Acta Numerica (2001), pp. 313–355 © Cambridge University Press, 2001

Data mining techniques

Markus Hegland
Centre for Mathematics and its Applications,
School of Mathematical Sciences,
Australian National University,
Canberra ACT 0200, Australia
E-mail: Markus.Hegland@anu.edu.au

Methods for knowledge discovery in data bases (KDD) have been studied for more than a decade. New methods are required owing to the size and complexity of data collections in administration, business and science. They include procedures for data query and extraction, for data cleaning, data analysis, and methods of knowledge representation. The part of KDD dealing with the analysis of the data has been termed data mining. Common data mining tasks include the induction of association rules, the discovery of functional relationships (classification and regression) and the exploration of groups of similar data objects in clustering. This review provides a discussion of and pointers to efficient algorithms for the common data mining tasks in a mathematical framework. Because of the size and complexity of the data sets, efficient algorithms and often crude approximations play an important role.

CONTENTS

1. Introduction

The following is an attempt at an introduction and review of some current data mining techniques. The main focus is on the computational aspects of data processing aspects, and not on statistics, data management or data retrieval. Data mining is a new and rapidly growing field. It draws ideas and resources from several disciplines, including machine learning, statistics, database research, high-performance computing and commerce. This explains the multifaceted and rapidly evolving nature of the data mining

discipline. While there is a broad consensus that the abstract goal of data mining is to *discover new and useful information in databases*, this is where the consensus ends, and the means of achieving this goal are as diverse as the communities contributing. Thus any reasonably sized treatment of data mining techniques necessarily has to be selective, and perhaps biased towards a particular approach. Despite this, we hope that the following discussion will provide useful information for readers wishing to get some understanding of ideas and challenges underlying a selection of data mining techniques. This selection includes some of the most widely used data mining problems such as association rule mining, predictive models, and clustering. The focus is on fundamental concepts and on the challenges posed by data size, dimensionality, and data complexity. It is hoped that this overview gives the computational mathematician, in particular, some starting points for further exploration. We believe that data mining does provide many new and challenging questions for approximation theory, stochastic analysis, numerical analysis and parallel algorithm design.

Data mining techniques are used to find patterns in large datasets. A necessary property of algorithms capable of handling large and growing datasets is their scalability, or linear complexity with respect to the data size. Patterns in the database are described by relations between the attributes. In a sense, a relational database itself defines a pattern. However, the size of the relations makes it impossible to use them directly for further predictions or decisions. On the other hand, these relations only provide information about the available observations and cannot be directly applied to future observations. The power of generalization from specific observation is obtained from statistics and machine learning.

The variables, or attributes, considered here are assumed to be either continuous or categorical. However, more general data types are frequently analysed in data mining; see Bock and Diday (2000). The techniques discussed here are not based on sampling, and access every item in the full dataset.

The different disciplines are also reflected in different goals of data mining techniques (Ramakrishnan and Grama 1999):

Induction.
Find general patterns and equations that characterize the data.
Compression.
Reduce the complexity of the data, replace by simpler concepts.
Querying.
Find better ways to query data, *i.e.*, extract information and properties.
Approximation.
Find models that approximate the observations well.
Search.
Look for recurring patterns.

Furthermore, we may classify data mining algorithms according to three key elements (Ramakrishnan and Grama 1999):

Model representation.
Decision trees, regression functions and associations.
Data.
Continuous, time series, discrete, labelled, multimedia or nominal data.
Application areas.
Finance and economy, biology, web logs, web text mining.

The discovered patterns can be characterized according to accuracy and precision, expressiveness, interpretability, parsimony, 'surprisingness', 'interestingness', or 'actionability' (Ramakrishnan and Grama 1999).

Assume that the initial raw data are stored in a relational database, that is, a collection of tables or relations. Each table contains a sequence of records, each of which consists of a number of attribute values. These tables are typically used for transactional purposes, that is, for the management of a business. For example, a health insurance company would store a table containing information on all the doctors, another for the patients, and maybe a third for claims, containing pointers into the other two tables. Each record in these tables contains information about the individuals: for example, the patient table would contain the age, sex and location of a patient. In order to guarantee consistency of the information and avoid redundancy, the tables are normalized (Date 1995).

Assume now that in a data mining project the claims of patients are to be further examined. In particular, we might be interested in differences between claims for different doctors of a particular specialization. Thus a first step is to restructure the original database into a database where each new 'record' now contains many records from several tables and the data mining investigation is to compare these 'records' in order to find patterns and outliers.

In a next step, major characteristics of these records are extracted. In our health insurance example this might be the number of patients, the total claim or the service offered by a particular doctor. These characteristics may be *symbolic objects* (Bock and Diday 2000) including sets, intervals and frequency distributions, but in this discussion we will mainly contain our discussion to *features* which are either continuous variables or categorical. The choice of these features is very important and often the main reason for failure or success of a data mining project. If, in our example, the features do not provide information about doctors and patients, the analysis of their interrelations will not provide any interesting insights. One data mining task is indeed the identification of features containing information that can contribute to a particular research question. We will assume that the features have been chosen, and focus here instead on the algorithms used

to analyse them. Thus, after these preliminary steps, we are left with one table or relation which is a sequence of feature vectors, and the data mining task is to explore and describe how these feature vectors are interrelated.

From a statistical perspective the data are a sequence of independent vectors described by a probability distribution. The goal of data mining is then to uncover interesting structures or aspects of the underlying probability distribution. Questions which are addressed are as follows.

- Are there areas that have a higher probability? This is addressed by clustering techniques and association rules.
- Can some of the variables be explained by others, and how well? This is addressed by classification and regression.
- Where are the areas where these functional relationships are better/worse? The analysis of areas of high misclassification rates and the residuals of regression address this.

The sizes of databases analysed are growing exponentially. In fact, it has been suggested that data grow at the same rate as computational resources, which, according to Moore's law, double every 18 months (Bell and Gray 1997). Today data collections in the megabyte range are very common. Gigabyte data collections are becoming available, including many business data collections, some of which easily extend into the terabyte range, particularly when the world wide web is involved. Further, data collections in the petabyte range are now emerging. The largest challenge is not so much the absolute size of the databases but their constant growth. Thus one of the largest challenges in computer science in general is the generation of systems which are capable of handling such growing datasets, that is, the systems need to be *scalable* (Bell and Gray 1997). Even if hardware and software performance scale at the same rate, the computational complexity may destroy overall scalability. Indeed, if one were to use an $\mathcal{O}(n^3)$ algorithm, then a typical data mining task in 10 years time would require roughly 10,000 times as long compared to a similar typical data mining task today, as both the data size (n) and the computational speed would have increased about 100-fold during this period, but the computational complexity would have increased by 10^6. Thus a typical data mining task may take one hour today, but in 10 years time a similar typical data mining task would take over one year due to the combined effects of data growth, increase in computing speed and $\mathcal{O}(n^3)$ complexity. Thus it is essential that all data mining algorithms have near-linear time complexity with data size ($\mathcal{O}(n \log(n)$ is usually acceptable).

The areas selected for the current discussion are association rules (Section 2), classification and regression (Section 3) and clustering (Section 5). The reader wishing to get further pointers to these and other areas of data mining is encouraged to search for relevant literature on the web, a useful

guide being the book on data mining by Han and Kamber (2001). In addition to discussing a vast collection of data mining research it also provides a database perspective. Association rule discovery is perhaps data mining at its purest, and the techniques, while possibly motivated by earlier work in rule discovery, have evolved within the data mining community. As this area is the newest of the three covered, the mathematical foundations are less well studied. In contrast, there has been a lot of theoretical work in the area of classification in the machine learning community: in particular, we would like to point to the book by Devroye, Györfi and Lugosi (1996). While linear regression is widely studied in statistics, non-parametric regression is still a relatively new field, particularly in the case of high-dimensional data. Clustering techniques have been used for many years and the literature on clustering is quite formidable. We will only reveal the tip of the iceberg here.

2. Association rules

The motivation for association rules comes from market basket analysis (see Agrawal, Imielinski and Swami (1993)). A market basket is a collection of items purchased by a customer in one transaction. (In the case of retail shops this is the content of a shopping trolley.) Typically we are faced with a large number of possible items and market baskets which only cover a small proportion of all items. It is observed that items are purchased in groups, and the aim of the market basket analysis is to detect these groups. More specifically, we would like to be able to predict groups of items that occur in a market basket given that certain items are present. This information is then used for product placement in shops and on web pages.

The underlying data are modelled as a sequence of data records or *transactions* $\omega \in \Omega$. It is assumed that the records are distributed according to a distribution function $p : \Omega \to \mathbb{R}$. Rules are a very popular way to express knowledge about the data because they are comprehensible. Rules consist of 'if-then clauses' which depend on the data. The probabilities are normalized; in the case of a finite set Ω we have

$$P(\Omega) = \sum_{\omega \in \Omega} p(\omega) = 1. \tag{2.1}$$

In this discussion it is assumed that the probability is time-invariant; in practice the distributions may vary. Market basket analysis that takes this into account is discussed in Ramaswamy, Mahajan and Silbershatz (1998).

In the case of association rules, the records ω are subsets of a set of items $J = i_1, \ldots, i_m$, which is here assumed to be finite. *Association rules* are statements about itemsets $A \subset J$ and $B \subset J$; in particular, the rule

$$A \Rightarrow B$$

states that, for many transactions ω, we may conclude $B \subset \omega$ if we know that $A \subset \omega$. Such a rule is only useful if $A \subset \omega$, and we say that a transaction ω supports A in this case. The probability that a randomly selected transaction ω contains A is the *support* $s(A)$:

$$s(A) := P\left(\{\omega : A \subset \omega \subset J\}\right). \tag{2.2}$$

In practice, the support is obtained from the database as the ratio of transactions ω supporting A to the total number of transactions N:

$$s(A) \approx \frac{N_A}{N}, \tag{2.3}$$

where $N_A = |\{\omega_i : A \subset \omega_i\}|$ is the number of transactions supporting A, and $|X|$ denotes the cardinality of the set X.

Often, there is no distinction made in practice between the probability and this ratio of frequencies, which is then also denoted with $s(A)$.

The support of the association rule $A \Rightarrow B$ is defined by

$$s(A \Rightarrow B) := s(A \cup B), \tag{2.4}$$

the probability that both A and B are subsets of a random ω. For association rule mining, the user imposes a minimal support s_0 for any rule that is further explored, that is, $s(A \Rightarrow B) > s_0$. This limits the number of spurious rules generated.

A second measure of the 'interestingness' of a rule is the *confidence*

$$c(A \Rightarrow B) := \frac{s(A \cup B)}{s(A)}, \tag{2.5}$$

which is the (conditional) probability that $B \subset \omega$ given that $A \subset \omega$. In addition to the minimal support, the user selects a minimal confidence in order to prune out random rules. An association rule which satisfies both $s(A \Rightarrow B) \geq s_0$ and $c(A \Rightarrow B) \geq c_0$ is called a *strong association rule*. High confidence means that, in all the transactions containing A, a large proportion also contains B. However, this is only interesting if, in the subset of itemsets containing A, the itemsets also containing B are more frequent than in the full set. The ratio of these proportions is called the *lift* of the association rule:

$$\gamma(A \Rightarrow B) := \frac{c(A \Rightarrow B)}{s(B)} = \frac{s(A \cup B)}{s(A)s(B)}.$$

Note that $\gamma(A \Rightarrow B) = \gamma(B \Rightarrow A)$.

The co-occurrence of A and B in the transactions ω is further analysed in the *contingency table* (see Table 2.1), and it turns out that all the frequencies can be computed from N, N_A, N_B and $N_{A \cup B}$. The rule $A \Rightarrow B$ is irrelevant if the association of $A \subset \omega$ with $B \subset \omega$ only occurs owing to coincidence,

Table 2.1. Contingency table for $A \Rightarrow B$

	$A \subset \omega$	$A \not\subset \omega$	Σ
$B \subset \omega$	$N_{A\cup B}$	$N_B - N_{A\cup B}$	N_B
$B \not\subset \omega$	$N_A - N_{A\cup B}$	$N - N_A - N_B + N_{A\cup B}$	$N - N_B$
Σ	N_A	$N - N_A$	N

that is, the two events are statistically independent. This hypothesis can be tested using a χ^2 test statistic (see, *e.g.*, Dobson (1990)) because

$$X^2(A \Rightarrow B) = N \frac{(s(A \cup B) - s(A)s(B))^2}{s(A)s(B)(1 - s(A))(1 - s(B))}$$

can be seen to have a χ^2 distribution with 2 degrees of freedom if the supports are estimated with the frequencies. Note that all the measures so far could be expressed in terms of the support $s(A)$ of itemsets A.

