****DRAFT**** ALGORITHMS FOR ASSOCIATION RULES

Markus Hegland
Mathematical Sciences Institute, Australian National University
John Dedman Building, Canberra ACT 0200, Australia
E-mail: Markus.Heglananu.edu.au

Association rules are "if-then rules" with two measures which quantify the support and confidence of the rule for a given data set. Having their origin in market basked analysis, association rules are now one of the most popular tools in data mining. This popularity is to a large part due to the availability of efficient algorithms following from the development of the Apriori algorithm.

In these lectures we will introduce basic concepts of association rule discovery including support, confidence, interestingness, the apriori property, and hierarchical and quantitative association rules. The core of the lectures consists of a review of the most important algorithms for association rule discovery. Some familiarity with concepts like predicates, probability, expectation and random variables is assumed.

1. Introduction

Data mining work has a long tradition where data mining refers to computational, often exploratory, approaches in data analysis. Early methods which are now considered core topics in data mining include methods for clustering (like the k-means algorithm), classification and regression trees, classification rules and artificial neural nets. Regular data mining conferences started to apear in the late 1980s. However, the introduction of association rule mining in in 1993 by Agrawal, Imielinski and Swami [2] and, in particular the publication of an efficient algorithm by Agrawal and Srikant [3] and, independently by Mannila, Toivonen and Verkamo [9] opened the field to a wider audience and, in particular, to the data base community. Research in data mining on computational efficiency, the interface between computational methods and data access, applications in science and business and

relations with statistical analysis have strengthened the young discipline and helped it establish itself as an important and exciting research area in computer science and data processing.

An association rule is an *implication* or *if-then-rule* which is supported by data. The motivation given in [2] for the development of association rules is *market basket analysis* which deals with the contents of point-of-sale transactions of large retailers. A typical association rule resulting from such a study could be "90 percent of all customers who buy bread and butter also buy milk". While such insights into customer behaviour may also be obtained through customer surveys, the analysis of the transactional data has the advantage of being much cheaper and covering all current customers. The disadvantage compared to customer surveys is in the limitation of the given transactional data set. For example, point-of-sale data typically does not contain any information about personal interests, age and occupation of customers.

Understanding the customer is core to business and ultimately may lead to higher profits through better customer relations, customer retention, better product placements, product development but also fraud detection. While originating from retail, association rule discovery has also been applied to other business data sets including

- credit card transactions
- telecommunication service purchases
- banking services
- insurance claims and
- medical patient histories.

However, the usefulness of association rule mining is not limited to business applications. It has also been applied in genomics and text (web page) analysis.

In these and many other areas, association rule mining has lead to new insights and new business opportunities. Of course the concept of a market basket needs to be generalised for these applications. For example, a market basket is replaced by the collection of medical services received by a patient during an episode of care, the subsequence of a sequence of amino acids of a protein or the set of words or concepts used in a web page. Thus when applying association rule mining to new areas one faces two core questions:

- what are the "items" and
- what are the "market baskets".

The answer of these questions is facilitated if one has an abstract mathematical notion of items and market baskets. This will be developed in the next section.

The efficiency of the algorithms will depend on the particular characteristics of the data sets. An important feature of the retailer data sets is that they contain a very large number of items (tens of thousands) but every market basket typically contains only a small subset.

A simple example in Table **??** illustrates the type of results association rule mining produces. From the table one can induce the following "rules":

market basket id	market basket content
1	orange juice, soda water
2	milk, orange juice, bread
3	orange juice, butter
4	orange juice, bread, soda water
5	bread

- 80 % of all transactions contain orange juice (1,2,3,4)
- 40 % of all transactions contain soda water (1,4)
- 50 % of the transactions containing orange juice also contain soda water (1,4) while
- all transactions which contain soda water also contain orange juice

These rules are very simple which is typical for association rule mining. Only simple rules will ultimately be understandable and useful. However, the determination of these rules is a major challenge due to the very large data sets and the large number of potential rules.

Large amounts of data have been collected routinely in the course of day-to-day management in businesss, administration, banking, the delivery of social and health services, environmental protection, policing and in politics. This data is primarily used for accounting and management of the customer base. However, it is also one of the major assets to the owner as it contains a wealth of knowledge about the customers which can assist in the development of marketing strategies, political campaigns, policies and product quality controll. Data mining techniques help process this data which is often huge, constantly growing and complex. The discovered patterns point to underlying mechanisms which help understand the customers and can give leads to better customer satisfaction and relations.

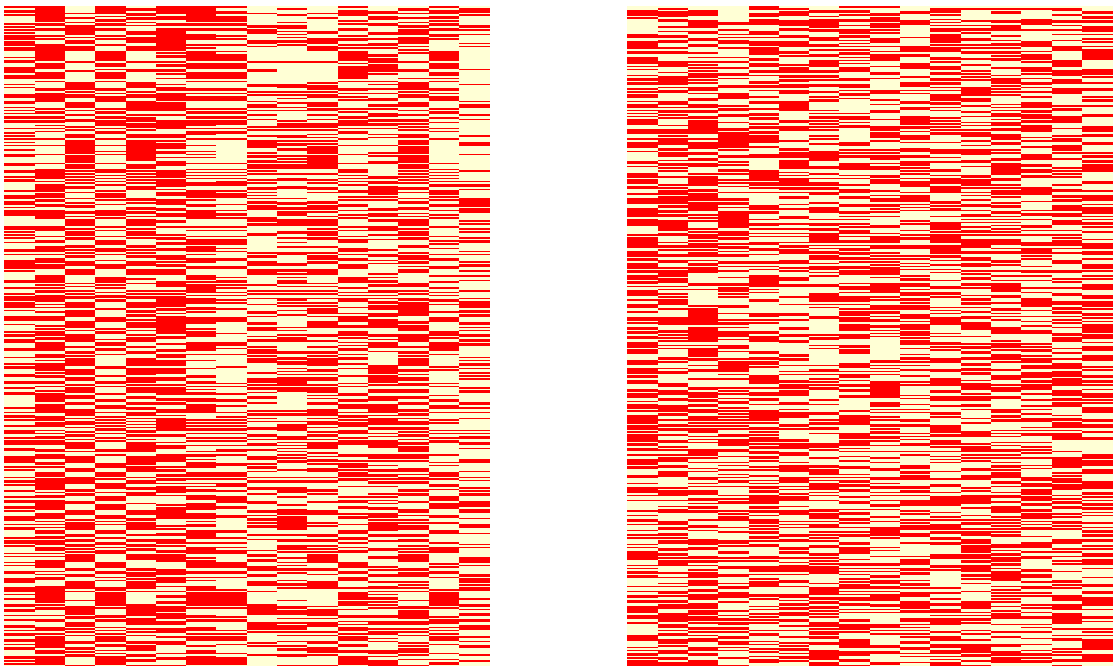

Fig. 1. 1984 US House of Representatives Votes, with 16 items voted on

As an illustrative (small) example consider the US Congress voting records from 1984 [8], see figure 1. The 16 columns of the displayed bit matrix correspond to the 16 votes and the 435 rows to the members of congress. We have simplified the data slightly so that a matrix element is one (pixel set) in the case of votes which contains "voted for", "paired for" and "announced for" and the matrix element is zero in all other cases. Data mining aims to discover patterns in this bit matrix. In particular, we will find columns or items which display similar voting patterns and we aim to discover rules relating to the items which hold for a large proportion of members of congress. We will see how many of these rules can be explained by underlying mechanisms (in this case party membership).

The choice of the columns and the rows is in this example somewhat arbitrary. One might actually be interested on patterns which relate to the members of congress. For example one might be interested in statements like "if member x and member y vote yes then member z votes yes as well. Statements like this may reveal some of the connections between the members of congress. For this the data matrix is thus "transposed". The duality of observations and objects occurs in other areas of data mining as well and illustrates that data size and data complexity are really two dual concepts which can be interchanged. This is in particular exploited in some newer association rule discovery algorithms which are related to formal concept analysis [6].

One now finds that the items have received between 34 to 63.5 percent yes votes. Pairs of items have received between 4 and 49 percent yes votes. The pairs with the most yes votes (over 45 percent) are in the columns 2/6, 4/6, 13/15, 13/16 and 15/16. Some rules obtained for these pairs are: 92 percent of the yes votes in column 2 are also yes votes in column 6, 86 percent of the yes votes in column 4 are also yes votes in column 6 and, on the other side, 88 percent of the votes in column 13 are also in column 15 and 89 percent of the yes votes in column 16 are also yes votes in column 15. These figures suggest combinations of items which could be further investigated in terms of causal relationships between the items. Only a careful statistical analysis may provide some certainty on this. This, however, is beyond the scope of these lectures.

2. Mathematical Models

Association rule discovery has originated in market basket analysis. Here the object is a market basket of items purchased by a customer. While many features may be of interest in market basket analysis, the main features studied are the types of items in the market basket.

The market basket example is just one incidence where association rule discovery is used. In general, it is used whenever the objects are sets of items, and, more generally, a collection of properties $X_i(\omega)$ of the objects, i.e., statements which are either true or false. In all these cases the features take the form of bitvectors.

2.1. *Modelling the Data*

Thus one has a set of possible items which we number and represent by their number. For example, a "micromarket" may sell the following items which

are given together with a map into integers: This allocation is somewhat

item	item number
juice	0
bread	1
milk	2
cheese	3
potatos	4

arbitrary and for particular applications one may wish to choose specific applications, for example one may order the items such that the itemnumbers indicate the rank in which the items were purchased, in the case of the micromarket juice would be the most frequently purchased, followed by bread etc.

More generally, we map the items onto integers $0, \ldots, d-1$. Any market basket is a set of items and we will model them as a set of integers $x \in Z_d = \{0, \ldots, d-1\}$. Thus the mathematical problem of finding association rules from data consists of finding structure in a sequence of subsets of Z_d. So the set $\mathbb{X}$ of features of objects is the powerset of Z_d.

Sets can be represented in various ways and all representations have their strengths. The representation of a collection of integers is very intuitive and compact. An alternative representation are bitvectors. With any subset of Z_d one associates a vector of length d containing except at the positions indicated by the elements of the subset where the component of the vector is one. So, for example, in the case of the microshop items one may have a market basket $\{bread, milk, potatos\}$ which is mapped onto the set $\{1, 2, 4\}$ which corresponds to the bitvector $(0, 1, 1, 0, 1)$. Thus in a bitvector not only can one easily extract what the elements are but also which elements are not contained in the set.

Using this representation we define our set of features to be

$$\mathbb{X} = \{0, 1\}^d$$

but occasionally will refer to the vectors as itemsets. The underlying probability space is discrete. The size of an itemset will be important as it gives a complexity measure. Using the bitvector representation, the size of x is then

$$|x| = \sum_{i=0}^{d-1} x_i.$$

In some cases one would like to enumerate the sets. The bitvector can be interpreted as the vector of bits of an integer and one thus has the following mapping of bitvectors (or itemsets) onto integers:

$$\phi(x) = \sum_{i=0}^{d-1} x_i 2^i.$$

Note that this actually gives a bijective map between the set of finite sets of integers onto the set of integers. Table **??** shows a collection of micromarket itemsets, the corresponding set of integers, the representing bitvector and the number in which this subset occurs in the collection of all subsets.

itemset	set of numbers	bitvector	number of subset
{juice, bread, milk }	$\{0, 1, 2\}$	$(1, 1, 1, 0, 0)$	7
{ potatos }	$\{4\}$	$(0, 0, 0, 0, 1)$	16
{bread, potatos }	$\{1, 4\}$	$(0, 1, 0, 0, 1)$	18

The data is thus represented by a sequence of bitvectors or a Boolean matrix. For example, corresponding to table **??** with the micromarket items one gets the matrix

$$\begin{bmatrix} 1\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1 \\ 0\,1\,0\,1\,0 \end{bmatrix}.$$

In the example of congressional voting the first few records are

```
1   0  1  1  1  1   0  1   0  1  0   0   1   1  0  0
1   1  1  0  0  1   0  1   1  1  1   0   1   1  0  0
0   0  0  1  1  1   0  1   1  0  0   0   0   0  1  0
0   1  0  1  0  1   0  1   1  0  1   1   0   0  1  0
```

or the same as sets

```
1  3  4  5  6  8  10  13  14
1  2  3  6  8  9  10  11  13  14
4  5  6  8  9  15
2  4  6  8  9  11 12  15
1  3  7  9 10  11 13  16
```

2.2. *Modelling the Patterns in the Database*

The structure association rules model in these matrices stems from the similarity of the rows $x^{(i)}$ which is based on some underlying mechanism. We will now illustrate this for the case of the congressional data. If there is no underlying mechanism, i.e., the samples are generated independent random components then the data mining algorithm should not find any structure. (As any "discovered patterns" would purely reflect the sample and not the underlying distribution.) If we take to rows at random then they are modelled by two random vectors $X^{(i)}$ and $X^{(j)}$. Even when they are at random, there will still be some components which these vectors share. The number of differing components is also a random variable. This is defined by the *Hamming distance*:

$$d_H(x, y) = \sum_{i=1}^{d} |x_i - y_i|.$$

If the probability that any component is one is p then the probabilty that this component is the same for both vectors is $p_2 := p^2 + (1-p)^2$. The number of different components is the sum of i.i.d. binary random variables and thus it has a binomial distribution with expectation dp_2 and variance $dp_2(1-p_2)$. Note that the ratio of the standard deviation (or square root of the variance) to the expectation goes to zero when d goes to infinity. This is an example of the concentration effect or curse of dimensionality. In figure 2 we display the distribution of the the actual distances together with the display of the binomial distribution of distances in the random case (upper two histograms). Clearly, the histogram in figure 2 shows that we do not have the random situation and thus we may expect to find meaningful patterns in the data. These patterns will often relate to the party membership as is shown in the lower two histograms, in the case of the same party one gets a random distribution with lower expectation and in the case of different parties one with higher expectation. In the distribution of all the differences in voting behaviour one has both these effects and thus a much broader distribution as in the random case. This example is not so exciting as it is expected that differences accross parties are larger than differences within a party. Such a "discovery" would be judged to be less than exciting. We have chosen this example, however, as it illustrates the case of an underlying "hidden" variable which may be able to explain the distribution. In practical cases both known and unknown such hidden mechanims are common.

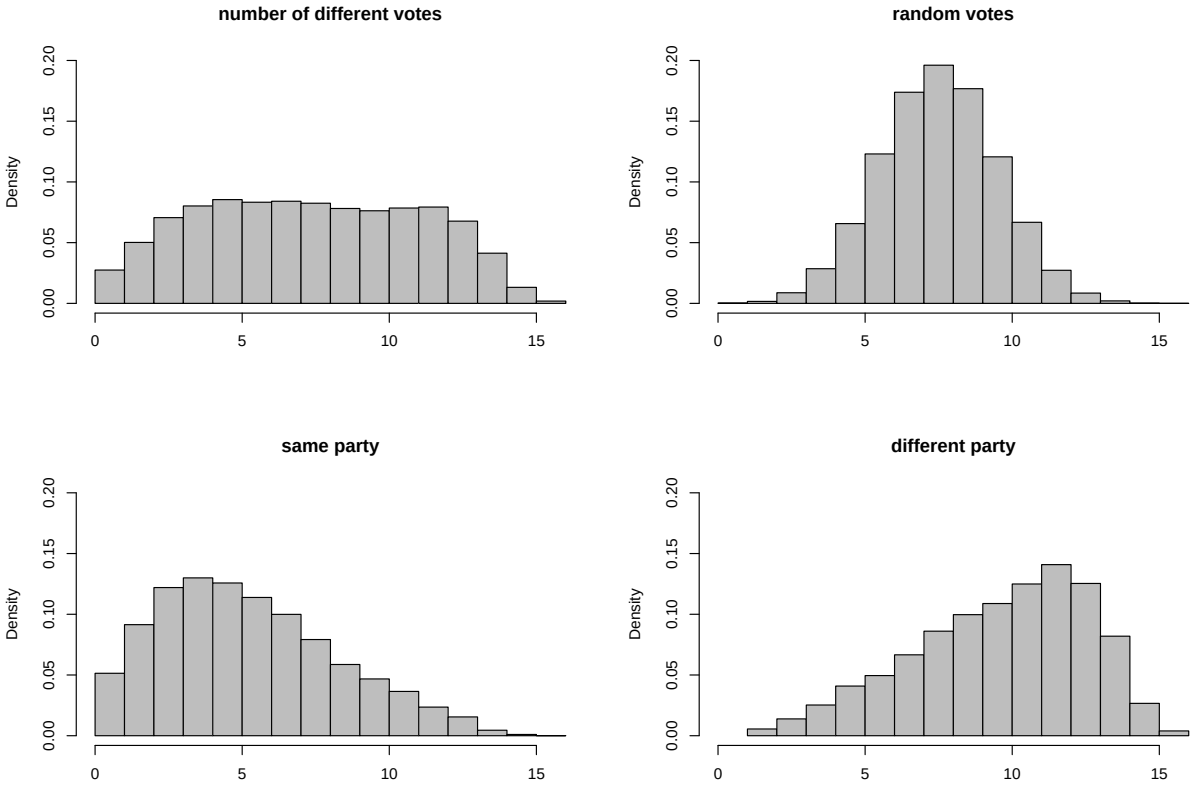

Fig. 2. Distribution of distances of two randomly chosen records in the congress voting data, compared to the binomial distribution and the distribution of the distances in the case of the same party and different parties.

In the case of the biological data the same distribution together with the binomial one is displayed in figure 3. The data set has much more noise and only a few data points which is a big problem, however, it appears that the distribution of the actual distances is still wider than the random case and thus on may be able to find some patterns. The "hidden" variable in this case is the occurrence of tumor and it is conjectured that different genes are active for tumor cells than for others, in fact, this is what some think could be used to find tumor. We can expect from this first display that finding meaningful and correct patterns will be much more difficult for this data set.

