In conclusion, we see that many different techniques can be used to implement data mining algorithms that are efficient in both time and space when we deal with very large data sets. Indeed there are several other approaches we have not even mentioned here, including the use of online algorithms that see each data point only once (useful for applications where data are arriving rapidly in a continuous stream over time) and more hardware-oriented solutions such as parallel processing implementations of algorithms (in cases when both the algorithm and the data permit efficient parallel approaches). Choice of a particular technique often depends on quite practical aspects of the data mining application—i.e., how quickly must the data mining algorithm produce an answer? Does the model need to be continually updated? and so forth. Research on scalable data mining algorithms is likely to continue for some time, and we can expect more developments in this area. The reader should be cautioned to be aware that, as in everything else, there is no free lunch! In other words, there are typically trade-offs involving model accuracy, algorithm speed and memory, and so forth. Informed judgment on which type of algorithm and data structures best suit your problem will require careful consideration of both algorithmic issues and application details about how the algorithm and model will be used in practice.

12.13 Further Reading

There are several high-quality yearly database conferences, such as ACM's SIGMOD Conference on Management of Data (SIGMOD), and the SIGACT-SIGMOD-SIGART Symposium on Principles of Database and Knowledge-base Systems (PODS), the Very Large Database Conference (VLDB), and the International Conference on Data Engineering (ICDE).
There are several fine database textbooks, including Ullman (1988), Abiteboul, Hull, and Vianu (1995), and Ramakrishnan and Gehrke (1999). A recent survey of query optimization is Chaudhuri (1998). The data cube is presented in Gray et al. (1996) and Gray et al. (1997). A good introduction to OLAP is Chaudhuri and Dayal (1997).
Implementation of database management systems is described in detail in Garcia-Molina et al. (1999). A nice discussion of OLAP and statistical databases is given by Shoshani (1997). Issues in using database management systems to implement mining algorithms are considered in Sarawagi et al. (2000) and Holsheimer et al. (1995).
Madigan et al. (in press) discuss various extensions of the the original squashing approach. Provost and Kolluri (1999) provide an overview of different techniques for scaling up data mining algorithms to handle very large data sets. Provost, Jensen, and Oates (1999), and Domingos and Hulten (2000) give examples of sampling problems with very large databases in data mining.

Chapter 13: Finding Patterns and Rules

13.1 Introduction

In this chapter we consider the problem of finding useful patterns and rules from large data sets. Recall that a pattern is a local concept, telling us something about a particular aspect of the data, while a model can be thought of as giving a full description of the data.
For a data set describing customers of a supermarket, a pattern might be "10 percent of the customers buy wine and cheese," and for a data set of telecommunication alarms a pattern could be "if alarms *A* and *B* occur within 30 seconds of each other, then alarm *C* occurs within 60 seconds with probability 0.5." For the Web log data set in chapter 1, an example pattern could be "if a person visits the CNN Web site, there is a 60% chance the person will visit the ABC News Web site in the same month." In each of these cases, the pattern is a potentially interesting piece of information about part of the data.

How do we find such patterns from data? Given some way of representing patterns and the set of all possible patterns in this representation, the trivial method is to try each

pattern in turn and see whether it occurs in data and/or whether it is significant in some sense. If the number of possible patterns is small, then this method might be applicable, but typically it is completely infeasible. For example, in the supermarket example we could define a pattern for each possible subset of the set of all products. For 1,000 products this yields 2^{1000} patterns. In the case of images or sequences of alarms, there is a potentially infinite number of patterns.

If the patterns were completely unrelated to each other, we would have no other choice but to use the trivial method. However, the set of patterns typically has a great deal of structure. We have to use this structure of the patterns to guide the search. Typically, there is a generalization/specialization relation between patterns. A pattern a is more general than pattern ß, if whenever ß occurs in the data, a occurs too. For example, the pattern "At least 10 percent of the customers buy wine" is more general than the pattern "At least 5 percent of the customers buy wine and cheese." Use of such generalization relationships between patterns leads to simple algorithms for finding all patterns of a certain type that occur in the data.

In this chapter we present a number of methods for finding local patterns from large classes of data. We start from very simple pattern classes and relatively straightforward algorithms, and then discuss some generalizations. The basic theme in the chapter is the discovery of interesting patterns through the refinement of more general ones.

Scalability of pattern and rule discovery algorithms is obviously an important issue. The algorithms that we describe in this chapter typically carry out only a limited number of passes through the data, and hence they scale rather nicely for large data sets. In addition, if we are interested in finding only patterns or rules that apply to relatively large fractions of the data set, we can effectively use sampling. The frequency of a pattern in the sample will be approximately the same as in the whole data set, so pattern discovery from the sample produces reasonably good results. If our interest is in patterns that occur only rarely in the data, for example, finding very rare and unusual stars or galaxies among tens of millions of objects in the night sky, then sampling will be insufficient.

