The Apriori Algorithm – a Tutorial

Markus Hegland
CMA, Australian National University
John Dedman Building, Canberra ACT 0200, Australia
E-mail: Markus.Hegland@anu.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. The first and arguably most influential algorithm for efficient association rule discovery is Apriori.

In the following we will review basic concepts of association rule discovery including support, confidence, the apriori property, constraints and parallel algorithms. The core 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

Large amounts of data have been collected routinely in the course of day-to-day management in business, administration, banking, the delivery of social and health services, environmental protection, security and in politics. Such data is primarily used for accounting and for management of the customer base. Typically, management data sets are very large and constantly growing and contain a large number of complex features. While these data sets reflect properties of the managed subjects and relations, and are thus potentially of some use to their owner, they often have relatively low information density. One requires robust, simple and computationally efficient tools to extract information from such data sets. The development and understanding of such tools is the core business of data mining. These tools are based on ideas from computer science, mathematics and statistics.

The introduction of association rule mining in 1993 by Agrawal, Imielinski and Swami [**?**] and, in particular, the development of an efficient algorithm by Agrawal and Srikant [**?**] and by Mannila, Toivonen and Verkamo [**?**] marked a shift of the focus in the young discipline of data mining onto rules and data bases. Consequently, besides involving the traditional statistical and machine learning community, data mining now attracted researchers with a variety of skills ranging from computer science, mathematics, science, to business and administration. The urgent need for computational tools to extract information from data bases and for manpower to apply these tools has allowed a diverse community to settle in this new area. The data analysis aspect of data mining is more exploratory than in statistics and consequently, the mathematical roots of probability are somewhat less prominent in data mining than in statistics. Computationally, however, data mining frequently requires the solution of large and complex search and optimisation problems and it is here where mathematical methods can assist most. This is particularly the case for association rule mining which requires searching large data bases for complex rules. Mathematical *modelling* is required in order to generalise the original techniques used in market basket analysis to a wide variety of applications. Mathematical *analysis* provides insights into the performance of the algorithms.

An association rule is an *implication* or *if-then-rule* which is supported by data. The motivation given in [**?**] 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". Insights into customer behaviour may also be obtained through customer surveys, but the analysis of the transactional data has the advantage of being much cheaper and covering all current customers. Compared to customer surveys, the analysis of transactional data does have some severe limitations, however. For example, point-of-sale data typically does not contain any information about personal interests, age and occupation of customers. Nonetheless, market basket analysis can provide new insights into customer behaviour and has led to higher profits through better customer relations, customer retention, better product placements, product development and fraud detection.

Market basket analysis is not limited to retail shopping but has also been applied in other business areas including

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

Association rule mining generalises market basket analysis and is used in many other areas including genomics, text data analysis and Internet intrusion detection. For motivation we will in the following mostly focus on retail market basket analysis.

When a customer passes through a point of sale, the contents of his market basket are registered. This results in large collections of market basket data which provide information about which items were sold and, in particular, which combinations of items were sold. The small toy example in the table of figure 1 shall illustrate this further. Each row corresponds to a

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

Fig. 1. Five grocery market baskets

market basket or transaction containing popular retail items. An inspection of the table reveals that:

- Four of the five baskets contain orange juice,
- two baskets contain soda water
- half of the baskets which contain orange juice also contain soda
- all the baskets which contain soda also contain juice

These rules are very simple as is typical for association rule mining. Simple rules are understandable and ultimately useful. In a large retail shop there are usually more than 10,000 items on sale and the shop may service thousands of customers every day. Thus the size of the collected data is substantial and even the detection of simple rules like the ones above requires sophisticated algorithms. The efficiency of the algorithms will depend on the particular characteristics of the data sets. An important feature of

many retail data sets is that an average market basket only contains a small subset of all items available.

The simplest model for the customers assumes that the customers choose products from the shelves in the shop at random. In this case the choice of each product is independent from any other product. Consequently, association rule discovery will simply recover the likelihoods for any item to be chosen. While it is important to compare the performance of other models with this "null-hypothesis" one would usually find that shoppers do have a more complex approach when they fill the shopping basket (or trolley). They will buy breakfast items, lunch items, dinner items and snacks, party drinks, and Sunday dinners. They will have preferences for cheap items, for (particular) brand items, for high-quality, for freshness, low-fat, special diet and environmentally safe items. Such goals and preferences of the shopper will influence the choices but can not be directly observed. In some sense, market basket analysis should provide information about how the shoppers choose. In order to understand this a bit further consider the case of politicians who vote according to party policy but where we will assume for the moment that the party membership is unknown. Is it possible to see an effect of the party membership in voting data? For a small but real illustrative example consider the US Congress voting records from 1984 [**?**], see figure 2. 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. The left data matrix in figure 2 is the original data where only the rows and columns have been randomly permuted to remove any information introduced through the way the data was collected. The matrix on the right side is purely random such that each entry is independent and only the total number of entries is maintained. Can you see the difference between the two bit matrices? We found that for most people, the difference between the two matrices is not obvious from visual inspection alone.

Data mining aims to discover patterns in the left bit matrix and thus differences between the two examples. 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).