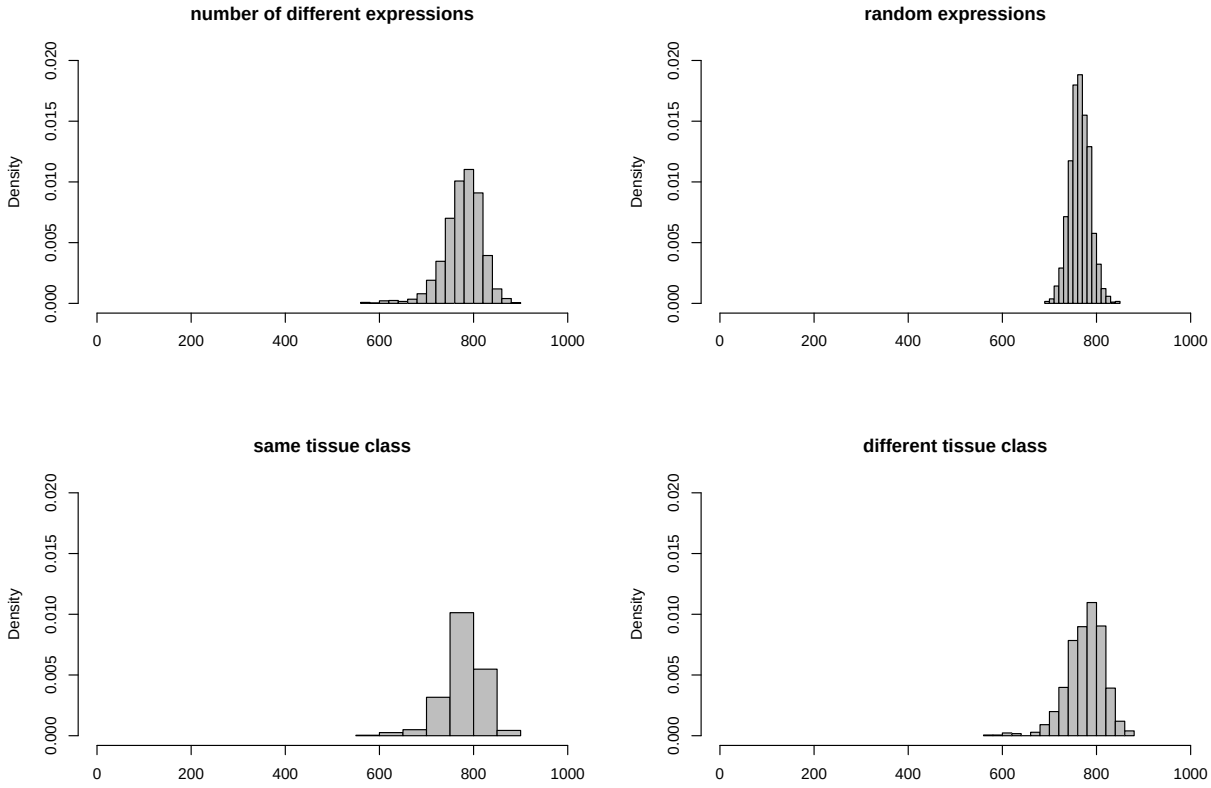

Fig. 3. Distribution of distances of two randomly chosen records in the tumor data.

In association rule discovery, one would like to make statements about the objects which are not necessarily universally true but which hold for a good proportion of the data. More specifically, one is interested to find mappings $a : \mathbb{X} \to \{0, 1\}$ which are true on a subset of $\mathbb{X}$ with large measure, or equivalently, one would like to find subsets of $\mathbb{X}$ which have a large probability.

The set S_a associated with a predicate a is its *support*, i.e., the subset of $\mathbb{X}$ where the predicate is one:

$$S_a = \text{supp}(a) = \{x | a(x) \neq 0\}.$$

Unfortunately, the probability $P(S_a)$ of the support is also called support

we call it:

$$s(a) = P(S_a).$$

We will be looking for statements, predicates a, or alternatively, for subsets S_a for which the support is high, say, more than a given lower bound σ which is dictated by the application. Thus we are looking for a with $s(a) \geq \sigma$. There are trivial examples which satisfy this, including the case $a = 1$ where $S_a = \mathbb{X}$ or the case where $S_a = \{x^{(1)}, \ldots, x^{(N)}\}$ being the data set. In both cases one would have a support $s(a) = 1$ or at least close to one. One thus needs to constrain the addimissible predicates a to ones which are interesting such that these cases are eliminated. Basically, one would like to have small and simple sets (a la Occam's razor) with a large support.

A problem related to the curse of dimensionality is that the total number of predicates is very large. As a predicate can be identified with its support the total set of predicates is equivalent to the powerset of $\mathbb{X}$. As the size of $\mathbb{X}$ is 2^d the size of the powerset of $\mathbb{X}$ is 2^{2^d}. Finding all the predicates which have at least support σ seems to be an unsurmountable task. Thus again, one needs to constrain the types of predicates.

On the one side, many predicates will not be of interest, on the other side many of the predicates found will be very similar to other ones found, and, say, differ just for one $\mathbb{X}$. One could thus introduce similarity classes of predicates. First, however, we can identify a generic property which the predicates we are interested in should have. For this we go back to the market basket example again. A property of a market basket relates to the contents, the price etc. If a market basket contains certain items then a larger market basket will contain the same items plus some more and thus, for the purposes of marketing the larger market basket would be just as interesting as the smaller one. Thus we would like our predicates to be *monotone* in x, i.e., if $x \leq y$ one has $a(x) \leq a(y)$. Of course this is not allways the case but this constraint does reduce the number of possible predicates a lot. So now we have the data mining problem: *Find all monotone predicates a which are frequent.* In order to do this we will have to understand more about the structure of the monotone predicates which we will do in a later section. Note that this monotonicity constraint does depend on the application and in other cases one might use different constraints.

The simplest case of such monotone constrains are given by the function which extracts the i-th component of the vector x, i.e.,

$$a_i(x) = x_i.$$

In the case of the US cogress votes the support for all the single votes is displayed in figure 4.

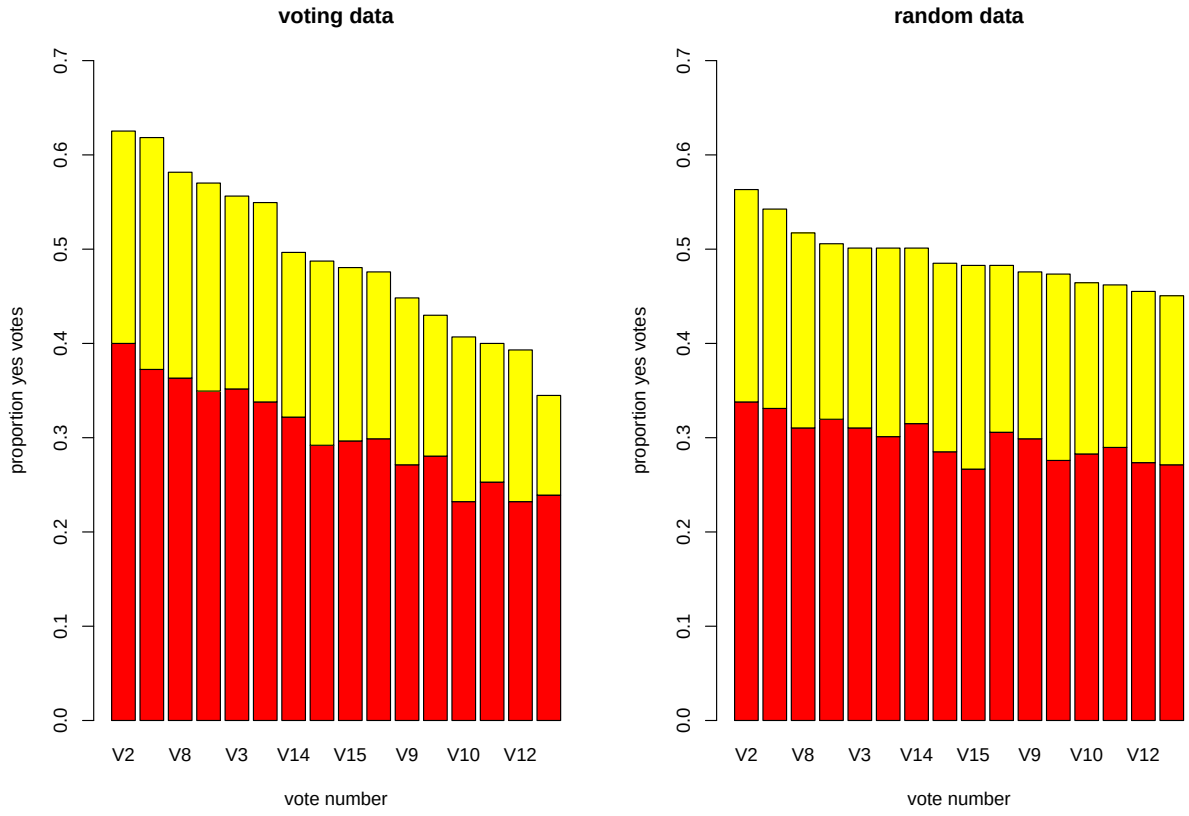

Fig. 4. Supports for all the votes in the US congress data (split by party).

A different, slightly more complex case is the product (or and) of two components:

$$a_{i,j}(x) = x_{i,j}.$$

For the voting example lets consider votes of the the type "V13 and Vx". Here the distributions do look quite different. figure 5. In particular, while the distribution of the single votes did not show us much about the parties, the combination certainly does. We can see that there are votes which

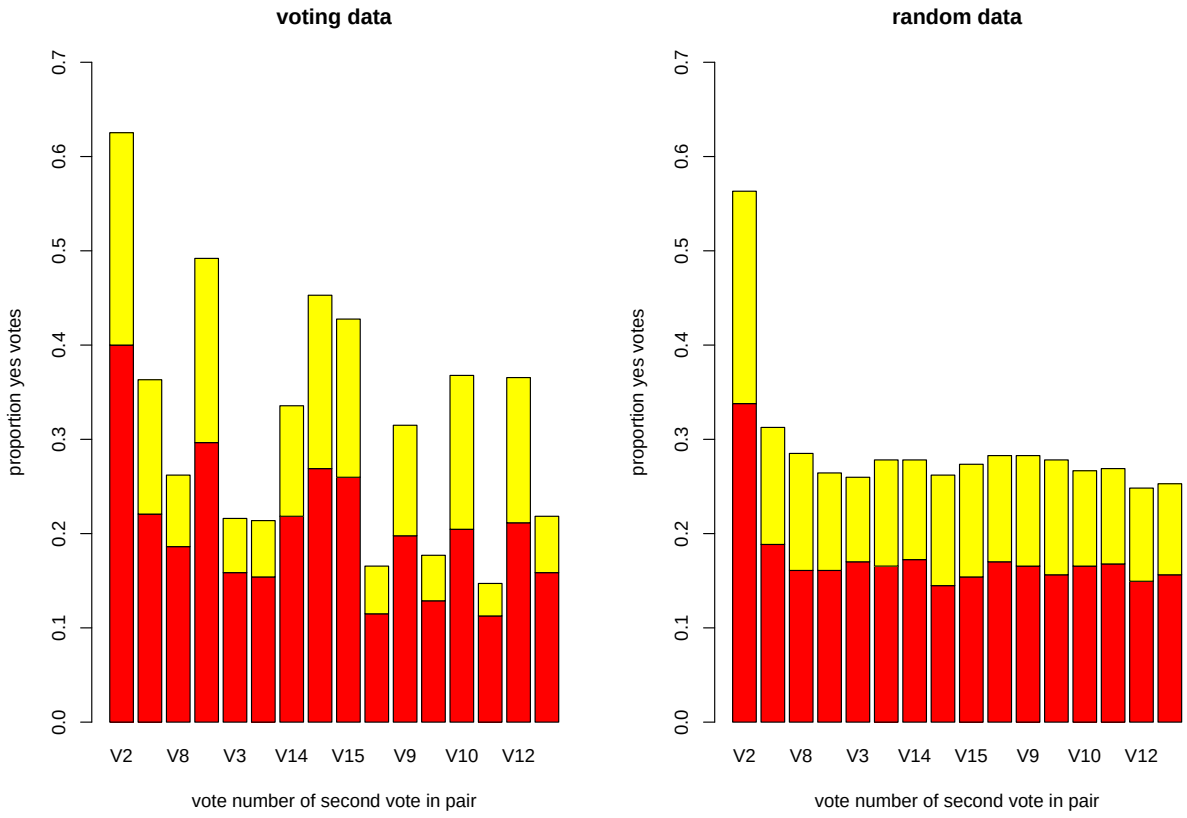

Fig. 5. Supports for pairs of votes in the US congress data (split by party).

do have similar voting behaviours. These similarities seem to go beyond party boundaries, i.e., high supports of a combination of two votes often means high supports from both parties. So maybe there is an underlying mechanism which is more fundamental than the party membership. In the random case there is only the case where the first vote is combined with itself (which actually is only one vote) and the case where two votes are combined.

Finally, the confidences of the rules "if voted yes for V13 then voted yes for Vx" is displayed in figure 6.

So far we have only considered structure which can be described by one predicate. However, one is also interested in multiple predicates, in

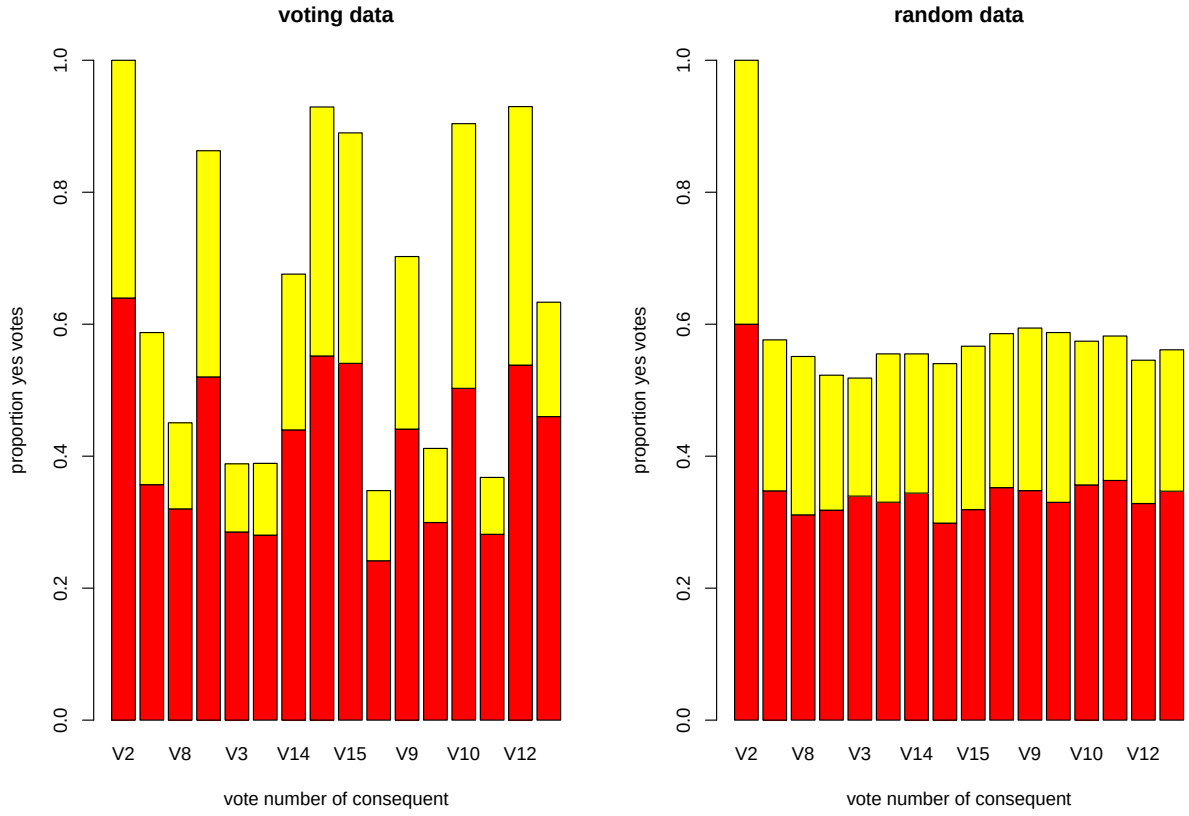

Fig. 6. Confidence of rule "if V13 then Vx" for the US congress data (split by party).

particular, pairs of predicates which interact. These are the *association rules* which have initiated the work in this area. They are described by a pair of predicates a, b of which the conjunction has a good support, and for which the conjunction has a good conditional support in the support S_a. This conditional support is also called confidence. Again, one is interested in monotone predicates a and b. The association rules intuitively model "if-then-rules", are denoted by $a \Rightarrow b$ and a is called antecedent and b is the consequent.

Note that these "if-then-rules have nothing to do with implications in classical propositional logic where an implication is equivalent to $\neg a \vee b$.

So an association rule is a pair of predicates with two numbers, the

support $s(a \Rightarrow b) = s(a \wedge b)$ and the *confidence* $c(a \Rightarrow b) = s(a\text{and } b)/s(a)$. Thus one has

Definition 1: An association rule, denoted by $a \Rightarrow b$ consists of a pair of monotone predicates a and b and a measure of support s and a measure of confidence c such that $s(a \Rightarrow b) = s(a \wedge b)$ and $c(a \Rightarrow b) = s(a \wedge b)/s(a)$.

In the following we will discuss algorithms for two data mining problems for any given probability space $\mathcal{P}(\mathbb{X}, 2^{\mathbb{X}}, P)$:

- Frequent set mining problem. For given $\sigma > 0$ find all (all) monotone predicates a such that $s(a) \geq \sigma$.
- Association rule mining problem. For given $\sigma > 0$ and $\gamma > 0$ find all association rules $a \Rightarrow b$ for which $s(a \Rightarrow b) \geq \sigma$ and $c(a \Rightarrow b) \geq \gamma$.