If the number of transactions supporting N_A is small, the confidence estimate can become unreliable owing to large random fluctuations. In this case the *Laplace estimator*

$$c(A \Rightarrow B) \approx \frac{N_{A\cup B} + 1}{N_A + k},$$

where $1/k$ is an *a priori* guess of the confidence, may improve accuracy (Segal and Etioni 1994); an alternative is the Yates correction (Quinlan 1987) for an application to rule induction.

In addition to confidence, support, lift and χ^2, many other criteria have been used to describe the interestingness of an association rule. Many such measures depend on the application. However, the basic techniques rely on properties of support and confidence and thus these will be the only two measures discussed further here. Alternatives and further discussion of such measures can be found in Chen, Han and Yu (1996), Srikant and Agrawal (1995) and Silberschatz and Tuzhilin (1996).

The enumeration of all strong association rules is a considerable *search problem*. In the general case, the determination of optimal rules is known to be NP-hard (Morishita and Nakaya 2000). For the determination of strong association rules, however, efficient algorithms have been found. There are $\mathcal{O}(m^k)$ k-itemsets with a low number k of items from a total number of m items. Evaluating all the rules generated from these itemsets requires $\mathcal{O}(Nm^k)$ tests for support, which is not feasible computationally. Fortunately, not all the itemsets need to be checked, as most have a very small support. The foundation is the anti-monotonicity property of the support

function. Even though the following properties follow readily, because of the importance of the application we formulate them as lemmata.

Lemma 1. The function $s(A)$ is *anti-monotone*, that is, if $A \subset B$ then $s(A) \geq s(B)$ and furthermore $s(\emptyset) = 1$.

Proof. The anti-monotonicity follows from the monotonicity of the probability P because, if $A \subset B$, then

$$\{\omega : A \subset \omega\} \supset \{\omega : B \subset \omega\}.$$

Furthermore, by equation (2.1), we have

$$s(\emptyset) = P(\{\omega : \emptyset \subset \omega\}) = 1. \qquad \square$$

Setting $B - A := \{i \in B : i \notin A\}$, we derive the following result from the definition.

Lemma 2. If $B \subset A$ then $s(A \Rightarrow B) = s(A)$ and $c(A \Rightarrow B) = 1$. Furthermore, $s(A \Rightarrow B) = s(A \Rightarrow (B - A))$ and $c(A \Rightarrow B) = c(A \Rightarrow (B - A))$. In particular,

$$s(\emptyset \Rightarrow \emptyset) = 1 \tag{2.6}$$
$$s(\emptyset \Rightarrow A) = s(A \Rightarrow \emptyset) = s(A) \tag{2.7}$$
$$c(\emptyset \Rightarrow \emptyset) = 1 \tag{2.8}$$
$$c(\emptyset \Rightarrow A) = s(A) \tag{2.9}$$
$$c(A \Rightarrow \emptyset) = 1. \tag{2.10}$$

Proof. Firstly, $s(A \Rightarrow B) = s(A \cup B) = s(A)$ and $c(A \Rightarrow B) = s(A \cup B)/s(A) = s(A)/s(A) = 1$.

Then $s(A \Rightarrow B) = s(A \cup B) = s(A \cup (B - A)) = s(A \Rightarrow (B - A)$ and $c(A \Rightarrow B) = s(A \cup B)/s(A) = s(A \cup (B - A))/s(A) = c(A \Rightarrow (B - A))$. $\square$

Basically, rules with $A \cap B \neq \emptyset$ do not do anything new, and so we only consider rules $A \Rightarrow B$ for which

$$A \cap B = \emptyset.$$

The association rule $\emptyset \Rightarrow \emptyset$ is the trivial rule, and the rule $\emptyset \Rightarrow A$, sometimes also written as '$\Rightarrow A$', can be identified with the itemset A, thus making the itemsets just a special case of an association rule.

An itemset A satisfying a condition $s(A) \geq s_0$ for a given s_0 is called a *frequent itemset*. (Note that this definition is not absolute: it depends on the choice of the parameter s_0.) The main property of frequent itemsets is as follows.

Lemma 3. (*a priori* property) Every subset of a frequent itemset is frequent.

Proof. Let $A \subset J$ be frequent, that is, $s(A) \geq s_0$, and let $B \subset J$. Then, by the anti-monotonicity property we have $s(B) \geq s(A)$ and thus $s(B) \geq s_0$, and thus B is also a frequent itemset. □

The *a priori* property has been the strongest tool in the development of efficient association mining algorithms. Once the frequent itemsets are determined, finding strong association rules is very efficient.

Lemma 4. Once all the frequent itemsets and their supports are known, the strong association rules can be found without any further scans of the database.

Proof. By definition, for any strong association rule $A \Rightarrow B$ the itemset $A \cup B$ is frequent. By the *a priori* property, the itemset A is frequent too.

Thus all the the strong association rules can be found by searching through all partitions $C = A \cup B$ for all frequent itemsets C and selecting the ones for which the confidence $c(A \Rightarrow B) = s(C)/s(A) > c_0$. This does not require any further scans as both C and A are frequent and thus their supports $s(C)$ and $s(A)$ were evaluated earlier. □

So far, a general algorithm for the determination of strong association rules consists of two steps:

(1) find all frequent itemsets A and their support $s(A)$;
(2) from this determine the strong itemsets.

Only the first step requires scanning the database. Thus the first step is usually orders of magnitude more expensive than the second step. Most improvements of the algorithm focus on the first step; the second step is achieved simply by enumerating all possible rules from partitions of the frequent itemsets.

As the size $|\omega|$ of the transactions is finite, the size of the frequent itemsets $|A|$ is limited. Let a *k-itemset* be an itemset A with size $|A| = k$. Then it follows from the *a priori* property that there are many more frequent k-itemsets than there are frequent $(k+1)$-itemsets. A systematic search for the frequent itemsets may thus look for frequent itemsets by order of their size. Let the set of all frequent k-itemsets be denoted by L_k, that is,

$$L_k := \{A \subset J : s(A) \geq s_0, \quad |A| = k\},$$

in particular, $L_0 = \emptyset$. Assume in the following, that the items $i_k \in J$ are ordered (in an arbitrary but fixed way). Given this ordering we define the *join* $L_k * L_k$ of two sets of k-itemsets by

$$L_k * L_k = \{\{i_1, \ldots, i_{k+1}\} : \{i_1, \ldots, i_k\},\ \{i_1, \ldots, i_{k-1}, i_{k+1}\} \in L_k\},$$

where the i_s are ordered, that is, $i_s \leq i_{s+1}$, for $s = 1, \ldots, k$.

This provides a first upper bound for the set of frequent k-itemsets.

Lemma 5. We have

$$L_{k+1} \subset L_k * L_k, \text{for } k \geq 0.$$

Proof. For any $\{i_1, \ldots, i_{k+1}\} \in L_{k+1}$, by the *a priori* property all subsets of size k are frequent k-itemsets; in particular, $\{i_1, \ldots, i_k\} \in L_k$ and $\{i_1, \ldots, i_{k-1}, i_{k+1}\} \in L_k$ and so $\{i_1, \ldots, i_{k+1}\} \in L_k * L_k$. □

This set can thus be easily constructed. While it may have much fewer than $\mathcal{O}(m^k)$ elements, it can still be large and may be pruned further using the *a priori* property. This *pruning step*, which does not require any database scanning, generates a *candidate itemset* C_{k+1} from the $L_k * L_k$ by

$$C_{k+1} := \{A \in L_k * L_k : \text{all } k\text{-itemsets } B \subset A \text{ satisfy } B \in L_k\}.$$

Application of the *a priori* property again gives

$$L_{k+1} \subset C_{k+1}, \quad k = 1, 2, \ldots.$$

This candidate itemset is the best we can get using the *a priori* property as any element contains only frequent itemsets.

Lemma 6. Any $B \subset A$ of any $A \in C_k$ is a frequent itemset.

Proof. By construction any k-itemset $B \subset A \in C_k$ is a frequent k-itemset. Any j-itemset with $j < k$ has to be by construction a subset of a frequent k itemset and is thus also frequent. □

Thus the *a priori* property provides ways to reduce the size of the *negative border*, that is, the points which are in C_k but not in L_k. While the supports of the itemsets in L_k are required for the determination of the confidence, the supports of the elements in the negative border are 'wasted', and so we would like to keep the size of $|C_k| - |L_k|$ to a minimum.

The *a priori algorithm* implements the repeated application of join, prune and evaluation of the support of itemsets to determine all itemsets. It is described in Algorithm 1, opposite.

The most expensive step is the scanning of the database. If we assume that determining whether a k-itemset is supported by a transaction ω requires $\mathcal{O}(k)$ operations, then the complexity of the *a priori* algorithm is $\mathcal{O}(N \sum_{k=1}^{\infty} |C_k| k)$. The performance of the *a priori* algorithm has been improved in numerous ways (see, *e.g.*, Han and Kamber (2001)), including

- reduction of the size of scans;
- reduction of the number of scans.

The reduction of the size of scans prunes transactions ω that do not contain frequent itemsets, using another consequence of the *a priori* property.

Lemma 7. Every item of a frequent $(k+1)$-itemset is contained in at least k frequent k-itemsets and thus in at least k candidate k-itemsets.

Algorithm 1 *a priori* algorithm

$C_1 := \{\{i\} : i \in J\}$ {set of all 1-itemsets}, $k = 1$
while $C_k \neq \emptyset$ **do**
  For all $A \in C_k$ determine $s(A)$ by scanning database.
  $L_k :=$ set of all frequent itemsets in C_k.
  Construct join $L_k * L_k$.
  Prune result by removing sets with infrequent subsets to get C_{k+1}.
  $k := k + 1$
end while
return all frequent itemsets $\bigcup L_k$

Proof. Let i be an item of a frequent $(k+1)$-itemset A. Then A has exactly k subsets of size k that contain i. Because of the *a priori* property these subsets have to be frequent k-itemsets.

The set of candidate k-itemsets C_k contains all frequent k-itemsets because $L_{k+1} \subset C_{k+1}$, as derived in the proof of Lemma 5. □

Using this lemma, the *a priori* algorithm is extended to contain a pruning of elements of transactions that are not in k different candidate k-itemsets. This pruning is done during the scanning step and no additional scanning is required. Note that whole transactions may be removed from the database and a substantial reduction in complexity may result. For more details and an implementation of this idea see Park, Chen and Yu (1995*a*), where a reduction of the number of counting steps based on hashing has been suggested.

Smaller association mining tasks may fit into memory and so multiple scans may not be necessary. However, for very large databases this is not realistic, and in this case the database has to be completely scanned from disk k times, where k is the maximal number of elements of the occurring frequent itemsets. A partitioning approach can reduce this number to two scans, which can substantially speed up the algorithm. For this let the database $D = (T_1, \ldots, T_N)$ be partitioned into p pairwise disjoint subsets D_i, that is,

$$D = D_1 \cup \cdots \cup D_p$$

such that $D_i \cap D_j = \emptyset$ for $i \neq j$. This is called a *partition of* D. We define the support of an itemset A in the partition D_j by

$$s_j(A) := \frac{|\{\omega \in D_j : A \subset \omega\}|}{|D_j|},$$

and an itemset A is said to be *frequent in* D_j if it satisfies

$$s_j(A) \geq s_0.$$

With this we get the following result.

Lemma 8. (Invariant partitioning property) Let $D_1, \ldots, D_p$ be a partition of D. Then each itemset that is frequent in D is frequent in at least one subset D_i.

Proof. Let A be an itemset that is frequent in D. Then

$$\begin{aligned} s_0 \le s(A) &= \frac{|\{\omega \in D : A \subset \omega\}|}{|D|} \\ &= \sum_{i=1}^{p} \frac{|D_i|}{|D|} \frac{|\{\omega \in D_i : A \subset \omega\}|}{|D_i|} \qquad \text{as } \{D_i\} \text{ are disjoint} \\ &= \sum_{i=1}^{p} \frac{|D_i|}{|D|}\, s_i(A) \\ &\le \max_{1 \le i \le p} s_i(A). \end{aligned}$$

Thus A is least frequent in the set D_i in which it has maximal support. □

The modified *a priori* algorithm in a first scan gathers all the frequent itemsets and their supports in the partitions D_i which are chosen such that they fit comfortably in memory. Because of the invariant partitioning property the local frequent itemsets are potential frequent itemsets over the whole database. A second scan through the data returns the supports for all these potentially frequent itemsets over D. This can be done very efficiently, as the supports of many of these itemsets are already known for many partitions. See Savasere, Omieski and Navanthe (1995) for a further discussion of this approach. Other approaches to the reduction of the number of scans are found in Toivonen (1996) and Brin *et al.* (1997).

