DATA MINING TECHNIQUES

Mohammed J. Zaki
Department of Computer Science, Rensselaer Polytechnic Institute
Troy, New York 12180-3590, USA
E-mail: zaki@cs.rpi.edu

Limsoon Wong
Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore 119613
E-mail: limsoon@i2r.a-star.edu.sg

Data mining is the semi-automatic discovery of patterns, associations, changes, anomalies, and statistically significant structures and events in data. Traditional data analysis is assumption driven in the sense that a hypothesis is formed and validated against the data. Data mining, in contrast, is data driven in the sense that patterns are automatically extracted from data. The goal of this tutorial is to provide an introduction to data mining techniques. The focus will be on methods appropriate for mining massive datasets using techniques from scalable and high performance computing. The techniques covered include association rules, sequence mining, decision tree classification, and clustering. Some aspects of preprocessing and postprocessing are also covered. The problem of predicting contact maps for protein sequences is used as a detailed case study.

The material presented here is compiled by LW based on the original tutorial slides of MJZ at the 2002 Post-Genome Knowledge Discovery Programme in Singapore.

Keywords: Data mining; association rules; sequence mining; decision tree classification; clustering; massive datasets; discovery of patterns; contact maps.

Organization:

1. Data Mining Overview

Data mining is generally an iterative and interactive discovery process. The goal of this process is to mine patterns, associations, changes, anomalies, and statistically significant structures from large amount of data. Furthermore, the mined results should be valid, novel, useful, and understandable. These "qualities" that are placed on the process and outcome of data mining are important for a number of reasons, and can be described as follows:

(1) Valid: It is crucial that the patterns, rules, and models that are discovered are valid not only in the data samples already examined, but are generalizable and remain valid in future new data samples. Only then can the rules and models obtained be considered meaningful.
(2) Novel: It is desirable that the patterns, rules, and models that are discovered are not already known to experts. Otherwise, they would yield very little new understanding of the data samples and the problem at hand.
(3) Useful: It is desirable that the patterns, rules, and models that are discovered allow us to take some useful action. For example, they allow us to make reliable predictions on future events.
(4) Understandable: It is desirable that the patterns, rules, and models that are discovered lead to new insight on the data samples and the problem being analyzed.

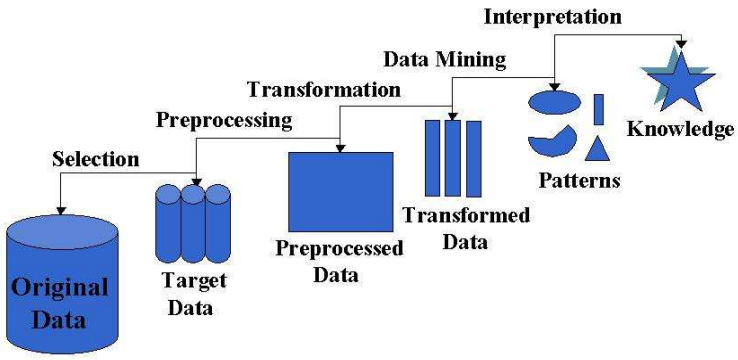

Fig. 1. The data mining process.

In fact, the goals of data mining are often that of achieving reliable prediction and/or that of achieving understandable description. The former answers the question "what", while the latter the question "why". With respect to the goal of reliable prediction, the key criteria is that of accuracy of the model in making predictions on the problem being analyzed. How the prediction decision is arrived at may not be important. With respect to the goal of understandable description, they key criteria is that of clarity and simplicity of the model describing the problem being analyzed.

There is sometimes a dichotomy between these two aspects of data mining in the sense that the most accurate prediction model for a problem may not be easily understandable, and the most easily understandable model may not be highly accurate in its predictions. For example, on many analysis and prediction problems, support vector machines are reported to hold world records in accuracy [22]. However, the maximum error margin models constructed by these machines and the quadratic programming solution process of these machines are not readily understood to the non-specialists. In contrast, the decision trees constructed by tree induction classifiers such as C4.5 [64] are readily grasped by non-specialists, even though these decision trees do not always give the most accurate predictions.

The general data mining process is depicted in Figure 1. It comprises the following steps [1, 36, 80], some of which are optional depending on the problem being analyzed:

(1) Understand the application domain: A proper understanding of the application domain is necessary to appreciate the data mining outcomes

desired by the user. It is also important to assimilate and take advantage of available prior knowledge to maximize the chance of success.

(2) Collect and create the target dataset: Data mining relies on the availability of suitable data that reflects the underlying diversity, order, and structure of the problem being analyzed. Therefore, the collection of a dataset that captures all the possible situations that are relevant to the problem being analyzed is crucial.

(3) Clean and transform the target dataset: Raw data contain many errors and inconsistencies, such as noise, outliers, and missing values. An important element of this process is the de-duplication of data records to produce a non-redundant dataset. For example, in collecting information from public sequence databases for the prediction of protein translation initiation sites [60], the same sequence may be recorded multiple times in the public sequence databases; and in collecting information from scientific literature for the prediction of MHC-peptide binding [40], the same MHC-binding peptide information may be reported in two separate papers. Another important element of this process is the normalization of data records to deal with the kind of pollution caused by the lack of domain consistency. This type of pollution is particularly damaging because it is hard to trace. For example, MHC-binding peptide information reported in a paper might be wrong due to a variety of experimental factors. In fact, a detailed study [72] of swine MHC sequences found that out of the 163 records examined, there were 36 critical mistakes. Similarly, clinical records from different hospitals may use different terminologies, different measures, capture information in different forms, or use different default values to fill in the blanks. As a last example, due to technology limitations, gene expression data produced by microarray experiments often contain missing values and these need to be dealt with properly [78].

(4) Select features, reduce dimensions: Even after the data have been cleaned up in terms of eliminating duplicates, inconsistencies, missing values, and so on, there may still be noise that is irrelevant to the problem being analyzed. These noise attributes may confuse subsequent data mining steps, produce irrelevant rules and associations, and increase computational cost. It is therefore wise to perform a dimension reduction or feature selection step to separate those attributes that are pertinent from those that are irrelevant. This step is typically achieved using statistical or heuristic techniques such as Fisher criterion [29], Wilcoxon rank sum test [70], principal component analysis [42], en-

tropy analysis [28], *etc.*

(5) Apply data mining algorithms: Now we are ready to apply appropriate data mining algorithms—association rules discovery, sequence mining, classification tree induction, clustering, and so on—to analyze the data. Some of these algorithms are presented in later sections.
(6) Interpret, evaluate, and visualize patterns: After the algorithms above have produced their output, it is still necessary to examine the output in order to interpret and evaluate the extracted patterns, rules, and models. It is only by this interpretation and evaluation process that we can derive new insights on the problem being analyzed.

As outlined above, the data mining endeavor involves many steps. Furthermore, these steps require technologies from other fields. In particular, methods and ideas from machine learning, statistics, database systems, data warehousing, high performance computing, and visualization all have important roles to play. In this tutorial, we discuss primarily data mining techniques relevant to Step (5) above.

There are several categories of data mining problems for the purpose of prediction and/or for description [1, 36, 80]. Let us briefly describe the main categories:

(1) Association Rules: Given a database of transactions, where each transaction consists of a set of items, association discovery finds all the item sets that frequently occur together, and also the rules among them. An example of an association could be that, 90% of the people who buy cookies, also buy milk (60% of all grocery shoppers buy both).
(2) Sequence mining (categorical): The sequence mining task is to discover sequences of events that commonly occur together, e.g., in a set of DNA sequences ACGTC is followed by GTCA after a gap of 9, with 30% probability.
(3) Similarity search: An example is the problem where we are given a database of objects and a "query" object, and we are then required to find those objects in the database that are similar to, i.e., within a user-defined distance of, the query object. Another example is the problem where we are given a database of objects, and we are then required to find all pairs of objects in the databases that are within some distance of each other.
(4) Deviation detection: An example is the problem of finding outliers. That is, given a database of objects, we are required to find those objects that are the most different from the other objects in the database. These

objects may be thrown away as noise, or they may be the "interesting" ones, depending on the specific application scenario.

(5) Classification and regression: This is also called supervised learning. In the case of classification, we are given a database of objects that are labeled with predefined categories or classes. We are required to learning from these objects a model that separates them into the predefined categories or classes. Then, given a new object, we apply the learned model to assign this new object to one of the classes. In the more general situation of regression, instead of predicting classes, we have to predict real-valued fields.

(6) Clustering: This is also called unsupervised learning. Here, we are given a database of objects that are usually without any predefined categories or classes. We are required to partition the objects into subsets or groups such that elements of a group share a common set of properties. Moreover the partition should be such that the similarity between members of the same group is high and the similarity between members of different groups is low.

Some of the research challenges for data mining from the perspectives of scientific and engineering applications [33] are issues such as:

(1) Scalability. How does a data mining algorithm perform if the dataset has increased in volume and in dimensions? This may call for some innovations based on efficient and sufficient sampling, or a trade-off between in-memory vs. disk-based processing, or an approach based on high performance distributed or parallel computing.

(2) Automation. While a data mining algorithm and its output may be readily handled by a computer scientist, it is important to realize that the ultimate user is often not the developer. In order for a data mining tool to be directly usable by the ultimate user, issues of automation—especially in the sense of ease of use—must be addressed. Even for the computer scientist, the use and incorporation of prior knowledge into a data mining algorithm is often a tricky challenge; (s)he too would appreciate if data mining algorithms can be modularized in a way that facilitate the exploitation of prior knowledge.

2. Data Mining Techniques

We now review some of the commonly used data mining techniques for the main categories of data mining problems. We touch on the following in

sequence: association rules in Subsection 2.2., sequence mining in Subsection 2.3., classification in Subsection 2.4., clustering in Subsection 2.5., and k-nearest neighbors in Subsection 2.6.

2.1. *Terminology*

In the tradition of data mining algorithms, the data being analyzed are typically represented as a table, where each row of the table is a data sample and each column is an attribute of the data. Hence, given such a table, the value stored in jth column of the ith row is the value of the jth attribute of the ith data sample. An attribute is an item that describes an aspect of the data sample—*e.g.*, name, sex, age, disease state, and so on.