In addition to these questions in particular other types of constraints for the association rules have been considered.

Thus we now have a complete model of the data and the research questions we can now move on to discussing the structure of the data and pattern space which will form the basis for the development of efficient algorithms.

2.3. *Structure of* $\mathbb{X}$

In order to systematically search throught the set $\mathbb{X}$ to find the y mentioned we will need to exploit the structure of $\mathbb{X}$. As $\mathbb{X}$ consists of sets, one has a natural *partial ordering* which is induced by the subset relation. In terms of the components one can define this componentwise as

$$x \leq y :\Leftrightarrow x_i \leq y_i.$$

Thus if, for any i, x contains i then $x_i = 1$ and, if $x \leq y$ then $y_i = 1$ and so y contains i as well which shows that $x \leq y$ means that x is a subset of y. We use the symbol $\leq$ here instead of $\subset$ as we will use the $\subset$ symbol for sets of itemsets.

Of course, subsets have at most the same number of elements as their supersets, i.e., if $x \leq y$ then $|x| \leq |y|$. Thus the size of a set is a *monotone function* of the set wrt the partial order. One can also easily see that the function ϕ is monotone as well. Now as ϕ is a bijection it induces a total order on $\mathbb{X}$ defined as $x \prec y$ iff $\phi(x) < \phi(y)$. This is the *colex order* and the colex order extends the partial order.

The partial order (and the colex) order have a smallest element which consists of the empty set, as bitvector $x = (0, \ldots, 0)$ and a largest element

wich is just the set of all items Z_d, as bitvector $(1, \ldots, 1)$. Furthermore, for each pair $x, y \in \mathbb{X}$ there is a greatest lower bound and a least upper bound. These are just

$$x \vee y = z$$

where $z_i = \max\{x_i, y_i\}$ for $i = 0, \ldots, d-1$ and similar for $x \wedge y$. Basically, the partial order forms a Boolean lattice. We denote the maximal and minimal elements by 1 and 0.

More generally, a partially ordered set is defined as

Definition 2: A partially ordered set $(\mathbb{X}, \leq)$ consists of a set $\mathbb{X}$ with a binary relation $\leq$ such that for all $x, x', x'' \in \mathbb{X}$:

- $x \leq x$ (reflexivity)
- If $x \leq x'$ and $x' \leq x$ then $x = x'$ (antisymmetry)
- If $x \leq x'$ and $x' \leq x''$ then $x \leq x''$ (transitivity)

A lattice is a partially ordered set with glb and lub:

Definition 3: A lattice $(\mathbb{X}, \leq)$ is a partially ordered set such that for each pair of of elements of $\mathbb{X}$ there is greatest lower bound and a least upper bound.

We will call a lattice *distributive* if the distributive law holds:

$$x \wedge (x' \vee x'') = (x \wedge x') \vee (x \wedge x'').$$

where $x \vee y$ is the maximum of the two elements which contains ones whenever at least one of the two elements and $x \wedge x'$ contains ones where both elements contain a one. Then we can define:

Definition 4: A lattice $(\mathbb{X}, \leq)$ is a Boolean lattice if

(1) $(\mathbb{X}, \leq)$ is distributive
(2) It has a maximal element 1 and a minimal element 0 such that for all $x \in \mathbb{X}$:

$$0 \leq x \leq 1.$$

(3) Each element x as a (unique) complement x' such that $x \wedge x' = 0$ and $x \vee x' = 1$.

The maximal and minimal elements 0 and 1 satisfy $0 \vee x = x$ and $1 \wedge x = x$. In algebra one considers the properties of the conjunctives $\vee$ and $\wedge$ and a set which has conjunctives which have the properties of a Boolean lattice

is called a *Boolean algebra*. We will now consider some of the properties of Boolean algebras.

The smallest nontrivial elements of $\mathbb{X}$ are the *atoms*:

Definition 5: The set of atoms $\mathcal{A}$ of a lattice is defined by

$$\mathcal{A} = \{x \in \mathbb{X} | x \neq 0 \text{ and if } x' \leq x \text{ then } x' = x\}.$$

The atoms generate the lattice, in particular, one has:

Lemma 6: *Let $\mathbb{X}$ be a finite Boolean lattice. Then, for each $x \in \mathbb{X}$ one has*

$$x = \bigvee \{z \in \mathcal{A}(\mathbb{X}) \mid z \leq x\}.$$

Proof: Let $A_x := \{z \in \mathcal{A}(B) | z \leq x\}$. Thus x is an upper bound for A_x, i.e., $\bigvee A_x \leq x$.

Now let y be any upper bound for A_x, i.e., $\bigvee A_x \leq y$. We need to show that $x \leq y$.

Consider $x \wedge y'$. If this is 0 then from distributivity one gets $x = (x \wedge y) \vee (x \wedge y') = x \wedge y \leq y$. Conversely, if it is not true that $x \leq y$ then $x \wedge y' > 0$. This happens if what we would like to show doesn't hold.

In this case there is an atom $z \leq x \wedge y'$ and it follows that $z \in A_x$. As y is an upper bound we have $y \geq z$ and so $0 = y \wedge y' \geq x \wedge y'$ which is impossible as we assumed $x \wedge y' > 0$. Thus it follows that $x \leq y$. □

The set atoms associated with any element is unique, and the Boolean lattice itself is isomorph to the set of sets of atoms. This is the key structural theorem of Boolean lattices and is the reason why we can talk about sets (itemsets) in general for association rule discovery.

Theorem 7: A finite Boolean algebra $\mathbb{X}$ is isomorphic to the power set $2^{\mathcal{A}(\mathbb{X})}$ of the set of atoms. The isomorphism is given by

$$\eta : x \in \mathbb{X} \mapsto \{z \in \mathcal{A}(\mathbb{X}) \mid z \leq x\}$$

and the inverse is

$$\eta^{-1} : S \mapsto \bigvee S.$$

In our case the atoms are the d basis vectors $e_1, \ldots, e_d$ and any element of $\mathbb{X}$ can be represented as a set of basis vectors, in particular $x = \sum_{i=1}^{d} \xi_i e_i$ where $\xi_i \in \{0, 1\}$. For the proof of the above theorem and further information on lattices and partially ordered sets see [5]. The significance of the

lemma lays in the fact that if $\mathbb{X}$ is an arbitrary Boolean lattice it is equivalent to the powerset of atoms (which can be represented by bitvectors) and so one can find association rules on any Boolean lattice which conceptially generalises the association rule algorithms.

In figure 7 we show the lattice of patterns for a simple market basket case which is just a power set. The corresponding lattice for the bitvectors

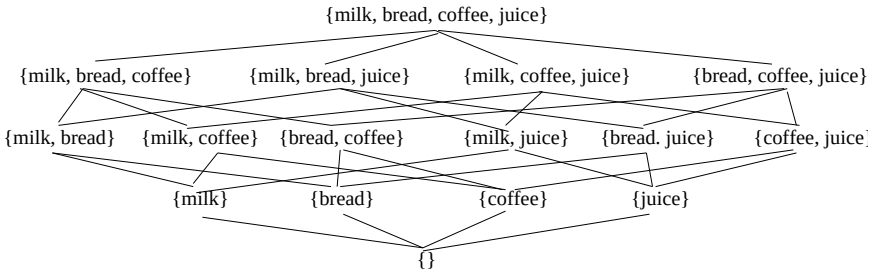

Fig. 7. Lattice of breakfast itemsets

is in figure 8. We represent the lattice using an undirected graph where the

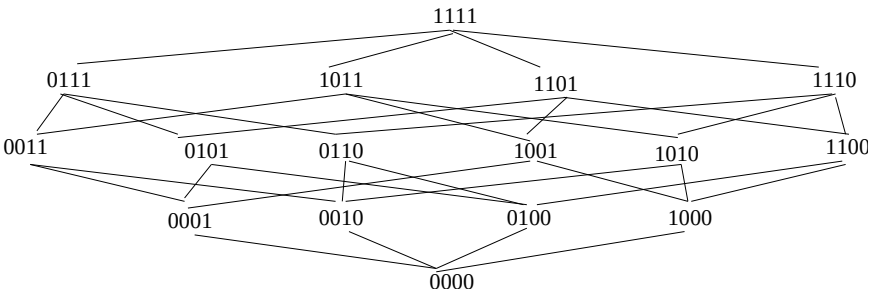

Fig. 8. Lattice of bitvectors.

nodes are the elements of $\mathbb{X}$ and edges are introduced between any element and its *covering* elements. A covering element of $x \in \mathbb{X}$ is an $x' \geq x$ such that no element is "in between" x and x', i.e., any element x'' with $x'' \geq x$ and $x' \geq x''$ is either equal to x or to x'.

In Figure 9 we display the graphs of the first few Boolean lattices. We will graph specific lattices with (Hasse) diagrams [5] and later use the positive plane $\mathbb{R}^2_+$ to illustrate general aspects of the lattices.

In the following we may sometimes also refer to the elements x of $\mathbb{X}$

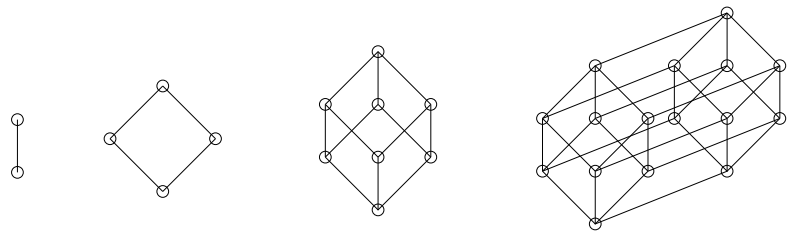

Fig. 9. The first Boolean Lattices

as item sets, market baskets or even patterns depending on the context. As the data set is a finite collection of elements of a lattice the closure of this collection with respect to $\wedge$ and $\vee$ (the inf and sup operators) in the lattice forms again a boolean lattice. The powerset of the set of elements of this lattice is then the sigma algebra which is fundamental to the measure which is defined by the data.

The partial order in the set $\mathbb{X}$ allows us to introduce the *cumulative distribution function* as the probability that we observe a bitvector less than a given $x \in \mathbb{X}$:

$$F(x) = P\left(\{x' | x' \leq x\}\right).$$

By definition, the cumulative distribution function is monotone, i.e.

$$x \leq x' \;\Rightarrow\; F(x) \leq F(x').$$

A second cumulative distribution function is obained from the dual order as:

$$F^{\partial}(x) = P\left(\{x' | x' \leq x\}\right).$$

It turns out that this dual cumulative distribution function is the one which is more useful in the discussion of association rules and frequent itemsets.

Typically, the set $\mathbb{X}$ is very large, for example, with $10,000$ items one can in principle have $2^{10,000} \approx 10^{3,000}$ possible market baskets. It can be said with certainty that not all possible market baskets will occur, and, in particular, one would only be interested in market baskets with a limited number of items, say, with less than 100 items.

In summary, the data is a set of points, see 10. For any given number of items d and number of records or data points n the number of possible data sets is very large, for example, there are $10^{3 \cdot 10^9}$ possible data sets with $n = 10^6$ market baskets containing a items from a collection of $10,000$ items. It is thus important to be very systematic when one would like to

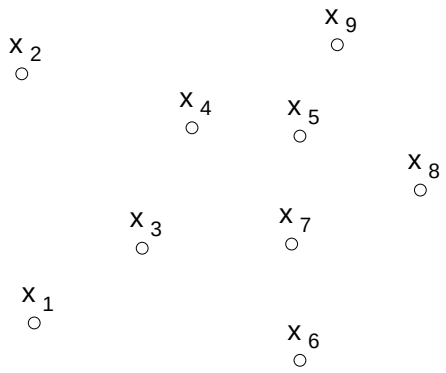

Fig. 10. Data points

compare different data sets and determine their essential properties. Now this complexity is a bit misleading, as we have not taken into account that mostly simple market baskets will occur. Note, however, that 10^6 market baskets and $10,000$ items corresponds to a relatively "small scenario".

2.4. *Structure of the Predicates on* $\mathbb{X}$

We would like to make statements about the x which would hold in with some probability. These probable statements are one type of structure we would like to discover for a given distribution. The statements are formulated as predicates

$$a : \mathbb{X} \to \{0, 1\}.$$

Such a predicate can be interpreted as a characteristic function of a subset $A \subset \mathbb{X}$, for which the predicate is true (or 1). We are looking for predicates which are true for a large proportion of the points, equivalently, we are looking for subsets which have a large probability.

In extremely simple cases one can consider all possible predicates (or subsets) and determine the support $s(a) = P(\{x|a(x) = 1\}$ and select the ones which have large enough support to be interesting. There are a total of $2^{|\mathbb{X}|} = 2^{2^d}$ predicates, which, in the case of market basket analysis with $d = 10,000$ items (which corresponds to a smaller supermarket) is $2^{10^{3,000}} \approx 10^{3 \cdot 10^{2,999}}$. Thus it is impossible to consider all the subsets.

One suggestion is to search through the predicates in a systematic way. The predicates themselves have an order which is given by the order on the range of a and is the subset relationship of the sets for which the predicates

are the characteristic functions. We have

$$a \leq b \Leftrightarrow a(x) \leq b(x) \text{ for all } x$$

so a is less than b if a is a consequence of b, i.e, if b is true whenever a is true. With respect to $\vee$ and $\wedge$ (here in their original logical meaning) the predicates form a Boolean lattice. So we could systematically start a the bottom or the top of the lattice and work our way through the lattice following the edges of the Hasse diagram. Any predicate is defined by the set of elements on which it is true, i.e., it can be written as an indicator function of that set as

$$a(x) = \chi_{\{z|a(z)=1\}}(x)$$

and if the indicator function for the set containing just x is χ_x we have

$$a = \bigvee_{a(y)=1} \chi_y.$$

It follows directly that the atoms of the Boolean algebra of the predicates are the indicators χ_y.

Unfortunately, at the bottom end of the lattice the support for any predicates will be very low. Thus we need to start at the top and use the dual ordering. With respect to this dual ordering the atoms are now the indicator functions of the sets containing everything except y. Thus we get the opposite situation as before and for a very long time all the predicates we get have a very large support. Thus it is not feasible to search just along the edges of the graph of the lattice of the predicates.

The observations indicate that we need to look for a different ordering of the predicates. Here is where Occam's razor does give us some intuition. We are interested in simple predicates or simple sets which we may be able to interprete. This does not necessarily mean sets with few elements but sets which have a simple structure. As the full powerset does not lead to anything usefull we should consider a different set of subsets. In order to do this we go back to the example of the market baskets. A property which will make a market basket interesting will typically also make a market basket which contains the original market basket interesting as well. This leads to properties which are *monotone*, i.e. for which we have

$$x \leq y \Rightarrow a(x) \leq a(y).$$

The set A for which a is an indicator function is an *up-set*, also called *increasing set* or *order filter* and it satisfies

$$x \in A \text{ and } x \leq y \Rightarrow y \in A.$$

(The dual of up-sets are *down-sets*, *decreasing sets* or *order ideals*.)

The set of up-sets $\mathcal{F}(\mathbb{X})$ is a subset of the powerset but it is still quite large. The sizes of some of the smallest sets of up-sets are given in [5] for the cases of $d = 1, \ldots, 7$. For example, in the case where $d = 6$ the set $\mathbb{X}$ has 64 elements and the size of the set of up-sets is $7,828,354$. The simplest up-sets are the ones generated by one element:

$$\uparrow x := \{z | z \geq x\}.$$

In order to better understand the lattice of up-sets $\mathcal{F}(\mathbb{X})$ one considers *join-irreducible sets* in this lattice. They are defined as

$$\mathcal{J}(\mathcal{F}(\mathbb{X})) = \{A \neq 0 | \text{ if } A = B \cup C \text{ for some } B, C \in \mathcal{F}(\mathbb{X}) \text{ then } A = B \text{ or } A = C\}.$$

While join-irreducibility is defined for more general lattices, in our case we can characterise the join-irreducible up-sets exactly:

Proposition 8:

$$\mathcal{J}(\mathcal{F}(\mathbb{X})) = \{\uparrow x \mid x \in \mathbb{X}\}$$

Proof: As $\mathbb{X}$ is finite there are is for each $U \in \mathcal{F}(\mathbb{X})$ a sequence $x_i \in \mathbb{X}, i = 1, \ldots, k$ such that $x_i \not\leq x_j$ and $U = \bigcup_{i=1}^{k} \uparrow x_i$. By definition, at most the U for which $k = 1$ can be join-irreducible.

Let $\uparrow x = U \cup V$ and, say, $x \in U$. It then follows that $\uparrow x \subset U$. As $U \subset \uparrow x$ it follows $\uparrow x = U$. Thus all $\uparrow x \in \mathcal{J}(\mathcal{F}(\mathbb{X}))$. □

Note that in our case the sets $\uparrow x \setminus \{x\}$ are the lower covers of $\uparrow x$, i.e., the closest up-sets which are subsets. In fact, this is a different characterisation of the join-irreducible elements of finite lattices, see [5].

The join-irreducible elements are essential in Birkhoff's representation theorem for finite distributive lattices [5] which states that the finite distributive lattices are isomorph to the lattice of down-sets generated by join-irreducible elements. This representation is also closely related to the *disjunctive normal form* in logic.

We will now consider the join irreducible elements $\uparrow y$ further. Lets denote their characteristic function by $a_y := \chi_{\uparrow y}$ such that

$$a_y(x) = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{else.} \end{cases}$$

Note that

$$a_{y \vee z} = a_y \wedge a_z$$

but $a_y \vee a_z$ is in general not of the form a_z. We can systematically search through the space of all a_y using the ordering induced by the ordering of the index y. We start with the a_x where the x are the atoms of $\mathbb{X}$. Note that the induced ordering is reversed, i.e., one has

$$y \leq z \Leftrightarrow a_y \geq a_z,$$

and more generally one has

Proposition 9: *$a_y(x)$ is a monotone function of y and an anti-monotone function of x and furthermore:*

$$a_y(x) = \bigwedge_{z \in \mathcal{A}(\mathbb{X})} (a_z(x) \vee \neg a_z(y)) .$$

Proof: The monotonicity/antimonotonicity properties follow directly from the definition.