A common problem in the determination of strong association rules is that the items may be too specific to produce any rules of interest. In the case of market basket analysis, the brand names and quantities as typically represented by a bar code may not reveal anything interesting. Choosing a more general concept, such as a class of consumer goods, may lead to more interesting results. If items are included in the set J which refer to more general concepts the relationship between the items is modelled by a *directed acyclic graph with nodes in* J. This can represent multiple hierarchies or taxonomies. Of particular interest is the transitive closure of this graph. An edge (i_1, i_2) in this transitive closure means that the the item i_2 is a *generalization* of i_1, or i_1 is of type i_2. For example, i_1 could be rye bread, and i_2 could be bread, in the example of market basket analysis. The graph is defined by the application and is a type of domain knowledge or constraint on the data.

Using this we can generalize the concept of support. An *itemset B supports an itemset A* if, for any element $i \in A$, either

- $i \in B$, or
- there is a $j \in B$ such that i is an ancestor of j.

We say that A generalizes B, or $A \leq B$. We see immediately that if $B \subset A$ then $B \leq A$. Also, the relationship $\leq$ is transitive owing to the transitivity of $\subset$, and it can be seen that this defines a partial ordering on Ω that extends the ordering given by $\subset$. The relationship with set operations is very close and we have the following.

Lemma 9. For any itemsets A, B and C, we have

$$C \geq A \quad \text{and} \quad C \geq B \quad \text{if and only if} \quad C \geq A \cup B.$$

Proof. If $C \geq A$ and $C \geq B$ then each $i \in A \cup B$ is an element of one of A or B (or both) and so it is either an element of C or an ancestor of an element of C, which means that $A \cup B$ generalizes C.

Conversely, if $C \geq A \cup B$ then, as any element of A is also an element of $A \cup B$, it is either an element of C or an ancestor of an element of C. Thus $A \leq C$, and the same argument holds for B. □

In the special case where $C = A$ the above lemma gives $A \geq A \cup B$ if $A \geq B$. Because $A \subset A \cup B$ we also have $A \leq A \cup B$. Thus the supports of the two itemsets are identical, and for our purposes they are thus indistinguishable. An example of such an itemset $A \cup B$ would contain both an item and an ancestor of this item. It is suggested that the itemsets are normalized to exclude this case, so that the only itemsets we will consider are modulo any ancestors of the elements in the itemset. We define $A \vee B$ to be the normalization of $A \cup B$ and we will still denote the normalized itemsets by A and B. The support of an itemset in this hierarchical context is now defined by

$$s(A) := P(\omega \in \Omega \mid A \leq \omega). \tag{2.11}$$

Note that the probability distribution has to be compatible with the hierarchy so that a distribution over the normalized itemsets can be defined. It follows that the support is also *anti-monotone*, that is, for $A \leq B$ we have $s(B) \leq s(A)$. A *frequent itemset* is defined as before by $s(A) \geq s_0$. The *a priori* property is obtained as above from the *a priori* algorithm. Further, frequent k-itemsets, the join operation, pruning and the candidate itemsets are straightforward generalizations. Normalization only has to be done explicitly in the *a priori* algorithm when generating the C_2, because all the itemsets are normalized automatically in the process (Srikant and Agrawal 1995).

Lemma 10. If s is defined as in equation (2.11) and C_2 is pruned such that it contains only normalized itemsets, then both C_k and L_k generated in the *a priori* algorithm contain only normalized itemsets.

Proof. By construction C_1, C_2 and L_1, L_2 are normalized. We use induction to show that the others are normalized as well. Assume now that $L_1, \ldots, L_k$ are normalized and $k \geq 2$. If there were an unnormalized set $A \in C_{k+1}$, then there would be two items $i_1, i_2 \in A$ such that i_1 is an ancestor of i_2. By Lemma 6, however, any pair of elements of a subset of C_{k+1} is frequent, that is, $\{i_1, i_2\} \in L_2$ and, because L_2 is normalized, i_1 can never be an ancestor of i_2 and thus C_{k+1} has to be normalized. □

As a consequence of this lemma the *a priori* algorithm can easily be modified to include hierarchies. The importance of this approach becomes apparent for the case of *quantitative association rules*. In this case the transactions contain sets of real numbers. Typically, the probability of any particular combination of real numbers is zero for continuous probability distributions and thus there are no frequent itemsets in the original sense. However, if we include intervals as items we can get frequent itemsets again. We will not go into the theory of quantitative association rules here but instead we will show how hierarchies provide a framework for approximation of association rules, in particular for discretization in the case of quantitative association rules. Further discussion of the case of hierarchical association rules can be found in Agrawal and Srikant (1995), Mannila, Toivonen and Verkamo (1995), Mannila and Toivonen (1996) and Shintani and Kitsuregawa (2000).

In the following, hierarchy-based approximation will be discussed in the case of finite sets J but this can be generalized to more general cases. So, given a set of items J with a graph, we would like to understand how well a subset of items $J' \subset J$ covers the space of strong association rules. The set J' has to satisfy several properties: in particular, for any element in J there has to be at least an ancestor in J'. We can then approximate any item $i \in J$ by the closest ancestor $\pi(i) \in J'$. This mapping induces a mapping between itemsets on J and J' which is defined elementwise, and again denoted by π as

$$\pi(A) := \{\pi(i) : i \in J\}.$$

From this we get

$$\pi(A \cup B) = \pi(A) \cup \pi(B) \tag{2.12}$$

$$\pi(A) \leq A \tag{2.13}$$

$$\pi(A) \cup \pi(B) \leq A \cup B \tag{2.14}$$

$$s(A) \leq s(\pi(A)). \tag{2.15}$$

If we can also bound the support of this 'projection' from below by

$$s(\pi(A)) \leq K s(A)$$

for some $K \geq 1$, we say that the triple (J, J', K) is K-*complete*. In this case we can give a bound on the effect this approximation has on the confidence of rules. We assume that an association rule $A \Rightarrow B$ is approximated by $\pi(A) \Rightarrow \pi(B)$.

Lemma 11. If (J, J', π) is K-complete then

$$\frac{c(\pi(A) \Rightarrow \pi(B))}{c(A \Rightarrow B)} \in [1/K, K].$$

Proof. This bound was proved by Srikant and Agrawal (1996*a*), who also provide further analysis and a choice of J' for the case of quantitative association rules, where bins or intervals are used to approximate the itemsets. □

Association rule mining can be generalized to the analysis of sequences, which has been termed *sequence mining*. Typically, a sequence is defined as a sequence of itemsets and frequent k-sequences containing k items can be found with a variant of the *a priori* algorithm. Several algorithms have been discussed in Srikant and Agrawal (1996*b*), Agrawal and Srikant (1995), Oates, Schmill, Jensen and Cohen (1997) and Zaki (1998, 2000). Related to this is the discovery of frequent episodes (Mannila *et al.* 1995, Mannila and Toivonen 1996).

Ultimately, itemsets are described by predicates $A(\omega)$ defined on the transactions. In *multidimensional association rule mining* the algorithms are generalized for sets of predicates. Predicate sets are conjunctions of predicates and, using the *a priori* approach, frequent predicate sets are found from which if-then rules $A \Rightarrow B$ with precedent A and antecedent B can be derived. In general such *rule induction* is an area which has been of much interest in the machine learning community. The rule induction process selects rules that satisfy certain *constraints* depending on the dataset. These constraints relate to the quality or the interestingness of the rules. Typical data mining tasks are as follows.

- Find the set of all rules that satisfy the constraints. Note in particular that the consequent C is not determined either.
- For a given consequent and a given ordering of the rules, find the best antecedents A.

In general, finding best rules is an optimization problem that is known to be NP-hard (Morishita 1998). One heuristic assumes the availability of a LEARN-ONE-RULE subroutine which is capable of generating one rule from any dataset. After this is applied for the first time to the dataset all the true positives are removed and the LEARN-ONE-RULE is applied again. This greedy algorithm is the *sequential covering algorithm* (Mitchell 1997, p. 275).

A different approach is based on *beam search* and a very general algorithm for rule induction is suggested in Provost, Aronis and Buchanan (2000). At each node of a search tree, a set of possible specializations consisting of conjunctions with new predicates is chosen. Good efficiency is obtained by pruning off unpromising paths. This approach was also applied to association rule mining in Webb (2000). Classical rule induction techniques include RL (Clearwater and Provost 1990) or RIPPER (Cohen 1995). Parallel and distributed rule induction algorithms can be found in Hall, Chawla, Bowyer and Kegelmayer (2000), Williams (1990) and Hall, Chawla and Bowyer (1998).

Association rule mining is a recent development and has its origins in the data mining community. This is an area also undergoing constant change and development. At the core of the developments are the focus on frequent itemsets and the *a priori* property. Current research is dealing with the following questions:

- Which interestingness criteria produce the most useful rules?
- What are the most effective algorithms? Of interest are scalability with respect to data size, parallelism, higher dimensionality, disk access and memory hierarchies.
- How are complex data analysed? This problem includes multimedia and web data.

A good collection of pointers to the literature up to around 2000 can be found in Han and Kamber (2001).

3. Classification and regression

Functional relationships

$$Y = f(X_1, \ldots, X_d)$$

between the features Y and X_i form a powerful tool for summarizing aspects of the data in order to extract important data classes or to predict future data trends. The data is again modelled as a set of transactions $D = (\omega_1, \ldots, \omega_N)$ where $\omega_i \in \Omega$, but in addition it is assumed that we also know some features or random variables Y and X_i defined on Ω. If Y is categorical, the problem of determining the model f from the data is called classification, and in the case of continuous Y we speak of regression.

3.1. Classification

Classification is treated in depth in the machine learning literature; see, *e.g.*, Mitchell (1997). Both associations $A \Rightarrow B$ and functions $Y = f(X)$ provide relations between features of the data. The following is a comparison of some properties of associations and functions.

- The dependent variable Y is fixed for classification, while finding the consequent is part of association rule mining.
- The features of association rules are the predicates, which are Boolean, while in classification the features can take arbitrary types. The right-hand side Y of a classifier has values in a finite set $\{C_0, \ldots, C_m\}$ and the most commonly discussed case is $m = 2$, and in this case Y is basically a predicate.
- The performance of a classifier is judged by its accuracy, rather than by support and confidence.
- One function f summarizes all the data compared with a set of association rules.
- As in the case of association rule induction, the search for good classifiers often starts with a simple classifier, and successively more complex ones are considered until the desired performance is achieved.

In terms of the distribution of the probability the *misclassification rate* is defined by

$$L(f) = P(\{\omega : Y \neq f(X)\}).$$

This is an important measure of the quality of the classifier and is also called the Bayes risk. In the case where Y, and thus $f(X)$, is Boolean, the misclassification rate can be determined from the support defined in the previous section by

$$L(f) = s(Y) + s(f(X)) - 2s(Y \wedge f(X)).$$

This suggests that association rule technology can be applied to classification. Examples of this approach can be found in Dong and Li (1999), Li, Dong and Ramamohanrarao (2000), Lent, Swami and Widom (1997), Liu, Hsu and Ma (1998), and Meretakis and Wüthrich (1999).

3.2. Decision trees

Decision or classification trees use a partitioning of the domain Ω into hyperrectangles parallel to the values of the features X_i. They can be defined recursively by

$$f(X) = \begin{cases} f_1(X), & \text{if } A(X) \text{ holds,} \\ f_2(X), & \text{else,} \end{cases}$$

where f_1 and f_2 are either constant or again decision trees. The predicate $A(X)$ defines the split. In most cases the predicate $A(X)$ only depends on one of the features X_i. Decision trees originated in machine learning and statistics (Breiman, Friedman, Olshen and Stone 1984, Quinlan 1986, Quinlan 1993). During the induction of the decision tree the data needs to be revisited for each new level of the tree. For N data points, a balanced tree

will have $\mathcal{O}(\log(N))$ levels. As the tree building algorithm has to scan the data for each level, the average complexity of the tree building algorithm is $\mathcal{O}(N \log(N) m)$, where m is the number of attributes or features considered. Thus the algorithm is close to scalable in the number of data points, especially under the assumption that the levels of the tree are kept constant for growing data sizes.

