Statistical Explanation of the Influence of Noise on Apriori

Nele Dexters

Abstract

This technical report discusses statistically the influence of noise on the output of data mining algorithms for finding frequent itemsets. The results of this research are algorithm independent but for simplicity, we focus on the results for the best known frequent itemset mining algorithm, Apriori. First, we consider the effects of small changes in the support value on the output of the algorithm for three real-life datasets. Second, we introduce noise and rerun Apriori with the same support values. The most important results of the comparison "before noise" and "after noise" will be shown and explained.

1 Introduction

In the last decades, with the development of relational databases, companies had the opportunity to store huge amounts of data efficiently. Not only this huge amount of data itself, but also the possibility to research and query it makes it so precious, valuable and interesting. The necessity of analysis is a motivation for data mining. After all, data mining *is* the analysis of huge amounts of observed data to discover interesting, unexpected relations, patterns or regularities [16].

For the discovery of these previously unknown but interesting patterns in data, lots of algorithms were developed. Usually, these algorithms handle the so called "frequent itemset mining problem" [2, 3]. The problem is, given a large database of basket data, i.e. subsets of a fixed set of items $\mathcal{I}$, and a user-defined support threshold k, to determine which sets of items occur together in at least k baskets. This is a well known and interesting basic problem at the core of many data mining problems [1, 2, 3, 4, 10, 13, 15, 19]; once the frequent items are found, this information can be used to find other interesting regularities between the data, for example Association Rules. In the last two decades, several different algorithms for finding frequent itemsets were proposed [2, 3, 17, 21, 24] and compared.

There is a fundamental link between data mining and statistics [26, 18, 23]. Both research disciplines are searching for interesting relationships in datasets. They are both trying to analyze data to find interesting patterns and use these results for their specific purposes. In data mining, the analysis of the algorithms is usually empirical, based on the execution time. In this technical report, we are trying to look at the statistical side of the game and consider the robustness of the existing algorithms in the case of noise and small changes in parameters.

Noise is a term that is used to indicate that a certain percentage of the data is not correct, not known, or missing. We cannot use it for the analysis, because we just don't know what it is or if it is correct. We are extremely interested in the output of frequent itemset mining algorithms when there is noise in the datasets. Because all algorithms (have to) find the same frequent itemsets, we focuse on the best known data mining algorithm for finding all frequent itemsets: Apriori [3].

It is known that a bit of noise has an enormous influence on the frequency of long patterns [22]. What happens with wrong data, incomplete data, missing information? When we are looking for periodic, returning regularities in the dataset, it could be possible that these different cases disturb the whole process and cause an overfitting of the dataset. It is also interesting to look whether a small change in the parameter support (k) has a large impact on the output of the data mining algorithms. For example, if the support value decreases, how is the amount of frequent itemsets behaving?

Outline
In Section 2, we shortly review the basics for the search for frequent itemsets and some definitions needed for the rest of this technical report. Section 3 describes the experiment that gave rise to this technical report. The idea is to compare the results of Apriori before noise and after noise. In Section 4, some details about the datasets and the algorithms used (the Apriori algorithm and the algorithm used to introduce noise) are discussed. Section 5 groups the results of the comparison before noise and after noise and tries to give an explanation. In Section 6, another way of making noise is introduced, different from the way it was introduced in 4. Section 7 concludes this technical report.

2 Refreshing Some Concepts

In the recent years, data mining (or $\mathcal{K}$nowledge $\mathcal{D}$iscovery in $\mathcal{D}$atabases) has attracted a lot of interest in the database community, caused by the large amounts of computerized data that organisations have about their businesses. The discovery of association rules and frequent itemsets [2, 3] is an interesting subfield of data mining. The motivation for this kind of research has come from the desire to analyze large amounts of supermarket basket data. Association rules describe how often items are purchased together. For instance, "bread $\Rightarrow$ butter (87%)" states that 87% of the customers that bought bread also bought butter. Such rules can be useful for decisions concerning product pricing or store layout.

We shortly review some definitions related to frequent itemset mining and association rules that are necessary for the rest of the paper. Consider the supermarket domain, the so called "market basket analysis". The goal here is to discover associations between items that are often bought together. $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$

represents the set of all items that are sold in the supermarket. Collections of these items are called "itemsets" and are denoted by I_1, I_2, $\ldots$. The baskets of the shoppers are represented by an identifier "TID". Let us assume that there are n shoppers, so n baskets or n transactions ($TID \in \{T_1, T_2, \ldots, T_n\}$). Each transaction combines a TID with all the items from that basket. To store this efficiently in a database $\mathcal{D}$, we split the transaction across several lines, for each item a line; we get a two-column database with in the first column the TID and in the second column the itemID. There will be b tuples in the database.

Here is an example to make it more concrete: John is buying bread, butter and milk, and Lisa is buying chips, beer and bread too. First, we number the baskets: John's basket is 1 and Lisa's is 2, representing the two transactions. Second, we number the products: bread is 1, butter 2, milk 3, chips 4 and beer 5; the number of items sold, m, is 5. Transaction 1 denotes John's basket: $(1; 1, 2, 3)$. Transaction 2 denotes Lisa's basket: $(2; 4, 5, 1)$. In the database $\mathcal{D}$, it is stored as

TID	**ItemID**
1	1
1	2
1	3
2	4
2	5
2	1

As we can see, the number of tuples in the database, b, equals 6 in this example.

The support of set X, $support(X)$, is the fraction of the baskets or transactions that contain all the items of X. An itemset X is called frequent if $support(X) \geq minsup$, a user-defined support threshold. Let, in our example, $minsup$ be equal to 2. We see that $support(\{bread\}) = 2 \geq minsup$, so bread is frequent. We also have $support(\{chips\}) = 1 < minsup$, so chips is not frequent. For describing the support count, we are working absolute, in terms of the amount of transactions. Other approaches in data mining are working relative (in percent) but we do not consider that approach in this technical report.

An association rule $X \Rightarrow Y$ is used to denote that when all the items in set X are in a basket, also the items in Y tend to be in the basket, thus when X occurs, so does Y. The strength of this rule is called the confidence

$$c(X \Rightarrow Y) =_{def} \frac{support(X \cup Y)}{support(X)}.$$

A user-defined confidence threshold is used to exclude rules that are not strong enough to be interesting. In our example the confidence of

$$bread \Rightarrow butter$$

is

$$\frac{support(\{bread, butter\})}{support(bread)} = \frac{1}{2} = 50\%.$$

The problem of finding association rules can be divided into two subproblems:

1. Find all the frequent itemsets $X \subseteq \mathcal{I}$.
2. Find all the association rules: For each frequent set X, test for all non-empty subsets $Y \subset X$ if $X \setminus Y \Rightarrow Y$ holds with sufficient confidence.