In this example the selection of what are rows and what columns is

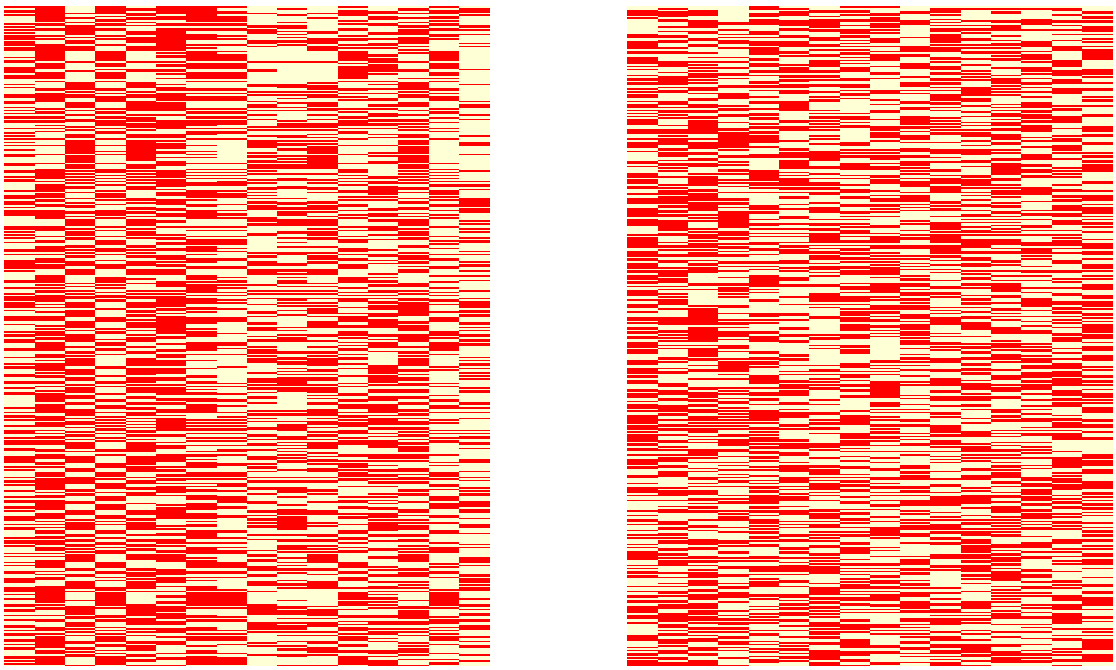

Fig. 2. 1984 US House of Representatives Votes, with 16 items voted on

somewhat arbitrary. Instead of patterns regarding the items voted on one might be interested in patterns relating 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 interactions between the members of Congress. 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 in many cases. This is in particular exploited in some newer association rule discovery algorithms which are based on formal concept analysis [**?**].

By inspecting the data matrix of the voting example, one 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 and other issues concerning inference belong to statistics and are beyond the scope of this tutorial which focusses on computational issues.

2. Itemsets and Associations

In this section a formal mathematical model is derived to describe itemsets and associations to provide a framework for the discussion of the apriori algorithm. In order to apply ideas from market basket analysis to other areas one needs a general description of market baskets which can equally describe collections of medical services received by a patient during an episode of care, subsequences of amino acid sequences of a protein, and collections or words or concepts used on web pages. In this general description the items are numbered and a market basket is represented by an indicator vector.

2.1. *The Datamodel*

In this subsection a probabilistic model for the data is given along with some simple model examples. For this, we consider the voting data example again.

First, the items are enumerated as $0, \ldots, d-1$. Often, enumeration is done such that the more frequent items correspond to lower numbers but this is not essential. Itemsets are then sets of integers between 0 and $d-1$. The itemsets are represented by bitvectors $x \in \mathbb{X} := \{0,1\}^d$ where item j is present in the corresponding itemset iff the j-th bit is set in x. Consider the "micromarket" with the items juice, bread, milk, cheese and potatoes with item numbers $0, 1, 2, 3$ and 4, respectively. The market basket containing bread, milk and potatoes is then mapped onto the set $\{1, 2, 4\}$ and is represented by the bitvector $(0, 1, 1, 0, 1)$. From the bitvector it is clear which elements are in the market basket or itemset and which are

not.

The data is a sequence of itemsets which is represented as a bitmatrix where each row corresponds to an itemset and the columns correspond to the items. For the micromarket example a dataset containing the market baskets {juice,bread, milk}, {potato} and {bread, potatoes} would be represented by the matrix

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

In the congressional voting example mentioned in the previous section the first few rows of matrix 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
```

and they correspond to the following "itemsets" (or sets of yes-votes):

