PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 87(101) (2010), 109–119 DOI: 10.2298/PIM1001109S

AN EFFICIENT PROCEDURE FOR MINING STATISTICALLY SIGNIFICANT FREQUENT ITEMSETS

Predrag Stanišić and Savo Tomović

Communicated by Žarko Mijajlović

Abstract. We suggest the original procedure for frequent itemsets generation, which is more efficient than the appropriate procedure of the well known Apriori algorithm. The correctness of the procedure is based on a special structure called Rymon tree. For its implementation, we suggest a modified sort-merge-join algorithm. Finally, we explain how the support measure, which is used in Apriori algorithm, gives statistically significant frequent itemsets.

1. Introduction

Association rules have received lots of attention in data mining due to their many applications in marketing, advertising, inventory control, and numerous other areas. The motivation for discovering association rules has come from the requirement to analyze large amounts of supermarket basket data. A record in such data typically consists of the transaction unique identifier and the items bought in that transaction. Items can be different products which one can buy in supermarkets or on-line shops, or car equipment, or telecommunication companies' services etc.

A typical supermarket may well have several thousand items on its shelves. Clearly, the number of subsets of the set of items is immense. Even though a purchase by a customer involves a small subset of this set of items, the number of such subsets is very large. For example, even if we assume that no customer has more than five items in her/his shopping cart, there are $\sum_{i=1}^{5}\binom{10000}{i} = 832916875004174500 \approx 10^{18}$ possible contents of this cart, which corresponds to the subsets having no more than five items of a set that has 10,000 items, and this is indeed a large number.

The supermarket is interested in identifying associations between item sets; for example, it may be interesting to know how many of the customers who bought

2010 *Mathematics Subject Classification*: Primary 03B70; Secondary 68T27, 68Q17.
Key words and phrases: data mining, knowledge discovery in databases, association analysis, Apriori algorithm.

bread and cheese also bought butter. This knowledge is important because if it turns out that many of the customers who bought bread and cheese also bought butter, the supermarket will place butter physically close to bread and cheese in order to stimulate the sales of butter. Of course, such a piece of knowledge is especially interesting when there is a substantial number of customers who buy all three items and a large fraction of those individuals who buy bread and cheese also buy butter.

For example, the association rule bread $\wedge$ cheese $\Rightarrow$ butter [support = 20%, confidence = 85%] represents facts:

- 20% of all transactions under analysis contain bread, cheese and butter;
- 85% of the customers who purchased bread and cheese also purchased butter.

The result of association analysis is strong association rules, which are rules satisfying a minimal support and minimal confidence threshold. The minimal support and the minimal confidence are input parameters for association analysis.

The problem of association rules mining can be decomposed into two sub-problems [**1**]:

- Discovering frequent or large itemsets. Frequent itemsets have a support greater than the minimal support;
- Generating rules. The aim of this step is to derive rules with high confidence (strong rules) from frequent itemsets. For each frequent itemset l one finds all nonempty subsets of l; for each $a \subset l \wedge a \neq \emptyset$ one generates the rule $a \Rightarrow l - a$, if $\frac{\text{support}(l)}{\text{support}(a)} >$ minimal confidence.

We do not consider the second sub-problem in this paper, because the overall performances of mining association rules are determined by the first step. Efficient algorithms for solving the second sub-problem are exposed in [**12**].

The paper is organized as follows. Section 2 provides formalization of frequent itemsets mining problem. Section 3 describes Apriori algorithm. Section 4 presents our candidate generation procedure. Section 5 explains how to use hypothesis testing to validate frequent itemsets.

2. Preliminaries

Suppose that I is a finite set; we refer to the elements of I as items.

Definition 2.1. A transaction dataset on I is a function $T : \{1, \dots, n\} \rightarrow P(I)$. The set $T(k)$ is the k-th transaction of T. The numbers $1, \dots, n$ are the transaction identifiers (TIDs).

Given a transaction data set T on the set I, we would like to determine those subsets of I that occur often enough as values of T.