The term "feature" is often used interchangeably with "attribute". Some time the term "dimension" is also used. A feature f and its value v in a sample are often referred to as an "item". A set of items is then called an "itemset" and can be written in notation like $\{f_1 = v_1, ..., f_n = v_n\}$ for an itemset containing features $f_1, ..., f_n$ and associated values $v_1, ..., v_n$. Given such an itemset x, we denote by $[x]_{f_i}$ the value of its feature f_i. An itemset can also be represented as a vector $\langle v_1, ..., v_n \rangle$, with the understanding that the value of the feature f_i is kept in the ith position of the vector. Such a vector is usually called a feature vector. Given such a feature vector x, we write $[x]_i$ for the value in its ith position. An itemset containing k items is called a k-itemset. The number k is the "length" or "cardinality" of the itemset.

It is also a convention to write an itemset as $\{f'_1, ..., f'_m\}$, if all the features are Boolean—*i.e.*, either 1 or 0—and $\{f'_1, ..., f'_m\} = \{f_i \mid v_i = 1, 1 \leq i \leq n\}$. Under this convention, the itemset is also called a "transaction". Note that transactions contain only those items whose feature values are 1 and not those whose values are 0.

2.2. *Association Rule Mining*

We say a transaction T contains an item x if $x \in T$. We also say an itemset X occurs in a transaction T if $X \subseteq T$. Let a dataset D of transactions and an itemset X be given. We denote the dataset cardinality by $|D|$. The count of X in D, denoted $count^D(X)$, is the number of transactions in D that contains X. The support of X in D, denoted $support^D(X)$, is the percentage of transactions in D that contain X. That is,

$$support^D(X) = \frac{|\{T \in D \mid X \subseteq T\}|}{|D|}$$

An association rule is pair that we write as $X \Rightarrow Y$, where X and Y are two itemsets and $X \cap Y = \emptyset$. The itemset X is called the antecedent of the rule. The itemset Y is called the consequent of the rule.

There are two important properties associated with rules. The first property is the support of the rule. The second property is the confidence of the rule. We define them below.

Definition 1: The support of the rule $X \Rightarrow Y$ in a dataset D is defined as the percentage of transactions in D that contain $X \cup Y$. That is,

$$support^D(X \Rightarrow Y) = support^D(X \cup Y)$$

Definition 2: The confidence of the rule $X \Rightarrow Y$ in a dataset D is defined as the percentage of transactions in D containing X that also contain Y. That is,

$$confidence^D(X \Rightarrow Y) = \frac{support^D(X \cup Y)}{support^D(X)} = \frac{count^D(X \cup Y)}{count^D(X)}$$

We are now ready to define the objective of data mining for association rules. Association rule mining [3] is: Given dataset D of objects and thresholds *minsupp* and *minconf*, find every rule $X \Rightarrow Y$ so that $support^D(X \Rightarrow Y) \geq minsupp$ and $confidence^D(X \Rightarrow Y) \geq minconf$. An association rule $X \Rightarrow Y$ can be interpreted as "if a transaction contains X, then it is also likely to contain Y." The thresholds *minsupp* and *minconf* are parameters that are specified by a user to indicate what sort of rules are "interesting".

Given the threshold *minsupp*, an itemset X is said to be frequent in a dataset D if $support^D(X) \geq minsupp$. Furthermore, a frequent itemset X is said to be maximal in a dataset D if none of its proper supersets is frequent. Clearly all subsets of a maximal frequent itemset are frequent. Also a frequent itemset X is said to be closed if none of its supersets has the same frequency.

Figure 2 shows an example of mining frequent itemsets. Here we have a database of 5 transactions and the items bought in each. The table on the right hand shows the itemsets that are frequent at a given level of support. With a threshold of $minsupp = 50\%$ all of the itemsets shown are frequent, with ACTW, and CDW as the maximal frequent itemsets.

An obvious approach to finding all association rules in a dataset satisfying the thresholds *minsupp* and *minconf* is the following:

(1) Generate all frequent itemsets. These itemsets satisfy *minsupp*.

OID	Items
1	A C T W
2	C D W
3	A C T W
4	A C D W
5	A C D T W
6	C D T

Maximal Frequent Itemsets
ACTW, CDW

Support	Itemsets
100%(6)	C
83% (5)	CW
67% (4)	A, D, T, AC, AW, CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW, CDW, CTW, ACTW

Fig. 2. An example of frequent itemsets, where we use a threshold of $minsupp \geq 50\%$.

(2) Generate rules from these frequent itemsets and eliminate those rules that do not satisfy *minconf*.

The first step above generally requires searching an exponential space with respect to the length of itemsets, and is thus computationally expensive and I/O intensive. Therefore, a key challenge in association rules mining is a solution to the first step in the process above. This problem has been studied by the data mining community quite intensively. One of the first efficient approach was the Apriori algorithm [3], which inspired the development of many other efficient algorithms[14, 35, 50, 58, 71, 73, 81]. Another promising direction was the development of methods to mine maximal [9, 15, 32, 49] and closed itemsets [59, 61, 83].

The Apriori algorithm [3] achieves its efficiency by exploiting the fact that if an itemset is known to be not frequent, than all its supersets are also not frequent. Thus it generates frequent itemsets in a level-wise manner. Let us denote the set of frequent itemsets produced at level k by L_k. To produce frequent itemset candidates of length $k+1$, it is only necessary to "join" the frequent itemsets in L_k with each other, as opposed to trying all possible candidates of length $k+1$. This join is defined as $\{i \mid i_1 \in L_k, i_2 \in L_k, i \subseteq (i_i \cup\ i_2), |i| = k+1, (\nexists i' \subset i, (|i'| = k) \wedge (i' \notin L_k))\}$. The support of each candidate can then be computed by scanning the dataset to confirm if the candidate is frequent or not.

The second step of the association rules mining process is relatively cheaper to compute. Given the frequent itemsets, one can form a frequent itemset lattice as shown in Figure 3. Each node of the lattice is a unique

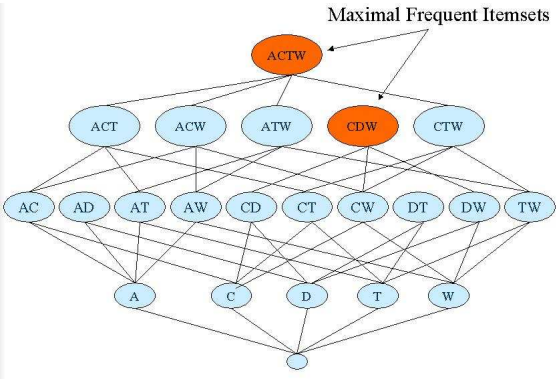

Fig. 3. An example of a frequent itemset lattice, based on the maximal frequent itemsets from Figure 2.

frequent itemset, whose support has been mined from the database. There is an edge between two nodes provided they share a direct subset-superset relationship. After this, for each node X derived from an itemset $X \cup Y$, we can generate a candidate rule $X \Rightarrow Y$, and test its confidence.

As an example, consider Figures 2 and 3. For the maximal itemset $\{CDW\}$, we have:

- $count^D(CDW) = 3$,
- $count^D(CD) = 4$,
- $count^D(CW) = 5$,
- $count^D(DW) = 3$,
- $count^D(C) = 6$,
- $count^D(D) = 4$, and
- $count^D(W) = 5$.

For each of the above subset counts, we can generate a rule and compute its confidence:

- $confidence^D(CD \Rightarrow W) = 3/4 = 75\%$,
- $confidence^D(CW \Rightarrow D) = 3/5 = 60\%$,
- $confidence^D(DW \Rightarrow C) = 3/3 = 100\%$,
- $confidence^D(C \Rightarrow DW) = 3/6 = 50\%$,
- $confidence^D(D \Rightarrow CW) = 3/4 = 75\%$, and
- $confidence^D(W \Rightarrow CD) = 3/5 = 60\%$.

Then those rules satisfying *minconf* can be easily selected.

	$Y \subseteq T$	$Y \not\subseteq T$
$X \subseteq T$	TP	FP
$X \not\subseteq T$	FN	TN

Fig. 4. Contingency table for a rule $X \Rightarrow Y$ with respect to a data sample T. According to the table, if X is observed and Y is also observed, then it is a true positive prediction (TP); if X is observed and Y is not, then it is a false positive (FP); if X is not observed and Y is also not observed, then it is a true negative (TN); and if X is not observed but Y is observed, then it is a false negative (FN).

Recall that support and confidence are two properties for determining if a rule is interesting. As shown above, these two properties of rules are relatively convenient to work with. However, these are heuristics and hence may not indicate whether a rule is really interesting for a particular application. In particular, the setting of *minsupp* and *minconf* is *ad hoc*. For different applications, there are different additional ways to assess when a rule is interesting. Other approaches to the interestingness of rules include rule templates [44], which limits rules to only those fitting a template; minimal rule cover [77], which eliminates rules already implied by other rules; and "unexpectedness" [51, 74].

As mentioned earlier, a rule $X \Rightarrow Y$ can be interpreted as "if X is observed in a data sample T, then Y is also likely to be observed in T." If we think of it as a prediction rule, then we obtain the contingency table in Figure 4.

With the contingency table of Figure 4 in mind, an alternative interestingness measure is that of odds ratio, which is a classical measure of unexpectedness commonly used in linkage disequilibrium analysis. It is defined as

$$\theta^D(X \Rightarrow Y) = \frac{TP^D(X \Rightarrow Y) * TN^D(X \Rightarrow Y)}{FP^D(X \Rightarrow Y) * FN^D(X \Rightarrow Y)}$$

where $TP^D(X \Rightarrow Y)$ is the number of data sample $T \in D$ for which the rule $X \Rightarrow Y$ is a true positive prediction, $TN^D(X \Rightarrow Y)$ is the number of data sample $T \in D$ for which the rule $X \Rightarrow Y$ is a true negative prediction, $FP^D(X \Rightarrow Y)$ is the number of data sample $T \in D$ for which the rule $X \Rightarrow Y$ is a false positive prediction, and $FN^D(X \Rightarrow Y)$ is the number of data sample $T \in D$ for which the rule $X \Rightarrow Y$ is a false negative prediction. The value of the odds ratio $\theta^D(X \Rightarrow Y)$ varies from 0 to infinity. When $\theta(X \Rightarrow Y) << 1$, then X and Y are indeed associated and the rule may be of interest.