The models from (small) decision trees are comprehensible and can quite easily be cast as standard database queries (which may be expressed in SQL, the standard for relational database queries). They can be generated rapidly and show good accuracy. These properties make them well suited to data mining. They can also be used as a starting point for the generation of rules. In fact, there is one rule per leaf of the tree, which is obtained by taking the conjunction of all the predicates $A(X)$ used to split the tree on a path from the leaf to the root. With the topmost split we get, in the case of Boolean f,

$$f(X) = A(X) \wedge f_1(X) \vee \overline{A(X)} \wedge f_2(X),$$

where $\overline{A(X)}$ denotes the negation of $A(X)$. This provides a recursion to build the rules. We start with constant values at the leaves. Then, let the $A_i^{(1)}(X) \Rightarrow B_i^{(1)}$ and $A_i^{(2)}(X) \Rightarrow B_i^{(2)}$ be the rules generated for both descendants of the node; then

$$A(X) \wedge A_i^{(1)}(X) \Rightarrow B_i^{(1)}, \quad \text{and} \quad \overline{A(X)} \wedge A_i^{(2)}(X) \Rightarrow B_i^{(2)}$$

are the rules for the top node. The antecedents are constants and do not change during the extraction process. They are used to initialize the rules for each leaf to True $\Rightarrow B_i^{(s)}$.

These initial rules, however, are often not very useful and may have low support, as the supports of the antecedents of the rules are mutually exclusive. Several steps are required to create rules that are useful for data mining from the initial rules (Quinlan 1993):

- Remove predicates in the rule which do not improve the rule. This pruning is often done based on a χ^2 test.
- Rules with overall low quality are removed and similar rules are merged.
- A default rule is required which covers all the cases not covered by other rules.

These steps do require further assessment of the quality of the generated rules and thus need further scanning of the data. As the pruning step may need to be repeated several times, the time spent on this data scanning may be substantial. See Quinlan (1987) for implementation of the rule induction from decision trees.

The basic algorithm for the construction of the decision tree consists of repeated assessment of the purity of the nodes in each partition and decisions on which partition to split into subpartitions and how; see Algorithm 2.

Algorithm 2 Basic decision tree algorithm

```
Partition(Ω)
if all the points in Ω are of the same class then
  return
else
  for each attribute A in Ω do
    evaluate all splits of A
  end for
  find best split Ω = Ω1 ∪ Ω2
  Partition(Ω1)
  Partition(Ω2)
end if
```

The evaluation of the splits is often based on an *impurity function* $\phi(p)$ based on the proportion p of transactions which are in the class of interest. If the proportion of class 1 cases is p in Ω, and p_1 and p_2 in the subsets $\Omega 1$ and $\Omega 2$ respectively, then the decrease of the impurity due to the splitting is modelled as

$$\phi(p) - s\phi(p_1) - (1-s)\phi(p_2),$$

where s is the proportion of cases in Ω_1. Examples of impurity functions are:

- $\phi(p) = \min(p, (1-p))$ (based on misclassification rate)
- $\phi(p) = 2p(1-p)$ (Gini (Breiman *et al.* 1984))
- $\phi(p) = -p\log(p) - (1-p)\log(1-p)$ (entropy (Quinlan 1986))

Alternatively, the χ^2 value is used. The construction of the tree has two steps. First the tree is grown until each leaf is either pure or contains very few points. Then the tree is pruned again in order to reduce overfitting using an error estimator.

In order to handle the large and growing data sizes, scalable parallel algorithms are required. An example is the SPRINT algorithm (Shafer, Agrawal and Mehta 1996). The computing platform is a 'shared nothing' parallel computer (*e.g.*, a Beowulf cluster) where the processors have their own memories and disks and the data are distributed evenly over these local disks. The data records are assumed to be sequences of features $(x_1^{(s)}, \ldots, x_d^{(s)}, y^{(s)})$ for $s = 1, \ldots, N$ are stored with some redundancy in sorted *attribute lists* which for each attribute x_i consist of sequences $(i, x_s^{(i)}, y^{(i)})$ sorted on $x_s^{(i)}$. This allows the efficient evaluation of splits in the case of real attributes. This evaluation is done based on histograms of the values of y for each x_s, and can be done in parallel. Hash tables are used to associate the transaction record identifier i with both the processor and the partition, and the Gini index is used for splitting. Experiments demonstrate the scalability of this

approach. In fact, SPRINT performs well with respect to three important criteria:

Scale-up. Fix the data per processor and study time as a function of the number of processors.

Speed-up. For a constant data size the time is studied as a function of p.

Size-up. The number of processors is fixed and the time is studied as a function of the data size n.

If the misclassification rate is determined from the dataset used for training, we get an estimate that is systematically too low. Better estimates are obtained from the *hold out* method, where the data are partitioned into training data for computation and test data for the estimation of the error. A popular alternative is *generalized cross-validation* or GCV (Golub, Heath and Wahba 1979), which emulates the hold-out technique without the reduction of the size of the training data.

3.3. Bayesian classifiers

The best classifier minimizes $L(f)$ and can be characterized in terms of the probability distribution as follows (see Devroye *et al.* (1996, p. 10)).

Theorem 1. Let $Y \in \{0, 1\}$ and

$$f^*(\mathbf{x}) = \begin{cases} 1, & \text{if} \quad P(Y = 1|\mathbf{x}) \geq 0.5, \\ 0, & \text{else.} \end{cases}$$

Then $L(f^*) \leq L(f)$.

The classifier f^* minimizing $L(f)$ is called *Bayes' classifier*. The minimum $L^* = L(f^*)$ is the Bayes error and characterizes how difficult a certain classification problem is. Other measures have been considered in the literature but are seen to be closely related to the Bayes error (Devroye *et al.* 1996, p. 21ff).

In practice, the distribution of the random variables is unknown and so the classifier has to be determined from observations. The observations are modelled as a sequence $(\mathbf{X}_i, Y_i)_{i=1}^n$ of identically distributed independent random variables. Practical classifiers are then functions $f_n(\mathbf{x}; \mathbf{X}_1, Y_1, \ldots, \mathbf{X}_n, Y_n)$.

An alternative, more general definition of the Bayesian classifier (Duda and Hart 1973) is

$$f^*(\mathbf{x}) = \text{argmax}_y \ p(y|\mathbf{x})$$

where, as usual, $p(y|\mathbf{x}) = P(Y = y|\mathbf{X} = \mathbf{x})$ is the conditional probability of Y being in class y when $\mathbf{X} = \mathbf{x}$. Practical estimators for f^* are obtained by the introduction of estimates for the conditional probabilities $p(y|\mathbf{x})$ in the above formula. The classifiers obtained in this way are called *plug-in*

decisions in the case of Boolean y and we have the following result (Devroye *et al.* 1996, p. 16).

Theorem 2. We have

$$P(f(X) \neq Y) \geq P(f^*(X) \neq Y) + 2E\,|\eta(\mathbf{X}) - \tilde{\eta}(\mathbf{X})|,$$

where $\eta(\mathbf{x}) = P(Y = 1|\mathbf{x})$, $\tilde{\eta}$ is the estimate of η, and $E(\cdot)$ denotes the expectation.

Many estimates of the conditional probability use *Bayes' formula*:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})},$$

where $p(y)$ is the probability of class y and $p(\mathbf{x})$ is the probability density function in the feature space in order to apply standard density estimators rather than have to estimate a family of probabilities.

In the case where the features are conditionally independent random variables we get

$$p(y|\mathbf{x}) = \frac{p(y)\Pi_{i=1}^{d} p(x_i|y)}{\sum_y p(y)\Pi_{i=1}^{d} p(x_i|y)}.$$

Techniques based on this assumption are called *naive Bayes* estimators. Langley, Iba and Thompson (1992) have demonstrated the effectiveness of naive Bayes in the case of independent Boolean features, and a newer analysis can be found in Langley and Sage (1999). It is also claimed that this approach works well even in cases where the features are not independent.

Note that f^* does not depend on the denominator. We obtain a *discriminant function* by removing the denominator and taking the logarithm

$$g(y|\mathbf{x}) = \log p(y) + \sum_{i=1}^{d} \log p(x_i|y),$$

and because of the monotonicity of the logarithm we get

$$f^b(\mathbf{x}) = \operatorname{argmax}_y g(y|\mathbf{x}).$$

The practical determination of f^b then uses estimates of the conditional probability densities $p(x_i|y)$ for all the variables x_i.

In the case of categorical x_i the estimate used for $p(x_i|y)$ is just the frequency, and for continuous features we often use a normal distribution for $p(x_i \mid y)$. In this case we get

$$g(y|\mathbf{x}) \approx k_y + \sum_{\text{categorical } x_i} k(x_i|y) + \sum_{\text{continuous } x_i} \frac{(x_i - \mu_i)^2}{2\sigma_i^2}.$$

There are two steps in the determination of $f(\mathbf{x})$:

(1) The determination of of the probability distributions. This step only has to be done once but requires the exploration of all the data points.
(2) The determination of the maximum of the discriminant function. This step has to be done for each evaluation of $f(\mathbf{x})$.

The determination of $p(y)$ requires n additions in order to count the frequencies of all classes. Only the c values of $p(y)$ need to be stored. In order to determine the distributions $p(x_i|y)$ for each feature x_i, the frequencies in the case of categorical x_i do require n additions; in the case of continuous variables and the normal distribution we require $2n$ additions and n multiplications to determine all the variances and means. Thus the total number of floating point operations required for the first step is

$$n + d_1 n + 3d_2 n,$$

where d_1 is the number of categorical variables and d_2 the number of continuous variables (the dimension is thus $d = d_1 + d_2$).

The evaluation of $f(\mathbf{x})$ does require the computation of the discriminant function for all the c possible values of y, which requires cd additions for the sum and $5cd_2$ floating point operations to evaluate the terms of the second sum.

Thus the naive Bayes classifier is computationally very efficient, as

- the data only have to be read once and do not need to be kept in memory,
- the algorithm is scalable in the number of observations,
- the algorithm is linear in the number of dimensions, and
- in addition, the main operations of the learning stage can be done in parallel both over the test dataset and over the dimensions.

However, the naive Bayes classifier does rely on two assumptions: first, it is based on the probabilistic model of the observations, and second, it requires that the features are independent for each class. However, in practice, features are mostly correlated. While it has been observed that, even in examples with correlations, naive Bayes is often as effective as tree-based classifiers (Langley and Sage 1994), it would appear that dealing with the correlations may improve the performance even further. Domingos and Pazzani (1997) provide the region where Bayesian classifiers are optimal as well as further evidence for the good performance.

Langley and Sage (1994) introduce the *selective Bayesian classifier*, which is able to remove redundant or heavily correlated features. The algorithm starts with no attributes, then adds one at a time until no more improvement is found. As, at each step, the effect of all the remaining attributes has to be

Table 3.2. Costs in time and space of the naive and flexible Bayes classifier

	Naive Bayes		Flexible Bayes	
	time	space	time	space
training on n cases	$\mathcal{O}(nk)$	$\mathcal{O}(k)$	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$
evaluation	$\mathcal{O}(k)$		$\mathcal{O}(nk)$	

recomputed, the complexity of this method is of $\mathcal{O}(d^2)$. In order to estimate the errors we could consider cross-validation techniques or hold-out, but the authors have simply used the accuracy on the training set and obtained good results. They included attributes as long as they did not degrade accuracy. In view of the complexity of the selective Bayesian classifier we lose the linear dependence on the dimension, and we have to read the data several times, once for each feature included.

One limitation of the approach is the use of normal distributions for the conditional probabilities in the case of numerical variables. John and Langley (1995) propose an approach using kernel estimators for the densities, specifically

$$p(X_i = x_i | Y = y) = (nh)^{-1} \sum_j K\left(\frac{x - \mu_j}{h}\right).$$

This increases, in particular, the number of operations for the evaluation stage. It is shown that in most cases a significant improvement in the classification performance is obtained. The time and space costs are given in Table 3.2.

A different way to include dependencies of the features are the *Bayesian networks*. A Bayesian network is a *directed acyclic graph* (DAG) where all the vertices are random variables X_i such that X_i and all the variables that are not descendants of X_i or parents of X_i are conditionally independent given the parents of X_i. The main property of Bayesian networks is that the probabilities of all the random variables can be computed almost as if they were independent (Bender (1996, pp. 308–313), Heckerman, Geiger and Chickering (1995), Friedman, Geiger and Goldszmidt (1997)).