```
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.
```

It is assumed that the data matrix $X \in \{0,1\}^{n,d}$ is random and thus the elements $x_j^{(i)}$ are binary random variables. One would in general have to assume correlations between both rows and columns. The correlations between the columns might relate to the type of shopping and customer, e.g., young family with small kids, weekend shopping or shopping for a specific dish. Correlations between the rows might relate to special offers of the retailer, time of the day and week. In the following it will be assumed that the rows are drawn independently from a population of market baskets. Thus it is assumed that there is a probability distribution function $p :\rightarrow [0,1]$ with

$$\sum_{x \in \mathbb{X}} p(x) = 1$$

where $\mathbb{X} = \{0,1\}^d$. The probability measure with distribution p is denoted by P and one has $P(A) = \sum_{x \in A} p(x)$.

The data can be represented as an empirical distribution with

$$p_{\text{emp}}(x) = \frac{1}{n}\sum_{i=1}^{n} \delta(x - x^{(i)})$$

where $\delta(x)$ is the indicator function where $\delta(0) = 1$ and $\delta(x) = 0$ if $x \neq 0$. (For simplicity the empty market basket is denoted by 0 instead of $(0, \ldots, 0)$.) All the information derived from the data is stored in p_{emp} but some sort of "smoothing" is required if one would like to generalise insights from the empirical distribution of one itemset collection to another to separate the noise from the noise from the signal. Association rule discovery has its own form of smoothing as will be seen in the following.

The task of *frequent itemset mining* consists of finding itemsets which occur frequently in market baskets. For this one recalls that the itemsets are partially ordered with respect to inclusion (the subset relation) and we write $x \leq y$ if the set with representation x is a subset of the set with representation y or $x = y$. With this partial order one defines the *support* of an itemset x to be

$$s(x) = P(\{z \mid x \leq z\}) \tag{1}$$

which is also called the *anticummulative distribution function* of the probability P. The support is a function $s : \mathbb{X} \to [0, 1]$ and $s(0) = 1$. By construction, the support is *antimonotone*, i.e., if $x \leq y$ then $p(x) \geq p(y)$. This antimonotonicity is the basis for efficient algorithms to find all *frequent itemsets* which are defined as itemsets for which $s(x) \geq \sigma_0 > 0$ for some user defined σ_0.

Equation 1 can be reformulated in terms of $p(x)$ as

$$s(x) = \sum_{z \geq x} p(z). \tag{2}$$

For given supports $s(x)$, this is a linear system of equations which can be solved recursively using $s(e) = p(e)$ (where $e = (1, \ldots, 1)$ is the maximal itemset) and

$$p(x) = s(x) - \sum_{z > x} p(z).$$

It follows that the support function $s(x)$ provides an alternative description of the probability measure P which is equivalent to p. However, for many examples the form of $s(x)$ turns out to be simpler. In the cases of market baskets it is highly unlikely, that market baskets contain large numbers of items and so approximations with $s(x) = 0$ for x with a large number of

nonzero components will usually produce good approximations of the itemset distribution $p(x)$. This leads to an effective smoothing mechanism for association rule discovery where the minimal support σ_0 acts as a smoothing parameter which in principle could be determined from a test data set or with crossvalidation.

2.1.1. *Example: The Random Shopper*

In the simplest case all the bits (items) in x are chosen independently with probability p_0. We call this the case of the "random shopper" as it corresponds to a shopper who fills the market basket with random items. The distribution is in this case

$$p(x) = p_0^{|x|}(1-p_0)^{d-|x|}$$

where $|x|$ is the number of bits set in x and d is the total number of items available, i.e., number of components of x. As any $z \geq x$ has at least all the bits set which are set in x one gets for the support

$$s(x) = p_0^{|x|}.$$

It follows that the frequent itemsets x are exactly the ones with few items, in particular, where

$$|x| \leq \log(\sigma_0)/\log(p_0).$$

For example, if one is interested in finding itemsets which are supported by one percent of the data records and if the probability of choosing any item is $p_0 = 0.1$ the frequent itemsets are the ones with at most two items. For large shops one would typically have p_0 much smaller. Note that if $p_0 < \sigma_0$ one would not get any frequent itemsets at all.

The random shopper is of course not a realistic model for shopping and one would in particular not expect to draw useful conclusions from the frequent itemsets. Basically, the random shopper is the market basket equivalent of noise. The above discussion, however, might be used to guide the choice of σ_0 to filter out the noise in market basket analysis. In particular, one could choose

$$\sigma_0 = \min_{|x|=1} s(x).$$

Note that in the random shopper case the rhs is just p_0. In this case, all the single items would be frequent.

A slight generalisation of the random shopper example above assumes that the items are selected independently but with different probabilities p_j. In this case one gets

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

and

$$s(x) = \prod_{j=1}^{d} p_j^{x_j}.$$

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 α. It follows again that itemsets with few popular items are most likely.