CID	Time	Items
1	10	A B
1	20	B
1	30	A B
2	20	A C
2	30	A B C
2	50	B
3	10	A
3	30	B
3	50	A
4	30	A B
4	40	A
4	50	B

Support	Sequences
100%	A
	B
	A→A
	A→B
75%	AB
	B→A
	B→B
	AB→B

Maximal Frequent Sequences

A→A B→A AB→B

Fig. 5. An example of maximal frequent sequences with 75% support.

2.3. *Sequence Mining*

Sequence mining is a data mining problem closely related to that of association rules mining. The key difference between sequence mining and association rules mining is in the input dataset. In the case of association rules, each row in the input table represent a single data sample. That is, each row is the entire transaction. In the case of sequence mining, the situation is more complicated. Here, a data sample—called a sequence—is split across multiple consecutive rows in the input table. Each row represents just one event of the sequence identified with a special identifier attribute. Each event is itself a transaction.

An example is shown in Figure 5. In the example, each sequence is identified by an identifier, recorded as the attribute `CID` (for customer ID); each event is identified by an identifier, recorded as the attribute `Time`; and the transaction associated with the event is recorded as a list of items, collectively recorded as the attribute `Items`. The table on the right shows the frequent sequences at different levels of *minsupp*, as well as the maximal sequences at the 75% *minsupp* level. As in association mining, these frequent sequences can be organized into a lattice as shown in Figure 6.

An example of applying sequence mining analysis to DNA sequences is As the sequence and event identifiers are typically unimportant, we write $i_1 \rightarrow \cdots \rightarrow i_n$ for a sequence in which the transactions i_1, ..., i_n—which are itemsets—occur in the same order. Naturally, $\cdot \rightarrow \cdot$ is to be viewed as an associative operation. We say that:

Definition 3: A sequence $a_1 \rightarrow \cdots \rightarrow a_n$ is contained in a sequence $b_1 \rightarrow \cdots \rightarrow b_m$ if there is a mapping $\varphi : \{1, ..., n\} \mapsto \{1, ..., m\}$ such that (1) for

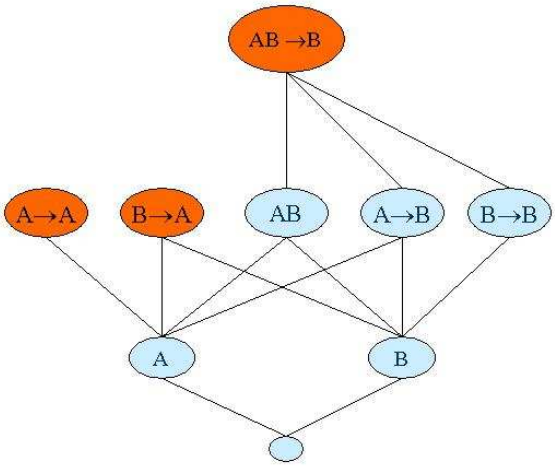

Fig. 6. An example of a frequent sequence lattice.

$1 \leq i \leq n$, $a_i \subseteq b_{\varphi(i)}$; and (2) for $1 \leq i \leq j \leq n$, $\varphi(i) \leq \varphi(j)$. A sequence is maximal if it is not contained in other sequences. We write $s_1 \subseteq s_2$ if the sequence s_1 is contained in the sequence s_2.

Note in particular that, according to this definition, the sequence $\{3\} \rightarrow \{5\}$ is not contained in the sequence $\{3, 5\}$ and vice versa.

The notion of support is extended to sequences as follows:

Definition 4: The support of a sequence s in a dataset of sequences D is the percentage of sequences in D that contain s. That is,

$$support^D(s) = \frac{|\{s' \in D \mid s \subseteq s'\}|}{|D|}$$

A sequence $i_1 \rightarrow \cdots \rightarrow i_n$ can generate $n-1$ rules of the form $X \Rightarrow Y$, *viz.* $i_1 \Rightarrow i_2 \rightarrow \cdots \rightarrow i_n$, $i_1 \rightarrow i_2 \Rightarrow i_3 \rightarrow \cdots \rightarrow i_n$, ..., and $i_1 \rightarrow \cdots \rightarrow i_{n-1} \Rightarrow i_n$. The notions of support and confidence on rules can then be defined in a manner analogous to Subsection 2.2. as follows.

Definition 5: The support of the rule $X \Rightarrow Y$ in a dataset D of sequences is defined as the percentage of sequences in D that contain $X \rightarrow Y$. That is,

$$support^D(X \Rightarrow Y) = \frac{|\{s' \in D \mid X \rightarrow Y \subseteq s'\}|}{|D|}$$

Definition 6: The confidence of the rule $X \Rightarrow Y$ in a dataset D of sequences is defined as the percentage of sequences in D containing X that also contain $X \rightarrow Y$. That is,

$$confidence^D(X \Rightarrow Y) = \frac{support^D(X \rightarrow Y)}{support^D(X)} = \frac{|\{s' \in D \mid X \rightarrow Y \subseteq s'\}|}{|\{s' \in D \mid X \subseteq s'\}|}$$

The goal of sequence mining [4, 75] is, given a dataset D of sequences, (1) find every sequence s so that its support in D satisfies the user-specified minimum support *minsupp*; and (2) find every rule $X \Rightarrow Y$ derived from a sequence so that its support and confidence satisfy the user-specified minimum support *minsupp* and minimum confidence *minconf*. Also, a sequence whose support satisfies *minsupp* is called a frequent sequence.

As in the closely related problem of finding association rules from transactions described in Subsection 2.2., finding frequent sequences is a computationally expensive task, while deriving rules from these sequences is a relatively cheaper task. A number of efficient algorithms for sequence mining have been proposed in the literature [6, 54, 62, 75, 82], but for purposes of exposition we focus on the original algorithm AprioriAll [4], which is similar to Apriori [3] (with a little bit of preprocessing of the input data samples). Let us outline this algorithm, assuming a dataset of sequences $S = \{s_1, ..., s_n\}$.

(1) Discover the frequent itemsets $i_1, ..., i_m$ from all the transactions in $s_1, ..., s_n$, using the Apriori algorithm already described in Subsection 2.2., with a minor modification so that each itemset is counted at most once for each $s_1, ..., s_n$.
(2) Map the frequent itemsets $i_1, ..., i_m$ to consecutive integers $\hat{i_1}, ..., \hat{i_m}$ respectively. This allows any two of these large itemsets i_j and i_k to be efficiently compared by comparing the integers $\hat{i_j}$ and $\hat{i_k}$.
(3) Transform each sequence $s_j \in S$ by mapping each transaction in s_j to the set of integers corresponding to the large itemsets contained in s_j. Let S' denote the result of transforming S as described.
(4) Generate the frequent sequences in a level-wise manner using a strategy similar to the Apriori algorithm. Let L_k denote all the frequent sequences generated at level k. First generate candidate frequent sequences of length $k+1$ at stage $k+1$ by taking the "join" of frequent sequences of length k generated at stage k. The join is defined as $\{\hat{i_1} \rightarrow \cdots \rightarrow \hat{i_k} \rightarrow \hat{i'_k} | \ \hat{i_1} \rightarrow \cdots \rightarrow \hat{i_k} \in L_k, \hat{i'_1} \rightarrow \cdots \rightarrow \hat{i'_k} \in L_k, \hat{i_1} = \hat{i'_1}, ..., \hat{i_{k-1}} = \hat{i'_{k-1}}, (\nexists s, s \subseteq \hat{i_1} \rightarrow \cdots \rightarrow \hat{i_k} \rightarrow \hat{i'_k} \wedge |s| = k \wedge s \notin L_k)\}$. Then we filter these candidates by scanning S' to check their support

to obtain frequent sequences of length $k+1$. Denote by L the set of resulting frequent sequences from all the levels.

(5) To obtain the maximal frequent sequences, as originally proposed in [4], we simply start from the longest frequent sequence $s \in L$, and eliminate all $s' \in L - \{s\}$ that are contained in s. Then we move on to the next longest frequent remaining sequence and repeat the process, until no more frequent sequences can be eliminated. The frequent sequences that remain are the maximal ones.

For the second task of generating rules that satisfy *minconf* from the maximal frequent sequences, an approach similar to that in Subsection 2.2. can be used. We form a frequent sequence lattice as shown in Figure 6. Each node of the lattice is a unique frequent sequence. After that, for each node X derived from a frequent sequence $X' \rightarrow Y'$ where $X \subseteq X'$, we obtain rules $X \Rightarrow Y$ where $Y \subseteq Y'$ and check their confidence.

An example of applying sequence mining analysis to DNA sequences is given in Figure 7. Here the database consists of 7 DNA sequences. At 4/7 *minsupp* threshold, we obtain the 6 maximal frequent sequences with length at least three. One possible rule derived from $AGTC$ is also shown; the rule $AGT \Rightarrow C$ has $confidence = count(AGTC)/count(AGT) = 4/5$.

SID	Sequence
1	ACGTCACG
2	TCGA
3	GACTGCA
4	CAGTC
5	AGCT
6	TGCAGCTC
7	AGTCAG

- **Maximal frequent sequences** (sup>=4, size>=3)
  - ACT, CAG, GCA, TCA, TCG, AGTC
  - AGT $\Rightarrow$ C (4/5 = 80% confidence)

Fig. 7. An example of a DNA sequence mining for maximal frequent sequences. The support threshold *minsupp* is 4/7 and confidence threshold is 4/5. We consider only frequent sequences of length at least 3. Here SID is the Sequence ID.

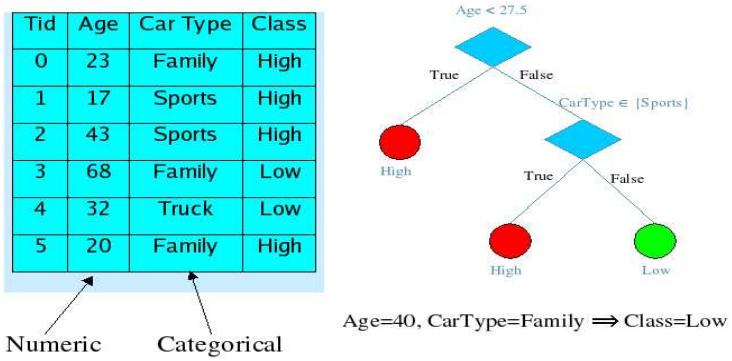

Fig. 8. An example of a decision tree learned from the given data.