Definition 2.2. Let $T : 1, \dots, n \rightarrow P(I)$ be a transaction data set on a set of items I. The support count of a subset K of the set of items I in T is the number $\text{suppcount}_T(K)$ given by: $\text{suppcount}_T(K) = |\{k \mid 1 \leqslant k \leqslant n \wedge K \subseteq T(k)\}|$.

The support of an item set K (in the following text instead of an item set K we will use an itemset K) is the number: $\text{supp}_T(K) = \text{suppcount}_T(K)/n$.

The following rather straightforward statement is fundamental for the study of frequent itemsets.

THEOREM 2.1. *Let $T : 1, \ldots, n \to P(I)$ be a transaction data set on a set of items I. If K and K' are two itemsets, then $K' \subseteq K$ implies* $\mathrm{supp}_T(K') \geqslant \mathrm{supp}_T(K)$.

PROOF. The theorem states that supp_T for an itemset has the anti-monotone property. It means that support for an itemset never exceeds the support for its subsets. For proof, it is sufficient to note that every transaction that contains K also contains K'. The statement from the theorem follows immediately. □

DEFINITION 2.3. An itemset K is μ-frequent relative to the transaction data set T if $\mathrm{supp}_T(K) \geqslant mu$. We denote by F_T^μ the collection of all μ-frequent itemsets relative to the transaction data set T and by $F_{T,r}^\mu$ the collection of μ-frequent itemsets that contain r items for $r \geqslant 1$ (in the following text we will use r-itemset instead of itemset that contains r items).

Note that $F_T^\mu = \bigcup_{r\geqslant 1} F_{T,r}^\mu$. If μ and T are clear from the context, then we will omit either or both adornments from this notation.

3. Apiori algorithm

We will briefly present the Apriori algorithm, which is the most famous algorithm in the association analysis. The Apriori algorithm iteratively generates frequent itemsets starting from frequent 1-itemsets to frequent itemsets of maximal size. Each iteration of the algorithm consists of two phases: candidate generation and support counting. We will briefly explain both phases in the iteration k.

In the candidate generation phase potentially frequent k-itemsets or candidate k-itemsets are generated. The anti-monotone property of the itemset support is used in this phase and it provides elimination or pruning of some candidate itemsets without calculating its actual support (candidate containing at least one not frequent subset is pruned immediately, before support counting phase, see Theorem 2.1).

The support counting phase consists of calculating support for all previously generated candidates. In the support counting phase, it is essential to efficiently determine if the candidates are contained in particular transaction, in order to increase their support. Because of that, the candidates are organized in hash tree [**2**]. The candidates, which have enough support, are termed as frequent itemsets.

The Apriori algorithm terminates when none of the frequent itemsets can be generated.

We will use $C_{T,k}$ to denote candidate k-itemsets which are generated in the iteration k of the Apriori algorithm. Pseudocode for the Apriori algorithm comes next.

Algorithm: Apriori
Input: A transaction dataset T; Minimal support threshold μ
Output: F_T^μ

```
Method:
    1.  C_{T,1} = {{i} | i ∈ I}
    2.  F^μ_{T,1} = {K ∈ C_{T,1} | supp_T(K) ⩾ μ}
    3.  for (k=2; F^μ_{T,k-1} ≠ ∅; k++)
           C_{T,k} = apriori_gen(F^μ_{T,k-1})
           F^μ_{T,k} = {K ∈ C_{T,k} | supp_T ⩾ μ}
        end for
    4.  F^μ_T = ⋃_{r⩾1} F^μ_{T,r}
```