The second part of the problem can be solved in a straightforward manner once the first part of the problem, the frequent itemset mining problem, is solved, so once the frequent sets and their corresponding supports are known. This first part is the hardest part and many algorithms were developed [3, 2, 24, 17] to handle this problem. Apriori [3] is one of them.

3 Experiment

In this section, the experiment is described that gave rise to this technical report. The experiment consists of two big parts: the idea is to compare the results of Apriori before noise and after noise, and this for different values of the parameter support k. This research shows that noise in a dataset is causing differences compared to the normal case without noise. For more details, we refer to Section 5.

The first part uses the Apriori algorithm on three real-life datasets (Section 4.1), for finding all frequent itemsets and their corresponding support counts. The similarities and the differences between the results for the three different datasets are discussed. We also try to find a pattern in the results of the Apriori algorithm when the input parameter, the user-defined support threshold k, is slightly changed. We decrease the support from 1% with steps of 0.20% until we reach 0.20%. Then we jump to 0.10% and decrease with steps of 0.02%. Additionally, we take 0.01% to end our support sequence.

The second part starts with adding noise in the three datasets. This noise means that every line in the database has $x\%$ chance of being removed. That is, for every basket B, every item i in it has a chance of $x\%$ to be removed from the basket. This corresponds with removing $x\%$ of the lines in the database at random (Section 4.1). The procedure of generating noise is done for different values of $x\% = 10\%, 20\%, 30\%, 40\%$ and 50%. Now, the experiment of part one is redone: the Apriori algorithm is run again and the results are compared.

The experiment is schematically shown in Table 1.

4 Datasets and Algorithm

4.1 Datasets, Before and After Noise

It is shown in [25] that association rule mining algorithms perform differently on real-world datasets then they do on artificial datasets. Therefore, for our exper-

Part I	
Datasets (3)	**Support**
BMS-WebView-1, BMS-WebView-2, BMS-POS	1%, 0.80%, 0.60%, 0.40%, 0.20%, 0.10%, 0.08%, 0.06%, 0.04%, 0.02%, 0.01%

Part II		
Noise		
Datasets (15)	**Percentage Noise**	**Support**
BMS-WebView-1, BMS-WebView-2, BMS-POS	10%, 20%, 30%, 40%, 50%	1%, 0.80%, 0.60%, 0.40%, 0.20%, 0.10%, 0.08%, 0.06%, 0.04%, 0.02%, 0.01%

Table 1: The two different stages of the experiment

iments, we have chosen to use three real-life datasets instead of synthetic datasets. Thanks to Blue Martini Software, Inc. we can use the sparse real-life datasets BMS-WebView-1, BMS-WebView-2 and BMS-POS.

BMS-WebView-1 and BMS-WebView-2 consist of several months worth of clickstream data from two e-commerce websites from a small dot-com company named Gazelle.com, specialised in legwear and legcare but not existing anymore. In this dataset, the items are the product detail views. Each transaction is a web-session consisting of all the product detail pages viewed in that session. The goal for working with this dataset is to find associations between products viewed by visitors in a single visit to the website.

BMS-POS is a dataset consisting of several years worth of point-of-sale data from a large electronics retailer. Every retailer has many different products, so for this dataset product categories are used as items. Each transaction in the database corresponds to a customer's purchase transaction consisting of all the product categories purchased at one time. The goal of this dataset is to find associations between product categories often purchased together by customers in a single visit to the retailer.

The datasets all have the same layout. Every line consists of two numbers. The first is the transaction identified and the second is the item identified. In this way, one transaction is spread across several lines, for each item in the transaction one line. We will denote the total number of lines in the dataset b, the number of transactions n and the number of items m. In Table 2, these characteristics can be found.

We now explain how we add noise to the three datasets. This noise means that every line in the database has $x\%$ chance of being removed; i.e., for every basket B, every item i in it has a chance of $x\%$ to be removed from B. This corresponds with removing approximately $x\%$ of the lines in the database at random. The procedure of generating noise is done for different values of $x\% = 10\%, 20\%, 30\%, 40\%$ and

Dataset	**Lines (*b*)**	**Transactions (*n*)**	**Items (*m*)**
BMS-WebView-1	149 641	59 602	497
BMS-WebView-2	358 318	77 512	3 340
BMS-POS	3 367 424	515 597	1 657

Table 2: Different characteristics of the three real-life datasets

50% to generate datasets with the same percentages of noise.

For the first two datasets, BMS-WebView-1 and BMS-WebView-2, this is done in S-PLUS, a statistical program for data analysis. A new column is generated in the dataset, with entries 0 (with $x\%$ chance) and 1 (with $(100-x)\%$ chance). This generation of 0 or 1, for each line in the dataset, is done with a random generator of a Bernoulli Distribution with probability $(100-x)/100$. Only the lines with 1 in the new column are kept. They form the new noise dataset with approximately $x\%$ lines less than the original dataset.

To handle the third dataset, BMS-POS, there is constructed a small C++ programm that makes the desired percentage of noise in the dataset used as input. A random generator is used to keep only $(100-x)\%$ of the data.

For each newly generated noise dataset, Table 3 gives the amount of lines, resulting from the noise algorithm; this is *approximately* $x\%$ less than the original dataset. We also give the theoretical amount of lines, i.e. the amount of lines that we would retain if we removed exactly $x\%$ of the lines in the database. We compare the generated amount with the exact amount to show that the way of making $x\%$ noise as explained in this section is accurate. Later on in this technical report (Section 6), another way of generating noise with *exact* removal of $x\%$ lines is explained.

4.2 The Apriori Algorithm

We have chosen for the implementation of Apriori by Bart Goethals [13]. His implementation is in C++ and generates all frequent itemsets for a given minimal (absolute) support threshold. The input format of the datasets is that of the Quest datagenerator ASCII. The support has to be given absolutely; this means that for using this implementation of the Apriori algorithm, it is important to know how many transactions there are in the datasets and to count how much the support is in terms of the amount of transactions. In Table 4, the used support counts are shown for the different datasets. The numbers are rounded to the nearest integer.