While the $\uparrow y$ are join-irreducible, they are not meet-irreducible. In particular, from the representation of Boolean algebras and the isomorphy of the lattice of join-irreducible elements of $\mathcal{F}(\mathbb{X})$ and $\mathbb{X}$ one gets

$$\uparrow y = \bigcap_{z \leq y,\ z \in \mathcal{A}(\mathbb{X})} \uparrow z.$$

If $z \not\leq y$ it follows that $\neg a_z(y)$ is true. □

Now we could use the logical theory of resolution to search for objects with particular properties or, alternatively, to find all possible predicates which either hold for all market baskets or for at least one market basket. These *deductions* are quite possible for smaller data sets but for larger data sets one can only hope to find a very limited subset of rules or predicates which occur very frequently.

Finally, we are interested in predicates a which have a support $s(a) \geq \sigma$, i.e., are frequent. Or more specifically, we are interested in frequent itemsets z such that $s(a_z) \geq \sigma$. By the apriori principle we have that if z is frequent and if $y \leq z$ then y is frequent as well. Thus the set of frequent itemsets forms a down-set, decreasing set or order ideal. It follows that we can represent the set of frequent itemsets F as

$$F = \bigcup_i \downarrow z^{(i)}$$

for a set of maximal elements $z^{(i)}$. The aim of the data mining for frequent itemsets is to find these maximal elements of F. The apriori algorithm

searches for these maximal element by looking at the lattice of z and, by starting at 0 moves to ever larger z until the maxima are found.

In summary, we now have an algebraic framework for the search for interesting predicates a_y.

2.5. *Support and antimonotonicity*

After having explored the space of predicates we intend to search we now discuss effects of the criterium used to select predicates further. Intuitively, we are looking for all predicates which are supported by the data, i.e., for which the support $s(a)$ is larger than a certain given value σ. Recall the definition of support

$$s(a) = P(\text{supp}(a)).$$

In case of the empirical distribution the relative frequency is the support, in the case where we have a sample (possibly from a larger population) the relative frequency is an estimator of the support and one has:

$$\hat{s}(a) = \frac{1}{n} \sum_{i=1}^{n} a(x^{(i)})$$

In the literature the term support is used some times also for the estimate $\hat{s}(a)$ and some times for the number of transactions for which a holds. In [1] support is reserved for the set of items in the data base which support the item, support count is used for the size of this set and support ratio for the estimate of $s(a)$. In Figure 11 the support is illustrated together with some data points. From the properties of the probability one gets properties of the support function s, in particular, one has $s(a \wedge b) \leq s(a)$.

We will now consider the support of the predicates a_y which considered in the previous section. Their support $s(a_y)$ is illustrated in figure 12, it is not the same as the support of the item set y, i.e., $s(a_y) \neq P(\{y\})$.

Note that the support of the item set $s(a_x)$ and the probability of the corresponding market basket x is not the same! The support of a_x is illustrated in However, the support is the the cumulative distribution function based on the dual lattice of $\mathbb{X}$.

Proposition 10: *Let s be the support and a_x be the predicate which indicates the presence of the item set x. Then one has*

$$s(a_x) = F^{\partial}(x)$$

where F^{∂} is the cumulative distribution function which is based on the dual partial order.

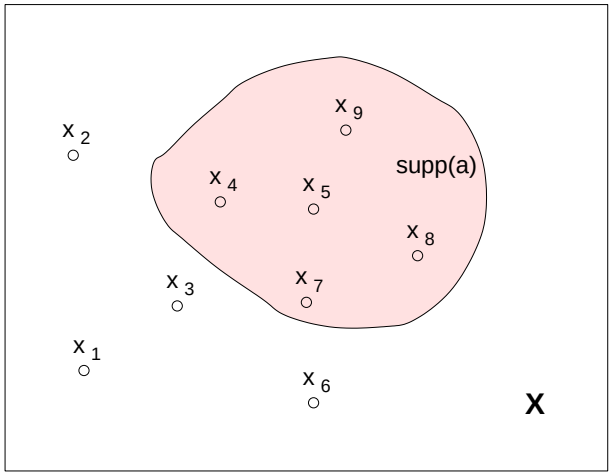

Fig. 11. Support of a general predicate

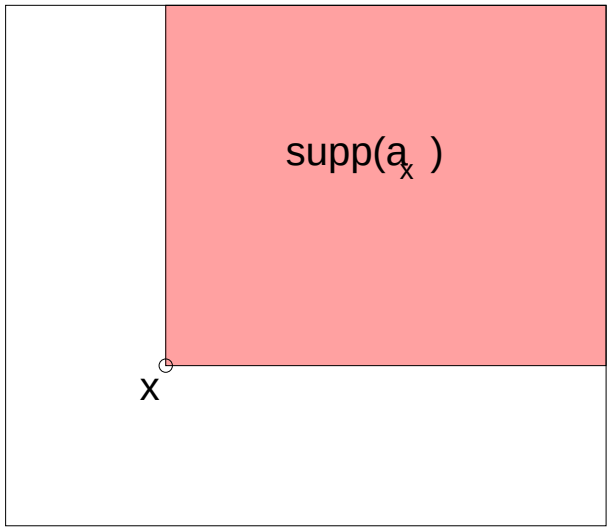

Fig. 12. Support of a_x.

Proof:

$$s(a_y) = \sum_{y \geq x} p(y).$$

□

While we have used probabilistic language, often, we are only interested in patterns which are frequent in the current data set and do not rely on any probabilistic interpretation of the data set or the generalisation of the rules.

A consequence of this previous proposition is the main property of the support function. Recall that a function $f(x)$ is *monotone* iff $f(x) \leq f(y)$ whenever $x \leq y$ and a function is *anti-monotone* iff $f(x) \geq f(y)$ whenever $x \leq y$. We now have the core property used for the association rule discovery algorithms:

Theorem 11: The support $s(a_y)$ is an anti-monotone function of y.

Proof: The cumulative distribution function F^δ is monotone wrt to the dual order and is thus anti-monotone wrt to the original order. The theorem then follows from the previous proposition. □

This simple anti-monotonicity property forms the basis of the apriori algorithm. Note that the cumulative distribution function uniquely defines the distribution and so either the probability distribution p or the support $s(a_x)$ can be used to determine the relevant properties of the probability space. Association rule mining uses the apriori property to find all the values of the cumulative distribution function which have at least a given size. Note that from a practical point of view, the predicates with small support are uninteresting.

A final comment on the relationship between the item sets and the *relational data model*. In the relational data model all the data is defined with tables or relations. In our case, this is based on two sets, one, the set of market baskets (stored as basket identifiers) which we call M and the set of items which we call I. The data base is then a relation $R \subset M \times I$. Thus we have one market basket table with one attribute "contains item" and the domain of this attribute are the items. The relation forms a bipartite graph. The contents of the market baskets x are the adjacency sets corresponding to the elements of M. The supports of item sets are the adjacent sets of sets of items in this model.

2.6. *Expectation and the size of itemsets*

For the analysis of the complexity of the association rule discovery algorithms we will require more general functions than predicates. In particular,

we will consider real functions $f : \mathbb{X} \to \mathbb{R}$. Such a function is a random variable and it also denotes a property of the objects we are investigating. In our case of finite $\mathbb{X}$ the function has an *expectation* which is defined as

$$E(f) = \sum_{x \in \mathbb{X}} p(x) f(x).$$

It follows that the expectation is a linear functional defined on the space of real functions over $\mathbb{X}$. The expectation is monotone in the sense that $f \geq g \Rightarrow E(f) \geq E(g)$ and the expectation of a constant function is the function value. The *variance* of the random variable corresponding to the function f is

$$\mathrm{var}(f) = E\left((f - E(f))^2\right).$$

A first class of functions f are the predicates a with values of 0 and 1 only. For them we have:

Proposition 12: *A predicate a is a random variable with expectation*

$$E(a) = s(a)$$

and variance

$$\mathrm{var}(a) = s(a)(1 - s(a)).$$

Note that $1 - s(a) = s(1 - a)$ is the support of the negation of a. One has $\mathrm{var}(a) \leq 0.25$ and the maximum is reached for $s(a) = 0.5$. In practice one would not expect any item to be in half of all market baskets and typically, $s(a)$ is often rather small so that the variance is close to the expectation. The previous proposition shows that the support can be defined as either a probability (of the supporting set) or an expectation (of the predicate). In fact, probability theory itself can be either based on the concept of probability or the concept of expectation [11].

A second class of examples is obtained when we consider a sample of independant observations as our objects. Thus the observed feature is the sequence $x^{(1)}, \ldots, x^{(n)}$. We consider the relative frequency of a on the observations, i.e.,

$$\hat{s} = \frac{1}{n} \sum_{i=1}^{n} a(x^{(i)})$$

as function defined on this object. The probability distribution of the observations is, and here we will assume that the samples are drawn at random

independantly:

$$p(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=1}^{m} p(x^{(i)}).$$

The expectation of the function $f(x^{(1)}, \ldots, x^{(n)}) = \hat{(}s)$ is

$$E(\hat{s}(a)) = s(a)$$

and the variance is

$$\mathrm{var}(\hat{s}(a)) = \frac{s(a)(1 - s(a))}{n}.$$

All these quantities are thus represented as a function of the support $s(a)$.

A third example of a random variable is the length of an itemset,

$$f(x) = |x| = \sum_{j=1}^{d} x_j.$$

The expectation of the length of an itemset is

$$E(|x|) = \sum_{j=1}^{d} p_j$$

where p_j is the probability that the bit j is set. Note that $p_j = s(a_j)$ where a_j is the predicate defined by

$$a_j(x) = \begin{cases} 1 & \text{if } x_j = 1 \\ 0 & \text{else} \end{cases}$$

is the indicator function for the up-set $\uparrow e_j$ of the j-th atom of $\mathbb{X}$. For the variance of the length of an itemset we need to also consider $a_i a_j$ with is the indicator function for sets of pairs of atoms or 2-itemsets and with $p_{ij} = E(a_i a_j)$ one gets

$$\mathrm{var}(|x|) = \sum_{i=1}^{d} p_i(1 - p_i) + \sum_{i \neq j} (p_{ij} - p_i p_j).$$

Note that the first sum is the sum of the variances of all the components. The second sum is a correction which originates from the fact that the components are not independant.

So far we have considered positive functions f only. One can now give a simple upper bound for the probabibility that the function values get large using the expectation only. This is useful as it allows the estimation of a distribution from a simple feature, the mean. It is unlikely that the the

random variable takes values which are very different from the expectation. A simple theorem gives an estimate of the probability for f to reach values larger than any given bound. It does not require any knowledge of the distribution. This is Markov's inequality:

Theorem 13: Let f be a non-negative function. Then, for $x > 0$:

$$p(f(x) \geq c) \leq \frac{E(f)}{c}$$

where $p(f(x) \geq c) := P(\{x|f(x) \geq c\})$.

Proof: From the definition of the expectation one gets

$$\begin{aligned} E(f) &= \sum_{x \in \mathbb{X}} f(x)p(x) \\ &= \sum_{f(x) \geq c} f(x)p(x) + \sum_{f(x) < c} f(x)p(x) \\ &\geq \sum_{f(x) \geq c} f(x)p(x) \\ &\geq c \sum_{f(x) \geq c} p(x) \\ &= cp(f(x) \geq c) \end{aligned}$$

from which we get the inequality by division by c on both sides. □

For the second example we can now see that the probability that $\hat{s}(a) > c$ is bounded from above by $s(a)/c$. This shows that we will not expect deviations which are too large but it does not tell us how this estimate improves with growing data size n. For example, we can see that the probability that the estimate is four times the size of $s(a)$ is less or equal than 0.25. If we consider $1/\hat{(}s)$ as our random variable we can get a similar lower bound, i.e., the probability that $\hat{s}(a)$ is less than $s(a)/4$ is also 0.25.

For the third example, the size of an itemset one gets similar bounds. We can again state that the probability that we get observations of the length of an itemset which is more than 4 times or less than 4 times the actual size are less or equal 0.25 each.

It turns out, that we can further modify the bounds obtained from the Markov inequality to include the variance as well. This is due to the fact that the Markov inequality holds for any positiv function. The variance now gives us a measure for how much the function values will likely spread. This is stated in the next theorem:

Theorem 14: For any real function f and $r > 0$ on has:

$$P(\{x \mid E(f) - r\sqrt{\text{var}(f)} < f(x) < E(f) + r\sqrt{\text{var}(f)}\}) \geq 1 - \frac{1}{r^2}.$$

Consider again the examples 2 and 3 from above. Choose $r = \rho E(f)/\sqrt{\text{var}(f)}$. Then we get

$$P(\{x \mid E(f)(1-\rho) < f(x) < E(f)(1+\rho)\}) \geq 1 - \frac{\text{var}(f)}{\rho^2 E(f)^2}.$$

As a consequence, we can now estimate the probability that the function value is between a bound of, say 25% around the exact value to be $1 - \frac{16\,\text{var}(f)}{E(f)^2}$.

In the second example we get for this bound $1 - \frac{16(1-s(a))}{ns(a)}$. This is small for small supports and large for large supports. Note that $ns(a)$ is just the number of positive cases. Thus for very small $s(a)$ we will need at 32 positive cases in order to get a 50 percent chance that the estimated support is within a 25 percent bound. This is actually not bad and thus the estimator for the support is relatively good.

In the third case the interpretation of the bound is a bit trickier but not much. We get the bound

$$1 - \frac{16\left(\sum_{j=1}^d p_j(1-p_j) + \sum_{i\neq j}(p_{ij} - p_i p_j)\right)}{\left(\sum_{j=1}^d p_j\right)^2}$$

which, in the case of small p_j and for indpedendant variables x_j is approximated by

$$1 - \frac{16}{\left(\sum_{j=1}^d p_j\right)}.$$

Consequently, in this case we can expect the length of an actual itemset to be at most 25 percent of its expectation with a probability of more than 0.5 if the expected size is larger than 32.

Note that the dependance of these bounds as a function of the number of variables d and the number of observations n are manifestations of the concentration phenomena.

2.7. *Examples of distributions*

The simplest case is where all $p_j = q$ for some $q > 0$ and

$$p(x) = q^{|x|}(1-q)^{d-|x|}$$

This is the case where every item is bought equally likely with probability q and independently. The length of the itemset has in this case a binomial distribution and the expected size of the itemset is

$$E(|x|) = qd.$$

The support of a_x can be seen to be

$$s(a_x) = q^{|x|}.$$

This models the observation that large data sets have low support (but not much else). The frequent itemsets correspond to itemsets which are limited in size by

$$|x| \leq \log(\sigma_0)/\log(q).$$

The distribution of the size is a binomial distribution

$$p(|x| = r) = \binom{d}{r} q^r (1-q)^{d-r}.$$

For very large numbers of items d and small q this is well approximated by the *Poisson distribution* with $\lambda := dq$:

$$p(|x| = r) = \frac{1}{r!} e^{-\lambda} \lambda^r.$$

The (dual) cumulative distribution function is

$$p(|x| \geq s) = \sum_{r=s}^{\infty} \frac{1}{r!} e^{-\lambda} \lambda^r.$$

In Figure 13 this probability is plotted as a function of x for the case of $\lambda = 4$ together with the Markov bound.

This example is not very exciting but may serve as a first test example for the algorithms. In addition, it is a model for "noise". A realistic model for the data would include noise and a "signal", where the signal models the fact that items are purchased in groups. Thus this example is an example with *no* structure and ideally we would expect a datamining algorithm not to suggest any structure either.

A second example, with just slightly more structure has again independant random variables but with variable distributions such that

$$p(x) = \prod_{j=1}^{m} p_j^{x_j} (1-p_j)^{1-x_j}.$$

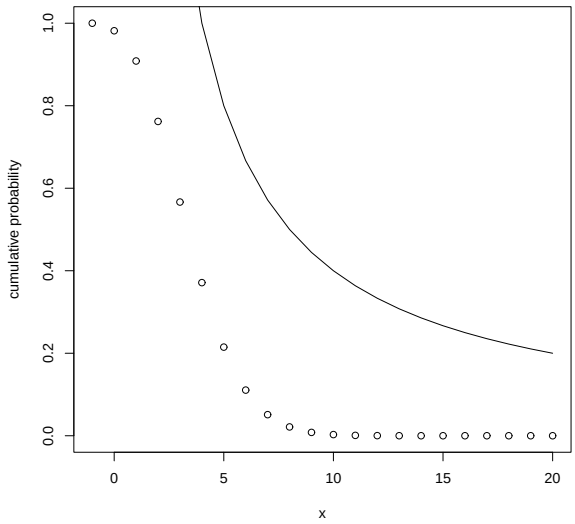

Fig. 13. Distribution of size of market baskets for the simplest case ("o") and the Markov bound for $\lambda = 4$.

The average market basket in this case is

$$\lambda = \sum_{j=1}^{m} p_j$$

and the support is

$$s(a_x) = \prod_{j=1}^{m} p_j^{x_j}.$$

In examples one can often find that the p_j when sorted are approximated by *Zipf's law*, i.e,

$$p_j = \frac{\alpha}{j}$$

for some constant

$$\alpha = \frac{\lambda}{\sum_{j=1}^{n} \frac{1}{j}}.$$

A third example is motivated by the voting data. In this case we assume that we have a "hidden variable" u which can take two values and which

has a distribution of, say $p(u = 0) = p(u = 1) = 0.5$. In addition, we have the conditional distributions $p(x|u = 0) = \prod_{j=1}^{m} p_j^{x_j}(1 - p_j)^{1-x_j}$ and $p(x|u = 1) = \prod_{j=1}^{m} q_j^{x_j}(1 - q_j)^{1-x_j}$. From this we get the distribution of the x to be

$$p(x) = \frac{1}{2}\left(\prod_{j=1}^{m} p_j^{x_j}(1 - p_j)^{1-x_j} + \prod_{j=1}^{m} q_j^{x_j}(1 - 1_j)^{1-x_j}\right).$$

Try to determine for this case what the distribution of the size of the itemset, and the variance are. Choose specific distributions p_j and q_j. What about the expected values for the a_j and the a_{jk}.