The `apriori_gen` is a procedure for candidate generation in the Apriori algorithm [**2**]. In the iteration k, it takes as argument $F^{\mu}_{T,k-1}$ – the set of all μ-frequent $(k-1)$-itemsets (which is generated in the previous iteration) and returns the set $C_{T,k}$. The set $C_{T,k}$ contains candidate k-itemsets (potentially frequent k-itemsets) and accordingly it contains the set $F^{\mu}_{T,k}$ – the set of all μ-frequent k-itemsets. The procedure works as follows. First, in the join step, we join $F^{\mu}_{T,k}$ with itself:

```
INSERT INTO C_{T,k}
SELECT R_1.item_1, R_1.item_2, ..., R_1.item_{k-1}, R_2.item_{k-1}
FROM F^μ_{T,k-1} AS R_1, F^μ_{T,k-1} AS R_2
WHERE R_1.item_1 = R_2.item_1 ∧ ··· ∧ R_1.item_{k-2} = R_2.item_{k-2} ∧ R_1.item_{k-1} <
R_2.item_{k-1}
```

In the previous query the set $F^{\mu}_{T,k}$ is considered as a relational table with $(k-1)$ attributes: $item_1, item_2, \dots, item_{k-1}$.

The previous query requires $k-2$ equality comparisons and is implemented with nested-loop join algorithm in the original Apriori algorithm from [**2**]. The nested-loop join algorithm is expensive, since it examines every pair of tuples in two relations:

```
for each l_1 ∈ F^μ_{T,k-1}
    for each l_2 ∈ F^μ_{T,k-1}
        if l_1[1] = l_2[1] ∧ ··· ∧ l_1[k-2] = l_2[k-2] ∧ l_1[k-1] < l_2[k-1]
            new_cand={l_1[1], l_1[2], ..., l_1[k-1], l_2[k-1]}
```

Consider the cost of nested-loop join algorithm. The number of pairs of itemsets (tuples) to be considered is $|F^{\mu}_{T,k-1}| * |F^{\mu}_{T,k-1}|$, where $|F^{\mu}_{T,k-1}|$ denotes the number of itemsets in $F^{\mu}_{T,k-1}$. For each itemset from $F^{\mu}_{T,k-1}$, we have to perform a complete scan on $F^{\mu}_{T,k-1}$. In the worst case the buffer can hold only one block of $F^{\mu}_{T,k-1}$, and a total of $|F^{\mu}_{T,k-1}| * n_b + n_b$ block transfers would be required, where denotes the number of blocks containing itemsets of $F^{\mu}_{T,k-1}$.

Next in the *prune* step, we delete all candidate itemsets $c \in C_{T,k}$ such that some $(k-1)$-subset of c is not in $F^{\mu}_{T,k-1}$ (see Theorem 2.1):

```
for each c ∈ C_{T,k}
    for each (k-1)-subset s in c
        if s ∉ F^μ_{T,k-1} delete c from C_{T,k}
```

In the next section we suggest a more efficient procedure for candidate generation phase and an efficient algorithm for its implementation.

4. New candidate generation procedure

We assume that any itemset K is kept sorted according to some relation $<$, where for all $x, y \in K$, $x < y$ means that the object x is in front of the object y. Also, we assume that all transactions in the database T and all subsets of K are kept sorted in lexicographic order according to the relation $<$.

For candidate generation we suggest the original method by which the set $C_{T,k}$ is calculated by joining $F^{\mu}_{T,k-1}$ with $F^{\mu}_{T,k-2}$, in the iteration k, for $k \geqslant 3$. A candidate k-itemset is created from one large $(k-1)$-itemset and one large $(k-2)$-itemset in the following way. Let $X = \{x_1, \dots, x_{k-1}\} \in F^{\mu}_{T,k-1}$ and $Y = \{y_1, \dots, y_{k-2}\} \in F^{\mu}_{T,k-2}$. Itemsets X and Y are joined if and only if the following condition is satisfied:

$$x_i = y_i,\ (1 \leqslant i \leqslant k-3) \quad \wedge \quad x_{k-1} < y_{k-2} \tag{4.1}$$

producing the candidate k-itemset $\{x_1, \dots, x_{k-2}, x_{k-1}, y_{k-2}\}$.