Theorem 3. If $(\mathbf{X}, E, P)$ is a Bayesian network, then

$$P(\mathbf{X}) = \prod_{P(C(X_i))\neq 0} P(X_i | C(X_i)).$$

Conversely, if $(\mathbf{X}, E)$ is a DAG and if f_i is a nonnegative function such that

$\sum f_i(X_i) = 1$, then

$$P(\mathbf{X}) = \prod f_i(X_i)$$

defines a probability space for which $(\mathbf{X}, E, P)$ is a Bayesian network. Furthermore, $P(X_i|C(X_i))$ is either 0 or $f_i(X_i)$.

Here $C(X_i)$ denotes the parents (or direct causes) of X_i.

4. Regression

Regression refers in data mining to the determination of functions

$$Y = f(X_1, \ldots, X_m),$$

where the response variable Y is real-valued. The features X_i are assumed to be given and can have any type. In logistic regression, these models are also used to model decision functions or data density distributions. Parametric models, in particular linear models, are immensely popular in data mining, but non-parametric models are also widely used and will be the focus of the following discussion. Data mining applications to regression pose some additional challenges when compared to smaller statistical applications:

- The data mining datasets are very large, with millions of transactions. Furthermore, they are high-dimensional, including up to several thousand features.
- The models found should be understandable in the application domain.
- The data are usually collected for some other purpose, such as management of accounts. Thus carefully designed experiments are an exception in data mining studies.

This shows that data mining must pay close attention to computational efficiency and simple models are often preferred. Furthermore, even though the data can be very large, the limitations of the information in the data have to be carefully studied, and the effect of skewed or long-tail distributions has to be assessed.

As before, let $\omega \in \Omega$ denote a transaction and $p(\omega)$ be the probability distribution on Ω. The features X_i and Y are real functions on Ω, *i.e.*, random variables. The function f is assessed according to how well it fits the data, and often the expected squared error is used as a measure of the error:

$$E((Y - f(\mathbf{X}))^2) = \int_\Omega (Y(\omega) - f(\mathbf{X}(\omega)))^2 \, \mathrm{d}p(\omega).$$

Of course, it is impossible to assess this error directly. Practical estimators use a special portion of the data selected as a test set to estimate the error

by the sum of squares

$$R(f) := \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - f(\mathbf{x}^{(i)}))^2.$$

In order to avoid over-optimistic error rates, a different independent part of the data is used for the evaluation of the error. Alternatively, particularly in the case of small datasets, we use *generalized cross-validation*, or GCV (Golub *et al.* 1979).

The best estimator with respect to the exact mean squared error is the conditional probability

$$f(\mathbf{x}) = E(Y|\mathbf{X} = \mathbf{x}).$$

The straightforward determination of this using the average is usually not feasible, as it is unlikely that enough (or even one) data points are available for every possible value of $\mathbf{x}$ such that $f(\mathbf{x})$ may be estimated by the mean of the corresponding values of Y. Assume, for example, $m = 30$ categorical features with 6 possible values each. Then the total number of possible values for the independent variable is $6^{30} \approx 2 \cdot 10^{23}$, which is way beyond the number of observations available. Thus smoothing, simplified and parametric models have to be used.

A seemingly simple approach to regression is to quantify or discretize Y and then use a classification technique to approximate f. This approach can make use of all the classification techniques; however, in order to get good performance we may have to use a very high number of classes. Consider, for example, the approximation of a simple one-dimensional linear function. A linear model requires only two parameters but a classification technique may require hundreds of parameters. Thus real models may be much simpler than their approximating (discretized) classification models.

A large class of regression functions has the following form:

$$f(\mathbf{x}) = f_0 + \sum_i f_i(x_i) + \sum_{i,j} f_{i,j}(x_i, x_j) + \sum_{i,j,k} f_{i,j,k}(x_i, x_j, x_k) + \cdots,$$

where the level of interactions may be determined by the data but an upper bound is often supplied by the user. It is seen that one or two levels are often enough and hardly any cases are found with interactions higher than level 5. These are called *ANOVA decompositions*, because of the similarity to the interaction models in ANOVA.

4.1. Regression trees

Regression trees (Breiman *et al.* 1984) are based on the same ideas as decision trees. They do have the same advantages and, in fact, they may be

used with logistic regression for the determination of classification probabilities. In contrast to techniques based on tensor products of one-dimensional spaces, regression trees are extremely successful in dealing with high dimensions as their complexity is proportional to the dimension. They have two shortcomings, however: they do not represent linear functions well and, more generally, they cannot approximate smooth functions accurately, as they are discontinuous, piecewise constant functions. This approach is described by Breiman *et al.* (1984).

The trees define, as in the case of classification, a partitioning of the space of the independent variable $\mathbf{x}$ and for each partition a function estimate is provided. The main questions in building the tree are how to split the domain into subdomains, how to prune the regression tree and how to associate functions to the leaves of the tree.

Addressing the last question first, we choose the function constant on all the subdomains Ω_i related to the leaves of the tree. Note that the mean

$$\overline{f}_j := \frac{1}{N_j} \sum_{\mathbf{x}^{(i)} \in \Omega_j} y^{(i)}$$

minimizes the sum of squared residuals

$$R_j(f) := \sum_{\mathbf{x}^{(i)} \in \Omega_j} (y^{(i)} - f(\mathbf{x}^{(i)}))^2$$

between constants and the data over the subdomain Ω_j, and, for all leaves Ω_j, we have the least squares approximation

$$\overline{f}(\mathbf{x}) = \overline{f}_j, \quad \mathbf{x} \in \Omega_j,$$

which is constant on Ω_j. Thus, once the partitioning of the domain is done, the determination of the 'best' function is straightforward. The splitting is selected based on the reduction of the sum of squares, or, in order to adapt the complexity of the model to the data, we use a penalized sum of squared residuals

$$R_\alpha(f) = R(f) + \alpha |L_f|,$$

where L_f is the number of leaves of f, that is, a complexity measure, possibly using a different dataset for its evaluation.

For every variable there are several splittings possible and we select the one that maximizes the decrease of error in ΔR. This splitting is continued until each of the leaf nodes contains at most $N_{\min}$ observations.

Typically, the resulting tree is substantially overfitting the data, and we would like to select a subtree that is less complex. Complexity is measured by the number of leaves of the tree, *i.e.*, $|L_f|$. Thus instead of minimizing the sum of squares $R'(f)$, we attempt to balance complexity with fit on the

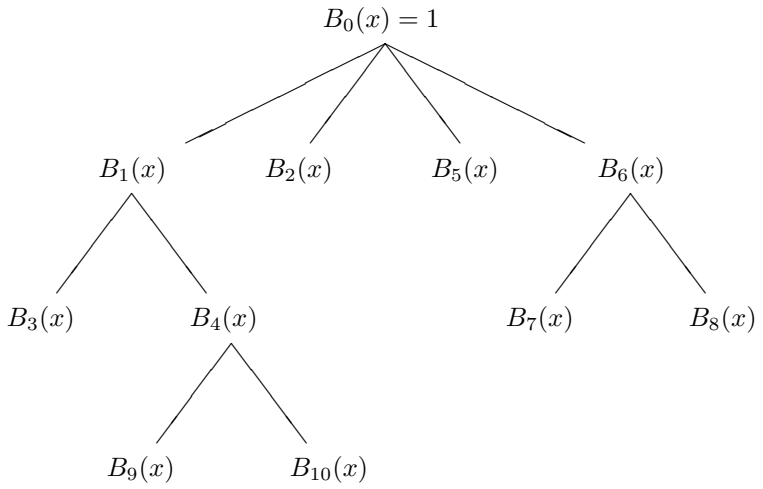

Fig. 4.1. Hierarchy of MARS basis functions

training dataset by minimizing

$$R_\alpha(f) = R'(f) + \alpha|L_f|.$$

Note that on the right-hand side an independent data size has to be used in practice for the evaluation of $R(f)$. For an N-dimensional space Ω, an approximation defined by a tree with M leaves and N data points, this algorithm does require $\mathcal{O}(nNM)$ operations (Friedman 1991). Thus the procedure is scalable, that is, linear in the number of observations N.

4.2. Regression splines

Better approximation properties are obtained by piecewise polynomial functions. They are used in the MARS algorithm (*multivariate adaptive regression splines*): see Friedman (1991). Instead of explicitly generating a hierarchy of domains, MARS generates a hierarchy of basis functions that implicitly define a hierarchy of domains. An example of a basis function hierarchy is displayed in Figure 4.1. At the root of the tree is the constant basis function $B_0 \equiv 1$. At each 'partitioning' step two new children are generated:

$$B_{\text{child}_1} = B_{\text{parent}}(x)(x_j - \xi)_+ \quad \text{and} \quad B_{\text{child}_2} = B_{\text{parent}}(x)(-x_j + \xi)_+,$$

where $(z)_+$ denotes the usual truncated linear function. It is equal to z for $z > 0$ and equal to zero if $z < 0$. The parent, the variable x_j and the value ξ are all chosen such that the sum of squared residuals is minimized.

While each node can have several children, there are some rules for the generation of this tree:

- The depth of the tree is bounded, typically by a value of 5 or less. This is thought to be sufficient for practical purposes (Friedman 1991), and the bound is important for controlling the computational work required for the determination of the function values.
- Each variable x_j can only appear once in a factor of a basis function B_k. This guarantees that the function is piecewise multilinear.

The partitioning of the domain is defined such that on each partition the function is multilinear. Thus this partitioning does not have the same interpretative value as in the case of classification and is not recovered by the algorithm. However, the MARS method generates an ANOVA decomposition.

The computational complexity of the algorithm, some clever updating ideas having been applied, is shown in Friedman (1991) to be $\mathcal{O}\big(dNM_{\max}^4/L\big)$ where d is the dimension, N the number of data points, $M_{\max}$ the number of basis functions considered – some might not be used later because of pruning – and L is the number of levels of the tree. Thus the algorithm is scalable, that is, the complexity is proportional to the data size. However, the proportionality constant can be very large owing to the dependence on $\mathcal{O}\big(M^4\big)$, which limits the number of basis functions that can be used, and thus limits the approximation power of this approach. Another limitation is due to the fact that a greedy algorithm is used and the choice of basis functions may not be a global optimum. One problem with the usage of truncated powers is stability. Bakin, Hegland and Osborne (1998) suggest a variant of the MARS algorithm where the truncated powers are replaced by a hierarchy of B-splines. In addition, a parallel implementation is provided.

4.3. Additive models

In the case where the tree of basis functions generated by MARS contains two levels, namely, the root $B_0 \equiv 0$ and its children, a model of the following form is generated:

$$f(x_1, \ldots, x_d) = f_0 + \sum_{i=1}^{d} f_i(x_i).$$

The univariate components f_i of this additive model are piecewise linear in the case of MARS. Other commonly used additive models are based on smoothing splines and local parametric (polynomial) approximations (Hastie and Tibshirani 1990). A unified treatment for all these approximations reveals that in all cases additive models are scalable with respect to data size

and, like the multivariate regression splines, conquer the curse of dimensionality. While additive models are computationally very competitive they also show good performance in practical applications. Like other regression models they provide good classification methods, especially if logistic regression is used.

If the probability distribution of the random variables $(X_1, \ldots, X_d, Y)$ which model the observations is known, the best (in the sense of expected squared error) approximation for the additive model is obtained when

$$f_i(x_i) = E\left(Y - f_0 - \sum_{\substack{k=1 \\ k \neq i}}^{d} f_k(X_k) \mid X_i = x_i\right) \tag{4.1}$$

and $f_0 = E(Y)$. These equations do not have a unique solution, but uniqueness can easily be obtained if we introduce constraints like $E(f_i(X_i)) = 0$.

In practical algorithms, the estimates are approximated by smooth functions. Let $\mathbf{f}_i$ be the vector of function values $\left(f_i(x_i^{(k)})\right)_{k=1}^{n}$. Furthermore, let S_i be the matrix representing the mapping between the data and the smooth $\mathbf{f}_i$. The matrix S_i depends on the observations $x_i^{(1)}, \ldots, x_i^{(n)}$. Replacement of the estimation operator by the matrix S_i in equation (4.1) leads to

$$\begin{bmatrix} I & S_1 & \cdots & S_1 \\ S_2 & I & \cdots & S_2 \\ \vdots & \vdots & & \vdots \\ S_d & S_d & \cdots & I \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_d \end{bmatrix} = \begin{bmatrix} S_1\mathbf{y} \\ S_2\mathbf{y} \\ \vdots \\ S_d\mathbf{y} \end{bmatrix}.$$

If the eigenvalues of the S_i are in $(0, 1)$ and this linear system of equations is nonsingular, it can be seen that the Gauss–Seidel algorithm for this system converges. This method of determining the additive model is the *backfitting algorithm* (Hastie and Tibshirani 1990): it has complexity $\mathcal{O}(Nqd)$, where q denotes the number of iteration steps. The backfitting algorithm is very general and, in fact, is used even for nonlinear smoothers. For very large datasets, however, the algorithm becomes very costly – even though it is scalable in the data size and does not suffer from the curse of dimensionality – because it needs to revisit the data qd times.