2.4. *Classification*

Predictive modeling can sometime—but not necessarily desirably—be seen as a "black box" that makes predictions about the future based on information from the past and present. Some models are better than others in terms of accuracy. Some models are better than others in terms of understandability; for example, the models range from easy-to-understand to incomprehensible (in order of understandability): decision trees, rule induction, regression models, neural networks.

Classification is one kind of predictive modeling. More specifically, classification is the process of assigning new objects to predefined categories or classes: Given a set of labeled records, build a model such as a decision tree, and predict labels for future unlabeled records

Model building in the classification process is a supervised learning problem. Training examples are described in terms of (1) attributes, which can be categorical—*i.e.*, unordered symbolic values—or numeric; and (2) class label, which is also called the predicted or output attribute. If the latter is categorical, then we have a classification problem. If the latter is numeric, then we have a regression problem. The training examples are processed using some machine learning algorithm to build a decision function such as a decision tree to predict labels of new data.

An example of a decision tree is given in Figure 8. The dataset, shown on the left, consists of 6 training cases, with one numeric attribute (Age), one categorical attribute (Car Type) and the class that we need to predict. The decision tree mined from this data is shown on the right. Each internal

node corresponds to a test on an attribute, whereas the leaf nodes indicate the predicted class. For example, assume we have a new record, $Age = 40, CarType = Family$ whose class we need to predict. We first apply the test at the root node. Since $Age < 27.5$ is not true, we proceed to the right subtree and apply the second test. Since $CarType \notin \{Sports\}$ we again to right in the tree where the leaf predicts the label to be Low.

Approaches to classification include decision trees [11, 63, 64], Bayes inference [26, 41], support vector machines [16, 79], emerging patterns [24, 25, 47, 48], neural networks [7, 20, 68], and so on. We describe the decision tree approach here. Although a large number of algorithms exist for this approach [11, 63, 64], they share two common main steps. The first is the "build tree" step:

(1) Start with data at root node.
(2) Select an attribute and formulate a logical test on that attribute. This selection is based on the so-called "split-points", which must be evaluated for all attributes.
(3) Create a child node on each outcome of the test, and move subset of examples satisfying that outcome to the corresponding child node.
(4) Recursively process each child node. The recursion stops for a child node if is "pure"—*i.e.*, all examples it contains are from a single class—or "nearly pure"—*i.e.*, most examples it contains are from the same class. The child node at which a recursion stops is called a leaf of the tree.

The second is the "prune tree" step which removes subtrees that do not improve classification accuracy, and helps avoid over-fitting. More details on tree pruning can be found in [55, 56, 65, 67]; we will focus on the build tree step.

Given a decision tree, each of its leaves corresponds to a rule. The rule is derived by simply conjoining the logical tests on each node on the path from the root of the tree to the corresponding leaf. Some rules derived from the decision tree of Figure 8 are given in Figure 9.

A visualization of how decision trees split the data is given in Figure 10. Assume that we have 42 points in the two dimensional space shown. The circles indicate low risk and the stars the high risk points. Decision trees try to formulate axis-parallel splits to best separate the points into "(relatively) pure" partitions. One such example is shown in the figure, where we split on $Age < 25$ first and then on $CarType \in \{Sports\}$.

The critical issue in the "build tree" step is the formulation of good split tests and selection measure for attributes in Substep (2). A general

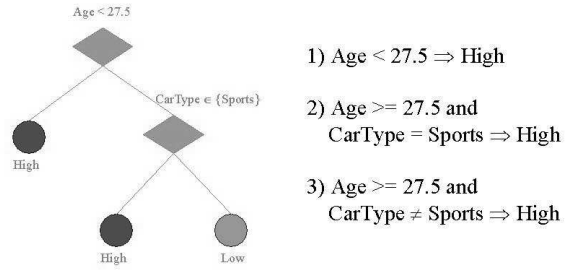

Fig. 9. Example of rules derived from the decision tree of Figure 8.

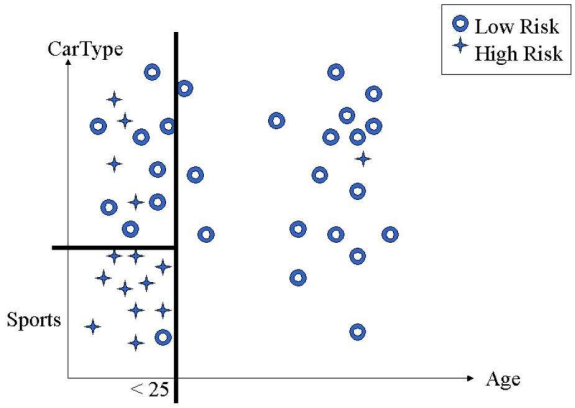

Fig. 10. A visualization of axis-parallel decision tree splits

idea is to prefer the simplest hypothesis that fits the data. One of simplest "hypothesis" is the so-called "minimum message length" principle.

Definition 7: Given a dataset D. Let there be hypotheses $\mathcal{H} = \{H_1, H_2, ..., H_n\}$ proposed for describing D. The minimal message length principle is to pick the hypothesis H that minimizes message length. That is,

$$H = \text{argmin}_{H \in \mathcal{H}}(ML(H_i) + ML(D|H_i))$$

where $ML(\cdot)$ is a measure of message length.

There are a number of ways to define message length, such as information gain [63], information gain ratio [64], Gini index [30], *etc.* Let us illustrate using the Gini index concept familiar from the study of economics.

Let $\mathcal{C} = \{c_1, ..., c_k\}$ be all the classes. Suppose we are currently at a node and D is the set of those samples that have been moved to this node. Let f be a feature and $[d]_f$ be the value of the feature f in a sample d. Let S be a range of values that the feature f can take. Then the gini index for f in D for the range S is defined as

$$gini_f^D(S) = 1 - \sum_{c \in \mathcal{C}} \left(\frac{|\{d \in D \mid d \in c,\ [d]_f \in S\}|}{|D|} \right)^2$$

The purity of a split of the value range S of an attribute f by some split-point into subranges S_1 and S_2 is then defined as

$$gini_f^D(S_1, S_2) = \sum_{S \in \{S_1, S_2\}} \frac{|\{d \in D \mid [d]_f \in S\}|}{|D|} * gini_f^D(S)$$

In Substep (2) of the "build tree" step, we choose the feature f and the split-point p that minimizes $gini_f^D(S_1, S_2)$ over all possible alternative features and split-points. An illustration is given in Figure 11. This example show the two best axis parallel split points on each of the two dimensions in our example: Age and Car-Type. In reality we have to test all possible split points for both dimensions. When we compare the number of points from each class on either side of the split and then derive the Gini Index, we find that the split on the left has a lower value (0.31) and is thus the best overall split. This also corresponds to our intuition, since it is easy to see that the split on Age results in a pure right partition (with the exception on one star), while the split on Car Type results in a less pure top partition (5 stars mixed with the majority of circles).

The performance of a classifier is often given in terms of accuracy, sensitivity, and precision. Let us assume that given a sample, we need to predict it as "positive" or as "negative". Then accuracy is defined as the proportion of predictions that are correct. Sensitivity is defined as the proportion of positive samples that we predict as positives. Precision is the proportion of samples that we predict as positive that are indeed positives.

2.5. *Clustering*

Given n samples without class labels. It is sometimes important to find a "meaningful" partition of the n samples into c subsets or groups. Each of the c subsets can then be considered a class by themselves. That is, we are discovering the c classes that the n samples can be meaningfully categorized into. The number c may be itself given or discovered. This task is called clustering.

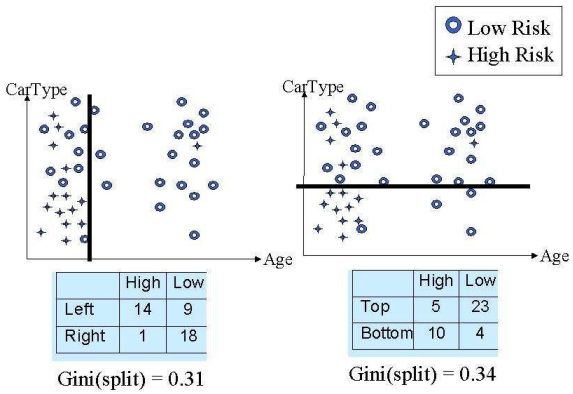

Fig. 11. An example of how to find good split-points using Gini index. The chosen split is the vertical one given on the left.

Given some notion of "similarity" between samples, the goal of clustering can be formulated as an optimization problem to maximize intra-cluster similarity and minimize inter-cluster similarity. If the feature vectors corresponding to the samples are all numeric, the notion of similarity can be defined in terms of distances. But if the features are categorical, then it is much harder to define a reasonable generic notion of similarity.

The main schemes of clustering are outlined below.

- Distance-based: For numeric feature vectors, the common notions of distance are the Euclidean distance and the angle between two vectors. For categorical feature vectors, a notion of distance that works in some situations is the number of common features.
- Partition-based: The points are partitioned into k subgroups and each such possible partition is scored. Since an exhaustive enumeration of all partitions is not feasible, heuristics are used to speed up the search.
- Hierarchical: This comes in two flavors. In a decisive clustering, all points are initially assumed to be in a single cluster. Then at each step a split is made into two clusters that are "far" apart. The process ends when each point is in a cluster by itself. The more common agglomerative clustering starts with each point in its own cluster, and it repeatedly merges the two most similar clusters until all points belong to one cluster.
- Model-based: Treat each cluster as a mixture of multivariate normal distributions, and compute the probability that each point belongs to

a given cluster.

- Density-based: Detect clusters of arbitrary shapes based on the density of points in adjoining regions. The clusters are grown as long as there are sufficient points in the neighborhood, which can be incorporated in to the cluster recursively.

As the notion of similarity or "closeness" used in a clustering is often defined as geometrical distances, it is advisable to consider normalizing the values of different features. For example, given 3 features A in micro seconds, B in milli seconds, and C in seconds, it is crucial to realize that it may be unwise to treat differences as the same in all dimensions or features. For such features, it may be necessary to scale or normalize the values for proper comparison, or to weigh the features according to their importance.