Dataset	Generated # Lines	Exact # Lines	Difference	
BMS-WebView-1				
Original		149 641.0		
10%	134 731	134 676.9	54.1	0.0362%
20%	119 808	119 712.8	95.2	0.0636%
30%	104 583	104 748.7	-165.7	0.111%
40%	89 878	89 784.6	93.4	0.0624%
50%	74 792	74 820.5	-28.5	0.0190%
BMS-WebView-2				
Original		358 318.0		
10%	322 445	322 486.2	-41.2	0.0115%
20%	286 730	286 654.4	75.6	0.0211%
30%	250 974	250 822.6	151.4	0.0423%
40%	215 032	214 990.8	41.2	0.0115%
50%	179 827	179 159.0	668.0	0.186%
BMS-POS				
Original		3 367 424.0		
10%	3 030 655	3 030 681.6	-26.6	0.000790%
20%	2 693 696	2 693 939.2	-243.2	0.00722%
30%	2 356 609	2 357 196.8	-587.8	0.0175%
40%	2 020 871	2 020 454.4	416.6	0.0124%
50%	1 682 388	1 683 712.0	-1324.0	0.0393%

Table 3: Different characteristics of the “noise” datasets

5 Results

5.1 Part I: Apriori on Real-Life Data

In Part I, the decrease of the parameter support is studied on the original datasets BMS-WebView-1, BMS-Web-View-2 and BMS-POS. We consider the similarities and differences in the results and try to find a pattern in the output of Apriori.

In Figure 1, the amount of frequent itemsets and the time used to compute these itemsets are plotted for each support level in percent for the three datasets on a logarithmic scale.

Plot (a) (Log Frequent Itemsets - Support in percent, for the three datasets) shows that the trend followed by the three datasets is the same. If the support is decreasing, the amount of frequent itemsets is increasing. This is exactly what we expect: by systematically lowering the support level, the itemsets have to appear in an ever smaller amount of transactions to be frequent. This causes a sudden explosion of itemsets that satisfy the support count and therefore are frequent and this causes an increase in plot (a). The increase in plot (b), when the support decreases, follows naturally because more itemsets also means more time needed

Support in Percent (Relative)	**Support in Transactions (Absolute)**		
	BMS-WebView-1	**BMS-WebView-2**	**BMS-POS**
1.00%	596	775	5 156
0.80%	477	620	4 125
0.60%	358	465	3 094
0.40%	238	310	2 062
0.20%	119	155	1 031
0.10%	60	78	516
0.08%	48	62	412
0.06%	36	47	309
0.04%	24	31	206
0.02%	12	16	103
0.01%	6	8	52

Table 4: The support values of the three different datasets, *relative* and *absolute*

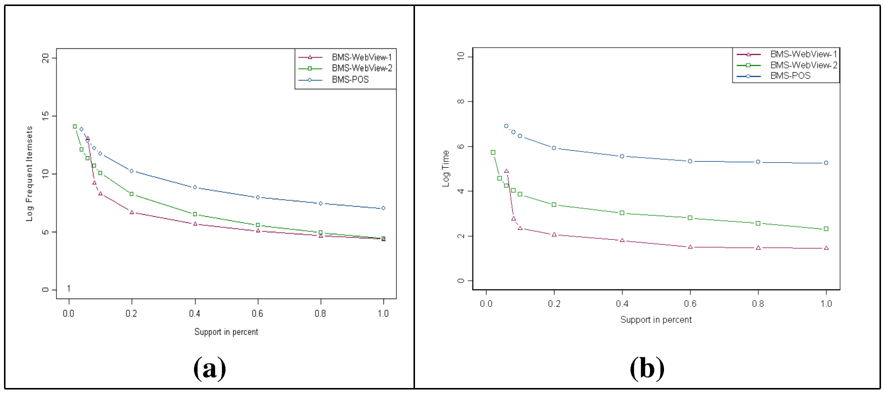


Figure 1: Frequent Itemsets and Time, related to Support in percent, on logarithmic axes

to compute them.

The increase in the amount of frequent itemsets when the support is decreasing is super-exponential. The graphs in the logarithmic scale are no straight lines but are still curved.

5.2 Part II: Apriori on Noisy Data

In Part II, we rerun Apriori, but this time on the noise versions of the datasets, to study the influence of noise on the output of the algorithm. In Section 5.2.1, the three datasets are compared for each noise level. Once the similarities and differences are discussed, we take a look at each of the three datasets separately for all noise levels (Section 5.2.2). Section 5.2.3 tries to find a mathematical formula to explain theoretically the effect of noise.

5.2.1 Comparison of the Datasets For Each Noise Level

For each noise level, the results for the three datasets are plotted in Figures 2 and 3.

The trend for the three datasets is the same for frequent itemsets as well as for the time used to compute these frequent itemsets, for each of the noise levels. The datasets behave like in the case with no noise (Section 5.1): when the support is decreasing, the amount of frequent itemsets and the time used to find them is increasing.

With increasing noise level, the amounts of frequent itemsets and time are decreasing, for a fixed percentage of support. To have a better idea of the effect of noise on the output of Apriori, the three datasets are considered individually in detail in the next section.

5.2.2 Comparison of the Noise Levels For Each Dataset

Figure 4 shows that the shape of the graph (Frequent Itemsets - Support in percent and Time - Support in percent) for each noise level for each of the three datasets is the same.

When focusing on one specific support value in the (a) plots, it is clear that an increase in the noise level causes a decrease in the amount of frequent itemsets. This is not strange, because increasing noise in fact means that the dataset becomes smaller, so the amount of frequent itemsets becomes smaller too.

The same behaviour can be seen in the time graphs (the (b) plots). On a fixed support level, an increasing amount of noise is causing a decreasing amount of time needed to compute these frequent itemsets. This happens for all three datasets.

The graphs also show that the most interesting region to consider in detail is the region where the support is very small. Noise has a strong influence for small support values. The (a) plots show that only a small increase in noise causes a large drop in frequent itemsets. For higher values of support, the different noise versions of each dataset are behaving practically the same. There is a small difference in the time needed to compute the frequent itemsets ((b) plots) but this is neglectable.

5.2.3 Interpretation

In this section, we try to capture the effect of noise in a mathematical formula. We therefore first focus on the specific case of BMS-WebView-1 10% noise and BMS-WebView-1 20% noise and show that when a concrete support level is fixed, an increasing amount of noise is causing a decreasing amount of frequent itemsets. Based on this example, an expression is derived that explains the relationship between the amount of frequent itemsets before and after the noise was added to the dataset.