The high cost of the solution of the previous linear system of equations resulted from the large size of its matrix. Smaller alternative systems are available for particular cases of smoothers S_i. For example, in the case of regression splines we can work with the system of normal equations. Consider the case of fitting with piecewise multilinear functions. These functions include the ones used in MARS and are, on each subdomain, linear combinations of products $x_{j_1} \cdots x_{j_k}$ where every variable x_j occurs at most once.

The basis functions of the full space of piecewise multilinear functions are products of hat functions of the form $b_1(x_1) \cdots b_d(x_d)$. For the full space, the normal matrix of the least squares fitting problem becomes sparse with nonzeros on 3^d of the diagonals. But the matrix is huge, being of order m^d if each of the d dimensions is discretized by m hat functions. This shows the computational advantage of the additive models where the dimension of the approximating function space is only md.

4.4. Radial basis functions

In order to explore further the challenge posed by the curse of dimensionality, we will investigate radial basis function approximation. In recent years, radial basis functions have received a lot of attention both theoretically and in applications. One of their outstanding features is that they are able to approximate high-dimensional functions very effectively. Thus they seem to be able to overcome the curse of dimensionality. In the case of real attributes $x \in \mathbb{R}^d$ a radial basis function is of the form

$$f(x) = \sum_{i=1}^{N} c_i \, \rho(\|x - x^{(i)}\|) + p(x),$$

where $x^{(i)}$ are the data points. Examples of the function ρ include Gaussians $\rho(r) = \exp(-\alpha r^2)$, powers and thin plate splines $\rho(r) = r^\beta$ and $\rho(r) = r^\beta \ln(r)$ (for even integers β only), multiquadrics $\rho(r) = (r^2 + c^2)^{\beta/2}$ and others. The function $p(x)$ is typically a polynomial of low degree and in many cases it is zero. The radial basis function approach may be generalized to metric spaces where the argument of ρ is replaced by the distance $d(x, x_i)$. Reviews of radial basis function research can be found in Dyn (1989), Powell (1992), and Buhmann (1993). Existence, uniqueness and approximation properties have been well studied.

The evaluation of $f(x)$ requires the computation of the distances between x and all the data points $x^{(i)}$. Thus the time required to compute one function value is $\mathcal{O}(dN)$; the complexity is linear in the number of attributes d and the curse of dimensionality has been overcome. However, if many function values need to be evaluated, this is still very expensive. Fast methods for evaluation of radial basis functions have been studied in Beatson, Goodsell and Powell (1996), Beatson and Light (1997), Beatson and Newsam (1992), and Beatson and Powell (1994). For example, a multipole method that reduces the complexity to $\mathcal{O}((m + N)\log(N))$ has been suggested in Beatson and Newsam (1992) for the evaluation of $f(x)$ for m values of x. For data mining applications, which have very large N, even this is still too expensive. What is required is an approximation for which the evaluation is independent of the data size N and does not suffer from the curse of dimensionality. In the following, we will revisit the determination of the function

from the data points and see how the geometry of high-dimensional spaces influences the computational costs.

The vector of coefficients of the radial basis functions $\mathbf{c} = (c_1, \ldots, c_N)$ and the vector of coefficients of the polynomial term $\mathbf{d} = (d_1, \ldots, d_m)$ are determined, in the case of smoothing, by a linear system of equations of the form

$$\begin{bmatrix} A + \alpha I & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix}. \tag{4.2}$$

The matrix $A = \left[\rho(\|x^{(i)} - x^{(j)}\|)\right]_{i,j=1\ldots n}$ has almost no zero elements for the case of thin plate splines and has to be treated as a dense matrix. However, the influence of the data points is local and it is mainly the observed points close to x that influence the value of $f(x)$. This locality is shared with nearest neighbour approximation techniques. However, in higher dimensions, points get more sparse. For example, Friedman (1994) observes that the expected distance of the nearest neighbour in a d-dimensional hypercube grows like $\mathcal{O}\left(N^{-1/d}\right)$ with the dimension and that large numbers of data points are required to maintain a uniform coverage of the domain. In particular, a constant distance between a point and its nearest neighbour is obtained if $\log(N) = \mathcal{O}(d)$, that is, the number of points has to grow exponentially with the dimension. This is just another aspect of the curse of dimensionality.

If the function to be approximated is smooth enough, the number of points available may be sufficient even in high dimensions. But a computational difficulty appears which is related to the concentration of measure phenomenon (Talagrand 1996, Milman and Schechtman 1986, Ball 1997). The concentration of measure basically tells us that in high dimensions the neighbours of any point are concentrated close to a sphere around that point.

The effect of this on the computation is severe as the determination of a good approximant will require visiting a large number of neighbouring points for each evaluation point. In an attempt to decrease the computational work, a compactly supported radial basis function may be used. However, the support will have to be chosen such that, for a very large number of points, the values of the radial basis function $\rho(\|x^{(i)} - x^{(j)}\|)$ will be nonzero. Thus the linear system of equations (4.2) will have a substantial number of nonzeros which will ultimately render the solution computationally infeasible. This, however, does not mean that radial basis functions cannot be used in data mining. In fact, it has been found (Powell 1992) that the approximation order of radial basis functions increases with the dimension. Further, there are new iterative algorithms that can be used to solve (4.2), sometimes in $\mathcal{O}(n \log n)$ operations (Faul and Powell 1999). Therefore there is some evidence that radial basis functions do not suffer from the curse of dimensionality, although further work remains to be done.

5. Cluster analysis

Clustering refers to the task of grouping objects into classes of similar objects (Kaufman and Rousseeuw 1990). Cluster analysis is important in business, for the identification of customers with similar interests for which similar solutions may be found (market segmentation: see Berry and Linoff (1997, p. 47)), but also in science, where taxonomies of animals, stars or archaeological artefacts build the foundations for the understanding of evolution and development (Gordon 1981, p. 1). Cluster analysis has a long history both in statistical analysis and machine learning (Berry and Linoff 1997, Berson and Smith 1997, Gordon 1981, Hartigan 1975, Ripley 1996). Clusters of similar objects form one type of information discovered in data mining and can lead to new insights and suggest further actions. However, in order to be applicable for data mining, clustering algorithms need to have the following properties (see Han and Kamber (2001)):

- *scalable* in order to work on large and growing data sizes,
- capable of handling a large number of attributes of various types and complex data,
- capable of handling noisy data,
- insensitive to the ordering of the data, and
- able to represent clusters of arbitrary shapes.

The result of a cluster analysis is one or several subsets $\Omega' \subset \Omega$ of the set of all possible observations. Many clustering algorithms partition the domain Ω into (possibly disjoint) subdomains Ω_i such that

$$\Omega = \bigcup \Omega_i.$$

In data mining explorations, clustering is often combined with classification and rule detection in a 3-step procedure:

(1) determination of clusters,
(2) induction of a decision tree to predict the cluster label,
(3) extraction of rules relating to the clusters.

In order to find clusters, the dissimilarity or similarity of transactions or objects need to be assessed and most commonly, this is done with metrics on feature vectors:

- Minkowski distances (including Euclidean and Manhattan), or p-norms, for real-valued features, that is,

$$\rho(\omega_1, \omega_2) = \left(\sum_{i=1}^{d} |X_i(\omega_1) - X_i(\omega_2)|^q \right)^{1/q}$$

- simple matching for categorical (and binary) features, that is,

$$\rho(\omega_1, \omega_2) = |\{X_i : X_i(\omega_1) \neq X_i(\omega_2)\}| .$$

Often the distances are normalized or weighted differences are used. The four main types of clustering algorithm used in data mining are as follows.

Partitioning methods. If the number of clusters is known in advance, the data is partitioned into sets of similar data records. Starting with some initial partitioning (often random), data points are moved between clusters until the differences of objects within the clusters is small and the difference between elements of different clusters is large.

Hierarchical clustering methods. These methods provide a hierarchical partitioning of the data. Clusters are determined from the hierarchical partitioning.

There are two basic approaches. In the *agglomerative* approach an initial partitioning is given by first identifying each separate data point with one cluster, and then repeatedly joining clusters together to form successively larger clusters until only one cluster is left.

In the *divisive* approach all points are first assumed to be part of one big cluster. The partitions are then successively divided into smaller clusters until every point forms one separate cluster.

Density-based methods. Neighbouring elements are joined together using a local density measure. Often, only one scan through the database is required for this algorithm.

Model-based methods. Each cluster is modelled, for instance by a simple distribution function, and the best fit of these models to the data is determined.

Many clustering algorithms have an $\mathcal{O}(N^2)$ complexity, which results from the fact that the distances between all the points are computed. There are two major approximation techniques that address this computational problem: sampling (Kaufman and Rousseeuw 1990), where a subset of the data is selected to perform the algorithm, and binning, which is based on discretization of the domain (Wang, Yang and Muntz 1997). In both approaches the full dataset may be used for the evaluation of the result, particularly since this evaluation often only requires $\mathcal{O}(N)$ complexity. Both algorithms can lead to scalable algorithms, if, in the first case, only $\mathcal{O}(\sqrt{N})$ elements are chosen from the sample, and, in the second case, the bin sizes are not chosen to be too large and the dimension is low. Note that binning becomes less useful in higher dimensions, because it suffers from the curse of dimensionality.

5.1. Partitioning techniques

The aim of the clustering is to find subsets which are more pure than the original set in the sense that, on average, their elements are much more similar than the elements of the original domain. A partition $\Omega_1, \ldots, \Omega_k$ is represented by the *centroids* $\mathbf{z}_1, \ldots, \mathbf{z}_k$ such that

$$\mathbf{x} \in \Omega_i \iff \rho(\mathbf{x}, \mathbf{z}_i) \leq \rho(\mathbf{x}, \mathbf{z}_j), \quad i, j = 1, \ldots, k.$$

The centroids define an *impurity measure* of the form

$$J(\mathbf{z}_1, \ldots, \mathbf{z}_p) = \frac{1}{N} \sum_{i=1}^{k} \sum_{\mathbf{x}^{(j)} \in \Omega_i} \rho(\mathbf{x}^{(j)}, \mathbf{z}_i) \tag{5.1}$$

$$= \frac{1}{N} \sum_{j=1}^{N} \min_{1 \leq i \leq k} \rho(\mathbf{x}^{(j)}, \mathbf{z}_i). \tag{5.2}$$

For simplicity we identify ω with the feature vector $\mathbf{x}$ here.

The two types of algorithm for partitioning differ in the way they estimate the centroid. In the k-means algorithm, the mean of the (real-valued) observations in Ω_i is used:

$$\mathbf{z}_i = \frac{1}{N_i} \sum_{\mathbf{x}^{(j)} \in \Omega_i} \mathbf{x}^{(j)},$$

where N_i denotes the number of data points in Ω_i. One disadvantage of the k-means approach is that the mean cannot be guaranteed to be close to any data point at all and the data are limited to real vectors. An alternative is the k-medoid approach, where the centroid is chosen to be the most central element of the set, *i.e.*,

$$\mathbf{z}_i = \mathbf{x}^{(s_i)}$$

such that

$$\sum_{\mathbf{x}^{(j)} \in \Omega_i} \rho(\mathbf{x}^{(j)}, \mathbf{x}^{(s_i)}) \leq \sum_{\mathbf{x}^{(j)} \in \Omega_i} \rho(\mathbf{x}^{(j)}, \mathbf{x}^{(m)}), \quad \text{for all } \mathbf{x}^{(m)} \in \Omega_i.$$

k-means algorithm

The *k-means* algorithm (see Algorithm 3) was discussed in MacQueen (1967). It can be seen that the k-means algorithm cannot increase the function J, and, in fact, if any clusters are changed, J is reduced. As J is bounded from below it converges, and as a consequence the algorithm converges. It is also known that the k-means will always converge to a local minimum (Bottou and Bengio 1995). The k-means algorithm may be viewed as a variant of the EM algorithm (McLachlan and Krishnan 1996).

