitsuregawa. Hash based parallel algorithms for mining association rules.* Proceedings of the Parallel and Distributed Information Systems Conference, *1996.*

[96] *Harry Singh.* Data Warehousing Concepts, Technologies, Implementations, and Management. *Prentice Hall PTR, 1998.*

[97] *P. H. A. Sneath and R. R. Sokal.* Numerical Taxonomy. *W. H. Freeman and Company, 1973.*

[98] *Ramakrishnan Srikant. Fast algorithms for mining association rules and sequential patterns. Technical report, 1996.*

[99] *Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules.* Proceedings of the International Very Large Databases Conference, *pages 407–419, 1995.*

[100] *Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables.* Proceedings of the ACM International Conference on Management of Data, *pages 1–12, 1996.*

[101] *Toby J. Teorey and James P. Fry.* Design of Database Structures. *Prentice-Hall, 1982.*

[102] *Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, and Sanjay Ranka. An efficient algorithm for the incremental updation of association rules in large databases.* Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD), *page 263, 1997.*

[103] *Hannu Toivonen. Sampling large databases for association rules.* Proceedings of the International Very Large Databases Conference, *pages 134–145, 1996.*

[104] *G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks.* Machine Learning, *13(1):71–101, 1993.*

[105] *R. C. Tryon and D. E. Bailey.* Cluster Analysis. *McGraw-Hill, Inc., 1970.*

[106] *Jr. W. T. McCormick, P. J. Sweitzer, and T. W. White. Problem decomposition and data reorganization by a clustering technique.* Operations Research, *20(5):993–1009, September-October 1922.*

[107] *Ian H. Witten and Eibe Frank.* Data Mining Practical Machine Learning Tools and Techniques. *Morgan Kaufmann, 2000.*

[108] *Osmar R. Zaiane, Jiawei Han, Ze-Niam Li, and Joan Hou. Mining multimedia data. Technical report, 1998.*

[109] Mohammed Javeed Zaki. Parallel and distributed association mining: A survey. IEEE Concurrency, *October-December 1999.*

[110] Mohammed Javeed Zaki, Mitsunori Ogihara, Srivivasan Parthasarathy, and Wei Li. Parallel data mining for association rules on shared-memory multiprocesors. Technical report, May 1996.

[111] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New parallel algorithms for fast discovery of association rules. Data Mining and Knowledge Discovery, *1(4):343–373, December 1997.*

[112] C. T. Zhan. Graph-theoretical methods for detecting and describing gestalt clusters. IEEE Transactions on Computers, *C(20):68–86, 1971.*

[113] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clusteirng method for very large databases. Proceedings of the ACM International Conference on Management of Data, *pages 103–114, 1996.*

[114] Jure Zupan. Clustering of Large Data Sets. *Research Studies Press, A Division of John Wiley & Sons Ltd., 1982.*

Index