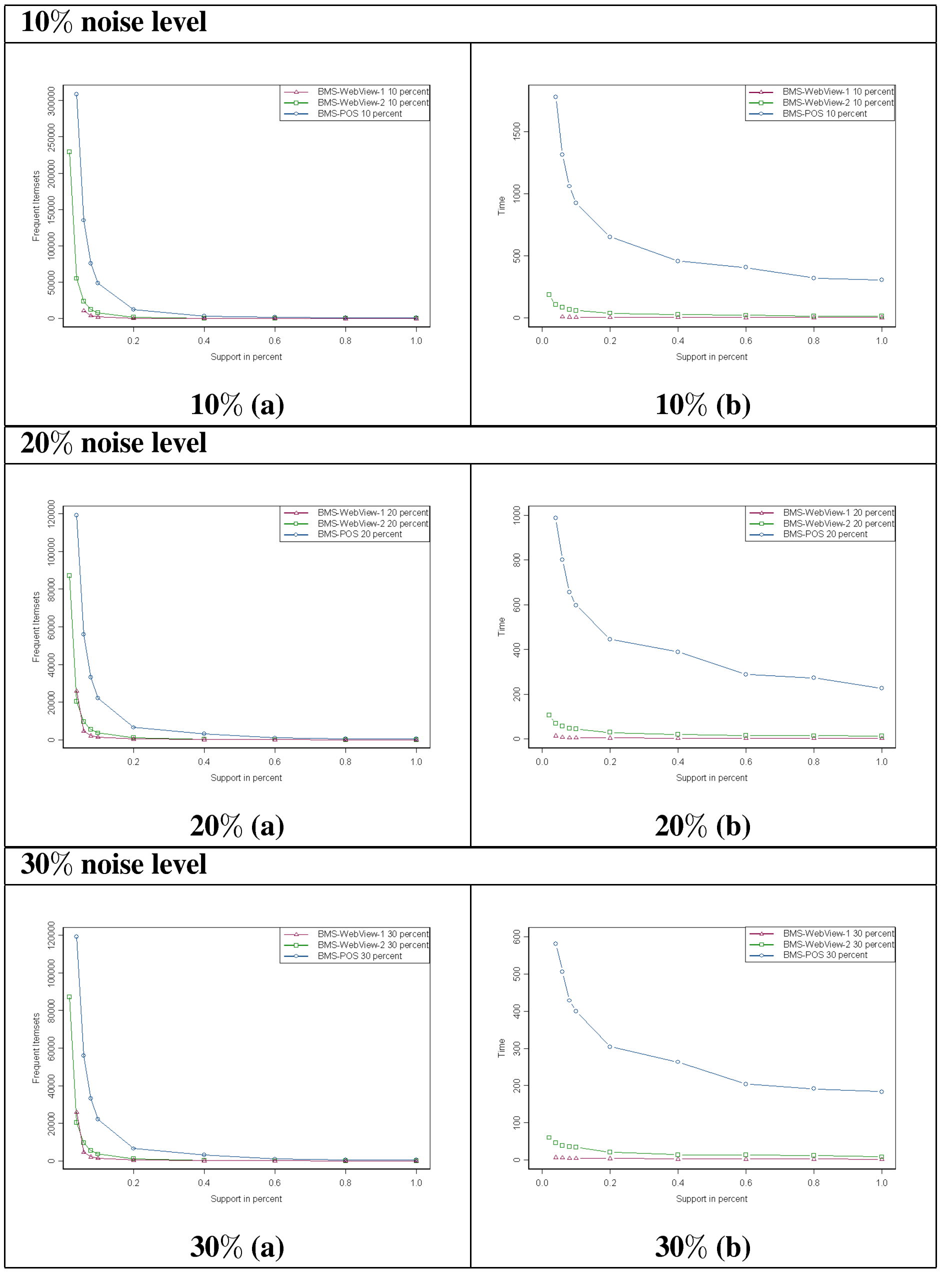
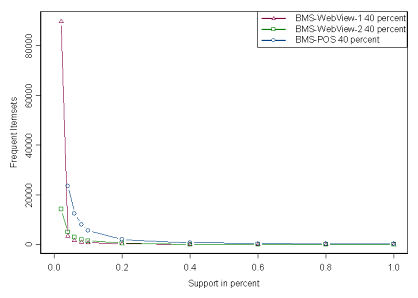


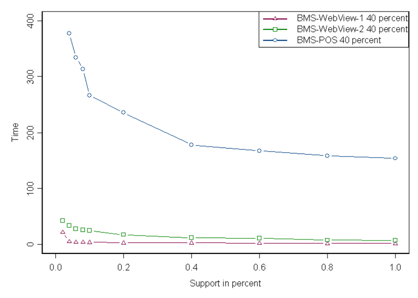
Figure 2: Frequent Itemsets and Time, related to Support in percent, for the three datasets and the different noise levels 10%, 20% and 30%

Let us fix a concrete support level. To illustrate the effect that an increasing amount of noise is causing a decrease in the amount of frequent itemsets, the exact results for BMS-WebView-1 10% noise and BMS-WebView-1 20% noise are given in Table 5. The amount of noise is increasing with 10%, so BMS-WebView-1 20%

40% noise level

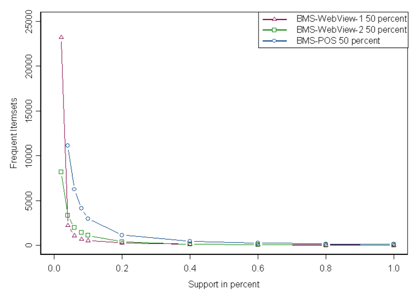


40% (a)

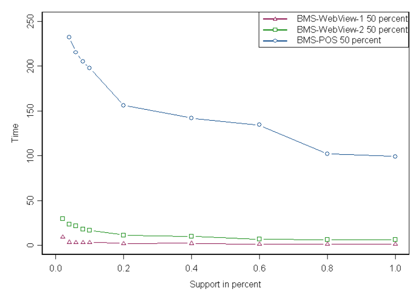


40% (b)

50% noise level



50% (a)



50% (b)

Figure 3: Frequent Itemsets and Time, related to Support in percent, for the three datasets and the different noise levels 40% and 50%

has approximately 10% lines less than BMS-WebView-1 10% (Table 3). For the support, it is not the amount of lines in the database that is important, but the amount of transactions; a certain itemset I has support s if I occurs in at least s transactions in the database. In Table 5 the amount of transactions for the two noise datasets are given, together with their absolute support count and the corresponding amount of frequent itemsets, obtained from the Apriori algorithm. For each support level, the amount of frequent itemsets for BMS-WebView-1 20% is (indeed) less than that for BMS-WebView-1 10%. The question is: How much less? What is the connection between the last two columns of Table 5? It looks like the decrease in support is exactly 10%, but this is not true. In the beginning, it is almost correct $(69 - 0.10 * 69 \approx 61)$, but at the end you can see that it has to be more than 10% $(0.90 * 3983 \approx 3585 \geq 2262)$.