A forth example is again motivated by the voting data. If strickt party discipline holds for some items, everyone would be voting the same for these items. Thus we would have random variables $x_1 = \cdots x_k$ with probability $p_1 = \cdots = p_k$ and the other random variables could be random with probability $p_i,\ i > k$. The probability distribution is then

$$p(x) = \delta(x_2 = x_1) \cdots \delta(x_k = x1) p_1^{x_1} \cdot p_{k+1}^{x_{k+1}} \cdots p_d^{x_d}.$$

Again, we leave the further analysis of this example to the reader. Hint: the support for any a_z with not components larger than k set but at least one below is p_1, independant of how many components are set.

The aim of association rule mining is to find itemsets which occur more frequently based on an underlying grouping of items into larger blocks of items which are frequently purchased together. In [3] the authors present a model for market basket data which is used to generate synthetic data. It is based on sets of items which are frequently purchased together. We will not discuss this further at this stage.

3. The Apriori Algorithm

The aim of association rule discovery is the derivation of *if-then-rules* based on the predicates a_x defined in the previous subsection. An example of such a rule is "if a market basket contains orange juice then it also contains bread".

One possible way to interpret such a rule is to identify it with the predicate "either the market basket contains bread or it doesn't contain orange juice". The analysis could then determine the support of this predicate. However, this is not a predicate in the class we have defined in the previous section. Of course it could be determined from the $s(a_x)$. However, this

measure also is not directly interpretable and thus not directly applicable in retail.

Association rules do not make the above identification but treat the rule

$$a \Rightarrow b$$

as an (ordered) pair of predicates (a, b) and characterise this pair by its *support*

$$s(a \Rightarrow b) := s(a \wedge b)$$

and its *confidence*

$$c(a \Rightarrow b) := \frac{s(a \wedge b)}{s(a)}.$$

The support does not require further discussion but the confidence is a measure of how well the predicate a is able to predict the predicate b. More specifically, the confidence is the conditional probability that a market basket contains the items in the item set b given that it contains the item set a.

While the disadvantage of this approach is that it needs two numbers for each rule, the determination of these numbers is straight-forward as the first number is just a support of an item set and the second number is the quotient of two supports. Thus one way to determine these numbers and all rules for which both numbers are larger than given bounds is to first determine all item sets with large enough supports, the *frequent item sets* and within these search for rules with high confidence. These rules are called the *strong rules*. This is indeed the approach taken by many algorithms and we will discuss this further in later sections. For now we note that in the case of the predicates defined by the market baskets ω one has

$$c(a_x \Rightarrow a_y) = F^\delta(y|x)$$

where $F^\delta(y|x) = F^\delta(y \wedge x)/F^\delta(x)$ is a conditional cumulative distribution function and $\wedge$ denotes the glb. The relationship between conditional probability and confidence show one case where one can tell the confidence with having to compute a quotient, namely, when x and y are two independent samples such that

$$F^\delta(y \wedge x) = F^\delta(y)F^\delta(x)$$

and it follows that $c(a_x \Rightarrow a_y) = F^\delta(y)$. With other words, the proportion of the events containing x which also contain y is the same as the proportion

of events which contain y overall. Thus the restriction to ω does not provide any advantage. A simple way to check for this situation is the *lift* coefficient

$$\gamma(a \Rightarrow b) := \frac{c(a \Rightarrow b)}{s(b)}$$

which is one for independent a and b and less otherwise.

Now there are several problems with strong association rules which have been addressed in the literature:

- The straight-forward interpretation of the rules may lead to wrong inferences.
- The number of strong rules found can be very small.
- The number of strong rules can be very large.
- Most of the strong rules found are expected and do not lead to new insights.
- The strong rules found do not lend themselves to any actions and are hard to interpret.

We will address various of these challenges in the following. At this stage the association rule mining problem consists of the following:

Find all strong association rules in a given data set D.

We will now discuss the classical Apriori algorithm as suggested by Agrawal et al. in [3] and some of the most important derived algorithms.

3.1. *Time complexity – computing supports*

The data is a sequence $x^{(1)}, \ldots, x^{(n)}$ of binary vectors. We can thus represent the data as a n by d binary matrix. The number of nonzero elements is $\sum_{i=1}^{n} |x^{(i)}|$. This is approximated by the expected length $E(|x|)$ times n. So the proportion of nonzero elements is $E(|x|)/d$. This can be very small, especially for the case of market baskets, where out of the 10,000 items usually less than 100 items are purchased. Thus less than one percent of all the items are nonzero. In this case it makes sense to store the matrix in a sparse format. Here we will consider two ways to store the matrix, either by rows or by columns. The matrix corresponding to an earlier example is

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

First we discuss the *horizontal organisation*. A row is represented simply by the indices of the nonzero elements. And the matrix is represented as a tuple of rows. For example, the above matrix is represented as

$$\big[(1,2)\ (1,3,4)\ (1,5)\ (1,2,4)\ (5)\big]$$

In practice we need in addition to the indices of the nonzero elements in each row we need pointers which tell us where the row starts if contiguous locations are used in memory.

Now assume that we have any row x and a a_z and would like to find out if x supports a_z, i.e., if $z \leq x$. If both the vectors are represented in the sparse format this means that we would like to find out if the indices of z are a subset of the indices of x. There are several different ways to do this and we will choose the one which uses an auxillary bitvector $v \in \mathbb{X}$ (in full format) which is initialised to zero. The proposed algorithm has 3 steps:

(1) Extract x into v, i.e., $v \leftarrow x$
(2) Extract the value of v for the elements of z, i.e., $v[z]$. If they are all nonzero then $z \leq x$.
(3) Set v to zero again, i.e., $v \leftarrow 0$

We assume that the time per nonzero element for all the steps is the same τ and we get for the time:

$$T = (2|x| + |z|)\tau.$$

Now in practice we will have to determine if $a_z(x)$ holds for m_k different vectors z which have all the same length k. Rather than doing the above algorithm m_k times one only needs to extract x once and one thus gets the algorithm

(1) Extract x into v, i.e., $v \leftarrow x$
(2) Extract the values of v for the elements of $z^{(j)}$, i.e., $v[z^{(j)}]$. If they are all nonzero then $z^{(j)} \leq x$. Do this step for $j = 1, \ldots, m_k$.
(3) Set v to zero again, i.e., $v \leftarrow 0$

With the same assumptions as above we get ($|z^{(j)}| = k$):

$$T = (2|x| + m_k k)\tau.$$

Finally, we need to run this algorithm for all the rows $x^{(i)}$ and we need to consider vectors $z^{(j)}$ of different lengths. Considering this one gets the total time

$$T = \sum_k (2\sum_{i=1}^{n} |x^{(i)}| + m_k k n)\tau$$

and the expected time is

$$E(T) = \sum_k (2E(|x|) + m_k k) n\tau.$$

Note that the sum over k is for k between one and d but only the k for which $m_k > 0$ need to be considered. The complexity has two parts. The first part is proportional to $E(|x|)n$ which corresponds to the number of data points times the average complexity of each data point. This part thus encapsulates the data dependency. The second part is proportional to $m_k kn$ where the factor $m_k k$ refers to the complexity of the search space which has to be visited for each record n. For $k = 1$ we have $m_1 = 1$ as we need to consider all the components. Thus the second part is larger than $dn\tau$, in fact, we would probably have to consider all the pairs so that it would be larger than $d^2 n\tau$ which is much larger than the first part as $2E(|x|) \leq 2d$. Thus the major cost is due to the search through the possible patterns and one typically has a good approximation

$$E(T) \approx \sum_k m_k kn\tau.$$

An alternative is the *vertical organisation* where the binary matrix (or Boolean relational table) is stored column-wise. This may require slightly less storage as the row wise storage as we only needs pointers to each column and one typically has more rows than columns. In this vertical storage scheme the matrix considered earlier would be represented as

$$\big[(1,2,3,4)\ (1,4)\ (2)\ (2,4)\ (3,5)\big]$$

The savings in the vertical format however, are offset by extra costs for the algorithm as we will require an auxillary vector with n elements.

For any a_z the algorithm considers only the columns for which the components z_j are one. The algorithm determines the intersection (or elementwise product) of all the columns j with $z_j = 1$. This is done by using the auxiliary array v which holds the current intersection. We initially set it to the first column j with $z_j = 1$, later extract all the values at the points defined by the nonzero elements for the next column j' for which $z_{j'} = 1$, then zero the original ones in v and finally set the extracted values into the v. More concisely, we have the algorithm, where x_j stands for the whole column j in the data matrix.

(1) Get j such that $z_j = 1$, mark as visited
(2) Extract x_j into v, i.e., $v \leftarrow x_j$
(3) Repeat until no nonzero elements in z unvisited:

(a) Get unvisited j such that $z_j = 1$, mark as visited
(b) Extract elements of v corresponding to x_j, i.e., $w \leftarrow v[x_j]$
(c) Set v to zero, $v \leftarrow 0$
(d) Set v to w, $v \leftarrow w$

(4) Get the support $s(a_z) = |w|$

So we access v three times for each column, once for the extraction of elements, once for setting it to zero and once for resetting the elements. Thus for the determination of the support of z in the data base we have the time complexity of

$$T = 3\tau \sum_{j=1}^{d} \sum_{i=1}^{n} x_j^{(i)} z_j.$$

A more careful analysis shows that this is actually an upper bound for the complexity. Now this is done for m_k arrays $z^{(s,k)}$ of size k and for all k. Thus we get the total time for the determination of the support of all a_z to be

$$T = 3\tau \sum_{k} \sum_{s=1}^{m_k} \sum_{j=1}^{d} \sum_{i=1}^{n} x_j^{(i)} z_j^{(s,k)}.$$

We can get a simple upper bound for this using $x_j^{(i)} \leq 1$ as

$$T \leq 3 \sum_{k} m_k k n \tau$$

because $\sum_{j=1}^{d} z_j^{(s,k)} = k$. This is roughly 3 times what we got for the previous algorithm. If the $x_j^{(i)}$ are random they all have an expectation $E(x_j^{(i)})$ which is typically much less than one and have an average expectation of $E(|x|)/d$. If we introduce this into the equation for T we get the approximation

$$E(T) \approx \frac{3E(|x|)}{d} \sum_{k} m_k k n \tau$$

which can be substantially smaller than the time for the previous algorithm.

Finally, we should point out that there are many other possible algorithms and other possible data formats. Practical experience and more careful analysis shows that one method may be more suitable for one data set where the other is stronger for another data set. Thus one carefully has to consider the specifics of a data set. Another consideration is also the size k and number m_k of the z considered. It is clear from the above that it is

essential to carefully choose the "candidates" a_z for which the support will be determined. This will further be discussed in the next sections. There is one term which occured in both algorithms above and which characterises the complexity of the search through multiple levels of a_z, it is:

$$C = \sum_k m_k k.$$

We will use this constant in the discussion of the efficiency of the search procedures.

3.2. *The algorithm*

Some principles of the apriori algorithm are suggested in [2]. In particular, the authors suggest a breadth-first search algorithm and utilise the apriori principle to avoid unnecessary processing. However, the problem with this early algorithm is that it generates candidate itemsets for each record and also cannot make use of the vertical data organisation. Consequently, two groups of authors suggested at the same conference a new and faster algorithm which before each data base scan determines which candidate itemset to explore [3,9]. This approach has substantially improved performance and is capable of utilising the vertical data organisation. We will discuss this algorithm using the currently accepted term *frequent itemsets*. (This was in the earlier literature called "large itemsets" or "covering itemsets".)

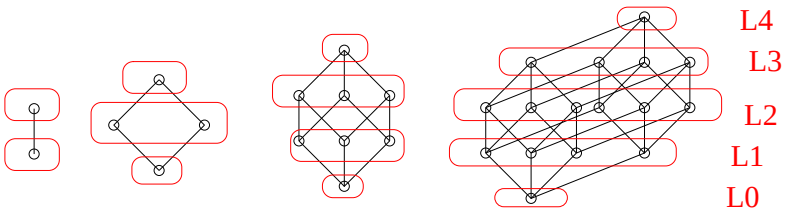

Fig. 14. Level sets of Boolean lattices

The apriori algorithm visits the lattice of itemsets in a level-wise fashion, see Figure 14 and Algorithm 1. Thus it is a *breadth-first-search* or BFS procedure. At each level the data base is scanned to determine the support of items in the *candidate itemset* C_k. Recall from the last section that the major determining parameter for the complexity of the algorithm is $C = \sum_k m_k k$ where $m_k = |C_k|$.

Algorithm 1 Apriori

$C_1 = \mathcal{A}(\mathbb{X})$ is the set of all one-itemsets, $k = 1$
while $C_k \neq \emptyset$ **do**
  scan database to determine support of all a_y with $y \in C_k$
  extract frequent itemsets from C_k into L_k
  generate C_{k+1}
  $k := k + 1$.
end while

It is often pointed out that much of the time is spent in dealing with pairs of items. We know that $m_1 = d$ as one needs to consider all single items. Furthermore, one would not have any items which alone are not frequent and so one has $m_2 = d(d-1)/2$. Thus we get the lower bound for C:

$$C \leq m_1 + 2m_2 = d^2.$$

As one sees in practice that this is a large portion of the total computations one has a good approximation $C \approx d^2$. Including also the dependence on the data size we get for the time complexity of apriori:

$$T = O(d^2 n).$$

Thus we have scalability in the data size but quadratic dependence on the dimension or number of attributes.

Consider the first (row-wise) storage where $T \approx d^2 n\tau$. If we have $d = 10,000$ items and $n = 1,000,000$ data records and the speed of the computations is such that $\tau = 10^{-9}$ the apriori algorithm would require 10^5 seconds which is around 30 hours, more than one day. Thus the time spent for the algorithm is clearly considerable.

In case of the second (column-wise) storage scheme we have $T \approx 3E(|x|)dn\tau$. Note that in this case for fixed size of the market baskets the complexity is now

$$T = O(dn).$$

If we take the same data set as before and we assume that the average market basket contains 100 items ($E(|x|) = 100$) then the apriori algorithm would require only 300 seconds or five minutes, clearly a big improvement over the row-wise algorithm.

3.3. *Determination and size of the candidate itemsets*

The computational complexity of the apriori algorithm is dominated by data scanning, i.e., evaluation of $a_z(x)$ for data records x. We found earlier that the complexity can be described by the constant $C = \sum_k m_k k$. As m_k is the number of k itemsets, i.e., bitvectors where exactly k bits are one we get $m_k \leq \binom{m}{k}$. On the other hand the apriori algorithm will find the set L_k of the actual frequent itemsets, thus $L_k \subset C_k$ and so $|L_k| \leq m_k$. Thus one gets for the constant C the bounds

$$\sum_k k|L_k| \leq C = \sum_k m_k k \leq \sum_{k=0}^{d} \binom{d}{k} k.$$

The upper bound is hopeless for any large size d and we need to get better bounds. This depends very much on how the candidate itemsets C_k are chosen. We choose C_1 to be the set of all 1-itemsets, and C_2 to be the set of all 2-itemsets so that we get $m_1 = 1$ and $m_2 = d(d-1)/2$.

The apriori algorithm determines alternatively C_k and L_k such that successively the following chain of sets is generated:

$$C_1 = L_1 \to C_2 \to L_2 \to C_3 \to L_3 \to C_4 \to \cdots$$

How should we now choose the C_k? We know, that the sequence L_k satisfies the *apriori property*, which can be reformulated as

Definition 15: If y is a frequent k-itemset (i.e., $y \in L_k$) and if $z \leq y$ then z is a frequent $|z|$-itemset, i.e., $z \in L_{|z|}$.

Thus in extending a sequence $L_1, L_2, \ldots, L_k$ by a C_{k+1} we can choose the candidate itemset such that the extended sequence still satisfies the apriori property. This still leaves a lot of freedom to choose the candidate itemset. In particular, the empty set would allways be admissible. We need to find a set which contains L_{k+1}. The apriori algorithm chooses the *largest* set C_{k+1} which satisfies the apriori condition. But is this really necessary? It is if we can find a data set for which the extended sequence is the set of frequent itemsets. This is shown in the next proposition:

Proposition 16: *Let $L_1, \ldots, L_m$ be any sequence of sets of k-itemsets which satisfies the apriori condition. Then there exists a dataset D and a $\sigma > 0$ such that the L_k are frequent itemsets for this dataset with minimal support σ.*

Proof: Set $x^{(i)} \in \bigcup_k L_k,\ i = 1, \ldots, n$ to be sequence of all maximal itemsets, i.e., for any $z \in \bigcup_k L_k$ there is an $x^{(i)}$ such that $z \leq x^{(i)}$ and $x^{(i)} \not\leq x^{(j)}$ for $i \neq j$. Choose $\sigma = 1/n$. Then the L_k are the sets of frequent itemsets for this data set. □

For any collection L_k there might be other data sets as well, the one chosen above is the minimal one. The sequence of the C_k is now characterised by:

(1) $C_1 = L_1$
(2) If $y \in C_k$ and $z \leq y$ then $z \in C_{k'}$ where $k' = |z|$.

In this case we will say that *the sequence C_k satisfies the apriori condition.* It turns out that this characterisation is strong enough to get good upper bounds for the size of $m_k = |C_k|$.