The better known cluster algorithms include k-means [53], k-medoids [43], expectation maximization (EM) [23], self organizing maps [46], competitive learning [69], and so on. We describe the k-means and EM algorithms as an illustration. For this algorithm, the number of clusters desired, k, must be specified in advance. The algorithm works like this:

(1) Guess k "seed" cluster centers.
(2) Iterate the following 2 steps until the centers converge or for a fixed number of times:
   (a) Look at each example and assign it to the center that is closest.
   (b) Recalculate the k centers, from all the points assigned to each center.

For Step (2)(a), "closest" is typically defined in terms of Euclidean distance $\sqrt{\sum_i^n([d_1]_{f_i} - [d_2]_{f_i})^2}$ between two n-dimensional feature vectors d_1 and d_2 if the feature values are numeric. For Step (2)(b), the center for each of the k clusters is recomputed by taking the mean $\langle(\sum_{d\in C}[d]_{f_1})/|C|$, ..., $(\sum_{d\in C}[d]_{f_n})/|C|\rangle$ of all the points d in the corresponding cluster C. Incidentally, the k-medoids algorithm is very similar to k-means, but for Step (2)(b) above, it uses the most centrally located sample of a cluster to be the new center of the cluster.

The K-Means algorithm is illustrated in Figure 12. Initially we choose random cluster center seeds as shown in the top figure. The boundary line between the different cluster regions are shown (these lines are the perpendicular bisectors of the line joining a pair of cluster centers). After we recompute the cluster centers based on the assigned points, we obtain the new centers shown in the bottom left figure. After another round the clus-

ters converge to yield the final assignment shown in the bottom right figure.

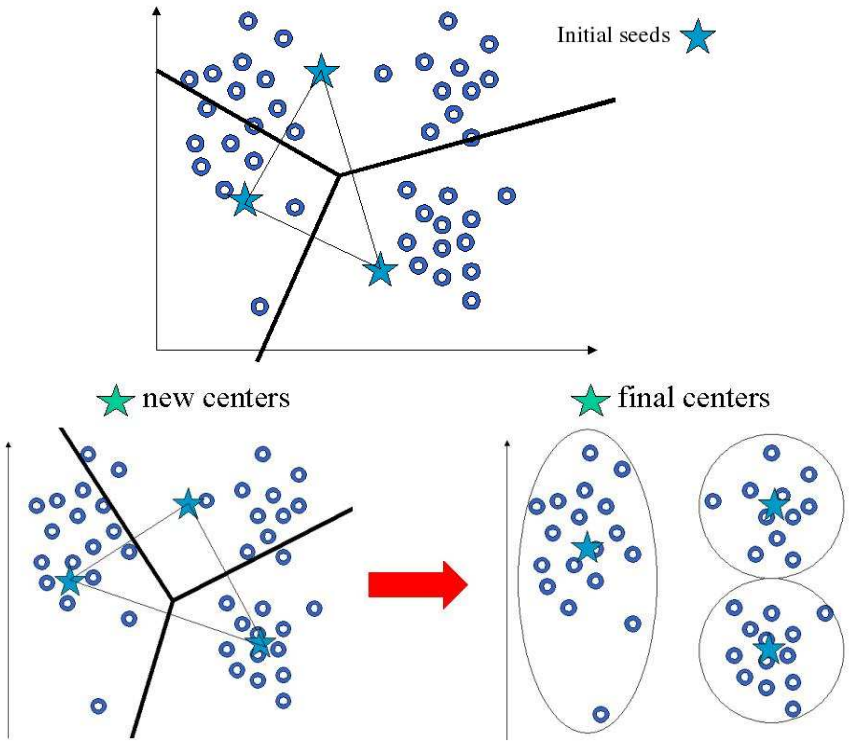

Fig. 12. The working of the k-means clustering algorithm.

While the k-means algorithm is simple, it does have some drawbacks. First of all, k-means is not designed to handle overlapping clusters. Second, the clusters are easily pulled off center by outliers. Third, each record is either in or out of a cluster—in some applications, a fuzzy or probabilistic view of cluster membership may be desirable.

These drawbacks bring us to the Expectation Maximization (EM) algorithm [10] for clustering. Rather than representing each cluster of a dataset D using a single point, the EM algorithm represents each cluster using a d-dimensional Gaussian distribution. This way, a sample x is allowed to "appear" in multiple clusters C_i with different probabilities $P(C_i|x)$. The algorithm [10] works like this:

(1) Choose as random seeds the mean vectors μ_1, ..., μ_k and $d \times d$ covariance matrices M_1, ..., M_k to parameterized the d-dimensional Gaussian distribution for the k clusters C_1, ..., C_k. Choose also k random weights W_1, ..., W_k to be the fraction of the dataset represented by C_1, ..., C_k respectively.

(2) Calculate probability $P(x|C_i)$ of a point x given C_i based on distance of x to the mean μ_i of the cluster as follows

$$P(x|C_i) = \frac{1}{\sqrt{(2 * \pi)^d * |M_i|}} * e \frac{(x - \mu_i)^T \cdot M_i^{-1} \cdot (x - \mu_i)}{2}$$

where $|M_i|$ denotes the determinant of M_i and M_i^{-1} its inverse.

(3) Calculate the mixture model probability density function

$$P(x) = \sum_{i=1}^{k} W_i * P(x|C_i)$$

(4) Maximize $E = \sum_{x \in D} \log(P(x))$ by moving the mean μ_i to the centroid of dataset, weighted by the contribution of each point. To do this move, we first compute the probability a point x belongs to C_i by

$$P(C_i|x) = W_i * \frac{P(x|C_i)}{P(x)}$$

Then we update W_i, μ_i, and M_i in that order like this:

$$W_i = \frac{1}{n} * \sum_{x \in D} P(C_i|x)$$

$$\mu_i = \frac{\sum_{x \in D} P(C_i|x) * x}{\sum_{x \in D} P(C_i|x)}$$

$$M_i = \frac{\sum_{x \in D} P(C_i|x) * (x - \mu_i) * (x - \mu_i)^T}{\sum_{x \in D} P(C_i|x)}$$

(5) Repeat Steps (2)–(4) until E converges or when the increase in E between two successive iterations is sufficiently small.

2.5.1. *Deviation Detection*

The problem of deviation detection is essentially the problem of finding outliers. That is, find points that are very different from the other points in the dataset. It is in some sense the opposite of clustering, since a cluster by definition is a group of similar points, and those points that do not cluster

well can be considered outliers, since they are not similar to other points in the data. This concept is illustrated in Figure 13; for the large cluster on the left, it is clear that the bottom left point is an outlier.

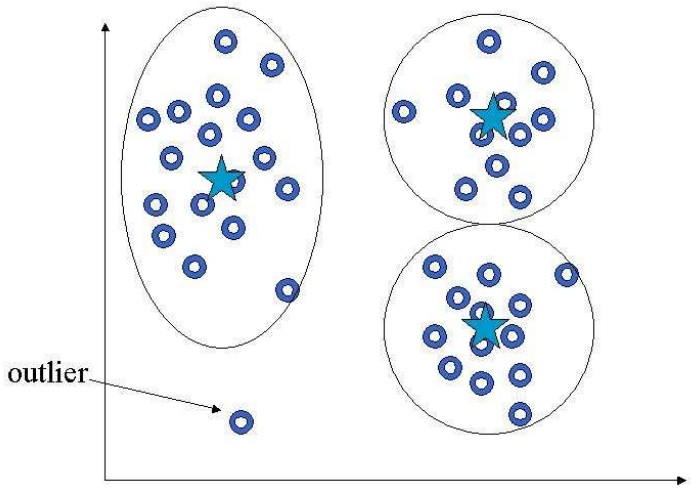

Fig. 13. This figure illustrate the concept of outliers, which are points that are far away from all the clusters.

The outlying points can be "noise", which can cause problems for classification or clustering. In the example above, we obtain a more compact cluster if we discard the outlying point. Or, the outlier points can be really "interesting" items—*e.g.*, in fraud detection, we are mainly interested in finding the deviations from the norm.

A number of techniques from statistics [8, 38] and data mining [2, 12, 13, 45, 66] have been proposed to mine outliers. Some clustering algorithms like BIRCH [85] also have a step to detect outliers.

2.6. *K-Nearest Neighbors*

In Subsection 2.4. we considered classification prediction from the perspective of first constructing a prediction model from training data and then using this model to assign class labels to new samples. There is another perspective to classification prediction that is quite different where no prediction model is constructed beforehand, and every new sample is assigned

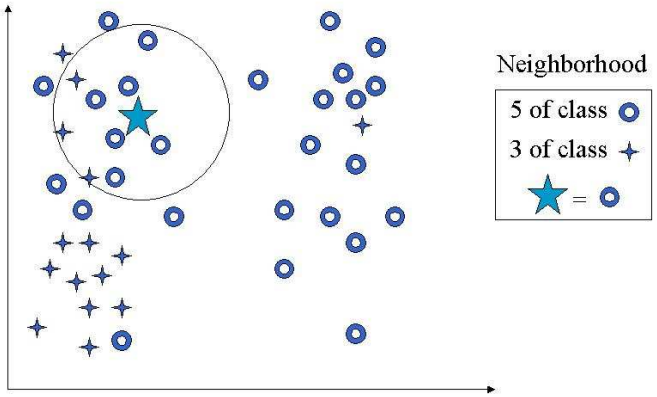

Fig. 14. The working of kNN. the new sample is the large "star". We consider $k = 8$ nearest neighbors. As can be seen in the diagram, 5 of the neighbors are in the "circle" class and 3 are in "cross" class. Hence the new sample is assigned to the "circle" class.

a class label in an instance-based manner.

Representatives of this perspective of classification prediction include the "k Nearest Neighbors" classifier (kNN) [21], which is particularly suitable for numeric feature vectors; the "Decision making by Emerging Patterns" classifier (DeEPs) [47], which is more complicated but also works well on categorical feature vectors; and other classifiers [5]. We describe the simplest of these classifiers—kNN—in this subsection.

The kNN classification technique to assign a class to a new example d is as follows:

(1) Find k nearest neighbors of d in the existing dataset, according to some distance or similarity measure. That is, we compare the new sample d to all known samples in the existing dataset and determine which k known samples are most similar to d.
(2) Determine which class c is the class to which most of those k known samples belong.
(3) Assign the new sample d to the class c.