13.2 Rule Representations

A *rule* consists of a left-hand side proposition (the antecedent or condition) and a right-hand side (the consequent), e.g., "If it rains then the ground will be wet." Both the left and right-hand sides consist of Boolean (true or false) statements (or propositions) about the world. The rule states that if the left-hand side is true, then the right-hand side is also true. A *probabilistic rule* modifies this definition so that the right-hand side is true with probability p, given that the left-hand side is true—the probability p is simply the conditional probability of the right-hand side being true given the left-hand side is true. Rules have a long history as a knowledge representation paradigm in cognitive modeling and artificial intelligence. Rules can also be relatively easy for humans to interpret (at least relatively small sets of rules are) and, as such, have been found to be a useful paradigm for learning interpretable knowledge from data in machine learning research. In fact, classification tree learning (discussed in chapters 6 and 10) can be thought of as a special case of learning a set of rules: the conditions at the nodes along the path to each leaf can be considered a conjunction of statements that make up the left-hand side of a rule, and the class label assignment at the leaf node provides the right-hand side of the rule.

Note that rules are inherently discrete in nature; that is, the left-and right-hand sides are Boolean statements. Thus, rules are particularly well matched to modeling discrete and categorical-valued variables, since it is straightforward to make statements about such variables in Boolean terms. We can, of course, extend the framework to real-valued variables by quantizing such variables into discrete-valued quanta, e.g., "if $X > 10.2$ then $Y < 1$" (this is precisely how classification trees handle real-valued variables, for example).

Typically the left-hand sides of rules are expressed as simple Boolean functions (e.g., conjunctions) of variable-value statements about individual variables (e.g., $A = a_1$ or $Y >$

0). The simplicity of conjunctions (compared to arbitrary Boolean functions) makes conjunctive rules by far the most widely used rule representation in data mining. For real-valued variables, a left-hand side such as $X > 1$? $Y > 2$ is defining a left-hand side region whose boundaries are parallel to the axes of the variables in (X, Y) space, i.e., a multidimensional "box" or hyperrectangle. Again, we could generalize to have statements about arbitrary functions of variables (leading to more complex left-hand side regions), but we would lose the interpretability of the simpler form. Thus, for handling real-valued variables in rule learning, simple univariate thresholds are popular in practice because of their simplicity and interpretability.

13.3 Frequent Itemsets and Association Rules

13.3.1 Introduction

Association rules (briefly discussed in chapters 5 and 12) provide a very simple but useful form of rule patterns for data mining. Consider again an artificial example of 0/1 data (an "indicator matrix") shown in figure 13.1. The rows represent the transactions of individual customers (in, for example, a "market basket" of items that were purchased together), and the columns represent the items in the store. A 1 in location (i, j) indicates that customer i purchased item j, and a 0 indicates that that item was not purchased.

[htb]

basket-id	A	B	C	D	E		
t_1	1	0	0	0	0		
t_2	1	1	1	1	0		
t_3	1	0	1	0	1		
t_4	0	0	1	0	0		
t_5	0	1	1	1	0		
t_6	1	1	1	0	0		
t_7	1	0	1	1	0		
t_8	0	1	1	0	1		
t_9	1	0	0	1	0		
t_{10}	0	1	1	0	1		

Figure 13.1: An Artificial Example of Basket Data.

We are interested in finding useful rules from such data. Given a set of 0,1 valued observations over variables A_1, ..., A_p, an association rule has the form

$$\Big((A_{i_1} = 1) \wedge \cdots \wedge (A_{i_k} = 1)\Big) \Rightarrow A_{i_{k+1}} = 1,$$

where $1 = i_i = p$ for all j. Such an association rule can be written more briefly as $\Big((A_{i_1} \wedge \cdots \wedge A_{i_k}\Big) \Rightarrow A_{i_{k+1}}$. A pattern such as

$$(A_{i_1} = 1) \wedge \cdots \wedge (A_{i_k} = 1)$$

is called an *itemset*. Thus, association rules can be viewed as rules of the form ? ? ?, where ? is an itemset pattern and ? is an itemset pattern consisting of a single conjunct. We could also allow conjunctions on the right-hand side of rules, but for simplicity we do not.

The framework of association rules was originally developed for large sparse transaction data sets. The concept can be directly generalized to non-binary variables taking a finite number of values, although we will not do so here (for simplicity of notation).
Given an itemset pattern ?, its *frequency fr*(?) is the number of cases in the data that satisfy ?. Note that the frequency *fr*(? ? ?) is sometimes referred to as the *support*.
Given an association rule ? ? ? , its *accuracy c*(? ? ?) (also sometimes referred to as the *confidence*) is the fraction of rows that satisfy ? among those rows that satisfy ?, i.e.,

(13.1) $$c(\theta \Rightarrow \varphi) = \frac{fr(\theta \wedge \varphi)}{fr(\theta)}.$$

In terms of conditional probability notation, the empirical accuracy of an association rule can be viewed as a maximum likelihood (frequency-based) estimate of the conditional probability that ? is true, given that ? is true. We note in passing that instead of a simple frequency-based estimate, we could use a maximum a posteriori estimate (chapter 4) to get a more robust estimate of this conditional probability for small sample sizes. However, since association rules are typically used in applications with very large data sets and with a large threshold on the size of the itemsets (that is, *fr*(?) is usually fairly large), the simple maximum likelihood estimate above will be quite sufficient in such cases.