The aim of this section is to find a relationship between the amount of frequent itemsets in a dataset after the $x\%$ noise algorithm in terms of the amount of frequent itemsets in the dataset before the $x\%$ noise algorithm. For now, we focus on the

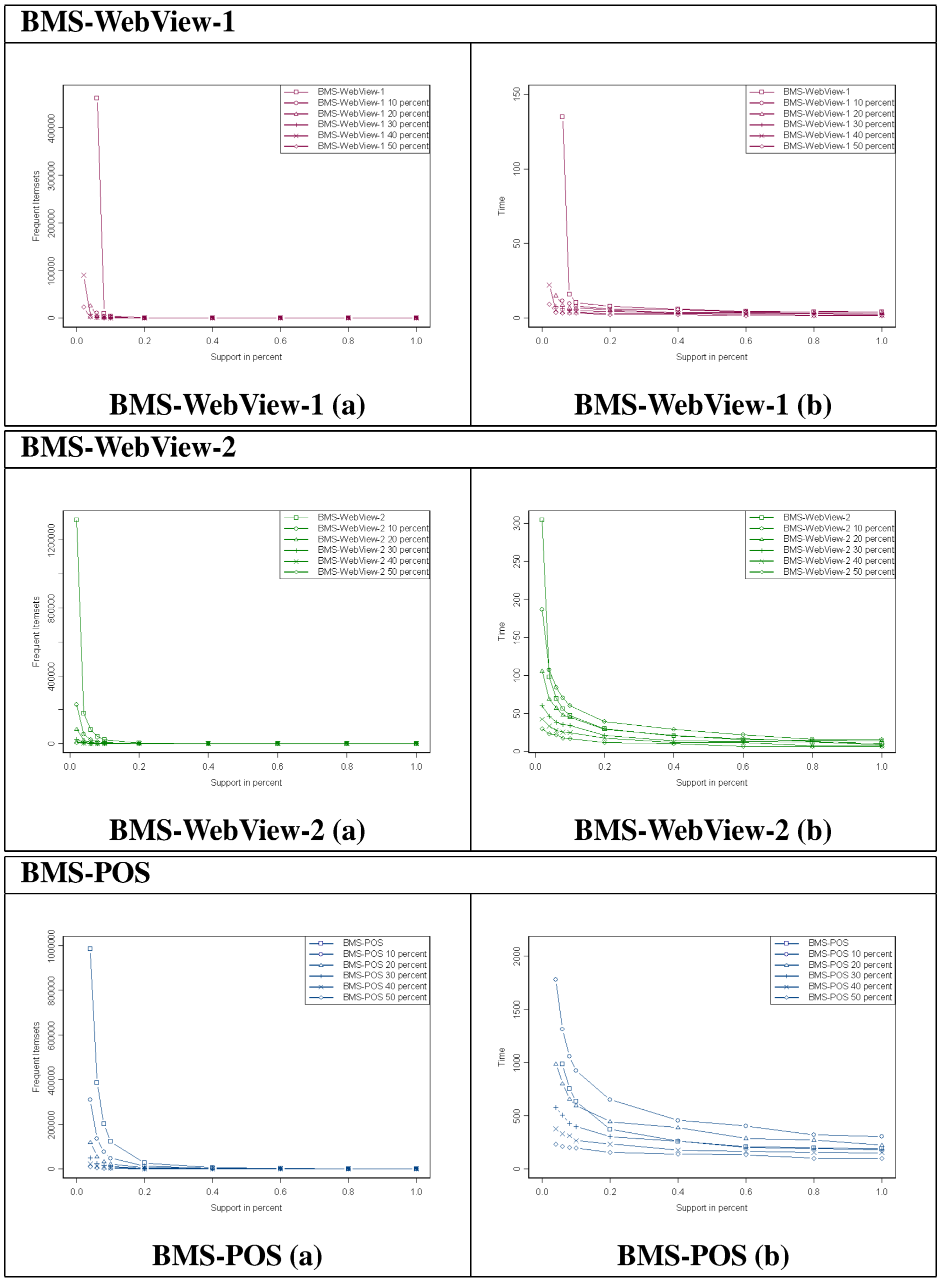


Figure 4: Frequent Itemsets and Time, related to Support in percent, for each support level, for each of the three datasets

10% noise case. The other percentages can be treated analoguously.

Consider an itemset I_1 consisting of one item i. What is the probability that a transaction that contained that itemset I does not contain it any more after the 10%

Datasets	**# transactions**
BMS-WebView-1 10% noise	56 160
BMS-WebView-1 20% noise	52 499

Relative Support	**Absolute Support**		**Frequent Itemsets**	
	10% **noise**	20% **noise**	10% **noise**	20% **noise**
1.00	562	525	69	61
0.80	449	420	95	86
0.60	337	315	151	140
0.40	225	210	234	208
0.20	112	105	610	490
0.10	56	52	2 327	1 504
0.08	45	42	3 983	2 262
0.06	34	31	10 920	4 737

Table 5: Frequent Itemsets for BMS-WebView-1, 10% and 20% noise

noise algorithm? So, what is the probability that a transaction that contained that specific item i, does not contain it anymore after the 10% noise algorithm? This is 10% because this is the probability that one line in the dataset is thrown away when carrying out the 10% noise algorithm:

$$P(\neg i) = 10\% \,.$$

Analoguously, the probability that a transaction that contained that specific item i still contains it after the 10% noise algorithm is

$$P(i) = 1 - 10\% = (100 - 10)\% = 90\% \,.$$

Let us now consider an itemset I_2 of length 2. What is the probability that a transaction that had this itemset I_2 before (so contained the two items together), won't have it anymore after the 10% noise algorithm? The event that two items, i_1 and i_2, won't be in the transaction anymore can be reached in three ways. The first way is that i_1 is still in the transaction but i_2 not. The second way is that i_1 is not in the transaction anymore, but i_2 still is and the third way is that neither i_1 nor i_2 are in the transaction. Because we know that

$$P(i_1 \text{ and } i_2) + P(i_1 \text{ and } \neg i_2) + P(\neg i_1 \text{ and } i_2) + P(\neg i_1 \text{ and } \neg i_2) = 1,$$

the probability we are looking for is 1 minus the probability that both items will stay in the transaction, so

$$P(i_1 \text{ and } \neg i_2) + P(\neg i_1 \text{ and } i_2) + P(\neg i_1 \text{ and } \neg i_2) = 1 - P(i_1 \text{ and } i_2).$$

In the $x\%$ noise algorithms that are used for this section, the removal of lines is based on independent Bernoulli distributions. This means that the removal of one line is independent of the removal of another line and is completely determined by the result of the Bernoulli trial with probability $x\%$ on the mother dataset BMS-WebView-1. The removal of one item is thus independent of the removal of another item. Because of the independent usage of the Bernoulli Distribution for each line separately in the database, this procedure for generating noise yields a dataset with *approximately* $x\%$ lines less than the original dataset. In Section 6, another approach for making noise is considered, this time with an *exact* removal of $x\%$ lines in the dataset.