The working of kNN classifier is illustrated in Figure 14

As mentioned earlier, it is a characteristic of kNN that it locates some training instances or their prototypes in the existing (training) dataset without any extraction of high-level patterns or rules. Such a classifier that is based solely on distance measure may be insufficient for certain types of

applications that require a comprehensible explanation of prediction outcomes. Some in-road to this shortcoming has been made in more modern instance-based classifiers such as DeEPs [47].

Instead of focusing on distance, DeEPs focuses on the so-called emerging patterns [24]. Emerging patterns are regular patterns contained in an instance that frequently occurs in one class of known data but that rarely—or even never—occurs in all other classes of known data. DeEPs then makes its prediction decision for that instance based on the relative frequency of these patterns in the different classes. These patterns are usually quite helpful in explaining the predictions made by DeEPs.

3. Data Preprocessing Techniques

Let us round out our general tutorial on data mining by a more detailed discussion on data preprocessing issues and techniques, especially with respect to data problems and data reduction.

3.1. *Data Problems*

As briefly mentioned in Section 1., collecting, creating, and cleaning a target dataset are important tasks of the data mining process. In these tasks, we need to be aware of many types of data problems such as—but not limited to—the followings:

(1) Noise. The occurrence of noise in the data are typically due to recording errors and technology limitations; or are due to uncertainty and probabilistic nature of specific feature and class values.
(2) Missing data. This can arise from conflicts in recorded data; or because the data was not originally considered important and hence not captured; or even practical reasons such as a patient missing a visit to the doctor.
(3) Redundant data. This has many forms such as the same data has been recorded under different names; the same data has been repeated; or the records contain irrelevant and information-poor attributes.
(4) Insufficient and stale data. Sometimes the data that we are looking for are rare events and hence we may have insufficient data. Sometimes the data may not be up to date and hence we may need to discard them and may end up with insufficient data.

To deal with these data problems, the following types of data preprocessing are performed where appropriate:

(1) Cleaning the data—which include tasks such as removing duplicates, removing inconsistencies, supplying missing values, *etc.*
(2) Selecting an appropriate dataset and/or sampling strategy.
(3) Reducing the dimension of the dataset by performing feature selection.
(4) Mapping of time-series (continuous) or sequence (categorical) data into a more manageable form.

We discuss techniques for item (3) in Subsection 3.2. below.

3.2. *Data Reduction*

As mentioned in the previous subsection, an important data problem in data mining is that of noise. In particular, noise can cause deterioration of data mining algorithms in two aspects. First, most data mining algorithms' time complexity grows exponentially with respect to the number of features that a data record has. If many of these features are polluted by noise, the exponential amount of extra time in processing them becomes a complete waste. Second, some data mining algorithms—especially classification and clustering algorithms—can be confused by noise. If many of these features are polluted by noise, the classification accuracy obtained becomes lower.

So it is worthwhile considering some ways to reduce the amount of noise in the data, provided that this can be done without sacrificing the quality of results, and that this can be done faster than running the main prediction algorithm itself. There are four major ways to data reduction, *viz.*

(1) feature selection, which removes irrelevant features;
(2) instance selection, which removes examples;
(3) discretization, which reduces the number of values of a feature; and
(4) feature transformation, which forms new composite features in a way that can be viewed as compression

Let us discuss feature selection and feature transformation in a little more detail. Feature selection is aimed at separating features that are relevant from features that are not relevant. Feature selection can be viewed as a search. There are three basic ways to do this search:

(1) An evaluation function is defined for each feature. Then each feature is evaluated according to this function. Finally, the best n features are selected for some fixed n; or all features satisfying some statistically significant level are selected. This approach is very reasonable especially if there is reason to believe that the features are independent of each

other. Techniques in this category include statistics-based methods such as t-test [19], signal-to-noise measure [31], Fisher criterion score [29], Wilcoxon rank sum test [70]; and entropy-based methods such as entropy measure [28], χ^2 measure [52], information gain measure [63], information gain ratio [64], *etc.*

(2) The evaluation function is defined using the estimated error rate of some fixed algorithm. Then we do step-wise elimination. That is, we start with the set F_0 of all the features. Then we evaluate $F_0 - \{f\}$ for each $f \in F$ and eliminate the worst f from F_0 to obtain F_1. The process is repeated with F_1 to obtain F_2, and so on. The process stops at some F_n if F_n contains a small enough number of features, has high enough accuracy, or meets some other stopping criteria.

(3) The possible combinations of features are enumerated. Then each combination of features is evaluated as a whole. Finally, the best combination of features is picked. This approach is sometimes necessary if there is reason to believe that the features are not independent of each other. As there are 2^d possible combinations of d features, exhaustive enumeration is impossible. Hence some heuristics are used in a branch-and-bound search. A well-known method in this category is the CFS method [34].

For feature transformation, the best-known method is probably principal component analysis (PCA), which is widely used in signal processing [42]. It works on numeric feature vectors like this.

(1) Let the set of n feature vectors of m dimensions be represented as a $n \times m$ matrix X.

(2) Compute the covariance matrix C of X so that $[C]_{i,j}$ is the linear correlation coefficient between columns i and j of X.

(3) Extract eigenvalues λ_i from $|C - \lambda_i * I| = 0$ for $i = 1, ..., m$, where I the identity matrix.

(4) Compute eigenvectors e_i from $(C - \lambda_i * I) \cdot e_i = 0$, for $i = 1, ..., m$. These are the principal components of X.

(5) Select those e_i with the largest λ_i, as these account for most of the variation in the data. Let $g_1, ..., g_{m'}$, where $m' << m$, be those e_i's selected.

(6) Transform each sample $x \in X$ into a lower m'-dimensional feature vector $\langle x \cdot g_1, ..., x \cdot g_{m'} \rangle$.

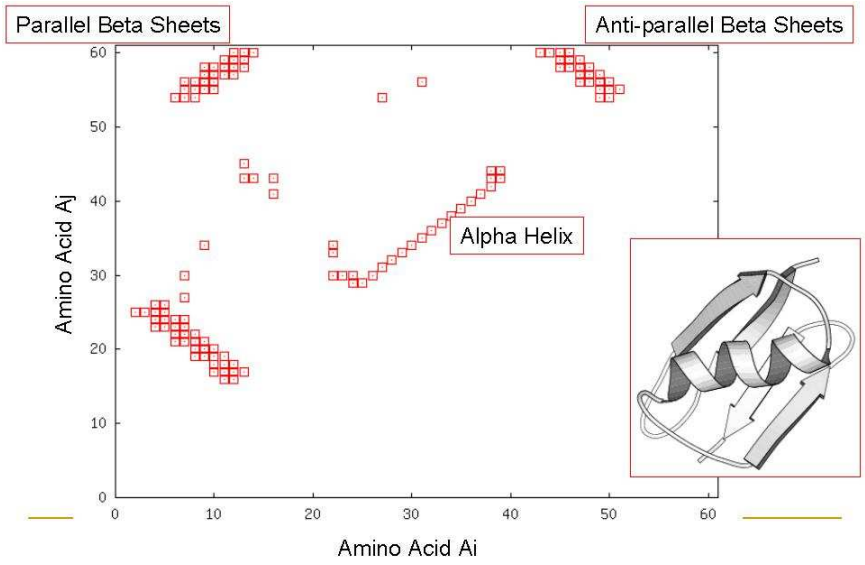

Fig. 15. An example contact map. The boxes indicate the amino acids in the protein—whose structure is shown in the inset diagram—that are in contact.

4. Example: Contact Mining

We have described a number of data mining techniques in the preceding sections. The successful application of these techniques to a specific data mining problem, however, also depends on an adequate understanding of the problem domain, a lot of experimentations, and some amount of experience. Here we illustrate using the example of mining residue contacts in proteins [84].

Definition 8: Two amino acids A_i and A_j of a protein P are said to be "in contact" if their 3D (structural) distance is less than some threshold, say 7Å, and their sequence separation is at least 4. We write the contact map C^P of a protein as a $n \times n$ matrix, where n is length of P, and $C^P(i,j) = 1$ if A_i and A_j are in contact and $C^P(i,j) = 0$ otherwise.

An example contact map is given in Figure 15. Our objective is to discover a set of rules for predicting such a contact map given the amino acid sequence of a protein. We describe the approach of Zaki *et al.* [84] to this problem. It is a hybrid approach that first uses a hidden Markov model (HMM) to predict local substructures within a protein and then uses

meta-level mining on the output of the HMM using association rule mining.

To tackle this problem, we transform the input protein sequence P of length n into a set of vectors $r_{1,1}$, ..., $r_{n,n}$. Each vector $r_{i,j}$ is an itemset $\{p_1 = i, p_2 = j, a_1 = A_i, a_2 = A_j, f_1 = v_1, ..., f_m = v_m\}$ so that p_1 and p_2 record the two positions in P, a_1 and a_2 record the two amino acids at these two positions, and $f_1 = v_1, ..., f_m = v_m$ are additional features derived from P that are helpful in deciding whether A_i are A_j are in contact. As it turns out using just the features p_1, p_2, a_1, and a_2 by themselves do not give rise to good prediction performance—Zaki *et al.* [84] reports a mere 7% precision and 7% accuracy.

In order to obtain better prediction performance, we need to derive some additional features from the protein sequence. These additional features can be chosen from some systematically generated candidate features, such as k-grams generated from the amino acids flanking positions i and j. They can also be motifs of local structural features predicted by some other means from the amino acids flanking position i and j, and so on.

Here, we use a HMM called HMMSTR [18] to derive these additional features. It is a highly branched HMM, as depicted in Figure 16, for general protein sequences based on the I-sites library of sequence structure motifs [17]. The model extends the I-site library by describing the adjacencies of different sequence-structure motifs as observed in protein databases. The I-site library consists of an extensive set of 262 short sequence motifs, each of which correlates strongly with a recurrent local structural motif in proteins, obtained by exhaustive clustering of sequence segments from a non-redundant database of known structures [17, 37].

Each of the 262 I-sites motif is represented as a chain of Markov states in HMMSTR. Each state emits 4 symbols—representing the amino acid, the secondary structure, the backbone angle region, and structural context observed—according to probability distributions specific to that state. These linear chains are hierarchically merged, based on the symbols they emit, into the HMM transition graph depicted in Figure 16. The probability distributions in the states are trained using about 90% of 691 non-redundant proteins from PDBselect [39].