The frequent itemsets are very simple patterns telling us that variables in the set occur reasonably often together. Knowing a single frequent itemset does not provide us with a great deal of information about the data: it only gives a narrow viewpoint on a certain aspect of the data. Similarly, a single association rule tells us only about a single conditional probability, and does not inform us about the rest of the joint probability distribution governing the variables.
The task of finding frequent itemset patterns (or, *frequent sets*) is simple: given a frequency threshold *s*, find all itemset patterns that are frequent, and their frequencies. In the example of figure 13.1, the frequent sets for frequency threshold 0.4 are {*A*}, {*B*}, {*C*}, {*D*}, {*AC*}, and {*BC*}. From these we could find, for example, the rule *A* ? *C*, which has accuracy 4/6 = 2/3, and the rule *B* ? *C*, with accuracy 5/5 = 1.

Algorithms for finding association rules find all rules satisfying the frequency and accuracy thresholds. If the frequency threshold is low, there might be many frequent sets and hence also many rules. Thus, finding association rules is just the beginning in a data mining effort: some of these rules will probably be trivial to the user, while others may be quite interesting. One of the research challenges in using association rules for data mining is to develop methods for selecting potentially interesting rules from among the mass of discovered rules.
The rule frequency tells us how often a rule is applicable. In many cases, rules with low frequency are not interesting, and this assumption is indeed built into the definition of the association rule-finding problem. The accuracy of an association rule is not necessarily a very good indication of its interestingness. For example, consider a medical application where the rule is learned that pregnancy implies that the patient is female with accuracy 1! A rule with accuracy close to 1 could be interesting, but the same is true for a rule with accuracy close to 0. We will return later to this question of measuring how interesting a rule is to a user. (We discussed issues of data quality in chapter 2. With a large data set we might well find that pregnancy implies that the patient is female with accuracy less than 1. This does not mean that there are pregnant men running around, but merely that data are not perfect.)
The statistical significance of an association rule *A* ? *B* can be evaluated using standard statistical significance testing techniques to determine whether the estimated probability $p(B = 1|A = 1)$ differs from the estimated background probability of $B = 1$, and whether this difference would be likely to occur by chance. This is equivalent to testing whether $p(B = 1|A = 1)$ differs from $p(B = 1|A = 0)$ (e.g., see example 4.14).
Although such testing is possible, the use of significance testing methods to evaluate the quality of association rules is problematic, due to the multiple testing problem discussed in chapter 4. If we extract many rules from the data and conduct significance tests on

each, then it is very likely, by chance alone, that we will find a rule that appears to be statistically significant, even if the data were purely random.

A set of association rules does not provide a single coherent model that would enable us to make inference in a systematic manner. For example, the rules do not provide a direct way of predicting what an unknown entry will be. Various rules might predict various values for a variable, and there is no central structure (as in decision trees) for deciding which rule is in force.
To illustrate, suppose we now obtain a further row for figure 13.1 with $A = 1$, $B = 1$, $D = 1$, and $E = 1$; then the set of rules obtained from that data could be used to suggest that (a) $C = 1$ with accuracy 2/3 (because of the rule $A \rightarrow C$) or (b) $C = 1$ with accuracy 1 (because of the rule $B \rightarrow C$). Thus the set of rules does not form a global and consistent description of the data set. (However, the collection of association rules or frequent sets can be viewed as providing a useful condensed representation of the original data set, in the sense that a lot of the marginal information about the data can be retrieved from this collection.)
Formulated in terms of the discussion of chapter 6, the model structure for association rules is the set of all possible conjunctive probabilistic rules. The score function can be thought of as binary: rules with sufficient accuracy and frequency get a score of 1 and all other rules have a score of 0 (only rules with a score of 1 are sought). In the next subsection we discuss search methods for finding all frequent sets and association rules, given pre-defined thresholds on frequency and accuracy.

13.3.2 Finding Frequent Sets and Association Rules

In this section we describe methods for finding association rules from large 0/1 matrices. For market basket and text document applications, a typical input data set might have 10^5 to 10^8 data rows, and 10^2 to 10^6 variables. These matrices are often quite sparse, since the number of 1s in any given row is typically very small, e.g., with 0.1% or less chance of finding a 1 in any given entry in the matrix.

The task in association rule discovery is to find all rules fulfilling given pre-specified frequency and accuracy criteria. This task might seem a little daunting, as there is an exponential number of potential frequent sets in the number of variables of the data, and that number tends to be quite large in, say, market basket applications. Fortunately, in real data sets it is the typical case that there will be relatively few frequent sets (for example, most customers will buy only a small subset of the overall universe of products).

If the data set is large enough, it will not fit into main memory. Thus we aim at methods that read the data as few times as possible. Algorithms to find association rules from data typically divide the problem into two parts: first find the frequent itemsets and then form the rules from the frequent sets.

If the frequent sets are known, then finding association rules is simple. If a rule $X \rightarrow B$ has frequency at least s, then the set X must by definition have frequency at least s.