However, this type of structure is not really what is of interest in association rule mining. To illustrate this consider again the case of the US Congress voting data. In figure **??** the support for single itemsets are displayed for the case of the actual data matrix and for a random permutation of all the matrix elements. The supports are between 0.32 and 0.62 for the original data where for the randomly permuted case the supports are between 0.46 and 0.56. Note that these supports are computed from the data, in theory, the permuted case should have constant supports somewhere slightly below 0.5. More interesting than the variation of the supports of single items is the case when 2 items are considered. Supports for the votes displayed in figure **??**. are of the form "V2 and Vx". Note that "V2 and V2" is included for reference, even though this itemset has only one item. In the random data case where the vote for any pair {V2,Vx} (where Vx is not V2) is the square of the vote for the single item V2 as predicted by the "random shopper theory" above. One notes that some pairs have significantly higher supports than the random ones and others significantly lower supports. This type of behaviour is not captured by the "random shopper" model above even if the case of variable supports for single items are allowed. The following example attempts to model some this behaviour.

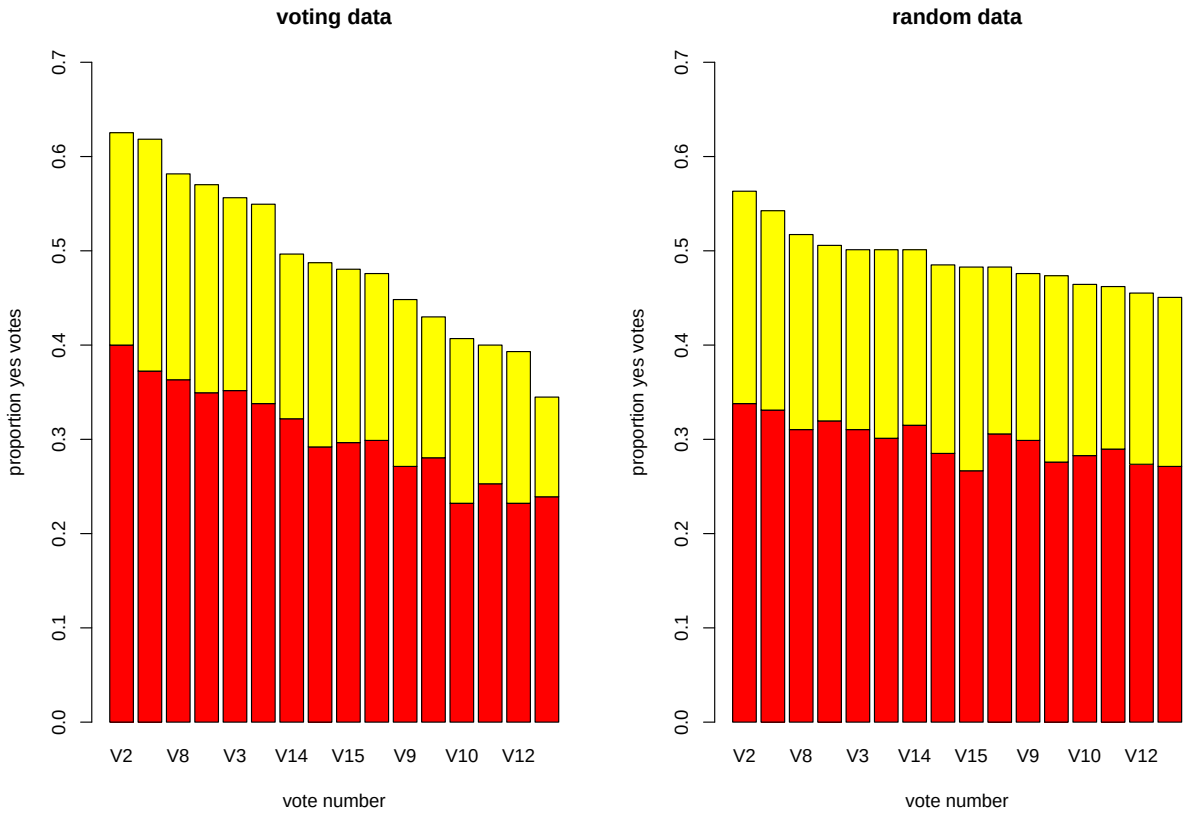

Fig. 3. Supports for all the votes in the US congress data (split by party).

2.1.2. *Example: Multiple Shopper and Item Classes*

Consider now the case of some simple structure. Assume that there are two types of shoppers and two types of items. Assume that the items are $x = (x_0, x_1)$ where the vectors x_i correspond to items of class i. In practice, the type of items might have to be discovered as well. Consider that the shoppers of type i have a probability π_i of filling a market basket where here $i = 0, 1$. Assume that it is not known to which type of shopper a market basket belongs. Finally, assume that the difference between the two types of shoppers relates to how likely they are to put items of the two types into their market basket. Let $p_{i,j}$ denote the probability that shopper of type i puts an item of type j into the market basket. In this case the probability

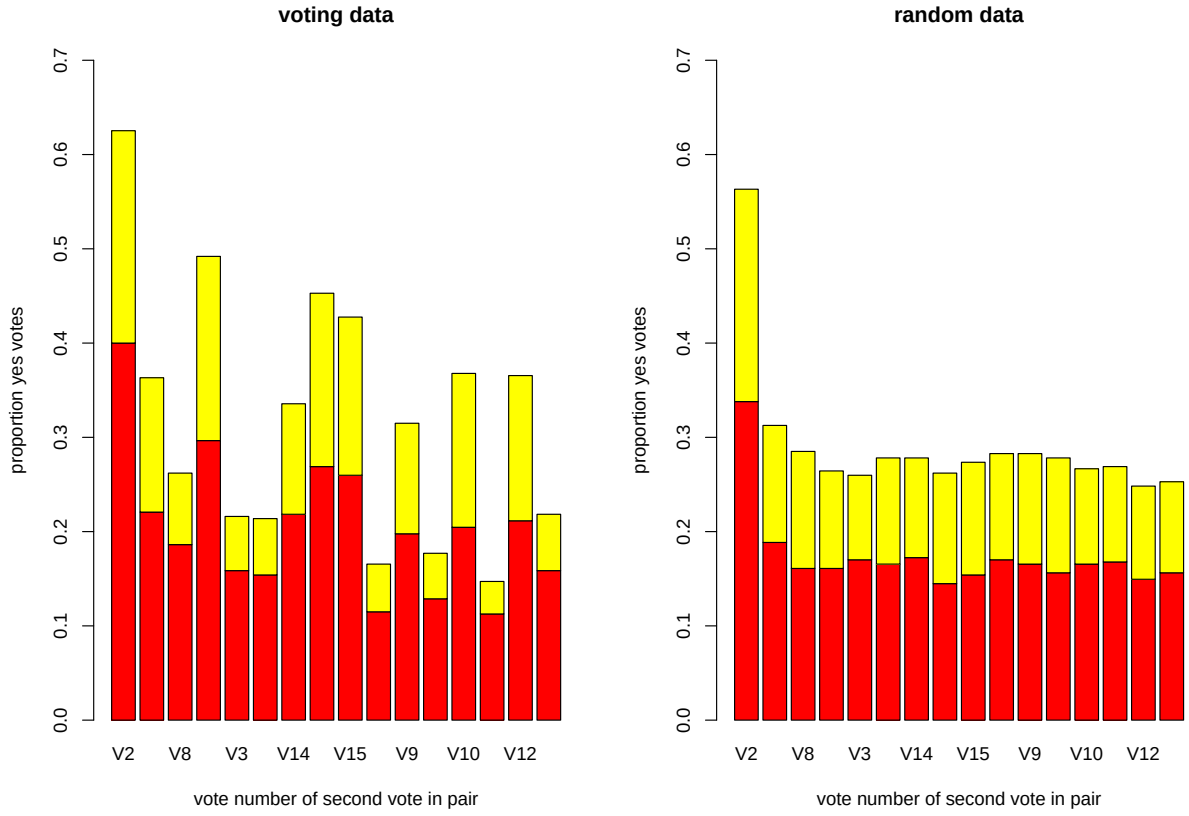

Fig. 4. Supports for pairs of votes in the US congress data (split by party).

distribution is

$$p(x) = \pi_0\, p_{00}^{|x_0|} p_{01}^{|x_1|} (1-p_{00})^{d_0-|x_0|} (1-p_{01})^{d_1-|x_1|} + \pi_1\, p_{10}^{|x_0|} p_{11}^{|x_1|} (1-p_{10})^{d_0-|x_0|} (1-p_{11})^{d_1-|x_1|}.$$

This is a *mixture model* with two components. Recovery of the parameters from the data of mixture models uses the EM algorithm and is discussed in detail in [**?**]. Note that $\pi_0 + \pi_1 = 1$.

For frequent itemset mining, however, the support function is considered and similarly to the random shopper case can be shown to be:

$$s(x) = \pi_0 p_{00}^{|x_0|} p_{01}^{|x_1|} + \pi_1 p_{10}^{|x_0|} p_{11}^{|x_1|}.$$

Assume that the shopper of type i is unlikely to purchase items of the other type. Thus p_{00} and p_{11} are much larger than p_{10} and p_{01}. In this case the

frequent itemsets are going to be small (as before), moreover, one has either $x_0 = 0$ or $x_1 = 0$, thus, frequent itemsets will only contain items of one type. Thus in this case frequent itemset mining acts as a filter to retrieve "pure" itemsets.

A simple application to the voting data could consider two types of politicians. A question to further study would be how closely these two types correspond with the party lines.

One can now consider generalisations of this case by including more than two types, combining with different probabilities (Zipf's law) for the different items in the same class and even use itemtypes and customer types which overlap. These generalisations lead to graphical models and Bayesian nets [**?**,**?**]. The "association rule approach" in these cases distinguishes itself by using support functions, frequent itemsets and in particular, is based on binary data. A statistical approach to this type of data is "discriminant analysis" [**?**].

2.2. *The Size of Itemsets*

The size of the itemsets is a key factor in determining the performance of association rule discovery algorithms. The size of an itemset is the equal to the number of bits set, one has

$$|x| = \sum_{i=1} x_i$$

if the components of x are interpreted as integers 0 or 1 this is a real valued random variable or function defined on $\mathbb{X}$. The expectation of a general real random variable f is

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

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

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

For an arbitrary but fixed itemset $x \in \mathbb{X}$ consider the function

$$a_x(z) = \prod_{i=1}^{d} z_i^{x_i}.$$

Thus function takes values which are either 0 or 1 and $a_x(z) = 1$ iff $x \leq z$ as in this case all the components z_i which occur to power 1 in $a_x(z)$

are one. Note that $a_x(z)$ is a monotone function of z and antimonotone function of x. Moreover, one has for the expectation $E(a_x) = s(x)$ and variance $\text{var}(a_x) = s(x)(1 - s(x))$. The term $1 - s(x)$ is the support of the complement of the itemset x. The values for the expectation and variance are obtained directly from the definition of the expectation and the fact that $a_x(z)^2 = a_x(z)$ (holds for any function with values 0 and 1).

Our main example of a random variable is the length of an itemset,

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

The average length of itemsets is the expectation of f and one can see that

$$E(|x|) = \sum_{|z|=1} s(z).$$

The variance of the length is also expressed in terms of the support as

$$\text{var}(|x|) = \sum_{|x||z|=1} (s(x \vee z) - s(x)s(z))$$