Algorithm 3 k-means algorithm

Select k arbitrary data points $\mathbf{z}_1, \ldots, \mathbf{z}_k$.
repeat
$\Omega_i := \{\mathbf{x}^{(j)} : \rho(\mathbf{x}^{(j)}, \mathbf{z}_i) \leq \rho(\mathbf{x}^{(j)}, \mathbf{z}_s), \quad s = 1, \ldots, p\ \}$
$\mathbf{z}_i := \frac{1}{|\Omega_i|} \sum_{\mathbf{x}^{(j)} \in \Omega_i} \mathbf{x}^{(j)}$.
until the $\mathbf{z}_i$ converge

Dhillon and Modha (2000) propose a parallel k-means algorithm. They also provide a careful analysis of the algorithm's computational complexity. There are two major steps in the algorithm: the determination of the distances between all the points, and the recalculation of the centroid. The determination of all the Euclidean distances between the N points and the k-cluster means requires $3Nkd$ floating point operations (one subtraction, square and one addition per component of all pairs). Finding the minimum for each point requires a total of kN comparisons; then we need to compute the new average for each cluster, which requires nd additions and kd divisions. In data mining the cost is usually dominated by the determination of all the distances, and thus the time is (Dhillon and Modha 2000)

$$T = \mathcal{O}\big(NkdI\big),$$

where I is the number of iterations.

For the parallel algorithm (shared nothing), the data are initially distributed over the discs of all the processors. Then each processor computes the distances of its elements to all cluster centres. This is done in parallel and so the most expensive computation gets a parallel speed-up of a factor of p (the number of processors). After the sums of all the elements are computed on all the processors, these sums are communicated, which requires an all-to-all communication with volume dk per iteration per processor. If this can be done in parallel, the time required is $\mathcal{O}(dkI\tau)$, where the time τ for reduction is typically of $\mathcal{O}\big(\sqrt{p}\big)$. Thus the total time is

$$\mathcal{O}\big(NkdI/p\big) + \mathcal{O}\big(dkI\sqrt{p}\big).$$

Now the communication time is small compared to the computation time for large problems, that is, if

$$N \gg p^{3/2}\kappa,$$

where κ is the ratio of communication time to computation time. Typically, p is in the order of 100 and κ around 1000, thus data sizes of a million do qualify. A scalable k-means method has been discussed in Bradley, Fayyad and Reina (1998).

k-medoids algorithm

Determining a set of elements $\mathbf{x}^{(j_s)}$ which minimize $J(\mathbf{x}^{(j_1)}, \ldots, \mathbf{x}^{(j_k)})$ defines a discrete optimization problem. The search graph of this problem has as nodes all possible sets of centroids, *i.e.*, all possible $\binom{N}{k}$ combinations of the points $\mathbf{x}^{(i)}$. The edges of the search graph join any two sets of centroids $\{\mathbf{x}^{(j_1)}, \ldots, \mathbf{x}^{(j_k)}\}$ and $\{\mathbf{x}^{(i_1)}, \ldots, \mathbf{x}^{(i_k)}\}$ if they only differ in one centroid. The k-medoids algorithm starts at a random node and moves to the adjacent node for which the reduction in impurity is maximal. This descent approach requires the determination of J for all adjacent nodes in order to determine the best direction. The algorithm finds a local minimum.

The algorithm stores all the distances between data points and all the centres. From this the minimal distance between any data point and the centres can be determined in $\mathcal{O}(k(N-k))$ comparisons. Thus the complexity of the algorithm is $\mathcal{O}(k^2(N-k)^2)$. However, for many data points the minimum can actually be found without comparing the distance between the new centroid and the data points and all the earlier computed distances. This is based on the following simple observation.

Lemma 12. If $\rho_1, \ldots, \rho_{k+1} \in \mathbb{R}$ satisfy $\rho_1 > \min(\rho_1, \ldots, \rho_k)$ or $\rho_{k+1} \leq \rho_1$, we have

$$\min(\rho_2, \ldots, \rho_{k+1}) = \min(\min(\rho_1, \ldots, \rho_k), \rho_{k+1}).$$

Proof. The conditions on ρ_1 imply that including ρ_1 does not change the minimum in both cases, that is,

$$\min(\rho_2, \ldots, \rho_{k+1}) = \min(\rho_1, \ldots, \rho_{k+1}). \qquad \square$$

Applying this to the k-medoid algorithm, we see that each term in the sum for J does have a minimum over all the centroids. If one centroid is replaced by another, say, centroid 1 is replaced by centroid $k+1$, then we have to compute the minimum $\min(\rho(x^{(i)}, z_2), \ldots, \rho(x^{(i)} z_{k+1}))$. This can be computed from the previously determined minimum over the original centroids using Lemma 12, except in the case, not covered by the lemma, for which the centroid is just the one that is replaced, and the distance between the new centroid and the data point is larger than the distance between the removed centroid and the data point. This algorithm is the PAM (partitioning around medoids) algorithm: see Kaufman and Rousseeuw (1990).

The complexity is further reduced by repeated clustering on a random sample of the data and always selecting the cluster with the lowest value of J. This is implemented in the CLARA (clustering large applications) algorithm and leads to an algorithm that is scalable in the number of data points N. However, there are still very many search directions and a random selection is used in the CLARANS (clustering large applications based on randomized

search) algorithm (Ng and Han 1994). This CLARANS algorithm is one of the early developments in clustering for data mining applications.

5.2. Hierarchical clustering

Hierarchical clustering algorithms transform a set of points with an associated dissimilarity metric into a tree structure known as a *dendrogram*. The nodes of the dendrogram correspond to sets of data points: for example, the leaves are single points. Like decision trees, dendrograms split the datasets at each node and the edges of the dendrogram correspond to set inclusions.

The two broad classes of hierarchical clustering algorithms are *agglomerative*, which start with single points and join them together whenever they are close, whereas *divisive* algorithms move from the top down and break the clusters up when they are dissimilar.

In contrast to decision trees, where the impurity of the nodes is determined through the value of a class attribute, in clustering the dissimilarity measure provides this measure of impurity.

An agglomerative hierarchical clustering algorithm consists of the following loop:

Initialize all points as single clusters
Determine all the dissimilarities between the clusters
while There is more than one cluster **do**
  Merge the two clusters with the smallest dissimilarity
  Update the dissimilarities
end while

It is important to assess the dissimilarity or impurity which is introduced through the joining of two partitions Ω_1 and Ω_2. The *single link* (or nearest neighbour) technique defines the dissimilarity as

$$\rho(A, B) = \min\{\rho(x, y) : x \in \Omega_1, y \in \Omega_2\}, \quad \text{for two partitions } \Omega_1 \text{ and } \Omega_2.$$

Other measures are used and they have a big effect on the computational load.

After the full generation of the dendrogram, it is pruned down to a desired level. The single link algorithm computes all the distances but does not require us to store them all. The classical SLINK (Sibson 1973) algorithm requires $\mathcal{O}(N^2)$ time and $\mathcal{O}(N)$ space.

In the case of a database distributed over p processors, Johnson and Kargupta (2000) present an algorithm for single-link clustering which has $\mathcal{O}(pN^2)$ time and $\mathcal{O}(pN)$ space complexity. The communication costs of the algorithm are $\mathcal{O}(N)$. The algorithm has the following three steps:

Apply the hierarchical clustering algorithm at each site.
Transmit the local dendrograms to the facilitator site.
Generate the global dendrogram.

The local dendrograms are communicated together with the distances of

each partition. Then an upper bound on the distance is obtained as the sum of the distances of the shortest path in each partition. This is an upper bound on the actual distance. We have

$$\rho(x_1, x_2) = \sqrt{\sum_{j=1}^{p} \left(\sum_{i \in P_j} (x_{1,i,j} - x_{i,2,j})^2 \right)} \leq \sqrt{\sum_{j=1}^{p} (\mathrm{dist}_{\mathrm{dendrogram}\ j}(x_1, x_2))}.$$

Basically, it uses the distance defined in the actual dendrograms. The merging algorithm is costly but the main advantage is that not all the data need to be communicated to one processor. For a parallel implementation of SLINK see Olson (1995).

There is a large variety of data mining clustering techniques including BIRCH (Zhang, Ramakrishnan and Livny 1996) and CURE (Guha, Rastogi and Shim 1998). A further class of clustering algorithms is based on density considerations, both using parametric models (Cheeseman and Stutz 1996) and non-parametric approaches in DBSCAN (Ester, Kriegel, Sander and Xu 1996).

Acknowledgement

The author would like to thank the editors and Brad Baxter (Imperial College, London), who carefully read the paper.

REFERENCES

R. Agrawal, T. Imielinski and T. Swami (1993), Mining association rules between sets of items in large databases, in *Proc. ACM–SIGMOD Conf. Management of Data*, ACM Press, pp. 207–216.

R. Agrawal and R. Srikant (1995), Mining sequential patterns, in *Proc. 11th Int. Conf. Data Engineering*, IEEE CS Press, Los Alamitos, CA, pp. 3–14.

S. Bakin, M. Hegland and M. Osborne (1998), Can MARS be improved with B-splines?, in *Computational Techniques and Applications: CTAC97* (B. J. Noye, M. D. Teubner and A. W. Gill, eds), World Scientific, pp. 75–82.

K. Ball (1997) 'An elementary introduction to modern convex geometry', in *Flavors of Geometry* (S. Levy, ed.), Cambridge University Press.

R. K. Beatson, G. Goodsell and M. J. D. Powell (1996), On multigrid techniques for thin plate spline interpolation in two dimensions, in *The Mathematics of Numerical Analysis* (Park City, UT, 1995), Vol. 32 of *Lectures in Appl. Math.*, American Mathematical Society, Providence, RI, pp. 77–97.

R. K. Beatson and W. A. Light (1997), Fast evaluation of radial basis functions: Methods for two-dimensional polyharmonic splines, *IMA J. Numer. Anal.*, **17** 343–372.

R. K. Beatson and G. N. Newsam (1992), Fast evaluation of radial basis functions, I, *Comput. Math. Appl.*, **24** 7–19.

R. K. Beatson and M. J. D. Powell (1994), An iterative method for thin plate spline interpolation that employs approximations to Lagrange functions, in *Numerical Analysis 1993*, Vol. 303 of *Pitman Res. Notes Math. Ser.*, Longman Sci. Tech., Harlow, pp. 17–39.

G. Bell and J. N. Gray (1997), The revolution yet to happen, in *Beyond Calculation* (P. J. Denning and R. M. Metcalfe, eds), Springer, pp. 5–32.

E. A. Bender (1996), *Mathematical Methods in Artificial Intelligence*, IEEE CS Press, Los Alamitos, CA.

M. J. A. Berry and G. Linoff (1997), *Data Mining Techniques: For Marketing, Sales and Customer Support*, Wiley.

A. Berson and S. J. Smith (1997), *Data Warehousing, Data Mining, and OLAP*, McGraw-Hill Series on Data Warehousing and Data Management, McGraw-Hill, New York.

H.-H. Bock and E. Diday, eds (2000), *Analysis of Symbolic Data*, Springer.

L. Bottou and Y. Bengio (1995), Convergence properties of the k-means algorithm, in *Adv. in Neural Info. Proc. Systems*, Vol. 7 (G. Tesauro and D. Touretzky, eds), MIT Press, Cambridge, MA, pp. 585–592.

P. Bradley, U. Fayyad and C. Reina (1998), Scaling clustering algorithms to large databases, in *Proc. 4th Int. Conf. KDD*, pp. 9–15.

L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone (1984), *Classification and Regression Trees*, Wadsworth International Group, Blemont, CA.

S. Brin *et al.* (1997), Dynamic itemset counting and implication rules for market basket data, in *Proc. ACM–SIGMOD Int. Conf. Management of Data*, ACM Press, New York, pp. 255–264.

M. D. Buhmann (1993), New developments in the theory of radial basis function interpolation, in *Multivariate Approximation: From CAGD to Wavelets* (Santiago, 1992), Vol. 3 of *Ser. Approx. Decompos.*, World Scientific, River Edge, NJ, pp. 35–75.

P. Cheeseman and J. Stutz (1996), Bayesian classification (Autoclass): Theory and results, in *Advances in Knowledge Discovery and Data Mining* (U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, eds), AAAI/MIT Press, Cambridge, MA, pp. 153–180.

M.-S. Chen, J. Han and P. S. Yu (1996), Data mining: An overview from a database perspective, *IEEE Trans. Knowledge and Data Engineering*, **8** 866–883.

S. Clearwater and F. Provost (1990), Rl4: A tool for knowledge-based induction.

W. W. Cohen (1995), Fast effective rule induction, in *Proc. 12th Int. Conf. Machine Learning*, Morgan Kaufmann, pp. 115–123.