Thus, if all frequent sets are known, we can generate all rules of the form $X \rightarrow B$, where X is frequent, and evaluate the accuracy of each of the rules in a single pass through the data.
A trivial method for finding frequent sets would be to compute the frequency of all subsets, but obviously that is too slow. The key observation is that a set X of variables can be frequent only if all the subsets of X are frequent. This means that we do not have to find the frequency of any set X that has a non-frequent proper subset. Therefore, we can find all frequent sets by first finding all frequent sets consisting of 1 variable. Assuming these are known, we build candidate sets of size 2: sets $\{A, B\}$ such that $\{A\}$ is frequent and $\{B\}$ is frequent. After building the candidate sets of size 2, we find by looking at the data which of them are really frequent. This gives the frequent sets of size 2. From these, we can build candidate sets of size 3, whose frequency is then computed from the data, and so on. As an algorithm, the method is as follows.

$i = 0$;

C_i = {{A} | A is a variable };
while C_i is not empty **do**
database pass:
for each set in C_i, test whether it is frequent;
let L_i be the collection of frequent sets from C_i;
candidate formation:
let C_{i+1} be those sets of size i + 1
whose all subsets are frequent;
End.

This method is known as the APriori algorithm. Two issues remain to be solved: how are the candidates formed? and how is the frequency of each candidate computed? The first problem is easy to solve in a satisfactory manner. Suppose we have a collection L_i of frequent sets, and we want to find all sets Y of size i + 1 that possibly can be frequent; that is, all sets Y whose all proper subsets are frequent. This can be done by finding all pairs {U, V} of sets from L_i such that the union of U and V has size i + 1, and then testing whether the union really is a potential candidate. There are fewer than $|L_i|^2$ pairs of sets in L_i, and for each one of them we have to check whether $|L_i|$ other sets are present. The worst-case complexity is approximately cubic in the size of L_i. In practice the method usually runs in linear time with respect to the size of L_i, since there are often only a few overlapping elements in L_i. Note that candidate formation is independent of the number of records n in the actual data.

Given a set C_i of candidates, their frequencies can be evaluated in a single pass through the database. Simply keep a counter for each candidate and increment the counter for each row that contains the candidate. The time needed for candidate C_i is $O(|C_i|np)$, if the test is implemented in a trivial way. Additional data structure techniques can be used to speed up the method.

The total time needed for the finding of the frequent sets is $O(\sum_i |C_i|np)$—that is, proportional to the product of the size of the data (np) and the number of sets that are candidates on any level. The algorithm needs k or k + 1 passes through the data, where k is the number of elements in the largest frequent set.

There exist many variants of the basic association rule algorithm. The methods typically strive toward one or more of the following three goals: minimizing the number of passes through the data, minimizing the number of candidates that have to be inspected, and minimizing the time needed for computing the frequency of individual candidates.

One important way of speeding up the computation of frequencies of candidates is to use data structures that make it easy to find out which candidate sets in C_i occur for each row in the data set. One possible way to organize the candidates is to use a tree-like structure with branching factor p (the number of variables). For each variable A, the child of the root of the tree labeled with A contains those candidates whose first variable (according to some ordering of the variables) is A. The child labeled A is constructed in a recursive manner.

Another important way of speeding up the computation of frequent sets is to use sampling. Since we are interested in finding patterns describing large subgroups, that is patterns having frequency higher than a given threshold, it is clear that just using a sample instead of the whole data set will give a fairly good approximation for the collection of frequent sets and their frequencies. A sample can also be used to obtain a method that with high probability needs only two passes through the data. First, compute from the sample the collection of frequent sets F using a threshold that is slightly lower than the one given by the user. Then compute the frequencies in the whole data set of each set in F. This produces the exact answer to the problem of finding the frequent sets in the whole data set, unless there is a set Y of variables that was not frequent in the sample but all of whose subsets turned out to be frequent in the whole data set; in this case, we have to make an extra pass through the database.

13.4 Generalizations

The method for finding frequently occurring sets of variables can also be applied to other types of patterns and data, since the algorithms described above do not use any special properties of the frequent set patterns. What we used were (1) the conjunctive structure of the frequent sets and the monotonicity property, so that candidate patterns could be formed quickly, and (2) the ability to test quickly whether a pattern occurs in a row, so that the frequency of the pattern can be computed by a fast pass through the data.
Next we formulate the same algorithms in terms of more abstract notions. Suppose that we have a class of *atomic patterns A*, and our interest is in finding conjunctions of these atomic patterns that occur frequently. That is, the pattern class *P* is the set of all conjunctions

$$a_1 ? \ \ldots ? \ a_k,$$

where $a_i \in A$ for all *i*.
Let the data set *D* consist of *n* objects $d_1, \ldots, d_n$, and assume we can test whether a pattern a is true about an object *d*. A conjunction $? = a_1 ? \ldots ? a_k \in P$ is true about *d* if all conjuncts a_i are true about *d*. Let s be a threshold. The goal is to find those conjunctions of patterns that occur frequently:

$$\{? \in P \mid ? \text{ is true for for at least s objects } d \in D\}.$$

In the case of frequent itemsets, the atomic patterns were conditions of the form $A = 1$, where *A* is a variable, and the frequent sets like *ABC* were just shorthand notations for the conjunctions of form $A = 1 \ ? \ B = 1 \ ? \ C = 1$.
Suppose we can decide how many times each atomic pattern a occurs in the data. Then we can apply the above algorithm for finding all the patterns from *P* that occur frequently enough. We simply first find the atomic patterns that occur frequently enough, then build the conjunctions of two atomic patterns that can possibly occur frequently, test to see which of those occur frequently enough, build conjunctions of size 3, etc. The method works in exactly the same way as before. If the patterns are complex, we may have to do some clever processing to build the new candidates and to test for occurrence of patterns.