The independence of the presence of the two items makes that this probability equals

$$P(i_1 \text{ and } \neg i_2) + P(\neg i_1 \text{ and } i_2) + P(\neg i_1 \text{ and } \neg i_2) = 1 - P(i_1)P(i_2) = \\ 1 - (1 - 10\%)^2.$$

If we generalize this to an itemset I_l of length l, we obtain the following expression $P(l, 10\%)$ for the probability that a transaction that contained an itemset of length l does not contain it anymore after the run of the 10% noise algorithm:

$$P(l, 10\%) = 1 - (1 - 10\%)^l.$$

We can find a formula for the frequency of I_l after the 10% noise algorithm in terms of the frequency in the dataset with approximately 10% more line. The "original" dataset is denoted by D, the dataset with 10% more noise (so with 10% less lines) is denoted by D'. Both datasets are generated based on the mother dataset BMS-WebView-1. The amount of lines in D is approximately $1 - x\% = (100-x)\%$ of the amount of lines in BMS-WebView-1. D' has approximately 10% lines less than D and is generated by taking $1 - (x + 10)\% = (100 - (x + 10))\%$ of lines from BMS-WebView-1. We thus have

$$\begin{array}{rcl} |D| & \cong & (100 - x)\% \ |\mathrm{BMS-WebView-1}| \\ |D'| & \cong & (100 - (x + 10))\% \ |\mathrm{BMS-WebView-1}| \end{array}$$

what leads to

$$\begin{array}{rcl} \dfrac{|D|}{(100 - x)\%} & \cong & \dfrac{|D'|}{(100 - (x + 10))\%} \\ |D'| & \cong & \dfrac{(100 - (x + 10))}{(100 - x)} \ |D|. \end{array}$$

If this information is combined in an expression relating the frequency of I_l in D' with the frequency of I_l in D, it yields

$$freq(I_l, D') \cong \left[\tfrac{(100-(x+10))}{100-x}\right]^l * freq(I_l, D).$$

l	**Itemset** I_l
1	$\{10295\}$
2	$\{47953, 47961\}$
3	$\{10295, 32201, 32205\}$
4	$\{10311, 12487, 12703, 32213\}$
5	$\{10311, 12487, 12703, 32213, 34893\}$

Table 6: The itemsets

l	$freq(I_l,$ BMS-WebView-1 $10\%)$	$freq(I_l,$ BMS-WebView-1 $20\%)$
1	1 811	1 606
2	87	75
3	65	50
4	128	84
5	58	38

Table 7: Results of the experiment

$freq(I_l,$ BMS-WV-1 $20\%)$	$(8/9)^l freq(I_l,$ BMS-WV-1 $10\%)$	**Relative Error**
1 606	1 428	0.1108
75	67	0.1067
50	44	0.1200
84	75	0.1071
38	34	0.1053

Table 8: Results of the formula

This formula is checked for our example considering BMS-WebView-1 10% and BMS-WebView-1 20%. The results can be found in Tables 7 and 8. We consider itemsets of length 1, 2, 3, 4, and 5 (Table 6) and the relative support level used is 0.06%. This corresponds to an absolute support of 34 for BMS-WebView-1 10% and an absolute support of 31 for BMS-WebView-1 20%. Numbers are rounded to the nearest integer.

The relative error is computed by subtracting the approximated value from the exact value and by deviding the result by the exact value. Table 8 shows that the new formula is approximately correct. The small changes in the results can be explained by knowing that BMS-WebView-1 20% does not have *exactly* 10% lines less than BMS-WebView-1 10%.

6 Another Way of Sampling

Until now, the noise algorithms used for the experiments were generating approximately $x\%$ of noise. In practice, approximately $x\%$ of lines in the original database were removed, based on independent Bernoulli trials. Therefore, the removal of one line is independent of the removal of another line. Because transactions are spread over several lines, this means that the removal of transaction T_i is independent of the removal of transaction T_j $(i \neq j)$.

In this section, another, more exact way of sampling and a more general way of computing the desired probabilities is considered. With this new procedure, exactly $x\%$ of the lines in the database is removed. Now, the removal of transaction T_i causes a decrease in the probability of the removal of transaction T_j $(i \neq j)$. On the other hand, when we do not remove transaction T_i, the probability of removing T_j increases.

This section also provides a comparison between the approximate removal and the exact removal, to show that both methods lead to the same results. In Section 3, the experiments were done, based on noise datasets generated with the approximate removal, because the exact removal is much more difficult to compute. Even more, the importance of the exact removal diminishes when the dataset becomes larger. In the case of a huge database, the difference between replacing lines and not replacing lines is negligible and the approximate results satisfy.

6.1 Example

This section starts with the illustration of the statement that the probability that a transaction that contained an itemset I_l of length l still contains that itemset I_l after the $x\%$ noise algorithm is dependent of the size of the dataset

Consider a dataset consisting of 3 lines and a dataset consisting of 6 lines. The percentage of lines to throw away is $1/3$. In the first case, it means that one line will be thrown away (and evidently, two lines will be kept) and in the second case, two lines will be thrown away (and four lines will be kept). We now consider an itemset I_2, so two lines in each dataset are particularly interesting for us.

Imagine in the first case that the two last lines are the interesting ones corresponding to the items from itemset I_2. In Figure 5, dots are used to show the different possibilities of single lines that can be removed (and also the pairs of lines that are left after the removal of one line). Each line that can be removed is expressed by a dot and the lines without dots are the lines that remain in the dataset in that case. There are three different possibilities of removing one line in the first dataset. To compute the probability that the two interesting lines are still in the new dataset consisting of two lines instead of three, the common rule

$$P = \frac{\sharp \text{ favorable pairs}}{\sharp \text{ possible selection of pairs}} = \frac{1}{3}$$

is used.