C. J. Date (1995), *An Introduction to Database Systems*, Addison-Wesley, Reading, MA.

L. Devroye, L. Györfi and G. Lugosi (1996), *A Probabilistic Theory of Pattern Recognition*, Vol. 31 of *Applications of Mathematics*, Springer.

I. S. Dhillon and D. S. Modha (2000), A data-clustering algorithm on distributed memory multiprocessors, in *Large-Scale Parallel Data Mining* (M. J. Zaki and C.-T. Ho, eds), Springer, pp. 245–260.

A. J. Dobson (1990), *An Introduction to Generalized Linear Models*, Chapman and Hall, London. Second edn of *Introduction to Statistical Modelling*.

P. Domingos and M. Pazzani (1997), On the optimality of the simple Bayesian classifier under zero–one loss, *Machine Learning*, **29** 103–130.
G. Dong and J. Li (1999), Efficient mining of emerging patterns: Discovering trends and differences, in *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining* (KDD'99), ACM Press, pp. 43–52.
R. O. Duda and P. E. Hart (1973), *Pattern Classification and Scene Analysis*, Wiley, New York.
N. Dyn (1989), *Interpolation and Approximation by Radial and Related Functions*, Vol. 1, Academic Press, pp. 211–234.
M. Ester, H.-P. Kriegel, J. Sander and X. Xu (1996), A density-based algorithm for discovering clusters in large spatial databases, in *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining* (KDD'96), AAAI Press, pp. 226–231.
A. C. Faul and M. J. D. Powell (1999) Proof of convergence of an iterative technique for thin plate spline interpolation in two dimensions, *Adv. Comput. Math.* **11** 183–192.
J. H. Friedman (1991), Multivariate adaptive regression splines, *The Annals of Statistics*, **19** 1–141.
J. H. Friedman (1994), Flexible metric nearest neighbor classification, Technical Report, Department of Statistics, Stanford University.
N. Friedman, D. Geiger and M. Goldszmidt (1997), Bayesian network classifiers, *Machine Learning*, **29** 131–163.
G. Golub, M. Heath and G. Wahba (1979), Generalized cross validation as a method for choosing a good ridge parameter, *Technometrics*, **21** 215–224.
A. D. Gordon (1981), *Classification*, Vol. 82 of *Monographs on Statistics and Applied Probability*, Chapman and Hall, London.
S. Guha, R. Rastogi and K. Shim (1998), CURE: An efficient clustering algorithm for large databases, in *Proc. ACM–SIGMOD Int. Conf. Management of Data*, ACM Press, pp. 73–84.
L. Hall, N. Chawla and K. Bowyer (1998), Decision tree learning on very large data sets, in *Int. Conf. Systems, Man and Cybernetics*, IEEE Press, pp. 2579–2584,
L. O. Hall, N. Chawla, K. W. Bowyer and W. P. Kegelmayer (2000), Learning rules from distributed data, in *Large-Scale Parallel Data Mining* (M. J. Zaki and C.-T. Ho, eds), Springer, pp. 211–220.
J. Han and M. Kamber (2001), *Data Mining, Concepts and Techniques*, Morgan Kaufmann.
J. A. Hartigan (1975), *Clustering Algorithms*, Wiley Series in Probability and Mathematical Statistics, Wiley, New York/London/Sydney.
T. J. Hastie and R. J. Tibshirani (1990), *Generalized Additive Models*, Vol. 43 of *Monographs on Statistics and Applied Probability*, Chapman and Hall.
D. Heckerman, D. Geiger and D. E. Chickering (1995), Learning Bayesian networks, *Machine Learning*, **20** 197–243.
G. H. John and P. Langley (1995), Estimating continuous distributions in Bayesian classifiers, in *Proc. 11th Conference on Uncertainty in Artificial Intelligence* (P. Besnard and S. Hanks, eds), Morgan Kaufmann, pp. 338–345.
E. L. Johnson and H. Kargupta (2000), Collective, hierarchical clustering from distributed heterogeneous data, in *Large-Scale Parallel Data Mining* (M. J. Zaki and C.-T. Ho, eds), Springer, pp. 221–244.

L. Kaufman and P. J. Rousseeuw (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, New York.

P. Langley, W. Iba and K. Thompson (1992), An analysis of Bayesian classifiers, in *Proc. 10th Nat. Conf. Artificial Intelligence* (W. Swartout, ed.), AAAI Press, pp. 223–228.

P. Langley and S. Sage (1994), Induction of selective Bayesian classifiers, in *Proc. 10th Conf. Uncertainty in Artificial Intelligence* (R. L. Mantaras and D. Poole, eds), Morgan Kaufmann, pp. 399–406.

P. Langley and S. Sage (1999), Tractable average-case analysis of naive Bayesian classifiers, in *Proc. 16th Int. Conf. Machine Learning*, Bled, Slovenia, Morgan Kaufmann, pp. 220–228,

B. Lent, A. Swami and J. Widom (1997), Clustering association rules, in *Proc. 1997 Int. Conf. Data Engineering* (ICDE'97), IEEE CS Press, pp. 220–231.

J. Li, G. Dong and K. Ramamohanrarao (2000), Making use of the most expressive jumping emerging patterns for classification, in *Proc. 2000 Pacific–Asia Conf. Knowledge Discovery and Data Mining* (PAKDD'00), Springer, pp. 220–232.

B. Liu, W. Hsu and Y. Ma (1998), Integrating classification and association rule mining, in *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining* (KDD'98), AAAI Press, pp. 80–86.

J. MacQueen (1967), Some methods for classification and analysis of multivariate observations, in *Proc. 5th Berkeley Sympos. Math. Statist. and Probability* (1965/66), Vol. I: *Statistics*, University of California Press, Berkeley, CA, pp. 281–297.

G. J. McLachlan and T. Krishnan (1996), *The EM Algorithm and Extensions*, Wiley.

H. Mannila, H. Toivonen and A. I. Verkamo (1995), Discovering frequent episodes in sequences, in *Proc. 1st Int. Conf. Knowledge Discovery Databases and Data Mining* (KDD'95), AAAI Press, pp. 210–215.

H. Mannila and H. Toivonen (1996), Discovering generalized episodes using minimal occurrences, in *Proc. 2nd Int. Conf. Knowledge Discovery Databases and Data Mining* (KDD'96), AAAI Press, pp. 146–151.

D. Meretakis and B. Wüthrich (1999), Extending naive Bayes classifiers using long itemsets, in *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining* (KDD'99), ACM Press, pp. 165–174.

V. D. Milman and G. Schechtman (1986) *Asymptotic Theory of Finite Dimensional Normed Spaces*, Vol. 1200 of *Lecture Notes in Mathematics*, Springer.

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

S. Morishita (1998), On classification and regression, in *Proc. 1st Int. Conf. Discovery Science*, Vol. 1532, Springer, pp. 40–57.

S. Morishita and A. Nakaya (2000), Parallel branch-and-bound graph search for correlated association rules, in *Large-Scale Parallel Data Mining* (M. J. Zaki and C.-T. Ho, eds), Springer, pp. 127–144.

R. Ng and J. Han (1994), Efficient and effective clustering methods for spatial data mining, in *Proc. 20th Int. Conf. Very Large Data Bases*, Morgan Kaufmann, pp. 144–155.

T. Oates, M. D. Schmill, D. Jensen and P. R. Cohen (1997), A family of algorithms

for finding temporal structure in data, in *Preliminary Papers of the 6th Int. Workshop on AI and Statistics*, Society for Artificial Intelligence and Statistics, pp. 371–378.

C. Olson (1995), Parallel algorithms for hierarchical clustering, *Parallel Computing*, **8** 1313–1325.

J. S. Park, M. S. Chen and P. S. Yu (1995*a*), An effective hash-based algorithm for mining association rules, in *Proc. 1995 ACM–SIGMOD Int. Conf. Management of Data* (SIGMOD'95), ACM Press, pp. 175–186.

J. S. Park, M. S. Chen and P. S. Yu (1995*b*), Efficient parallel data mining for association rules, in *Proc. 4th Int. Conf. Information and Knowledge Management*, ACM Press, pp. 31–36.

M. J. D. Powell (1992), The theory of radial basis function approximation in 1990, in *Advances in Numerical Analysis, Vol. II* (Lancaster, 1990), Oxford Sci. Publ., Oxford University Press, New York, pp. 105–210.

F. Provost, J. Aronis and B. Buchanan (2000), Rule-space search for knowledge-based discovery, CIIO Working Paper Number #IS 99-0012, Stern School of Business, New York University, NY. Available from:
`http://www.stern.nyu.edu/ fprovost/`

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

J. R. Quinlan (1987), Generating production rules from decision trees, in *Proc. 10th Int. Conf. Artificial Intelligence* (IJCAI-87), Morgan Kaufmann, pp. 304–307.

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

N. Ramakrishnan and A. Y. Grama (1999), Data mining: From serendipity to science, *Computer*, **32**(8) 34–37.

S. Ramaswamy, S. Mahajan and A. Silbershatz (1998), On the discovery of interesting patterns in association rules, in *Proc. 24th Int. Conf. Very Large Data Bases*, Morgan Kaufmann, pp. 368–379.

B. D. Ripley (1996), *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge.

A. Savasere, E. Omieski and S. Navanthe (1995), An efficient algorithm for mining association rules in large databases, in *Proc. 21st Int. Conf. Very Large Data Bases*, Morgan Kaufmann, pp. 432–444.

R. Segal and O. Etioni (1994), Learning decision lists using homogeneous rules, in *Proc. 12th Nat. Conf. Artificial Intelligence*, AAAI Press, pp. 619–625.

J. Shafer, R. Agrawal and M. Mehta (1996), SPRINT: A scalable parallel classifier for data mining, in *Proc. 1996 Int. Conf. Very Large Data Bases* (VLDB'96), Morgan Kaufmann, pp. 544–555.

T. Shintani and M. Kitsuregawa (2000), Parallel generalized association rule mining on large scale PC clusters, in *Large-Scale Parallel Data Mining* (M. J. Zaki and C.-T. Ho, eds), Springer, pp. 145–160.

R. Sibson (1973), SLINK: An optimally efficient algorithm for the single-link cluster method, *The Computer Journal*, **16** 30–34.

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

R. Srikant and R. Agrawal (1995), Mining generalized association rules, in *Proc. 1995 Int. Conf. Very Large Data Bases* (VLDB'95), pp. 407–419.

R. Srikant and R. Agrawal (1996*a*), Mining quantitative association rules in large relational tables, in *Proc. 1996 ACM–SIGMOD Int. Conf. Management of Data* (SIGMOD'96), ACM Press, pp. 1–12.

R. Srikant and R. Agrawal (1996*b*), Mining sequential patterns: Generalizations and performance improvements, in *5th Int. Conf. Extending Database Technology*, Vol. 1057 of *Lecture Notes in Computer Science*, Springer, pp. 3–17.

M. Talagrand (1996), A new look at independence, *Ann. Prob.*, **23** 1–37.

H. Toivonen (1996), Sampling large databases for association rules, in *Proc. 22nd Int. Conf. Very Large Data Bases*, Morgan Kaufmann, pp. 134–145.

W. Wang, J. Yang and R. Muntz (1997), STING: A statistical information grid approach to spatial data mining, in *Proc. 1997 Int. Conf. Very Large Data Bases* (VLDB'97), Morgan Kaufmann, pp. 186–195.

G. Webb (2000), Efficient search for association rules, in *Proc. of the 6th ACM–SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, ACM Press, pp. 99–107.

G. Williams (1990), Inducing and combining multiple decision trees, PhD thesis, Australian National University.

M. J. Zaki (1998), Efficient enumeration of frequent sequences, in *7th Int. Conf. Information and Knowledge Management*, ACM Press, pp. 68–75.

M. J. Zaki (2000), Parallel sequence mining on shared-memory machines, in *Large-Scale Parallel Data Mining* (M. J. Zaki and C.-T. Ho, eds), Springer, pp. 161–189.

M. J. Zaki and C.-T. Ho, eds (2000), *Large-Scale Parallel Data Mining*, Vol. 1759 of *Lecture Notes in Artificial Intelligence*, Springer.

T. Zhang, R. Ramakrishnan and M. Livny (1996), BIRCH: An efficient data clustering method for very large databases, in *Proc. 1996 ACM–SIGMOD Int. Conf. Management of Data* (SIGMOD'96), ACM Press, pp. 103–114.