13.5 Finding Episodes From Sequences

In this section we present another application of the general idea of finding association rules: we describe algorithms for finding *episodes* from sequences.
Given a set of *E* of *event types*, an *event sequence* s is a sequence of pairs (*e, t*), where $e \in E$ and *t* is an integer, the occurrence time of the event of type *e*. An *episode* a is a partial order of event types, such as the ones shown in figure 13.2. Episodes can be viewed as graphs.

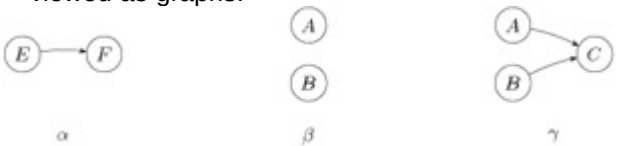

Figure 13.2: Episodes a, ß, and ?.

Given a window width *W*, the *frequency* of the episode a in event sequence *S* is the fraction of slices of width *W* taken from *S* such that the slice contains events of the types occurring in a in the order described by a. We now concentrate on the following discovery task: given an event sequence **s**, a set e of episodes, a window width *win*, and a frequency threshold *min_fr*, find the collection *F E*(**s**, *win, min_fr*) of all episodes from the set that occur at least in a fraction of *min_fr* of all the windows of width *win* on the sequence **s**. The display below gives an algorithm for computing the collection of frequent episodes.

The method is based on the same general idea as the association rule algorithms: the frequencies of patterns are computed by starting from the simplest possible patterns. New candidate patterns are built using the information from previous passes through the data, and a pattern is not considered if any of its subpatterns is not frequent enough. The

main difference compared to the algorithms outlined in the previous sections is that the conjunctive structure of episodes is not as obvious.

An episode ß is defined as a *subepisode* of an episode a if all the nodes of ß occur also in a and if all the relationships between the nodes in ß are also present in a. Using graph-theoretic terminology, we can say that ß is an induced subgraph of a. We write ß ? a if ß is a subepisode of a, and ß ? a if ß ? a and ß ? a.

Example 13.1

[redacted]

Given a set *E* of event types, an event sequence s over *E*, a set e of episodes, a window width *win*, and a frequency threshold *min_fr*, the following algorithm computes the collection *FE*(**s**, *win, min_fr*) of frequent episodes.

C_1 := {a ∈ e ||a| = 1};
l := 1;
while C_l ? f **do**
/* Database pass: */
compute F_l := {a ∈ C_l | *fr*(a, s, *win*) = *min_fr*};
l := *l* + 1;
/* Candidate generation: */
compute C_l := {a ∈ e ||a| = *l* and for all ß ∈ e such that ß ? a
and |ß| < *l* we have ß ∈ $F_{|ß|}$};
end;
for all *l* **do** output F_l;

[redacted]

The algorithm performs a levelwise (breadth-first) search in the class of episodes following the subepisode relation. The search starts from the most general episodes—that is, episodes with only one event. On each level the algorithm first computes a collection of candidate episodes, and then checks their frequencies from the event sequence.

The algorithm does at most *k* + 1 passes through the data, where *k* is the number of edges and vertices in the largest frequent episode. Each pass evaluates the frequency of $|C_l|$ episodes. The computation of the frequency of an episode requires that we find the windows in the sequence in which the episode occurs. This can be done in time linear in the length of the sequence and the size of the episode. The running time of the episode discovery algorithm is thus $O(n \sum_{l=1}^{k} |C_l|l)$, where *n* is the length of the sequence.

A similar approach can be used for any conjunctive class of patterns, as long as there are not too many frequent patterns.

13.6 Selective Discovery of Patterns and Rules

13.6.1 Introduction

The previous sections discussed methods that are used to find all rules of a certain type that fulfill simple frequency and accuracy criteria. While this task is useful in several applications, there are also simple and important classes of patterns for which we definitely do not want to see all the patterns. Consider, for example, a data set having variables with continuous values. Then, as in chapter 1, we can look at patterns such as

- *0*: if $X > x_1$ then $Y > y_1$ with probability *p*, and the frequency of the rule is *q*

Such a rule is a fine partial description of the data. The problem is that if in the data we have *k* different values of *X* and *h* different values of *Y* , there are *kh* potential rules, and many of these will have sufficient frequency to be interesting from that point of view. For example, from a data set with variables Age and Income we might discover rules

- a: if Age>40 then Income>62755 (probability 0.34)

- ß: if Age>41 then Income>62855 (probability 0.33)

First, the user will not be be happy seeing two rules that express more or less the same general pattern. Thus, even if we found both of these rules, we should avoid showing them to the user, The second problem is that in this example the pattern a is more general than ß, and there are very long sequences a_1, a_2, ... of patterns such that a_i is more general than a_{i+1}. Hence the basic algorithmic idea in the previous sections of starting from the most general pattern, looking at the data, and expanding the qualified patterns in all possible ways does not work, as there are many specializations of any single pattern and the pattern space is too large.