However, before we go any further in the study of bounds for $|C_k|$ we provide a construction of a sequence C_k which satisfies the apriori condition. A first method uses L_k to construct C_{k+1} which it chooses to be the maximal set such that the sequence $L_1, \ldots, L_k, C_{k+1}$ satisfies the apriori property. One can see by induction that then the sequence $C_1, \ldots, C_{k+1}$ will also satisfy the apriori property. A more general approach constructs $C_{k+1}, \ldots, C_{k+p}$ sucht that $L_1, \ldots, L_k, C_{k+1}, \ldots, C_{k+p}$ satisfies the apriori property. As p increases the granularity gets larger and this method may work well for larger itemsets. However, choosing larger p also amounts to larger C_k and thus some overhead. We will only discuss the case of $p = 1$ here.

The generation of C_{k+1} is done in two steps. First a slightly larger set is constructed and then all the elements which break the apriori property are removed. For the first step the join operation is used. To explain join let the elements of L_1 (the atoms) be enumerated as $e_1, \ldots, e_d$. Any itemset can then be constructed as join of these atoms. We denote a general itemset by

$$e(j_1, \ldots, j_k) = e_{j_1} \vee \cdots \vee e_{j_k}$$

where $j_1 < j_2 < \cdots < j_k$. The *join* of any k-itemset with itself is then defined as

$$L_k \bowtie L_k := \{e(j_1, \ldots, j_{k+1}) \mid e(j_1, \ldots, j_k) \in L_k, \quad e(j_1, \ldots, j_{k-1}, j_{k+1}) \in L_k\}.$$

Thus $L_k \bowtie L_k$ is the set of all $k+1$ itemsets for which 2 subsets with k items each are frequent. As this condition also holds for all elements in C_{k+1} one has $C_{k+1} \subset L_k \bowtie L_k$. The C_{k+1} is then obtained by removing an elements which contain infrequent subsets.

For example, if $(1,0,1,0,0) \in L_2$ and $(0,1,1,0,0) \in L_2$ then $(1,1,1,0,0) \in L_2 \bowtie L_2$. If we also know that $(1,1,0,0,0) \in L_2$ then $(1,1,1,0,0) \in C_3$.

Having developed a construction for C_k we can now determine the bounds for the size of the candidate itemsets based purely on combinatorial considerations. The main tool for our discussion is the the Kruskal-Katona theorem. The proof of this theorem and much of the discussion follows closely the exposition in Chapter 5 of [4].

We will denote the set of all possible k-itemsets or bitvectors with exactly k bits as $\mathcal{I}_k$. Subsets of this set are sometimes also called hypergraphs in the literature. The set of candidate itemsets $C_k \subset \mathcal{I}_k$.

Given a set of k-itemsets C_k the *lower shadow of* C_k is the set of all $k-1$ subsets of the elements of C_k:

$$\partial(C_k) := \{y \in \mathcal{I}_{k-1} \mid y < z \text{ for some } z \in C_k\}.$$

This is the set of bitvectors which have $k-1$ bits set at places where some $z \in C_k$ has them set. The shadow ∂C_k can be smaller or larger than the C_k. In general, one has for the size $|\partial C_k| \geq k$ independant of the size of C_k. So, for example, if $k = 20$ then $|\partial C_k| \geq 20$ even if $|C_k| = 1$. (In this case we actually have $|\partial C_k| = 20$.) For example, we have $\partial C_1 = \emptyset$, and $|\partial C_2| \leq d$.

It follows now that the sequence of sets of itemsets C_k satisfies the apriori condition iff

$$\partial(C_k) \subset C_{k-1}.$$

The Kruskal-Katona Theorem provides an estimate of the size of the shadow.

For our discussion we will map the bitvectors in the usual way onto integers. This is done with the mapping defined by

$$\phi(x) := \sum_{i=0}^{d-1} 2^{ix_i}.$$

For example, $\phi(\{1,3,5\}) = 42$. The induced order on the itemsets is then defined by

$$y \prec z :\Leftrightarrow \phi(y) < \phi(z).$$

This is the *colex (or colexicographic) order*. In this order the itemset $\{3,5,6,9\} \prec \{3,4,7,9\}$ as the largest items determine the order. In the lexicographic ordering the order of these two sets would be reversed.

Let $[m] := \{0, \ldots, m-1\}$ and $[m]^{(k)}$ be the set of all k-itemsets where k bits are set in the first m positions. As in the colex order any z with bits m and beyond set are larger than any of the elements in $[m]^{(k)}$ this is just the set of the first $\binom{m-1}{k}$ bitvectors with k bits set.

We will now construct the sequence of the first m bitvectors for any m. This corresponds to the first numbers, which, in the binary representation have m ones set. Consider, for example the case of $d = 5$ and $k = 2$. For this case all the bitvectors are in table **??**. (Printed reverse for legibility.)

(0,0,0,1,1)	3
(0,0,1,0,1)	5
(0,0,1,1,0)	6
(0,1,0,0,1)	9
(0,1,0,1,0)	10
(0,1,1,0,0)	12
(1,0,0,0,1)	17
(1,0,0,1,0)	18
(1,0,1,0,0)	20
(1,1,0,0,0)	24

As before, we denote by e_j the $j-th$ atom and by $e(j_1, \ldots, j_k)$ the bitvector with bits $j_1, \ldots, j_k)$ set to one. Furthermore, we introduce the element-wise join of a bitvector and a set C of bitvectors as:

$$C \vee y := \{z \vee y \mid z \in C\}.$$

For $0 \leq s \leq m_s < m_{s+1} < \cdots < m_k$ we introduce the following set of k-itemsets:

$$B^{(k)}(m_k, \ldots, m_s) := \bigcup_{j=s}^{k} \left([m_j]^{(j)} \vee e(m_{j+1}, \ldots, m_k) \right) \subset \mathcal{I}_k.$$

As only term j does not contain itemsets with item m_j (all the others do) the terms are pairwise disjoint and so the union contains

$$|B^{(k)}(m_k, \ldots, m_s)| = b^{(k)}(m_k, \ldots, m_s) := \sum_{j=s}^{k} \binom{m_j}{j}$$

k-itemsets. This set contains the first (in colex order) bitvectors with k bits set. By splitting off the last term in the union one then sees:

$$B^{(k)}(m_k, \ldots, m_s) = \left(B^{(k-1)}(m_{k-1}, \ldots, m_s) \vee e_{m_k} \right) \cup [m_k]^{(k)} \qquad (1)$$

and consequently

$$b^{(k)}(m_k, \ldots, m_s) = b^{(k-1)}(m_{k-1}, \ldots, m_s) + b^{(k)}(m_k).$$

Consider the example of table **??** of all bitvectors up to $(1, 0, 0, 1, 0)$. There are 8 bitvectors for which come earlier in the colex order. The highest bit set for the largest element is bit 5. As we consider all smaller elements we need to have all two-itemsets where the 2 bits are distributed between positions 1 to 4 and there are $\binom{4}{2} = 6$ such bitvectors. The other cases have the top bit fixed at position 5 and the other bit is either on position 1 or two thus there are $\binom{2}{1} = 2$ bitvectors for which the top bit is fixed. Thus we get a total of

$$\binom{4}{2} + \binom{2}{1} = 8$$

bitvectors up to (and including) bitvector $(1, 0, 0, 1, 0)$ for which 2 bits are set. This construction is generalised in the following.

In the following we will show that $b^{(k)}(m_k, \ldots, m_s)$ provides a unique representation for the integers. We will make frequent use of the identity:

$$\binom{t+1}{r} - 1 = \sum_{l=1}^{r} \binom{t-r+l}{l}. \tag{2}$$

Lemma 17: *For every $m, k \in \mathbb{N}$ there are numbers $m_s < \cdots < m_k$ such that*

$$m = \sum_{j=s}^{k} \binom{m_j}{j} \tag{3}$$

and the m_j are uniquely determined by m.

Proof: The proof is by induction over m. In the case of $m = 1$ one sees immediately that there can only be one in the sum (3), thus $s = k$ and the only choice is $m_k = k$.

Now assume that equation 3 is true for some $m' = m - 1$. We now show uniqueness for m. We only need to show that m_k is uniquely determined as the uniqueness of the other m_j follows from the induction hypothesis applied to $m' = m - m - \binom{m_k}{k}$.

So we assume a decomposition of the form 3 is given. Using equation 2

one gets:

$$\begin{aligned} m &= \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_s}{s} \\ &\leq \binom{m_k}{k} + \binom{m_k-1}{k-1} + \cdots + \binom{m_k-k+1}{1} \\ &= \binom{m_k+1}{k} - 1 \end{aligned}$$

as $m_{k-1} \leq m_k - 1$ etc. Thus we get

$$\binom{m_k}{k} \leq m \leq \binom{m_k+1}{k} - 1.$$

With other words, m_k is the largest integer such that $\binom{m_k}{k} \leq m$. This provides a unique characterisation of m_k which proves uniqueness.

Assume that the m_j be constructed according to the method outlined in the first part of this proof. One can check that equation 3 holds for these m_j using the characterisation.

What remains to be shown is $m_{j+1} > m_j$ and using inductions, it is enough to show that $m_{k-1} < m_k$. If, on the contrary, this does not hold and $m_{k-1} \geq m_k$, then one gets from 2:

$$\begin{aligned} m &\geq \binom{m_k}{k} + \binom{m_{k-1}}{k-1} \\ &\geq \binom{m_k}{k} + \binom{m_k}{k-1} \\ &= \binom{m_k}{k} + \binom{m_k-1}{k-1} + \cdots + \binom{m_k-k+1}{1} + 1 \\ &\geq \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_s}{s} + 1 \text{ by induction hyp.} \\ &= m+1 \end{aligned}$$

which is not possible. □

Let $N^{(k)}$ be the set of all k-itemsets of integers. It turns out that the $B^{(k)}$ occur as natural subsets of $N^{(k)}$:

Theorem 18: The set $B^{(k)}(m_k, \ldots, m_s)$ consists of the first $m = \sum_{j=s}^{k} \binom{m_j}{j}$ itemsets of $N^{(k)}$ (in colex order).

Proof: The proof is by induction over $k - s$. If $k = s$ and thus $m = \binom{m_k}{k}$ then the first elements of $N^{(k)}$ are just $[m_k]^{(k)}$. If $k > s$ then the first $\binom{m_k}{k}$

elements are still $[m_k]^{(k)}$. The remaining $m - \binom{m_k}{k}$ elements all contain bit m_k. By the induction hypothesis the first $b^{k-1}(m_{k-1}, \ldots, m_s)$ elements containing bit m_k are $B^{k-1}(m_{k-1}, \ldots, m_s) \vee e_{m_k}$ and the rest follows from 1. □

The shadow of the first k-itemsets $B^{(k)}(m_k, \ldots, m_s)$ are the first $k-1$-itemsets, or more precisely:

Lemma 19:

$$\partial B^{(k)}(m_k, \ldots, m_s) = B^{(k-1)}(m_k, \ldots, m_s).$$

Proof: First we observe that in the case of $s = k$ the shadow is simply set of all $k-1$ itemsets:

$$\partial [m_k]^k = [m_k]^{(k-1)}.$$

This can be used as anchor for the induction over $k - s$. As was shown earlier, one has in general:

$$B^{(k)}(m_k, \ldots, m_s) = [m_k]^k \cup \left(B^{(k-1)}(m_{k-1}, \ldots, m_s) \vee e_{m_k} \right)$$

and, as the shadow is additive, as

$$\partial B^{(k)}(m_k, \ldots, m_s) = [m_k]^{(k-1)} \cup \left(B^{(k-2)}(m_{k-1}, \ldots, m_s) \vee e_{m_k} \right) = B^{(k-1)}(m_k, \ldots, m_s).$$

Note that $B^{(k-1)}(m_{k-1}, \ldots, m_s) \subset [m_l]^{(k-1)}$. □

Recall that the shadow is important for the apriori property and we would thus like to determine the shadow, or at least its size for more arbitray k-itemsets as they occur in the apriori algorithm. Getting bounds is feasible but one requires special technology to do this. This is now what we are going to develop. We would like to reduce the case of general sets of k-itemsets to the case of the previous lemma, where we know the shadow. So we would like to find a mapping which maps the set of k-itemsets to the first k itemsets in colex order without changing the size of the shadow. We will see that this can almost be done in the following. The way to move the itemsets to earlier ones (or to "compress" them) is done by moving later bits to earlier positions.

So we try to get the k itemsets close to $B^{(k)}(m_k, \ldots, m_s)$ in some sense, so that the size of the shadow can be estimated. In order to simplify notation we will introduce $z + e_j$ for $z \vee e_j$ when $e_j \not\leq z$ and the reverse

operation (removing the j-th bit) by $z - e_j$ when $e_j \leq z$. Now we introduce *compression* of a bitvector as

$$R_{ij}(z) = \begin{cases} z - e_j + e_i & \text{if } e_i \not\leq z \text{ and } e_j \leq z \\ z & \text{else} \end{cases}$$

Thus we simply move the bit in position j to position i if there is a bit in position j and position i is empty. If not, then we don't do anything. So we did not change the number of bits set. Also, if $i < j$ then we move the bit to an earlier position so that $R_{ij}(z) \leq z$. For our earlier example, when we number the bits from the right, starting with 0 we get $R_{1,3}((0,1,1,0,0)) = (0,0,1,1,0)$ and $R_{31}((0,0,0,1,1)) = (0,0,0,1,1)$. This is a "compression" as it moves a collection of k-itemsets closer together and closer to the vector $z = 0$ in terms of the colex order.

The mapping R_{ij} is not injective as

$$R_{ij}(z) = R_{ij}(y)$$

when $y = R_{ij}(z)$ and this is the only case. Now consider for any set C of bitvectors the set $R_{ij}^{-1}(C) \cap C$. These are those elements of C which stay in C when compressed by R_{ij}. The compression operator for bitsets is now defined as

$$\tilde{R}_{i,j}(C) = R_{ij}(C) \cup (C \cap R_{ij}^{-1}(C)).$$

Thus the points which stay in C under R_{ij} are retained and the points which are mapped outside C are added. Note that by this we have avoided the problem with the non-injectivity as only points which stay in C can be mapped onto each other. The size of the compressed set is thus the same. However, the elements in the first part have been mapped to earlier elements in the colex order. In our earlier example, for $i, j = 1, 3$ we get

$$C = \{(0,0,0,1,1), (0,1,1,0,0), (1,1,0,0,0), (0,1,0,1,0)\}$$

we get

$$\tilde{R}_{i,j}(C) = \{(0,0,0,1,1), (0,0,1,1,0), (1,1,0,0,0), (0,1,0,1,0)\}.$$

Corresponding to this compression of sets we introduce a mapping $\tilde{R}_{i,j}$ (which depends on C) by $\tilde{R}_{i,j}(y) = y$ if $R_{i,j}(y) \in C$ and $\tilde{R}_{i,j}(y) = R_{i,j}(y)$ else. In our example this maps corresponding elements in the sets onto each other. In preparation, for the next lemma we need the simple little result:

Lemma 20: *Let C be any set of k itemsets and $z \in C, e_j \leq z, e_i \not\leq z$. Then*

$$z - e_j + e_i \in \tilde{R}_{i,j}(C).$$

Proof: There are two cases to consider:

(1) Either $z - e_j + e_i \notin C$ in which case $z - e_j + e_i = \tilde{R}_{i,j}(z)$.
(2) Or $z - e_j + e_i \in C$ and as $z - e_j + e_i = \tilde{R}_{i,j}(z - e_j + e_i)$ one gets again $z - e_j + e_i \in \tilde{R}_{i,j}(C)$. □

The next result shows that in terms of the shadow, the "compression" $\tilde{R}_{i,j}$ really is a compression as the shadow of a compressed set can never be larger than the shadow of the original set. We suggest therefor to call it compression lemma.

Lemma 21: *Let C be a set of k itemsets. Then one has*

$$\partial\tilde{R}_{i,j}(C) \subset \tilde{R}_{i,j}(\partial C).$$

Proof: Let $x \in \partial\tilde{R}_{i,j}(C)$. We need to show that

$$x \in \tilde{R}_{i,j}(\partial C)$$

and we will enumerate all possible cases.

First notice that there exists a $e_k \not\leq x$ such that

$$x + e_k \in \tilde{R}_{i,j}(C)$$

so there is an $y \in C$ such that

$$x + e_k = \tilde{R}_{i,j}(y).$$

(1) In the first two cases $\tilde{R}_{i,j}(y) \neq y$ and so one has (by the previous lemma)

$$x + e_k = y - e_j + e_i, \quad \text{for some } y \in C, e_j \leq y, e_i \not\leq y.$$

   (a) First consider $i \neq k$. Then there is a bitvector z such that $y = z + e_k$ and $z \in \partial C$. Thus we get

$$x = z - e_j + e_i \in \tilde{R}_{i,j}(\partial C)$$

   as $z \in \partial C$ and with lemma 20.
   (b) Now consider $i = k$. In this case $x + e_i = y - e_j + e_i$ and so $x = y - e_j \in \partial C$. As $e_j \not\leq x$ one gets

$$x = \tilde{R}_{i,j}(x) \in \tilde{R}_{i,j}(\partial C).$$

(2) In the remaining cases $\tilde{R}_{i,j}(y) = y$, i.e., $x + e_k = \tilde{R}_{i,j}(x + e_k)$. Thus $x + e_k = y \in C$ and so $x \in \partial C$. Note that $\tilde{R}_{i,j}$ actually depends on ∂C!
   (a) In the case where $e_j \not\leq x$ one has $x = \tilde{R}_{i,j}(x) \in \tilde{R}_{i,j}(\partial C)$.