HMMSTR is used to derive the additional features that we need as follows. Given a protein sequence $P = A_1A_2 \ldots A_m$, HMMSTR is applied to it, yielding a sequence of states $S = s_1s_2 \ldots s_m$. A sample of the output emitted by HMMSTR when given a protein sequence is shown in Figure 17. Such an output is then then extracted to form the additional features for the protein sequence. Thus, each vector $r_{i,j}$ becomes an itemset of the form $\{p_1 = i,$

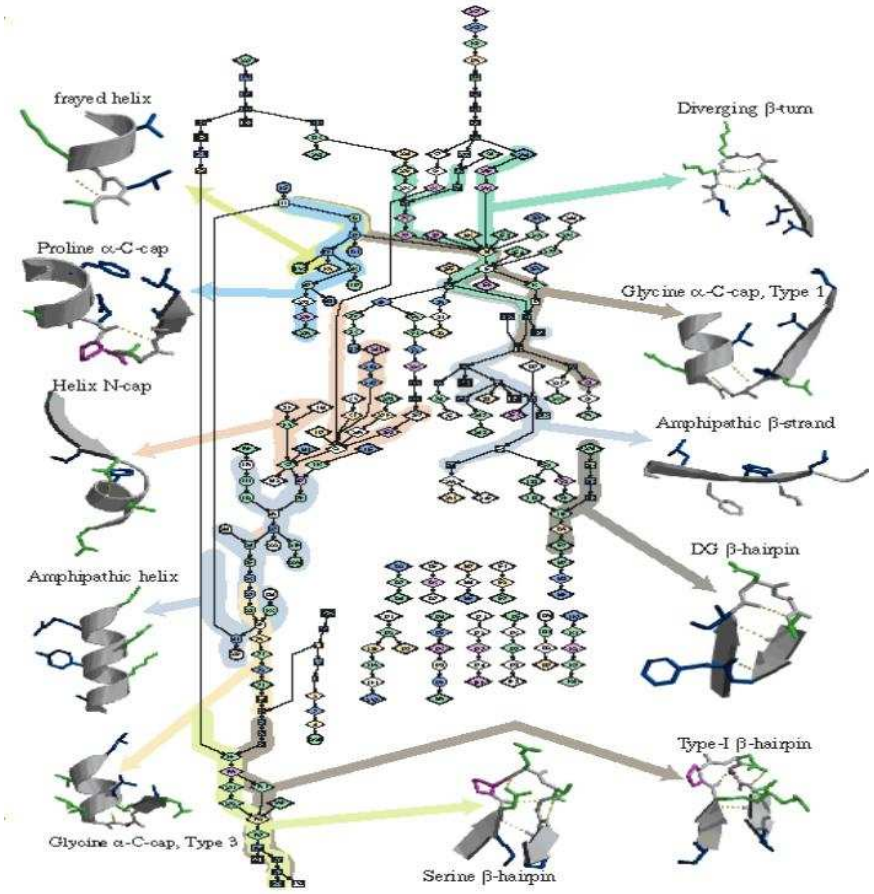

Fig. 16. The highly branched topology of the HMM underlying HMMSTR.

$p_2 = j$, $a_1 = A_i$, $a_2 = A_j$, $coordsD_1 = \cdot$, $coordsR_1 = \cdot$, $coordsC_1 = \cdot$, $coordsD_2 = \cdot$, $coordsR_2 = \cdot$, $coordsC_2 = \cdot$, $profile1_1 = \cdot$, ..., $profile20_1 = \cdot$, $profile1_2 = \cdot$, ..., $profile20_2 = \cdot$, $state1_1 = \cdot$, ..., $state282_1 = \cdot$, $state1_2 = \cdot$, ..., $state282_2 = \cdot\}$. The $\cdot$'s are extracted from the corresponding HMMSTR output. Each state in HMMSTR can produce, or "emit", amino acids and structure symbols according to a probability distribution specific to that state. There are four probability distributions defined for the states in HMMSTR, *profile*, D, R, and C, which describe the probability of observing a particular amino acid in a given state, secondary structure, backbone

```
PDB Name: 153l_
Sequence Length: 185

Position: 1
Residue: R
Coordinates: 0.0, -73.2, 177.6
AA Profile: 0.0 ... 1.0 ... 0.0 # 20 values
HMMSTR State Probabilities: 0.0 ... 0.7 ... 0.0 # 282 values
Distances: 0 3 5 ... 18 15 13 # 185 values
⋮
Position: 185
Residue: Y
Coordinates: -88.7, 0.0, 0.0
AA Profile: 0.0 ... 0.4 ... 0.6 ... 0.0
HMMSTR State Probabilities: 0.0 ... 0.2 ... 0.5 ... 0.3 ... 0.0
Distances: 15 13 10 ... 5 3 0
```

Fig. 17. A sample output from HMMSTR for one PDB protein (153l_) with length 185. For each position, it shows the amino acid (residue), the 3D coordinates, the amino acid profile (i.e., which amino acids are likely to occur at that position based on evolutionary information), the HMMSTR state probability of that position and the 3D distances to every other position (used to construct the contact map).

angle region, or structural context descriptor, respectively. A context descriptor represents the classification of a secondary structure type according to its context. Along with the probability of being in a given state (*state*), such vectors become the input to predict whether the amino acids at positions i and j of a protein are in contact.

The classification is performed via association rules mining. We use a training dataset comprising vectors of the form above. In addition, each vector $r_{i,j}$ in this training dataset is tagged to indicate whether the amino acids in positions i and j are in contact. This training dataset is partitioned into a set D_c consisting of those vectors tagged as "contact", and a set D_n consisting of those vectors tagged as "non-contact". The the process of building a discriminative rule sets is as follows.

(1) Mining: As we are primarily interested in detecting contacts, we apply association rules mining to mine the frequent itemsets in D_c based on a suitably chosen *minsupp* threshold. Let us denote the set of these itemsets by F.
(2) Counting: We compute the support of all itemsets in F in D_n. The support of these itemsets in D_c is computed in the course of the previous step already.
(3) Pruning: The probability of occurrence $P(X|D_c)$ of an itemset $X \in F$ in D_c is simply $support^{D_c}(X)/|D_c|$. Similarly, the probability of occurrence $P(X|D_n)$ of an itemset $X \in F$ in D_n is simply $support^{D_n}(X)/|D_n|$. We remove an itemset $X \in F$ if $P(X|D_c)/P(X|D_n)$ is less than some threshold ρ. That is, we keep only those itemsets that are highly predictive of contacts. Let us denote these remaining itemsets by R.

Now given an unknown protein P of length m. We generate candidate vectors $r_{1,1}$, ..., $r_{m,m}$. we make prediction like this:

(1) Evidence calculation: Let $S(r_{i,j}) = \{X \in R \mid X \subset r_{i,j}\}$. Next, calculate

$$S_c(r_{i,j}) = \sum_{X \in S(r_{i,j})} support^{D_c}(X)$$

$$S_n(r_{i,j}) = \sum_{X \in S(r_{i,j})} support^{D_n}(X)$$

Then define "evidence" as the ratio

$$\rho(r_{i,j}) = \frac{S_c(r_{i,j})}{S_n(r_{i,j})}$$

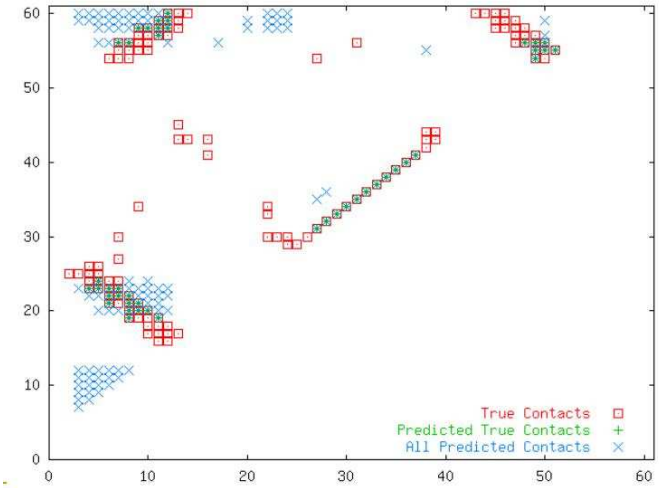

Fig. 18. The predicted contact map for the protein previously shown in Figure 15.

(2) Prediction: Sort $C = \{r_{i,j} | 1 \leq i \leq m,\ 1 \leq j \leq m,\ S_c(r_{i,j}) > 0\}$ in decreasing order of evidence. These are the candidates for "contact". The top γ fraction of C are predicted as "contact". All others are predicted as "non-contact". Here γ can be specified or determined empirically by the user.

Zaki *et al.* [84] test the method above on a test set having a total of 2,336,548 pairs, out of which 35,987 or 1.54% are contacts. This test set is derived from the 10% of the 691 proteins from PDBselect [39] not used in the training of the HMM and association rules. They use a *minsupp* threshold of 0.5% and a $\rho(r_{i,j})$ threshold of 4.

Figure 18 is the prediction result by Zaki *et al.* [84] on the protein given earlier in Figure 15. For this protein they achieve 35% precision and 37% sensitivity. Averaging their results over all the proteins in the test set, it is found that at the 25% sensitivity level, 18% precision can be achieved; this is about 5 times better than random. And at the 12.5% sensitivity level, about 44% precision can be achieved.

If we look at proteins at various lengths, we find that for length less than 100, we get 26% precision at 63% sensitivity, which is about 4 times better than random. For length between 100 and 170, we get 21.5% precision at 10% sensitivity, which is 6 times better than random. For length between

170 and 300, we get 13% precision at 7.5% sensitivity, which is 7.8 times better than random. For longer proteins, we get 9.7% precision at 7.5% sensitivity, which is 7.8 times better than random.