where $x \vee z$ corresponds to the union of the itemsets x and z.

With the expectation and using the Markov inequality one gets a simple bound on the probability of large itemsets as

$$P(\{x \mid |x| \geq m\}) \leq E(|x|)/m.$$

The expected length is easily obtained directly from the data and this bound gives an easy upper bound on probability of large itemsets. Of course one could just as easily get a histogram for the size of itemsets directly from the data. Using the above equation one gets an estimate for the average support of one-itemsets as $E(|x|)/d$.

Consider now the special example of a random shopper discussed previously. In this case one gets $E(|x|) = dp_0$. The distribution of the length is in this case binomial and one has:

$$P(|x| = r) = \binom{d}{r} p_0^r] \, (1 - p_0)^{d-r}.$$

Moreover, for very large d and small p_0 one gets a good approximation using the *Poisson distribution*

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

with $\lambda = E(|x|)$.

The apriori algorithm which will be discussed in the following works best when long itemsets are unlikely. Thus in order to choose a suitable algorithm, it is important to check if this is the case, e.g., by using the histogram for the length $|x|$.

2.3. *The Itemset Lattice*

The search for frequent itemsets benefits from a structured search space as the itemsets form a lattice. This lattice is also intuitive and itemsets close to 0 in the lattice are often the ones which are of most interest and lend themselves to interpretation and further discussion.

As $\mathbb{X}$ consists of sets or bitvectors, one has a natural *partial ordering* which is induced by the subset relation. In terms of the bitvectors one can define this component-wise as

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

Alternatively,

$$x \leq y :\Leftrightarrow (x_i = 1 \Rightarrow y_i = 1, \quad i = 1, \ldots, d)\,.$$

If $x_i = 1$ and $x \leq y$ then it follows that $y_i = 1$. Thus if the corresponding itemset to x contains item i then the itemset corresponding to y has to contain the same item. In other words, the itemset corresponding to x is a subset of the one corresponding to y.

Subsets have at most the same number of elements as their supersets and so if $x \leq y$ then $|x| \leq |y|$.

Bitvectors also allow a total order by interpreting it as an integer $\phi(x)$ by

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

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 as $x \leq y \Rightarrow x \prec y$.

The partial order order has a smallest element which consists of the empty set, corresponding to the bitvector $x = (0, \ldots, 0)$ and a largest element which is just the set of all items Z_d, corresponding to the 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 similarly for $x \wedge y$. Consequently, the partially ordered set $\mathbb{X}$ forms a Boolean lattice. We denote the maximal and minimal elements of $\mathbb{X}$ by e and 0.

In general, partial order is defined by

Definition 1: 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 2: 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 3: A lattice $(\mathbb{X}, \leq)$ is a *Boolean lattice* if

(1) $(\mathbb{X}, \leq)$ is distributive
(2) It has a *maximal element* e and a *minimal element* 0 such that for all $x \in \mathbb{X}$:
$$0 \leq x \leq e.$$
(3) Each element x as a (unique) *complement* x' such that $x \wedge x' = 0$ and $x \vee x' = e$.

The maximal and minimal elements 0 and e satisfy $0 \vee x = x$ and $e \wedge x = x$. 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 4: 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 5: *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 of atoms associated with any element is unique, and the Boolean lattice itself is isomorph to the powerset of the set 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 6: 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 [**?**]. The significance of the theorem 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 conceptually generalises the association rule algorithms.

In figure **??** we show the lattice of patterns for a simple market basket case which is just a power set. The corresponding lattice for the bitvectors is in figure **??**. We represent the lattice using an undirected graph where the nodes are the elements of $\mathbb{X}$ and edges are introduced between any element

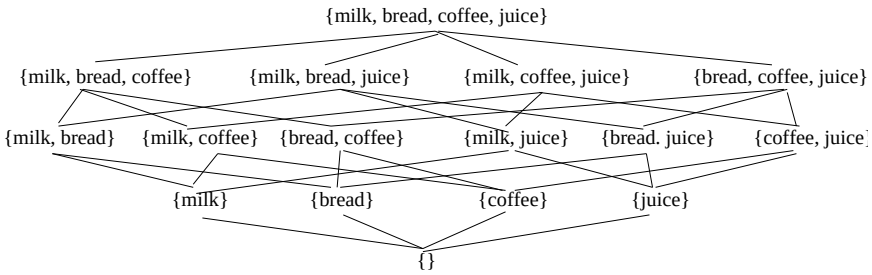

Fig. 5. Lattice of breakfast itemsets

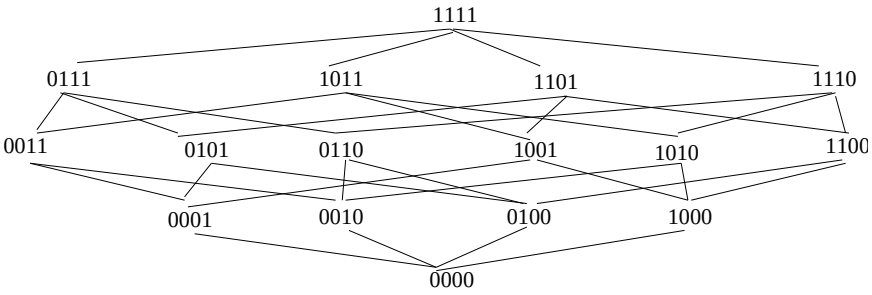

Fig. 6. Lattice of bitvectors.

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 **??** we display the graphs of the first few Boolean lattices. We will graph specific lattices with (Hasse) diagrams [**?**] and later use the

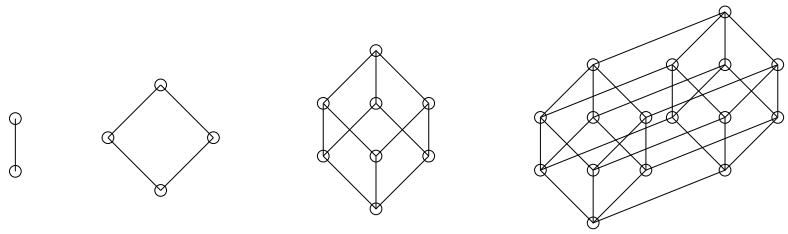

Fig. 7. The first Boolean Lattices

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 $\mathbb{X}$ 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 obtained 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 as one has for the support $s(x)$:

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

From this it follows directly that $s(x)$ is antimonotone, i.e., that

$$x \leq y \Rightarrow s(x) \geq s(y).$$

The aim of frequent itemset mining is to find sets of itemsets, i.e., subsets of $\mathbb{X}$. In particular, one aims to determine

$$L = \{x \mid s(x) \geq \sigma_0\}.$$

From the antimonotonicity of s, it follows that

$$x \in L \;\text{ and }\; x \geq y \Rightarrow y \in L.$$

Such a set is called an *down-set*, *decreasing set* or *order ideal*. This algebraic characterisation will turn out to be crucial for the development and analysis of algorithms.