All of this means that the search for patterns has to be pruned in addition to the use of frequency criteria. The pruning is typically done using two criteria:

1. interestingness: whether a discovered pattern is sufficiently interesting to be output;
2. promise: whether a discovered pattern has a potentially interesting specialization.

Note that a pattern can be promising even though it is not interesting. A simple example is any pattern that is true of all the data objects: it is not interesting, but some specialization of it can be. Interestingness can be quantified in various ways using the frequency and accuracy of the pattern as well as background knowledge.

13.6.2 Heuristic Search for Finding Patterns

Assume we have a way of defining the interestingness and promise of a pattern, as well as a way of pruning. Then a generic heuristic search algorithm for finding interesting patterns can be formulated as follows.

C = { the most general pattern };

while *C* ? f **do**

E = all suitable selected specializations of

elements of *C*;

for *q* ∈ *E* **do**

if *q* satisfies the interestingness criteria **then** output *q*;

if *q* is not promising **then** discard *q* **else** retain *q*

End;

additionally prune *E*;

End;

C = *E*;

end;

As instantiations of this algorithm, we get several more or less familiar methods:

1. Assume patterns are sets of variables, and define the interestingness and promise of an itemset *X* both by the predicate *fr*(*X*) > s. Do no additional pruning. Then this algorithm is in principle the same as the algorithm for finding association rules.
2. Suppose the patterns are rules of the form

$$a_1 ? \ldots ? a_k ? ß,$$

where a_i and ß are conditions of the form $X = c$, $X < c$, or $X > c$ for a variable X and a constant c. Let the interestingness criterion be that the rule is statistically significant in some sense and let the promise criteria be trivially true. The additional pruning step retains only one rule from *E*, the one with the highest statistical significance. This gives us a hill-climbing search for a rule with the highest statistical significance. (Of course, significance cannot be interpreted in a properly formal sense here because of the large number of interrelated tests.)

3. Suppose the interestingness criterion is that the rule is statistically significant, the promise test is trivially true, and the additional pruning retains the *K* rules whose significance is the highest. Above we had the case for *K* = 1; an arbitrary *K* gives us *beam search*.

13.6.3 Criteria for Interestingness

In the previous sections we referred to measures of interestingness for rules. Given a rule ? ? ? , its interestingness can be defined in many ways. Typically, background knowledge about the variables referred to in the patterns ? and ? have great influence in the interestingness of the rule. For example, in a credit scoring data set we might decide beforehand that rules connecting the month of birth and credit score are not interesting. Or, in a market basket database, we might say that the interest in a rule is directly proportional to the frequency of the rules multiplied by the prices of the items mentioned in the product, that is, we would be more interested in rules of high frequency that connect expensive items. Generally, there is no single method for automatically taking background knowledge into account, and rule discovery systems need to make it easy for the user to use such application-dependent criteria for interestingness.

Purely statistical criteria of interestingness are easier to use in an application-independent way. Perhaps the simplest criteria can be obtained by constructing a 2 × 2 contingency table by using the presence or absence of ? and ? as the variables, and having as the counts the frequencies of the four different combinations.

▪

	?	¬?
?	*fr*(? ? ?)	*fr*(? ? ¬?)
¬?	*fr*(¬? ? ?)	*fr*(¬? ? ¬?)

From the data in this table we can compute different types of measures of association between ? and ? , e.g., the $?^2$ score. One particularly useful measure of interestingness of a rule ? ? ? is the *J-measure*, defined as

(13.2) $$J(\theta \Rightarrow \varphi) = p(\theta)\left(p(\varphi|\theta)\log\frac{p(\varphi|\theta)}{p(\varphi)} + (1-p(\varphi|\theta))\log\frac{1-p(\varphi|\theta)}{1-p(\varphi)}\right).$$

Here, *p*(? |?) is the empirically observed confidence (accuracy) of the rule, and *p*(?) and *p*(?) are the empirically observed marginal probabilities of ? and ? respectively. This measure can be viewed as the cross-entropy between the binary variables defined by ? with and without conditioning on the event ?. The factor *p*(?) indicates how widely the rule is applicable. The other factor measures how dissimilar our knowledge about ? is from only knowing about the marginal *p*(?), compared and with knowing that ? holds, i.e., the conditional probability *p*(? |?). The *J*-measure has the advantage that it behaves well with respect to specializations, that is, it is possible to prove bounds on the value of the *J*-measure for specializations of a given rule.
In practice it has been found that different score functions for interestingness will often return largely the same patterns, as long as the score functions obey some basic properties (such as the score monotonically increasing as the frequency of a pattern increases, with the accuracy remaining constant). General issues relating to the "interestingness" of patterns were also discussed in chapter 7.

13.7 From Local Patterns to Global Models

Given a collection of patterns occurring in the data, is there a way of forming a global model using the patterns? In this section we briefly outline two ways of doing this. The first method forms a decision list or rule set for a classification task, and the second method constructs an approximation of the probability distribution using the frequencies of the patterns.