(b) In the other case $e_j \leq x$. We will show that $x = \tilde{R}_{i,j}(x)$ and, as $x \in \partial C$ one gets $x \in \tilde{R}_{i,j}(\partial C)$.
   i. If $k \neq i$ then one can only have $x + e_k = \tilde{R}_{i,j}(x + e_k)$ if either $e_i \leq x$, in which case $x = \tilde{R}_{i,j}(x)$, or $x - e_j + e_i + e_k \in C$ in which case $x - e_j + e_i \in \partial C$ and so $x = \tilde{R}_{i,j}(x)$.
   ii. Finally, if $k = i$, then $x + e_i \in C$ and so $x - e_j + e_i \in \partial C$ thus $x = \tilde{R}_{i,j}(x)$. □

The operator $\tilde{R}_{i,j}$ maps sets of k itemsets onto sets of k itemsets and does not change the number of elements in a set of k itemsets. One now says that a set of k itemsets C is *compressed* if $\tilde{R}_{i,j}(C) = C$ for all $i < j$. This means that for any $z \in C$ one has again $R_{ij}(z) \in C$. Now we can move to prove the key theorem:

Theorem 22: Let $k \geq 1, A \subset N^{(k)},\ s \leq m_s < \cdots < m_k$ and

$$|A| \geq b^{(k)}(m_k, \ldots, m_s)$$

then

$$|\partial A| \geq b^{(k-1)}(m_k, \ldots, m_s).$$

Proof: First we note that the shadow is a monotone function of the underlying set, i.e., if $A_1 \subset A_2$ then $\partial A_1 \subset \partial A_2$. From this it follows that it is enough to show that the bound holds for $|A| = b^{(k)}(m_k, \ldots, m_s)$.

Furthermore, it is sufficient to show this bound for compressed A as compression at most reduces the size of the shadow and we are looking for a lower bound. Thus we will assume A to be compressed in the following.

The proof uses double induction over k and $m = |A|$. First we show that the theorem holds for the cases of $k = 1$ for any m and $m = 1$ for any k. In the induction step we show that if the theorem holds for $1, \ldots, k-1$ and any m and for $1, \ldots, m-1$ and k then it also holds for k and m, see figure (15).

In the case of $k = 1$ (as A is compressed) one has:

$$A = B^{(1)}(m) = \{e_0, \ldots, e_{m-1}\}$$

and so

$$\partial A = \partial B^{(1)}(m) = \{0\}$$

hence $|\partial A| = 1 = b^{(0)}(m)$.

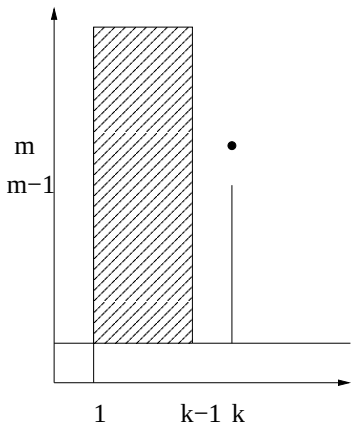

Fig. 15. Double induction

In the case of $m = |A| = 1$ one has:

$$A = B^{(k)}(k) = \{e(0, \ldots, k-1)\}$$

and so:

$$\partial A = \partial B^{(k)}(k) = [k]^{(k-1)}$$

hence $|\partial A| = k = b^{(k-1)}(k)$.

The key step of the proof is a partition of A into bitvectors with bit 0 set and such for which bit 0 is not set: $A = A_0 \cup A_1$ where $A_0 = \{x \in A | x_0 = 0\}$ and $A_1 = \{x \in A | x_0 = 1\}$.

(1) If $x \in \partial A_0$ then $x + e_j \in A_0$ for some $j > 0$. As A is compressed it must also contain $x + e_0 = R_{0j}(x + e_j) \in A_1$ and so $x \in A_1 - e_0$ thus

$$|\partial A_0| \leq |A_1 - e_0| = |A_1|.$$

(2) A special case is $A = B^{(k)}(m_k, \ldots, m_s)$ where one has $|A_0| = b^{(k)}(m_k - 1, \ldots, m_s - 1)$ and $|A_1| = b^{(k-1)}(m_k - 1, \ldots, m_s - 1)$ and thus

$$m = b^{(k)}(m_k, \ldots, m_s) = b^{(k)}(m_k - 1, \ldots, m_s - 1) + b^{(k-1)}(m_k - 1, \ldots, m_s - 1)$$

(3) Now partition ∂A_1 into 2 parts:

$$\partial A_1 = (A_1 - e_0) \cup (\partial(A_1 - e_0) + e_0).$$

It follows from previous inequalities and the induction hypothesis that $|\partial A_1| = |A_1 - e_0| + |\partial(A_1 - e_0) + e_0| = |A_1| + |\partial(A_1 - e_0)| \geq b^{(k-1)}(m_k -$

$1, \ldots, m_s - 1) + b^{(k-2)}(m_k - 1, \ldots, m_s - 1) = b^{(k-1)}(m_k, \ldots, m_s)$ and hence

$$|\partial A| \geq |\partial A_1| \geq b^{(k-1)}(m_k, \ldots, m_s) \qquad \square$$

This theorem is the tool to derive the bounds for the size of future candidate itemsets based on a current itemset and the apriori principle.

Theorem 23: Let the sequence C_k satisfy the apriori property and let

$$|C_k| = b^{(k)}(m_k, \ldots, m_r).$$

Then

$$|C_{k+p}| \leq b^{(k+p)}(m_k, \ldots, m_r)$$

for all $p \leq r$.

Proof: The reason for the condition on p is that the shadows are well defined.

First, we choose r such that $m_r \leq r + p - 1$, $m_{r+1} \leq r + 1 + p - 1$, ..., $m_{s-1} \leq s - 1 + p - 1$ and $m_s \geq s + p - 1$. Note that $s = r$ and $s = k + 1$ may be possible.

Now we get an upper bound for the size $|C_k|$:

$$\begin{aligned} |C_k| &= b^{(k)}(m_k, \ldots, m_r) \\ &\leq b^{(k)}(m_k, \ldots, m_s) + \sum_{j=1}^{s-1} \binom{j+p-1}{j} \\ &= b^{(k)}(m_k, \ldots, m_s) + \binom{s+p-1}{s-1} - 1 \end{aligned}$$

according to a previous lemma.

If the theorem does not hold then $|C_{k+p}| > b^{(k+p)}(m_j, \ldots, m_r)$ and thus

$$\begin{aligned} |C_{k+p}| &\geq b^{(k+p)}(m_j, \ldots, m_r) + 1 \\ &\geq b^{(k+p)}(m_k, \ldots, m_s) + \binom{s+p-1}{s+p-1} \\ &= b^{(k+p)}(m_k, \ldots, m_s, s+p-1). \end{aligned}$$

Here we can apply the previous theorem to get a lower bound for C_k:

$$|C_k| \geq b^{(k)}(m_k, \ldots, m_s, s+p-1).$$

This, however is contradicting the higher upper bound we got previously and so we have to have $|C_{k+p}| \leq b^{(k+p)}(m_j, \ldots, m_r)$. $\square$

As a simple consequence one also gets tightness:

Corollary 24: *For any m and k there exists a C_k with $|C_k| = m = b^{(k+p)}(m_k, \ldots, m_{s+1})$. such that*

$$|C_{k+p}| = b^{(k+p)}(m_k, \ldots, m_{s+1}).$$

Proof: The C_k consists of the first m k-itemsets in the colexicographic ordering. □

In practice one would know not only the size but also the contents of any C_k and from that one can get a much better bound than the one provided by the theory. A consequence of the theorem is that for L_k with $|L_k| \leq \binom{m_k}{k}$ one has $|C_{k+p}| \leq \binom{m_k}{k+p}$. In particular, one has $C_{k+p} = \emptyset$ for $k > m_p - p$.

3.4. *Apriori Tid*

The apriori algorithm discussed above computes supports of itemsets by doing intersections of columns. Some of these intersections are repeated over time and, in particular, entries of the Boolean matrix are revisited which have no impact on the support. The Apriori TID algorithm provides a solution to some of these problems. For computing the supports for larger itemsets it does not revisit the original table but transforms the table as it goes along. The new columns correspond to the candidate items. In this way each new candidate itemset only requires the intersection of two old ones.

The following shows an example on how this works. The example is adapted from [3]. In the first row the itemsets from C_k are depicted. The minimal support is 50 percent or 2 rows. The initial data is:

$$\left[\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline & 1 & 0 & 1 & 1 & 0 \\ & 0 & 1 & 1 & 0 & 1 \\ & 1 & 1 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Note that the column (or item) four is not frequent and thus not considered

for C_k. Thus one gets in the Apriori TID after one step the matrix:

$$\left[\begin{array}{cccccc} (1,2) & (1,3) & (1,5) & (2,3) & (2,5) & (3,5) \\ \hline 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

Here one can see directly that the itemsets $(1,2)$ and $(1,5)$ are not frequent. It turns out that there remains only one candidate, namely $(2,3,5)$ and the matrix is

$$\left[\begin{array}{c} (2,3,5) \\ \hline 0 \\ 1 \\ 1 \\ 0 \end{array}\right]$$

Let $z(j_1,\ldots,j_k)$ denote the elements of C_k. Then the elements in the transformed Boolean matrix are $a_{z(j_1,\ldots,j_k)}(x_i)$.

We will use again an auxiliary array $v \in \{0,1\}^n$. The apriori tid algorithm uses the join operation from earlier in order to construct a matrix for the frequent itemsets L_{k+1} from L_k. (This, as the previous algorithms are assumed to be in-memory, i.e., all data is stored in memory. The case of very large data sets which do not fit into memory will be discussed later.) The key part of the algorithm, i.e., the step from k to $k+1$ is then:

(1) Select pair of frequent k-itemsets (y,z), mark as read
(2) expand $x^y = \bigwedge_i x_i^{y_i}$, i.e., $v \leftarrow x^y$
(3) extract elements using x^z, i.e., $w \leftarrow v[x^z]$
(4) compress result and reset v to zero, $v \leftarrow 0$

There are three major steps where the auxiliary vector v is accessed. The time complexity for this is

$$T = \sum_{i=1}^{n} (2|(x^{(i)})^y| + |(x^{(i)})^z|)\tau.$$

This has to be done for all elements $y \vee z$ where $y, z \in L_k$. Thus the average complexity is

$$E(T) = \sum_k 3nm_k E(x^y)\tau$$

for some "average" y and $x^y = \bigwedge_i x_i^{y_i}$. Now for all elements in L_k the support is larger than σ, thus $E(x^y) \geq \sigma$. So we get a lower bound for the complexity:

$$E(T) \geq \sum_k 3nm_k\sigma\tau.$$

We can also obtain a simple upper bound if we observe that $E(x^y) \leq E(|x|)/d$ which is true "on average". From this we get

$$E(T) \leq \sum_k 3nm_k \frac{E(|x|)}{d}\tau.$$

Another (typically lower bound) approximation is obtained if we assume that the components of x are indepdendent. In this case $E(x^y) \approx (E(|x|)/d)^k$ and thus

$$E(T) \geq \sum_k 3nm_k(E(|x|)/d)^k\tau.$$

From this we would expect that for some $r_k \in [1, k]$ we get the approximation

$$E(T) \approx \sum_k 3nm_k(E(|x|)/d)^{r_k}\tau.$$

Now recall that the original column-wise apriori implementation required

$$E(T) \approx \sum_k 3nm_k k(E(|x|)/d)\tau$$

and so the "speedup" we can achieve by using this new algorithm is around

$$S \approx \frac{\sum_k km_k}{\sum_k (E(|x|)/d)^{r_k-1}m_k}.$$

which can be substantial as both $k \geq 1$ and $E(|x|)/d)^{r_k-1} < 1$. We can see that there are two reasons for the decrease in work: First we have reused earlier computations of $x_{j_1} \wedge \cdots \wedge x_{j_k}$ and second we are able to make use of the lower support of the k-itemsets for larger k. While this second effect does strongly depend on r_k and thus the data, the first effect allways holds, so we get a speedup of at least

$$S \geq \frac{\sum_k km_k}{\sum_k m_k},$$

i.e., the average size of the k-itemsets. Note that the role of the number m_k of candidate itemsets maybe slightly diminished but this is still the core parameter which determines the complexity of the algorithm and the need to reduce the size of the frequent itemsets is not diminished.

3.5. *Constrained association rules*

The amount of frequent itemsets found by the apriori algorithm will often be too large or too small. While the prime mechanism of controlling the number itemsets found is the minimal support σ, this may often not be enough as small collections of frequent itemsets may often contain mostly well known associations whereas large collections may reflect mostly random fluctuations. There are effective other ways to controll the amount of itemsets obtained. First, in the case of too many itemsets one can use constraints to filter out trivial or otherwise uninteresting itemsets. In the case of too few frequent itemsets one can also change the attributes or features which define the vector x. In particular, one can introduce new "more general" attributes. For example, one might find that rules with "ginger beer" are not frequent. However, rules with "soft drinks" will have much higher support and may lead to interesting new rules. Thus one introduces new more general items. However, including more general items while maintaining the original special items leads to duplications in the itemsets, in our example the itemset containing ginger beer and soft drinks is identical to the set which only contains ginger beer. In order to avoid this one can again introduce constraints, which, in our example would identify the itemset containing ginger beer only with the one containing softdrink and ginger beer.

Constraints are conditions for the frequent itemsets of interest. These conditions take the form "predicate = true" with some predicates

$$b_1(z), \ldots, b_s(z).$$

Thus one is looking for frequent k-itemsets L_k^* for which the b_j are true, i.e.,

$$L_k^* := \{z \in L_k \mid b_j(z) = 1\}.$$

These constraints will reduce the amount of frequent itemsets which need to be further processed, but can they also assist in making the algorithms more efficient? This will be discussed next after we have considered some examples. Note that the constraints are not necessarily simple conjunctions! Examples:

- We have mentioned the rule that any frequent itemset should not contain an item and its generalisation, e.g., it should not contain both soft drinks and ginger beer as this is identical to ginger beer. The constraint is of the form $b(x) = \neg a_y(x)$ where y is the itemset where the "softdrink and ginger beer bits" are set.

- In some cases, frequent itemsets have been well established earlier. An example are crisps and soft drinks. There is no need to rediscover this association. Here the constraint is of the form $b(x) = \neg\delta_y(x)$ where y denotes the itemset "softdrinks and chips".
- In some cases, the domain knowledge tells us that some itemsets are prescribed, like in the case of a medical schedule which prescribes certain procedures to be done jointly but others should not be jointly. Finding these rules is not interesting. Here the constraint would exclude certain z, i.e., $b(z) = \neg\delta_y(z)$ where y is the element to exclude.
- In some cases, the itemsets are related by definition. For example the predicates defined by $|z| > 2$ is a consequence of $|z| > 4$. Having discovered the second one relieves us of the need to discover the first one. This, however, is a different type of constraint which needs to be considered when defining the search space.

A general algorithm for the determination of the L_k^* determines at every step the L_k (which are required for the continuation) and from those outputs the elements of L_k^*. The algorithm is exactly the same as apriori or apriori tid except that not all frequent itemsets are output. See Algorithm 2. The work is almost exactly the same as for the original apriori

Algorithm 2 Apriori with general constraints

$C_1 = \mathcal{A}(\mathbb{X})$ is the set of all one-itemsets, $k = 1$
while $C_k \neq \emptyset$ **do**
  scan database to determine support of all $z \in C_k$
  extract frequent itemsets from C_k into L_k
  use the constraints to extract the constrained frequent itemsets in L_k^*
  generate C_{k+1}
  $k := k + 1$.
end while

algorithm.

Now we would like to understand how the constraints can impact the computational performance, after all, one will require less rules in the end and the discovery of less rules should be faster. This, however, is not straight-forward as the constrained frequent itemsets L_k^* do not necessarily satisfy the apriori property. There is, however an important class of constraints for which the apriori property holds:

Theorem 25: If the constraints $b_j,\ j = 1, \ldots, m$ are anti-monotone then the set of constrained frequent itemsets $\{L_k^*\}$ satisfies the apriori condition.

Proof: Let $y \in L_k^*$ and $z \leq y$. As $L_k^* \subset L_k$ and the (unconstrained frequent itemsets) L_k satisfy the apriori condition one has $z \in L_{\text{size}(z)}$.

As the b_j are antimonotone and $y \in L_k^*$ one has

$$b_j(z) \geq b_j(y) = 1$$

and so $b_j(z) = 1$ from which it follows that $z \in L^*_{\text{size}(z)}$. □

When the apriori condition holds one can generate the candidate itemsets C_k in the (constrained) apriori algorithm from the sets L_k^* instead of from the larger L_k. However, the constraints need to be anti-monotone. We know that constraints of the form $a_{z^{(j)}}$ are monotone and thus constraints of the form $b_j = \neg a_{z^{(j)}}$ are antimonotone. Such constraints say that a certain combination of items should not occur in the itemset. An example of this is the case of ginger beer and soft drinks. Thus we will have simpler frequent itemsets in general if we apply such a rule. Note that itemsets have played three different roles so far:

(1) as data points $x^{(i)}$
(2) as potentially frequent itemsets z and
(3) to define constraints $\neg a_{z^{(j)}}$.

The constraints of the kind $b_j = \neg a_{z^{(j)}}$ are now used to reduce the candidate itemsets C_k *prior* to the data base scan (this is where we save). Even better, it turns out that the conditions only need to be checked for level $k = |z^{(j)}|$. (This gives a minor saving.) This is summarised in the next theorem:

Theorem 26: Let the constraints be $b_j = \neg a_{z^{(j)}}$ for $j = 1, \ldots, s$. Furthermore let the candidate k-itemsets for L_k^* be sets of k-itemsets such that

$$C_k^* = \{y \in \mathcal{I}_k \mid \text{ if } z < y \text{ then } z \in L_{|z|} \text{ and } b_j(y) = 1\}$$

and a further set defined by

$$\tilde{C}_k = \{y \in \mathcal{I}_k \mid \text{if } z < y \text{ then } z \in L_{|z|} \text{ and if } |z^{(j)}| = k \text{ then } b_j(y) = 1\ \}.$$

Then $\tilde{C}_k = C_k^*$.