These results appear to be competitive to those reported so far in the literature on contact map prediction. For example, Fariselli and Casadio [27] use a neural network based approach—over a pairs database with contextual information like sequence context windows, amino acid profiles, and hydrophobicity values—to obtain 18% precision for short proteins with a 3 times improvement over random. Olmea and Valencia [57] use correlated mutations in multiple sequence alignments augmented with information on sequence conservation, alignment stability, contact occupancy, *etc.* to obtain 26% precision for short proteins. Note that these two previous works use smaller test sets and their sensitivity levels have not been reported. Thomas *et al.* [76] also use the correlated mutation approach and obtain 13% precision or 5 times better than random, when averaged over proteins of different lengths. Zhao and Kim [86] examine pairwise amino acid interactions in the context of secondary structural environment—*viz.* helix, strand, and coil—and achieve 4 times improvement better than random, when averaged over proteins of different lengths.

5. Summary

We have given an overview of data mining processes. We have described several data mining techniques, including association rules, sequence mining, classification, clustering, deviation detection, and k-nearest neighbors. We have also discussed some data preprocessing issues and techniques, especially data reduction. We have also illustrated the use of some of these techniques by a detailed example on predicting the residue contacts in proteins based on the work of Zaki *et al.* [84].

For association rules, we have in particular described the Apriori algorithm of Agrawal and Srikant [3] in some degree of detail. For sequence mining, we have in particular presented the a generalization of the Apriori algorithm by Agrawal and Srikant [4]. For classification, we have concentrated mostly on decision tree induction [63, 64] based on Gini index [30]. For clustering, we have described the k-means algorithm [53] and the EM algorithm [10]. For k-nearest neighbors, we presented the method of Cover and Hart [21]. For data reduction, we have described the principal component analysis approach [42] in some detail.

References

1. Pieter Adriaans and Dolf Zantinge. *Data Mining*. Addison Wesley Longman, Harlow, England, 1996.
2. C. Aggarwal and P.S. Yu. Outlier detection for high dimensional data. In *Int'l Conf. on Management of Data*, 2001.
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.
4. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of International Conference on Data Engineering*, pages 3–14, 1995.
5. D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
6. Jay Ayres, J. E. Gehrke, Tomi Yiu, and Jason Flannick. Sequential pattern mining using bitmaps. In *SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, July 2002.
7. Pierre Baldi and Soren Brunak. *Bioinformatics: The Machine Learning Approach*. MIT Press, Cambridge, MA, 1999.
8. V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley, New York, 1994.
9. R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD Conf. Management of Data*, June 1998.
10. P. S. Bradley, U. M. Fayyad, and C. A. Reina. Scaling EM (expectation-maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research, November 1998.
11. L. Breiman, L. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.
12. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jorg Sander. OPTICS-OF: Identifying local outliers. In *Int'l Conf. on Principles of Data Mining and Knowledge Discovery*, 1999.
13. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *Int. Conf. on Management of Data*, 2000.
14. S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, 1997. ACM Press.
15. D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: a maximal frequent itemset algorithm for transactional databases. In *Intl. Conf. on Data Engineering*, April 2001.
16. C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
17. C. Bystroff and D. Baker. Prediction of local structure in proteins using a library of sequence-structure motifs. *Journal of Molecular Biology*, 281(3):565–577, 1998.
18. C. Bystroff, V. Thorsson, and D. Baker. HMMSTR: A hidden Markov model

for local sequence-structure correlations in proteins. *Journal of Molecular Biology*, 301(1):173–190, 2000.
19. Mario Caria. *Measurement Analysis: An Introduction to the Statistical Analysis of Laboratory Data in Physics, Chemistry, and the Life Sciences*. Imperial College Press, London, 2000.
20. Y. Chauvin and D. Rumelhart. *Backpropagation: Theory, Architectures, and Applications*. Lawrence Erlbaum, Hillsdale, NJ, 1995.
21. T.M. Cover and P.E. Hart. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.
22. Nello Cristianini and Bernhard Scholkopf. Support vector machines and kernel methods: The new generation of learning machines. *AI Magazine*, pages 31–41, Fall 2002.
23. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.
24. Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of 5th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 15–18, San Diego, August 1999.
25. Guozhu Dong, Xiuzhen Zhang, Limsoon Wong, and Jinyan Li. CAEP: Classification by aggregating emerging patterns. In *Proceedings of 2nd International Conference on Discovery Science*, pages 30–42, Tokyo, Japan, December 1999.
26. R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
27. P. Fariselli and R. Casadio. A neural network nased predictor of residue contacts in proteins. *Protein Engineering*, 12(1):15–21, 1999.
28. U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of 13th International Joint Conference on Artificial Intelligence*, pages 1022–1029, 1993.
29. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.
30. Corrado Gini. Measurement of inequality of incomes. *The Economic Journal*, 31:124–126, 1921.
31. T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Misirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield, and E.S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(15):531–537, 1999.
32. K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *1st IEEE Int'l Conf. on Data Mining*, November 2001.
33. Robert L. Grossman, Chandrika Kamath, Philip Kegelmeyer, Vipin Kumar, and Raju R. Namburu. *Data Mining for Scientific and Engineering Applications*. Kluwer, Dordrecht, Netherlands, 2001.
34. M.A. Hall. *Correlation-based feature selection machine learning*. PhD thesis, Department of Computer Science, University of Waikato, New Zealand, 1998.
35. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidates

generation. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 1–12, 2000.
36. Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, 2000.
37. K. Han and D. Baker. Global properties of the mapping between local amino acid sequence and local structure in proteins. *Proc. Natl. Acad. Sci. USA*, 93(12):5814–5818, 1996.
38. D. Hawkins. *Identification of outliers*. Chapman and Hall, London, 1980.
39. U. Hobohm and C. Sander. Enlarged representative set of protein structures. *Protein Science*, 3(3):522–524, 1994.
40. M. C. Honeyman, V. Brusic, N. Stone, and L. C. Harrison. Neural network-based prediction of candidate T-cell epitopes. *Nature Biotechnology*, 16(10):966–969, 1998.
41. F. V. Jensen. *An Introduction to Bayesian Networks*. Springer-Verlag, New York, 1996.
42. I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, Berlin, 1986.
43. L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, New York, 1990.
44. M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of 3rd International Conference on Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, 1994. ACM Press.
45. E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *24th Intl. Conf. Very Large Databases*, August 1998.
46. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.
47. Jinyan Li, Guozhu Dong, and Kotagiri Ramamohanarao. DeEPs: Instance-based classification using emerging patterns. In *Proceedings of 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 191–200, Lyon, France, 2000.
48. Jinyan Li and Limsoon Wong. Geography of differences between two classes of data. In *Proceedings 6th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 325–337, Helsinki, Finland, August 2002.
49. D-I. Lin and Z. M. Kedem. Pincer-search: A new algorithm for discovering the maximum frequent set. In *6th Intl. Conf. Extending Database Technology*, March 1998.
50. J-L. Lin and M. H. Dunham. Mining association rules: Anti-skew algorithms. In *14th Intl. Conf. on Data Engineering*, February 1998.
51. Bing Liu and Wynne Hsu. Post-analysis of learned rules. In *Proceedings of AAAI*, pages 828–834, 1996.
52. H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of IEEE 7th International Conference on Tools with Artificial Intelligence*, pages 338–391, 1995.
53. J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1:281–297, 1967.

54. H. Mannila, H. Toivonen, and I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery: An International Journal*, 1(3):259–289, 1997.
55. Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. In *Proc. of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining*, August 1995.
56. Jon Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.
57. O. Olmea and A. Valencia. Improving contact predictions by the combination of correlated mutations and other sources of sequence information. *Folding & Design*, 2:S25–S32, June 1997.
58. J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 175–186, San Jose, CA, 1995. ACM Press.
59. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *7th Intl. Conf. on Database Theory*, January 1999.
60. Anders Pedersen and Henrik Nielsen. Neural network prediction of translation initiation sites in eukaryotes: Perspectives for EST and genome analysis. *Intelligent Systems for Molecular Biology*, 5:226–233, 1997.
61. J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *SIGMOD Int'l Workshop on Data Mining and Knowledge Discovery*, May 2000.
62. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefixprojected pattern growth. In *Int'l Conf. Data Engineering*, April 2001.
63. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
64. J. R. Quinlan. *C4.5: Program for Machine Learning*. Morgan Kaufmann, 1993.
65. J.R. Quinlan. Simplifying decision trees. *Interational Journal of Man-Machine Studies*, 27:221–334, 1987.
66. S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Int'l Conference on Management of Data*, 2000.
67. R. Rastogi and K. Shim. Public: A decision tree classifier that integrates building and pruning. In *24th Intl. Conf. Very Large Databases*, August 1998.
68. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
69. D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.
70. R. Sandy. *Statistics for Business and Economics*. McGrawHill, 1989.
71. A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of 21st International Conference on Very Large Data Bases*, pages 432–443, Zurich, Switzerland, 1995. Morgan Kaufmann.
72. C. Schoenbach, P. Kowalski-Saunders, and V. Brusic. Data warehousing in

molecular biology. *Briefings in Bioinformatics*, 1:190–198, 2000.
73. P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *ACM SIGMOD Intl. Conf. Management of Data*, May 2000.
74. A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):970–974, 1996.
75. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *5th Intl. Conf. Extending Database Technology*, March 1996.
76. D. Thomas, G. Casari, and C. Sander. The prediction of protein contacts from multiple sequenc alignments. *Protein Engineering*, 9(11):941–948, 1996.
77. H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hatonen, and H. Mannila. Pruning and grouping discovered association rules. In *Proceedings of ECML-95 Workshop on Statistics, Machine Learning, and Discovery in Databases*, pages 47–52, 1995.
78. O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17:520–525, 2001.
79. Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
80. Limsoon Wong. Datamining: Discovering information from bio-data. In Tao Jiang, Ying Xu, and Michael Zhang, editors, *Current Topics in Computational Biology*, chapter 13, pages 317–342. MIT Press, Cambridge, MA, 2002.
81. M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372-390, May-June 2000.
82. M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, Jan/Feb 2001.
83. M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining*, April 2002.
84. Mohammed J. Zaki and Chris Bystroff. Mining residue contacts in proteins. In R. L. Grossman et al., editors, *Data Mining for Scientific and Engineering Applications*, chapter 9, pages 141–164. Kluwer, Dordrecht, Netherlands, 2001.
85. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, June 1996.
86. C. Zhao and S.-H. Kim. Environment-dependent residue contact energies for proteins. *Proc. Natl. Acad. Sci.*, 97(6):2550–2555, 2000.