The set of downsets is a subset of the power set of $\mathbb{X}$, however, there are still a very large number of possible downsets. For example, in the case of $d = 6$, the set $\mathbb{X}$ has 64 elements and the power set has $2^{64} \approx 1.810^{19}$

elements and the number of downsets is 7,828,354. The simplest downsets are generated by one element and are

$$\downarrow x = \{z \mid z \leq x\}$$

For example, one has

$$\mathbb{X} = \downarrow e.$$

Consider the set of itemsets for which the (empirical) support as defined by a data base D is nonzero. This is the set of all itemsets which are at least contained in one data record. It is

$$L^{(0)} = \bigcup_{x \in D} \downarrow x.$$

This formula can be simplified by considering only the maximal elements $D_{\max}$ in D:

$$L^{(0)} = \bigcup_{x \in D_{\max}} \downarrow x.$$

Any general downset can be represented as a union of the "simple downsets" generated by one element. It follows that for some set $Z \subset \mathbb{X}$ of maximal elements one then has

$$L = \bigcup_{x \in Z} \downarrow x.$$

The aim of frequent itemset mining is to determine Z for a given σ_0. Algorithms to determine such Z will be discussed in the next sections. Note that $L \subset L^{(0}$ and that the representation of L can be considerably more complex than the representation of $L^{(0)}$. As illustration, consider the case where e is in the data base. In this case $L^{(0)} = \mathbb{X} = \downarrow e$ but e is usually not going to be a frequent itemset.

2.4. *General Search for Itemsets and Search for Rules*

The previous subsections considered the search for itemsets which were frequently occurring in a database. One might be interested in more general characterisations, maybe searching for itemsets for which the income from their sale amounted to some minimum figure or which combined only certain items together. Thus one has a criterion or predicate $a(x)$ which is true for items of interest. It is assumed that the evaluation of this criterion is expensive and requires reading the database. One would now like to find all "interesting" items and would like to do this without having to consider all

possible itemsets. This search for interesting itemsets is considerably more challenging. In some cases one finds, however, that the sets to be found are again downsets and then similar algorithms can be employed. Often, however, one has to resort to heuristics and approximate methods.

Once frequent itemsets are available one can find *strong rules*. These rules are of the type "if itemset x is in a record than so is itemset y. Such a rule is written as $x \Rightarrow y$ and is defined by a pair of itemsets $(x, y) \in \mathbb{X}^2$. The proportion of records for which this rule holds is called the *confidence*, it is defined formally as

$$c(x \Rightarrow y) = \frac{s(x \vee y)}{s(x)}.$$

A strong rule is given by a rule $x \Rightarrow y$ for which $x \vee y$ is frequent, i.e., $s(x \vee y) \geq \sigma_0$ for a given $\sigma_0 > 0$ and for which $c(x \Rightarrow y) \geq \gamma_0$ for a given $\gamma_0 > 0$. The constants σ_0, γ_0 are provided by the user and their careful choice is crucial to the detection of sensible rules. From the definition it follows that the confidence is the conditional anticumulative distribution, i.e.,

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

where $F^\delta(y|x) = F^\delta(y \vee x)/F^\delta(x)$ is the conditional cumulative distribution function. 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 can be inferred from domain knowledge 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.

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 7: 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 neighbours (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. See [**?**] for efficient algorithms for the direct search for association rules.

3. The Apriori Algorithm

The aim of association rule discovery is the derivation of *if-then-rules* based on the itemsets 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". In this section the Apriori algorithm to find all frequent itemsets is discussed. The classical Apriori algorithm as suggested by Agrawal et al. in [**?**] is one of the most important data mining algorithms. It uses a breadth first search approach, first finding all frequent 1-itemsets, and then discovering 2-itemsets and continues by finding increasingly larger frequent itemsets. The three subsections of this section consider first the problem of the determination of the support of any itemset and the storage of the data in memory, then the actual apriori algorithm and finally the estimation of the size of candidate itemsets which allows the prediction of computational time as the size of the candidates is a determining factor in the complexity of the apriori algorithm.

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 n times expected length, i.e., $nE(|x|)$. 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 often more than 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 also 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 auxiliary bitvector $v \in \mathbb{X}$ (in full format) which is initialised to zero. The proposed algorithm has 3 steps:

(1) Expand x into a bitvector v: $v \leftarrow x$
(2) Extract the value of v for the elements of z, i.e., $v[z]$. If they are all nonzero, i.e., if $v[z] = e$ 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:

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

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 can extract x once and one so gets the algorithm

(1) Extract x into v: $v \leftarrow x$
(2) For all $j = 1, \ldots, m_k$ check if $v[z^{(j)}] = e$, i.e., if $z^{(j)} \leq x$.
(3) Set v to zero again, i.e., $v \leftarrow 0$