Proof: We need to show that every element $y \in \tilde{C}_k$ satisfies the constraints $b_j(y) = 1$. Remember that $|y| = k$. There are three cases:

- If $|z^{(j)}| = |y|$ then the constraint is satisfied by definition
- If $|z^{(j)}| > |y|$ then $z^{(j)} \not\leq y$ and so $b_j(y) = 1$
- Consider the case $|z^{(j)}| < |y|$. If $b_j(y) = 0$ then $a_{z^{(j)}}(y) = 1$ and so $z^{(j)} \leq y$. As $|z^{(j)}| < |y|$ it follows $z^{(j)} < y$. Thus it follows that $z^{(j)} \in L^*_{z^{(j)}}$ and, consequently, $b_j(z^{(j)}) = 1$ or $z^{(j)} \not\leq z^{(j)}$ which is not true. It follows that in this case we have $b_j(y) = 1$.

From this it follows that $\tilde{C}_k \subset C^*_k$. The converse is a direct consequence of the definition of the sets. □

Thus we get a variant of the apriori algorithm which checks the constraints only for one level, and moreover, this is done to reduce the number of candidate itemsets. This is Algorithm 3.

Algorithm 3 Apriori with antimonotone constraints

$C_1 = \mathcal{A}(\mathbb{X})$ is the set of all one-itemsets, $k = 1$
while $C_k \neq \emptyset$ **do**
  extract elements of C_k which satisfy the constraints $a_{z^{(j)}}(x) = 0$ for $|z^{(j)}| = k$ and put into C^*_k
  scan database to determine support of all $y \in C^*_k$
  extract frequent itemsets from C^*_k into L^*_k
  generate C_{k+1} (as per ordinary apriori)
  $k := k + 1$.
end while

3.6. *Partitioned algorithms*

The previous algorithms assumed that all the data was able to fit into main memory and was resident in one place. Also, the algorithm was for one processor. We will look here into partitioned algorithms which lead to parallel, distributed and out-of-core algorithms with few synchronisation points and disk scans. The algorithms have been suggested in [10].

We assume that the data is partitioned into equal parts as

$$D = [D_1, D_2, \ldots, D_p]$$

where $D_1 = (x^{(1)}, \ldots, x^{(n/p)})$, $D_2 = (x^{(n/p+1)}, \ldots, x^{(2n/p)})$, etc. While we assume equal distribution it is simple to generalise the discussions below to non-equal distributions.

In each partition D_j an estimate for the support $s(a)$ of a predicate can be determined and we will call this $\hat{s}_j(a)$. If $\hat{s}(a)$ is the estimate of the support in D then one has

$$\hat{s}(a) = \frac{1}{p} \sum_{j=1}^{p} \hat{s}_j(a).$$

This leads to a straight-forward parallel implementation of the apriori algorithm: The extraction of the L_k can either be done on all the processors

Algorithm 4 Parallel Apriori

$C_1 = \mathcal{A}(\mathbb{X})$ is the set of all one-itemsets, $k = 1$
while $C_k \neq \emptyset$ **do**
  scan database to determine support of all $z \in C_k$ on each D_j and sum up the results
  extract frequent itemsets from C_k into L_k
  generate C_{k+1}
  $k := k + 1$.
end while

redundantly or on one master processor and the result can then be communicated. The parallel algorithm also leads to an out-of-core algorithm which does the counting of the supports in blocks. One can equally develop an apriori-tid variant as well.

There is a disadvantage of this straight-forward approach, however. It does require many synchronisation points, respectively, many scans of the disk, one for each level. As the disks are slow and synchronisation expensive this will cost some time. We will not discuss and algorithm suggested by [10] which substantially reduces disk scans or synchronisation points at the cost of some redundant computations. First we observe that

$$\min_k \hat{s}_k(a) \leq \hat{s}(a) \leq \max_k \hat{s}_k(a)$$

which follows from the summation formula above. A consequence of this is

Theorem 27: Each a which is frequent in D is at least frequent in one D_j.

Proof: If for some frequent a this would not hold then one would get

$$\max \hat{s}_j(a) < \sigma_0$$

if σ_0 is the threshold for frequent a. By the observation above $\hat{s}(a) < \sigma_0$ which contradicts the assumption that a is frequent. □

Using this one gets an algorithm which generates in a first step frequent k-itemsets $L_{k,j}$ for each D_j and each k. This requires one scan of the data, or can be done on one processor, respectively. The union of all these frequent itemset is then used as a set of candidate itemsets and the supports of all these candidates is found in a second scan of the data. The parallel variant of the algorithm is then Algorithm 5. Note that the supports for all the levels

Algorithm 5 Parallel Association Rules

determine the frequent k-itemsets $L_{k,j}$ for all D_j in parallel
$C_k^p := \bigcup_{j=1}^p L_{k,j}$ and broadcast
determine supports $\hat{s}_k$ for all candidates and all partitions in parallel
collect all the supports, sum up and extract the frequent elements from C_k^p.

k are collected simultaneously thus they require only two synchronisation points. Also, the apriori property holds for the C_k^p:

Proposition 28: *The sequence C_k^p satisfies the apriori property, i.e.,*

$$z \in C_k^p \quad \& \quad y \leq z \quad \Rightarrow y \in C_{|y|}^p.$$

Proof: If $z \in C_k^p \quad \& \quad y \leq z$ then there exists a j such that $z \in L_{k,j}$. By the apriori property on D_j one has $y \in L_{|y|,j}$ and so $y \in C_{|y|}^p$. □

In order to understand the efficiency of the algorithm one needs to estimate the size of the C_k^p. In the (computationally best case, all the frequent itemsets are identified on the partitions and thus

$$C_k^p = L_{k,j} = L_k.$$

We can use any algorithm to determine the frequent itemsets on one partition, and, if we assume that the algorithm is scalable in the data size the time to determine the frequent itemsets on all processors is equal to $1/p$ of the time required to determine the frequent itemsets on one processor as the data is $1/p$ on each processor. In addition we require to reads of the data base which has an expectation of $n\lambda\tau_{Disk}/p$ where λ is the average size of the market baskets and τ_{Disk} is the time for one disk access. There

is also some time required for the communication which is proportional to the size of the frequent itemsets. We will leave the further analysis which follows the same lines as our earlier analysis to the reader at this stage.

As the partition is random, one can actually get away with the determination of the supports for a small subset of C_k^p, as we only need to determine the support for a_z for which the supports have not been determined in the first scan. One may also wish to choose the minimal support σ for the first scan slightly lower in order to further reduce the amount of second scans required.

3.7. *Finding strong rules*

The determination of rules of the kind $a_z \Rightarrow a_y$ is the last step in the apriori algorithm. As it does not require any scanning of the data it has been given less attention. However, in [3] the authors show how to apply the antimonotonicity again to get some gains.

Remember that we are interested in *strong association rules* of the form

$$a_z \Rightarrow a_y$$

for which $z \vee y$ is a frequent itemset and for which the *confidence* is larger than a given threshold $\gamma_0 > 0$:

$$\mathrm{conf}(a_z \Rightarrow a_y) = \frac{s(a_{z\vee y})}{s(a_z)} \geq \gamma_0.$$

A simple procedure would now visit each frequent itemset z and look at all pairs z_1, z_2 such that $z = z_1 \vee z_2$ and $z_1 \wedge z_2 = 0$ and consider all rules $a_{z_1} \Rightarrow a_{z_2}$. This procedure can be improved by taking into account that

Theorem 29: Let $z = z_1 \vee z_2 = z_3 \vee z_4$ and $z_1 \wedge z_2 = z_3 \wedge z_4 = 0$. Then if $a_{z_1} \Rightarrow a_{z_2}$ is a strong association rule and $z_3 \geq z_1$ then so is $a_{z_3} \Rightarrow a_{z_4}$.

This is basically "the apriori property for the rules" and allows pruning the tree of possible rules quite a lot. The theorem is again used as a necessary condition. We start the algorithm by considering $z = z_1 \vee z_2$ with 1-itemsets for z_2 and looking at all strong rules. Then, if we consider a 2-itemset for z_2 both subsets $y < z_2$ need to be consequents of strong rules in order for z_2 to be a candidate of a consequent. By constructing the consequents taking into account that all their nearest neighbors (their cover in lattice terminology) need to be consequents as well. Due to the interpretability problem one is mostly interested in small consequent itemsets so that this is not really a big consideration.

4. Other Approaches

4.1. *Mining Sequences*

The following is an example of how one may construct more complex structures from the simple market baskets. We consider here a special case of sequences, see [7,1]. Let the data be of the form

$$(x_1, \ldots, x_m)$$

where each x_i is an itemset (not a component as in our earlier notation. Examples of sequences correspond to the shopping behaviour of customers of retailers over time, or the sequence of services a patient receives over time. The focus is thus not on individual market-baskets but on the customers. We do not discuss the temporal aspects, just the sequencial ones.

In defining our space of features we include the empty sequence () but not components of the sequences are 0, i.e.,

$$x_i \neq 0.$$

The rationale for this is that sequences correspond to actions which occur in some order and 0 would correspond to a non-action. We are not interested in the times when a shoper went to the store and didn't buy anything at all. Any empty component itemsets in the data will also be removed.

The sequences also have an intrinsic partial ordering

$$\mathbf{x} \leq \mathbf{y}$$

which holds for $(x_1, \ldots, x_m)$ and $(y_1, \ldots, y_k)$ when ever there is a sequence $1 \leq i_1 < i_2 < \cdots < i_m \leq k$ such that

$$x_i \leq y_{i_s}, \quad s = 1, \ldots, m.$$

One can now verify that this defines a partial order on the set of sequences introduced above. However, the set of sequences does not form a lattice as there are not necessarily unique lowest upper or greatest lower bounds. For example, the two sequences $((0,1),(1,1),(1,0))$ and $((0,1),(0,1))$ have the two (joint) upper bounds $((0,1),(1,1),(1,0),(0,1))$ and $((0,1),(0,1),(1,1),(1,0)$ which have now common lower bound which is still an upper bound for both original sequences. This makes the search for frequent itemsets somewhat harder.

Another difference is that the complexity of the mining tasks has grown considerably, with $|\mathcal{I}|$ items one has $2^{|\mathcal{I}|}$ market-baskets and thus $2^{|\mathcal{I}|m}$ different sequences of length $\leq m$. Thus it is essential to be able to deal with the computational complexity of this problem. Note in particular, that

the probability of any particular sequence is going to be extremely small. However, one will be able to make statements about the support of small subsequences which correspond to shopping or treatment patterns.

Based on the ordering, the *support* of a sequence $\mathbf{x}$ is the set of all sequences larger than $\mathbf{x}$:

$$\text{supp}(\mathbf{x}) = \{\mathbf{y} | \mathbf{x} \leq \mathbf{y}\}.$$

The probability of this set of sequences is again called support:

$$s(\mathbf{x}) = P(\text{supp}(\mathbf{x})).$$

This is estimated by the number of sequences in the data base which are in the support. Note that the itemsets now occur as length 1 sequences and thus the support of the itemsets can be identified with the support of the corresponding 1 sequence. As our focus is now on sequences this is different from the support we get if we look just at the distribution of the itemsets.

The length of a sequence is the number of non-empty components. Thus we can now define an apriori algorithm as before. This would start with the determination of all the frequent 1 sequences which correspond to all the frequent itemsets. Thus the first step of the sequence mining algorithm is just the ordinary apriori algorithm. Then the apriori algorithm continues as before, where the candidate generation step is similar but now we join any two sequences which have all components identical except for the last (non-empty) one. Then one gets a sequence of length $m + 1$ from two such sequences of length m by concatenating the last component of the second sequence on to the first one. After that one still needs to check if all subsequences are frequent to do some pruning.

There has been some arbitrariness in some of the choices. Alternatives choose the size of a sequence as the sum of the sizes of the itemsets. In this case the candidate generation procedure becomes slightly more complex, see [1].

4.2. *The FP tree algorithm*

The Apriori algorithm is very effective for discovering a reasonable number of small frequent itemsets. However it does show severe performance problems for the discovery of large numbers of frequent itemsets. If, for example, there are 10^6 frequent items then the set of candidate 2-itemsets contains $5 \cdot 10^{11}$ itemsets which all require testing. In addition, the Apriori algorithm has problems with the discovery of very long frequent itemsets.

For the discovery of an itemset with 100 items the algorithm requires scanning the data for all the 2^{100} subsets in 100 scans. The bottleneck in the algorithm is the creation of the candidate itemsets, more precisely, the number of candidate itemsets which need to be created during the mining. The reason for this large number is that the candidate itemsets are visited in a breadth-first way.

The FP tree algorithm addresses these issues and scans the data in a depth-first way. The data is only scanned twice. In a first scan, the frequent items (or 1-itemsets) are determined. The data items are then ordered based on their frequency and the infrequent items are removed. In the second scan, the data base is mapped onto a tree structure. Except for the root all the nodes are labelled with items, each item can correspond to multiple nodes. We will explain the algorithm with the help of an example, see table **??** for the original data and the records with the frequent itemsets only (here we look for support > 0.5).

items	$s > 0.5$
f, a, c, d, g, i, m, p	f, c, a, m, p
a, b, c, f, l, m, o	f, c, a, b, m
b, f, h, j, o, w	f, b
b, c, k, s, p	c, b, p
a, f, c, e, l, p, m, n	f, c, a, m, p

Initially the tree consists only of the root. Then the first record is read and a path is attached to the root such that the node labelled with the first item of the record (items are ordered by their frequency) is adjacent to the root, the second item labels the next neighbor and so on. In addition to the item, the label also contains the number 1, see Step 1 in figure 16. Then the second record is included such that any common prefix (in the example the items f,c,a is shared with the previous record and the remaining items are added in a splitted path. The numeric parts of the labels of the shared prefix nodes are increased by one, see Step 2 in the figure. This is then done with all the other records until the whole data base is stored in the tree. As the most common items were ordered first, there is a big likelihood that many prefixes will be shared which results in substantial saving or compression of the data base. Note that no information is lost with respect to the supports. The FP tree structure is completed by adding a header table which contains all items together with pointers to their first

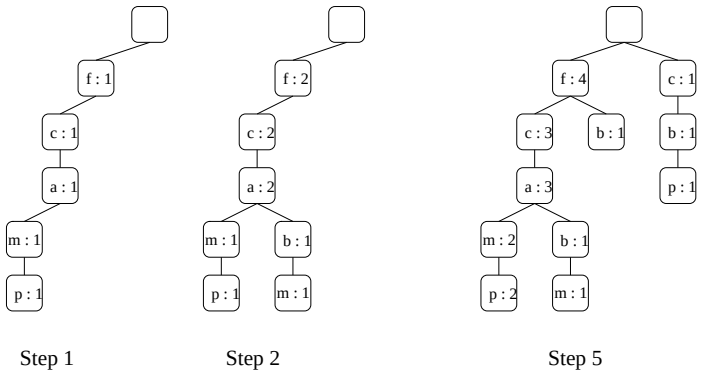

Fig. 16. Construction of the FP-Tree

occurrence in the tree. The other occurrences are then linked together so that all occurrences of an item can easily be retrieved, see figure 17.

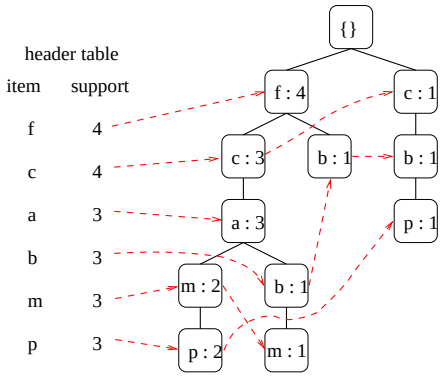

Fig. 17. Final FP-Tree

The FP tree does never break a long pattern into smaller patterns as this is done when searching the candidate itemsets. Long patterns can be directly retrieved from the FP tree. The FP does also contain the full relevant information about the data base. It is compact, as all infrequent items are removed and the highly frequent items share nodes in the tree. The number of nodes is never less than the size of the data base measured in the sum of the sizes of the records but there is anecdotal evidence that compression rates can be over 100.

The FP tree is used to find all association rules containing particular items. Starting with the least frequent items, all rules containing those items can be found simply by generating for each item the conditional data base which consists for each path which contains the item of those items which are between that item and the root. (The lower items don't need to be considered, as they are considered together with other items.) These *conditional pattern bases* can then again be put into FP-trees, the *conditional FP-trees* and for those trees all the rules containing the previously selected and any other item will be extracted. If the conditional pattern base contains only one item, that item has to be the itemset. The frequencies of these itemsets can be obtained from the number labels.

An additional speed-up is obtained by mining any long prefix paths and the rest separately and combine the results at the end. Of course any chain does not need to be broken into parts necessarily as all the frequent subsets, together with their frequencies are easily obtained directly.

References

1. Jean-Marc Adamo. *Data Mining for Association Rules and Sequential Patterns*. Springer, New York, 2001.
2. R. Agrawal, T. Imielinski, and T. Swami. Mining association rules between sets of items in large databases. In *Proc., ACM SIGMOD Conf. on Manag. of Data*, pages 207–216, Washington, D.C., 1993.
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. Int. Conf. on Very Large Data Bases*, pages 478–499, Santiago, Chile, 1994. Morgan Kaufman, San Francisco.
4. Béla Bollobaś. *Combinatorics*. Cambridge University Press, 1986.
5. B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, New York, second edition, 2002.
6. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*. Springer, 1999.
7. Jiawei Han and Micheline Kamber. *Data Mining, Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.
8. donor Jeff Schlimmer. *Congressional Quarterly Almanac, 98th Congress, 2nd session 1984*, volume XL. Congressional Quarterly Inc., 1985.
9. H. Mannila, H. Toivonen, and A.I. Verkamo. Discovering frequent episodes in sequences. In *Proc. 1st Int. Conf. Knowledge Discovery Databases and Data Mining*, pages 210–215, Menlo Park, Calif., 1995. AAAI Press.
10. A. Savasere, E. Omieski, and S. Navanthe. An efficient algorithm for mining association rules in large databases. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 432–444. Morgan Kaufmann, San Francisco, 1995.
11. Peter Whittle. *Probability via expectation*. Springer Texts in Statistics. Springer-Verlag, New York, fourth edition, 2000.