Let B be, for simplicity, a binary variable and suppose that we have a discovered a collection of rules of the form $?_i$? $B = 1$ and $?_i$? $B = 0$. How would we form a decision list for finding out or predicting the value of B? (A decision list for variable B is an ordered list of rules of the form $?_i$? $B = b_i$, where $?_i$ is a pattern and b_i is a possible value of B.) The accuracy of such a decision list can be defined as the fraction of rows for which the list gives the correct prediction. The optimal decision list could be constructed, in principle at least, by considering all possible orderings of the rules and checking for each one that produces the best solution. However, this would take exponential time in the number of rules. A relatively good approximation can be obtained by viewing the problem as a weighted set cover task and using the greedy algorithm.

That is one way to use local patterns to obtain information about the whole data set. Here is another. If we know that pattern $?_i$ has frequency $fr(?_i)$ for each $i = 1, ..., k$, how much information do we have about the joint distribution on all variables $A_1, ..., A_p$? In principle, any distribution f that satisfies the pattern frequencies could have generated the observations $fr(?_i)$. However, a reasonable model to adopt would be one that made no further assumptions about the general nature of the distribution (since nothing further is known). This would be the distribution that maximizes the entropy, subject to the pattern frequencies that have been observed. This distribution can be constructed reasonably efficiently using the iterative proportional fitting algorithm. Roughly speaking, this algorithm operates as follows. Start with a random distribution $p(\mathbf{x})$ on the variables A_j, and then enforce the frequency constraint for each pattern $?_i$. This is done by computing the sum of p over the states for which $?_i$ is true, and scaling these probabilities so that the resulting updated version of p has $?_i$ true in a set of measure $fr(?_i)$. The update step is carried out in turn for each pattern, until the observed frequencies of patterns agree with those provided by p. The method converges under fairly general conditions, and it is widely employed—for example, in statistical text modeling. The drawback of the method (at least if it is applied straightforwardly) is that it requires the construction of each of the states of the joint distribution, so that both the space and time complexity of the method are exponential in the number of variables.

13.8 Predictive Rule Induction

In this chapter we have so far focused primarily on association rules and similar rule formalisms. We began the chapter with a general definition of a rule, and we now return to this framework. Recall that we can interpret each branch of a classification tree as a rule, where the internal nodes on the path from the root to the leaf define the terms in the conjunctive left-hand side of the rule and the class label assigned to the leaf is the right-hand side. For classification problems the right-hand side of the rule will be of the form $C = c_k$, with a particular value being predicted for the class variable C.

Thus, we can consider our classification tree as consisting of a *set of rules*. This set has some rather specific properties—namely, it forms a mutually exclusive (disjoint) and exhaustive partition of the space of input variables. In this manner, any observation $\mathbf{x}$ will be classified by one and only one rule (namely the branch defining the region within which it lies). The set of rules is said to "cover" the input space in this manner.

We can see that it may be worth considering rule sets which are more general than tree-structured rule sets. The tree representation can be particularly inefficient (for example) at representing disjunctive Boolean functions. For example, consider the disjunctive mapping defined by $(A = 1$? $B = 1) \vee (D = 1$? $E = 1)$? $C = 1$ (and where $C = 0$ otherwise). We can represent this quite efficiently via the two rules $(A = 1$? $B = 1)$? $C = 1$ and $(D = 1$? $E = 1)$? $C = 1$. A tree representation of the same mapping would

necessarily involve a specific single root-node variable for all branches (e.g., A) even though this variable is relevant to only part of the mapping.

One technique for generating a rule set is to build a classification tree (using any of the techniques described in chapter 10) and then to treat each of the branches as individual candidate rules. Visiting each such rule in turn, the rule-induction algorithm determines whether each condition on the left-hand side of each rule affects the accuracy of that rule on the data that it "covers." For example, we could assess whether the accuracy of the rule (which is equivalent to the estimated conditional probability) improves for the better (or indeed shows no significant change at all) when a particular condition is removed from the left-hand side. If it improves or shows no change, the condition can be deemed not necessary and can be removed to yield a simpler and potentially more accurate rule. The process can be repeated until all conditions in all rules are examined. In practice this method has often been found to eliminate a large fraction of the initial rule conditions, conditions that are introduced in the tree-growing process because of their *average* contribution in terms of model improvement, but that are not necessary for some particular branches.

The final rule set produced in this manner is then used for classification. Since the original rules carved up the input space in a disjoint manner, and we have removed a subset of the conditions defining these disjoint regions, the boundaries have been broadened (the rules have been generalized), and these regions may now overlap. Thus, we might have two rules of the form $A = 1 \; ? \; C = 1$ and $B = 1 \; ? \; C = 1$. The natural question is: how do we use two such rules to classify a new observation vector $\mathbf{x}$ where both $A = 1$ and $B = 1$? One approach would be to view the two rules as constraints on the overall joint distribution of $p(A, B, C)$ and to infer an estimate for $p(C = 1 | A = 1, B = 1)$ using the maximum entropy approach described earlier in this chapter in section 13.7. However, since the maximum entropy approach is somewhat complex computationally, much simpler techniques tend to be used in practice. For example, we can find all the rules that "fire" (i.e., for which the conditions are satisfied) given the observation vector $\mathbf{x}$. If there is more than one rule, we can simply pick the one with the highest conditional probability. If no rules fire, we can simply pick the class value that is most likely a priori. Other more complex schemes are also possible, such as arranging the rules in an ordered decision list, or voting or averaging among multiple rules.