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

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

Finally, running this algorithm for all the rows $x^{(i)}$ and vectors $z^{(j)}$ of different lengths, 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 $m_k k n$ 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

$$E(T) \approx \sum_k m_k k n\tau.$$

An alternative is based on 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 storage savings in the vertical format however, are offset by extra storage costs for an auxiliary 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 column x_j into v: $v \leftarrow x_j$
(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. However, the $x_j^{(i)}$ are random with an expectation $E(x_j^{(i)})$ which is typically much less than one and have an average expected length

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 better 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 occurred in both algorithms above and which characterises the complexity of the search through multiple levels of a_z, it is:

$$C = \sum_{k=1}^{\infty} m_k k.$$

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

3.2. *The algorithm*

Some principles of the apriori algorithm are suggested in [**?**]. 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 determines candidate itemsets before each data base scan [**?**,**?**]. 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".)

The apriori algorithm visits the lattice of itemsets in a level-wise fashion, see Figure **??** and Algorithm **??**. 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

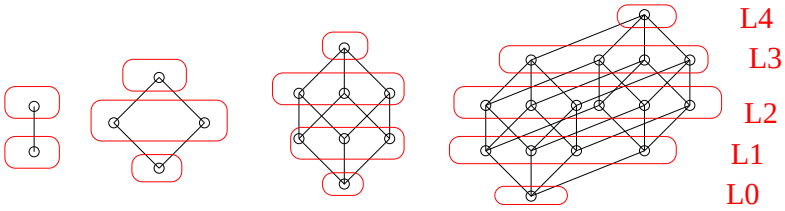

Fig. 8. Level sets of Boolean lattices

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

the major determining parameter for the complexity of the algorithm is $C = \sum_k m_k k$ where $m_k = |C_k|$.

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 $\tau = 1$ns 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 8: 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 always 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 9: *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 $C_{k+1}, \ldots, C_{k+p}$ such 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 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, in addition, $(1,1,0,0,0) \in L_2$ 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 Kruskal-Katona theorem. The proof of this theorem and much of the discussion follows closely the exposition in Chapter 5 of [**?**]. The bounds developed in this way have first been developed in [**?**].

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 $|\partial C_k| \geq k$ independent 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.

First, recall the mapping $\phi : \mathbb{X} \to \mathbb{N}$, defined by:

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

and the induced order

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

which 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 and all other bits can be either 0 or 1. In the colex order any z where bits m (and beyond) are set are larger than any of the elements in $[m]^{(k)}$. Thus $[m]^k$ 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 with the lowest significant bit on the right hand side 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:

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

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:

$$B^{(k)}(m_k, \ldots, m_s) = \left(B^{(k-1)}(m_{k-1}, \ldots, m_s) \vee e_{m_k}\right) \cup [m_k]^{(k)} \tag{3}$$

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 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:

$$\binom{t+1}{r} - 1 = \sum_{l=1}^{r} \binom{t-r+l}{l}. \tag{4}$$

Lemma 10: *For every $m, k \in \mathbb{N}$ there are numbers $m_s < \cdots < m_k$ such that*

$$m = \sum_{j=s}^{k} \binom{m_j}{j} \tag{5}$$

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 term in the sum of equation (**??**), thus $s = k$ and the only choice is $m_k = k$.

Now assume that equation **??** is true for some $m' = m - 1$. We 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 $m' = m - m - \binom{m_k}{k}$.

Assume a decomposition of the form **??** is given. Using equation **??** 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 **??** 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 **??**:

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

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 11: 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 **??**. □

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

Lemma 12:

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

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

The shadow is important for the apriori property and we would thus like to determine the shadow, or at least its size for more arbitrary k-itemsets as they occur in the apriori algorithm. Getting bounds is feasible but one requires special technology to do this. This is going to be developed further in the sequel. 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 13: *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 14: *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 **??**.

(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 15: 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 (**??**).

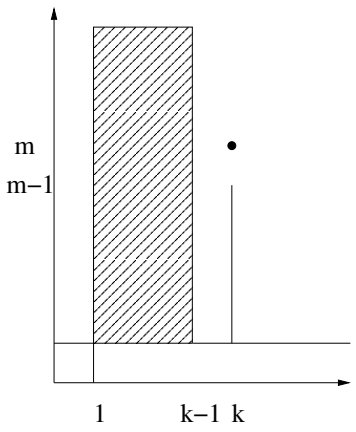

Fig. 9. Double induction

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

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 16: 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 $|C_{k+p}| \leq b^{(k+p)}(m_j, \ldots, m_r)$. □

As a simple consequence one also gets tightness:

Corollary 17: *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 $k > m_p - p$.

4. Extensions

4.1. *Apriori Tid*

One variant of 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 itemsets. In this way each new candidate itemset only requires the intersection of two old ones.

The following demonstrates with an example how this works. The example is adapted from [**?**]. In the first row the itemsets from C_k are depicted. The minimal support is 50 percent or 2 rows. The initial matrix of the tid

algorithm is equal to

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

Note that the column (or item) four is not frequent and is not considered for C_k. After one step of the Apriori tid one gets 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 follows that there remains only one candidate itemset with three items, 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 again use an auxiliary array $v \in \{0,1\}^n$. The apriori tid algorithm uses the join considered earlier in order to construct a matrix for the frequent itemsets L_{k+1} from L_k. (As in the previous algorithms it is assumed that all matrices are 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 a 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 approximation (typically a lower bound) is obtained if we assume that the components of x are independent. 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 always 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.

4.2. *Constrained association rules*

The number of frequent itemsets found by the apriori algorithm will often be too large or too small. While the prime mechanism of controlling the discovered itemsets is the minimal support σ, this may often not be enough. 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 control 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 including the item "ginger beer" are not frequent. However, rules including "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 **??**. The work is almost exactly the same as for the original apriori 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 18: 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.

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

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 scan (this is how we save most). Even better, it turns out that the conditions only need to be checked for level $k = |z^{(j)}|$ where k is the size of the itemset defining the constraint. (This gives a minor saving.) This is summarised in the next theorem:

Theorem 19: 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 **??**.

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

4.3. *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 little disk access. The algorithms have been suggested in [**?**].

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 [**?**] 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 20: 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 **??**. 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 21: *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.

4.4. *Mining Sequences*

The following is an example of how one may construct more complex structures from the market baskets. We consider here a special case of sequences, see [**?**,**?**]. 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 sequential 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 shopper 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 $\mathbf{x}$ is

$$s(\mathbf{x}) = P\left(\{\mathbf{x} | \mathbf{x} \leq y\}\right).$$

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 [**?**].

4.5. *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 neighbour and so on. In addition to the item, the label also contains the number 1, see Step 1 in figure **??**. Then the second record is included such that any common prefix (in the

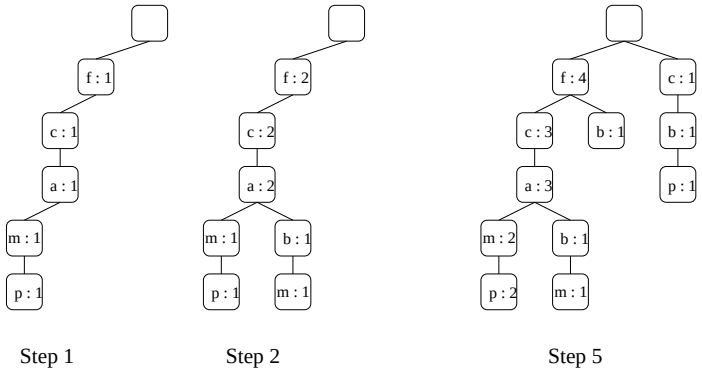

Fig. 10. Construction of the FP-Tree

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

The FP tree does never break a long pattern into smaller patterns the way the Apriori algorithm does. Long patterns can be directly retrieved from the FP tree. The FP tree also contains 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

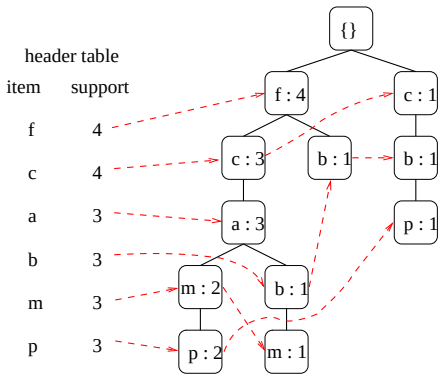

Fig. 11. Final FP-Tree

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 long prefix paths 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.

5. Conclusion

Data mining deals with the processing of large, complex and noisy data. Robust tools are required to recover weak signals. These tools require highly efficient algorithms which scale with data size and complexity. Association rule discovery is one of the most popular and successful tools in data mining. Efficient algorithms are available. The developments in association rule discovery combine concepts and insights from probability and combinatorics. The original algorithm "Apriori" was developed in the early years of data mining and is still widely used. Numerous variants and extensions exist of which a small selection was covered in this tutorial.

The most recent work in association rules uses concepts from graph

theory, formal concept analysis and and statistics and links association rules with graphical models and with hidden Markov models.

In this tutorial some of the mathematical basis of association rules was covered but no attempt has been made to cover the vast literature discussing with numerous algorithms.

Acknowledgements

I would like to thank Zuowei Shen, for his patience and support during the preparation of this manuscript. Much of the work has arisen in discussions and collaboration with John Maindonald, Peter Christen, Ole Nielsen, Steve Roberts and Graham Williams.