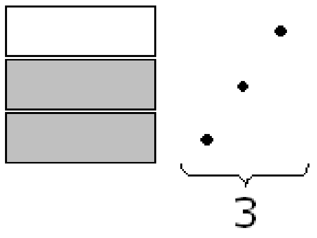


Figure 5: The possibities of removing 1 line in a dataset of 3 lines

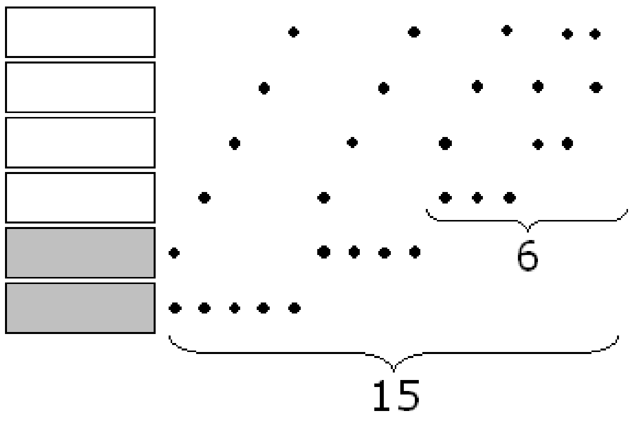


Figure 6: The possibities of removing 2 lines in a dataset of 6 lines

The same is now done for the second case; again, the last two lines are the interesting ones. For this example, two lines have to be thrown away and four lines will be kept. In Figure 6, the dots correspond to the possible combinations of lines that can be thrown away. The probability that the two interesting lines are still in the new dataset consisting of four lines is

$$P = \frac{\sharp \text{ favorable sets of four}}{\sharp \text{ possible sets of four}} = \frac{6}{15} = \frac{2}{5}.$$

Because $1/3 \neq 2/5$, it is shown with this example that the probability that two interesting items that were contained in a transaction are still contained in that transaction after the noise algorithm, is dependent of the size of the database.

6.2 Another Noise Algorithm

The way of thinking illustrated in Section 6.1 is only correct when we *exactly* remove $x\%$ (in our example 1/3) of the lines in the dataset. With the approach we used so far, we *approximately* removed $x\%$, which was pretty good (Table 3, Table 8). This section shows a theoretical way to remove exactly $x\%$ of the lines in a dataset. This method is not used for experiments because the first approach is good enough to confirm our expectations.

First, it is necessary to know how many lines there are in the dataset; this number is denoted by b. Then, $x\%$ of this amount b is computed to find the exact amount of lines that has to be thrown away: $xb/100$; this is denoted by X. Now, a random number between 0 and 1 is generated, multiplied by b and rounded to the nearest integer; this yields the number of the line that will be thrown away. This line is thrown away, the resulting lines are renumbered and the procedure is repeated. A new random number between 0 and 1 is generated but this time multiplied by $b-1$ because already one line is removed from the database. This causes dependency. The result is rounded and the corresponding line is removed. After each removal, the resulting lines are renumbered. This procedure is repeated until exactly X lines are removed from the dataset. In pseudocode:

```
for i = 1..$\lfloor \frac{xb}{100} \rfloor$
        remove line$_{\lfloor rand(b-i+1) \rfloor}$
```

in which $rand(s)$ means removing a line at random from a dataset with s lines, as described above.

6.3 Another Way of Computing Probabilities

Now we have a method to remove exactly $x\%$ of lines from our dataset, the way of thinking of our example (Section 6.1) can be abstracted to the general case. A mathematically correct way can be given to compute the probability that l items that where contained in a transaction are still contained in that transaction after the $x\%$ noise algorithm described above. Therefore, the Hypergeometric Distribution is used.

We start from a bag with N marbles; a fraction p of the marbles is white (so Np marbles) and a fraction q $(= 1-p)$ is black (so Nq marbles). Without replacement, r marbles are picked, so in fact r marbles are picked at the same time. If the jth marble is white, $X_j = 1$ and if the jth marble is black, $X_j = 0$ $(1 \leq j \leq r)$. We are interested in Y, the amount of times that we picked a white marble. Mathematically:

$$Y = X_1 + X_2 + \ldots + X_r$$

with the X_j NOT independent $(1 \leq j \leq r)$. After all, if we pick a white marble in the first stap (with probability $P = Np/N = p$), the probability to pick again a white one in the second stap is reducing to $(Np-1)/N-1$ because already one white marble is gone.

The probability of having i white marbles in a sample consisting of r marbles out of an original bag of N marbles, partitioned in Np white ones and Nq other ones can be computed by using the Hypergeometric distribution. This distribution tells us that

	N = 3	**N = 6**
♯ interesting lines **Np**	2	4
♯ not interesting lines **Nq**	1	2

Table 9: The two different examples

$$P(Y = i) = \frac{\sharp \text{ favorable}}{\sharp \text{ possible}} = \frac{\begin{pmatrix} Np \\ i \end{pmatrix}\begin{pmatrix} Nq \\ r-i \end{pmatrix}}{\begin{pmatrix} N \\ r \end{pmatrix}}.$$

6.4 Illustration

This section shows that the two examples in Section 6.1 are illustrations of this exact approach. We consider an interesting itemset of length $l = 2$, so two lines in our datasets. The idea is to remove $1/3$ of the datasets; in the first case one line (two lines left) and in the second case two lines (four lines left). In Table 9, the different parameters in each example are shown.

Example 1
Pick a sample of $r = 2$ lines and compute the probability that the two interesting lines are the two lines that are kept in the new dataset.

$$P(Y = 2) = \frac{\begin{pmatrix} 2 \\ 2 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}}{\begin{pmatrix} 3 \\ 2 \end{pmatrix}} = \frac{1}{3}$$

Example 2
Pick a sample of $r = 4$ lines and compute the probability that the two interesting lines are still in the four lines that are kept.

$$P(Y = 2) = \frac{\begin{pmatrix} 4 \\ 2 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix}}{\begin{pmatrix} 6 \\ 4 \end{pmatrix}} = \frac{6}{15} = \frac{2}{5}$$

6.5 Generalisation

In this section, the general case is considered. For each newly introduced parameter, the corresponding notation from Section 6.3 is given between brackets.

The startpoint is a database consisting of b (N) lines. Consider an itemset of length l ($Np = l$, $Nq = b - l$). Pick a sample of r lines; throw away $x\%$ of lines from the database to keep $(100 - x)\%$ lines ($r = \lfloor(100 - x)\%b\rfloor$).