We might ask: why start with a classification tree and then produce rules, rather than search for the rules directly? One advantage of looking at classification trees is that it automatically quantizes any real-valued variables in a relatively simple and computationally efficient fashion during the tree-building phase (although, of course, these quanta need not necessarily be optimal in any sense in the context of the final rule set). Another advantage is the ease of implementing the technique; there are many efficient techniques for producing trees (both for data in main memory and in secondary memory, as discussed in chapter 10) and, thus, it is relatively straightforward to add the rule selection component as a "postprocessing" step.

Nonetheless, producing rules from trees imposes a particular bias on how we search for rules, and with this in mind, there has also been a great deal of work in machine learning and data mining on algorithms that search for rules directly, particularly for discrete-valued data. It is worth noting once again of course that the number of possible conjunctive rules is immense, $O(m^p)$ for p variables each taking m values. Thus, in searching for an optimal set of such rules (or even just the best single rule) we will usually resort to using some form of heuristic search through this space (as already pointed out in section 13.6 in the context of finding interesting sets of rules).

Note here that, in the context of classification, optimality should be defined as the set of rules that are most accurate on average on new data (or have the minimum average loss when classification costs are involved). However, just as with classification trees, classification accuracy on the training data need not be the best score function to guide in the selection of rules. For example, we could define a specific rule for each training example containing all of the variable values present in that example. Such a specific rule may have high accuracy (indeed, even accuracy 1 if all examples with the same variable values have the same class label), but may generalize poorly since it is so specific. Thus, score functions other than simple accuracy are often used in practice, particularly for selecting the next rule to add to the existing set, e.g., some trade-off

between the coverage of the rule (the probability of the left-hand side expression) and the rule accuracy, such as the J-measure described earlier.

Having defined a suitable score function, the next issue is how to search for a set of rules to optimize this score on the training data. Many rule induction algorithms utilize a form of "general-to-specific" heuristic, of the same general form described earlier in searching for interesting rules, where now we replace the interestingness score function with a classification-related one. These algorithms start with a set containing the most general rule possible (that is, the left-hand side is empty) and proceed to add rules to this set in a greedy fashion by successively exploring more specific versions of the rules in the existing set. This can be viewed as a systematic search through the space of all subsets, starting from the null set and using an operator that can add only one condition to a rule at a time. A large variety of search techniques are applicable here, including any of the systematic heuristic search techniques (such as beam search) discussed in chapter 8. The opposite heuristic strategy of starting from the most specific rules and generalizing is also possible, although computationally this tends to be a bit more tricky, since it is not so obvious what set of rules to start from. For real-valued data we can either pre-quantize each real-valued variable into bins (for example by using a clustering algorithm on each variable), or quantize as one searches for rules. The latter option can be computationally quite demanding and tricky to implement; an interesting algorithm which operates in this manner is the PRIM algorithm (Friedman and Fisher, (1999)), that gradually "shrinks" the rule regions starting from the full range of the data for each variable.

There is of course a trade-off between the more computationally (and memory) intensive search techniques which search more of the rule space, and the simpler techniques that can search only a smaller fraction of the space. In practice, as with classification trees, relatively simple greedy search techniques with simple operators often seem to perform almost as well empirically as the more complex methods and are quite popular as a result. As with classification trees, there is also the problem of deciding when to stop adding rules to the rule set (the familiar problem of deciding how complex the model should be—here we can interpret our set of rules as a "model" for the data). Once again, the technique of cross-validation can be quite useful in estimating the true predictive accuracy of a set of rules, but again it can be quite computationally intensive, particularly if it is invoked repeatedly at various stages of the rule search.

We conclude this discussion on predictive rules by mentioning a few notable extensions to the basic classification paradigm. The first extension is that, just as we can extend the ideas of classification trees to produce regression trees, so we can also perform *rule-based regression*. The left-hand side condition of a rule defines a particular region of the input space. Given this region we can then estimate a local regression model on the data in this region (it can be as simple as the best-fitting constant for example). If the rules are disjoint we get a piecewise local regression surface; if the rules overlap we must again decide how to combine the various rule predictions in the overlapping regions. One particular advantage of the rule-based regression framework is the ease of interpretability, particularly in high-dimensional problems, since only a small fraction of the variables are often selected as being relevant for inclusion in the rules.

The second notable extension to the basic rule induction paradigm is that of using relational logic as the basis for the rules. A discussion in any depth of this topic is beyond the scope of this text, but essentially the idea is to generalize beyond the notion of propositional logic statements ("Variable = value") to what are known as first-order relational logic statements such as "Parent(X, Y) ? Male(X) ? Father (X, Y)." This type of learning is, in principle, extremely powerful, since it allows a much richer representational language to describe our data. A propositional version of a relational statement is typically quite awkward (and can be exponentially large), since there is no notion in the (simpler) propositional framework of objects and relations among objects. The extra representational power of relational logic comes at a cost of course, both in terms of reasoning with such rules and in terms of learning them from data. Algorithms for learning relational rules have been developed under the title "inductive logic programming," with some promising results, although largely on logical rather than probabilistic representations for data.