The probability that a transaction that contained these l items still contains them after the $x\%$ noise algorithm (in the dataset consisting of $\lfloor(100 - x)\%b\rfloor$ lines) is

$$\begin{aligned} P(Y = l) &= \frac{\begin{pmatrix} l \\ l \end{pmatrix}\begin{pmatrix} b-l \\ r-l \end{pmatrix}}{\begin{pmatrix} b \\ r \end{pmatrix}} \\ &= \frac{r!\ (b-l)!}{b!\ (r-l)!} \end{aligned}$$

leading to

$$P(i_1 \text{ and } i_2 \text{ and } \ldots \text{ and } i_l) = \frac{(r-l+1)(r-l+2)\cdots(r-1)r}{(b-l+1)(b-l+2)\cdots(b-1)b}.$$

This approach (using the dependency) gives us a more general formula instead of

$$P(i_1 \text{ and } i_2 \text{ and } \ldots \text{ and } i_l) = P(i_1)P(i_2)\ldots P(i_l) = ((100 - x)\%)^l,$$

the formula we found in Section 5.2.3 when we used independency.

The new formula converges to the old formula, when x is fixed. If $b \to \infty$, so the difference between replacing and not replacing is negligible, the above result becomes

$$\frac{r^l}{b^l} = \left(\frac{r}{b}\right)^l \cong \lfloor(100 - x)\%\rfloor^l.$$

The probability that a transaction that contained these l items does not contain them any more after the $x\%$ noise algorithm is

$$\begin{aligned} P(\neg i_1 \text{ or } \ldots \text{ or } \neg i_l) &= 1 - P(i_1 \text{ and } i_2 \text{ and } \ldots \text{ and } i_l) \\ &= 1 - \lfloor(100 - x)\%\rfloor^l \end{aligned}$$

In the 10% noise case, this gives us $1 - (90\%)^l$ as also found in the approximate approach!

7 Conclusion

In this technical report, the influence of noise on the output of the frequent itemset mining algorithm Apriori is studied statistically. First, the effects of changes in the support value on the output of Apriori on three real-life datasets are considered. Second, noise is introduced in these datasets and Apriori is rerun with the same sequence of support values. A comparison is made.

It is now showed that noise in a dataset is causing differences compared to the case with no noise. A formula is computed to predict the probability of the frequency of an itemset after the noise in terms of the probability of that itemset before the noise. An alternative way of sampling is explained.

Acknowledgements

We would like to thank Toon Calders for his active help with this research.

References

[1] K. Ali, S. Manganaris, R. Srikant. Partial Classification using association rules. In *Proc. KDD Int. Conf. Knowledge Discovery in Databases*, pages 115–118, 1997.

[2] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., 1993.

[3] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of the 1994 Very Large Data Bases Conference*, pages 487–499, 1994.

[4] R. Agrawal, R. Srikant. Mining Sequential Patterns. In *Proc. IEEE ICDE Int. Conf. on Data Engineering*, pages 3–14, 1995.

[5] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal. Mining Frequent Patterns with Counting Inference. In *SIGKDD Explorations,* **2***(2)*, pages 66-75, 2000.

[6] R. J. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proc. of the 1998 ACM SIDMOD Int. Conf. on Management of Data*, pages 85–93, 1998.

[7] D. Burdick, M. Calimlim, J. Gehrke. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *Proc. of the 17th Int. Conf. on Data Engineering*, pages 443–452, 2002.

[8] T. Calders. Computational Complexity of Itemset Frequency Satisfiability. In *Proc. of the 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 143–154, 2004.

[9] T. Calders, B. Goethals. Mining all Non-Derivable Frequent Itemsets. In *Proc. of the 6th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'02), Lecture Notes in Artificial Intelligence.*, pages 74–85, 2002.

[10] H. Chernoff. A Measure of Asymptotic Efficiency for Test of a Hypothesis Based on the Sum of Observations. In *Annals of Mathematical Statistics,* **23**, pages 493–507, 1942.

[11] W.G. Cochran Sampling Techniques. 1977, John Wiley and Sons.

[12] F. Geerts, B. Goethals, J. Van den Bussche. A Tight Upper Bound on the Number of Candidate Patterns. In *Proc. of the first IEEE Int. Conf. on Data Mining*, San Jose, CA, USA2001.

[13] B. Goethals. Survey of Frequent Pattern Mining. *[http://www.adrem.ua.ac.be/∼goethals/ publications.html]*, 2003

[14] B. Goethals. Efficient Frequent Pattern Mining. *PhD thesis*, transnational University of Limburg, Diepenbeek, Belgium, December 16 2002.

[15] B. Goethals, M. J. Zaki (eds.). Proc. of the Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, 2003.

[16] J. Han, M. kamber. Data Mining: Concepts and Techniques. 2001, Morgan Kaufman.

[17] J. Han, J. Pei, Y. Yin. Mining Frequent Patterns without Candidate Generation In *Proc. of the 2000 ACM SIGMOD Int. Conf. Management of Data*, Dallas, TX, pages 1–12, 2000.

[18] D. Hand. Statistics and Data Mining: Intersecting Disciplines. In *SIGKDD Explorations*, 1999.

[19] J. Hipp, U. Güntzer, G. Nakhaeizadeh. Algorithms for Association Rule Mining — A General Survey and Discovery of Association Rules — A Position Paper. In *ACM SIGKDD Explorations,* **2**, pages 65–69.

[20] W. A. Kosters, W. Pijls. Apriori, A Depth First Implementation. In *Proc. of the Workshop on Frequent Itemset Mining Implementations*, Melbourne, Florida, 2003.

[21] H. Mannila, H. Toivonen. Levelwise Search and Borders of Theories in Knowledge Discovery. In *Data Mining and Knowledge Discovery,* **1**, pages 241–258, 1997.

[22] P. W. Purdom, D. Van Gucht, D. P. Groth. Average Case Performance of the Apriori Algorithm. In *SIAM J. Computing,* **33***(5)*, pages 1223–1260, 2004.

[23] P. Smyth. Data Mining at the Interface of Computer Science and Statistics. Invited Chapter in *Data Mining for Scientific and Engeneering Applications*, 2001.

[24] M. J. Zaki. Scalable Algorithms for Association Mining. In *IEEE Transactions on Knowledge and Data Engineering,* **12***(3)*, pages 372–390, 2000.

[25] Z. Zheng, R. Kohavi, L. Mason. Real World Performance of Association Rule Algorithms. 2001.

[26] Z.-H. Zhou. Three Perspectives of Data Mining. 2003.