We will prove the correctness of the suggested method. In the following text we will denote this method by $C_{T,k} = F^{\mu}_{T,k-1} \times F^{\mu}_{T,k-2}$. Let $I = i_1, \dots, i_n$ be a set of items that contains n elements. Denote by $G_I = (P(I), E)$ the Rymon tree (see Appendix) of $P(I)$. The root of the tree is $\emptyset$. A vertex $K = \{i_{p_1}, \dots, i_{p_k}\}$ with $i_{p_1 < i_{p_2 < \dots < i_{p_k}}}$ has $n - i_{p_k}$ children $K \cup j$, where $i_{p_k} < j \leqslant n$. Let S_r be the collection of itemsets that have r elements. The next theorem suggest a technique for generating S_r starting from S_{r-1} and S_{r-2}.

THEOREM 4.1. *Let G be the Ryman tree of $P(I)$, where $I = i_1, \dots, i_n$. If $W \in S_r$, where $r \geqslant 3$, then there exists a unique pair of distinct sets $U \in S_{r-1}$ and $V \in S_{r-2}$ that has a common immediate ancestor $T \in S_{r-3}$ in G such that $U \cap V \in S_{r-3}$ and $W = U \cup V$.*

PROOF. Let u and v and p be the three elements of W that have the largest, the second-largest and the third-largest subscripts, respectively. Consider the sets $U = W - \{u\}$ and $V = W - \{v, p\}$. Note that $U \in S_{r-1}$ and $V \in S_{r-2}$. Moreover, $Z = U \cup V$ belongs to S_{r-3} because it consists of the first $r-3$ elements of W. Note that both U and V are descendants of Z and that $U \cup V = W$ (for $r = 3$ we have $Z = \emptyset$).

The pair (U, V) is unique. Indeed, suppose that W can be obtained in the same manner from another pair of distinct sets $U_1 in S_{r-1}$ and $V_1 \in S_{r-2}$ such that U_1 and V_1 are immediate descendants of a set $Z_1 \in S_{r-3}$. The definition of the Rymon tree G_I implies that $U_1 = Z_1 \cup \{i_m, i_q\}$ and $V_1 = Z_1 \cup \{i_y\}$, where the letters in Z_1 are indexed by a number smaller than $\min\{m, q, y\}$. Then, Z_1 consists of the first $r-3$ symbols of W, so $Z_1 = Z$. If $m < q < y$, then m is the third-highest index of a symbol in W, q is the second-highest index of a symbol in W and y is the highest index of a symbol in W, so $U_1 = U$ and $V_1 = V$. $\square$

The following theorem directly proves correctness of our method $C_{T,k} = F^{\mu}_{T,k-1} \times F^{\mu}_{T,k-2}$.

THEOREM 4.2. *Let T be a transaction data set on a set of items I and let $k \in N$ such that $k > 2$. If W is a μ-frequent itemset and $|W| = k$, then there exists* *a μ-frequent itemset Z and two itemsets $\{i_m, i_q\}$ and $\{i_y\}$ such that $|Z| = k-3$, $Z \subseteq W$, $W = Z \cup \{i_m, i_q, i_y\}$ and both $Z \cup \{i_m, i_q\}$ and $Z \bigcup \{i_y\}$ are μ-frequent itemsets.*

PROOF. If W is an itemset such that $|W| = k$, than we already know that W is the union of two subsets U and V of I such that $|U| = k-1$, $|V| = k-2$ and that $Z = U \cap V$ has $k-3$ elements (it follows from Theorem 2). Since W is a μ-frequent itemset and Z, U, V are subsets of W, it follows that each of these sets is also a μ-frequent itemset (it follows from Theorem 1). □

We have seen in the previous section that in the original Apriori algorithm from [**2**], the join procedure is suggested. It generates candidate k-itemset by joining two large $(k-1)$-itemsets, if and only if they have first $(k-2)$ items in common. Because of that, each join operation requires $(k-2)$ equality comparisons. If a candidate k-itemset is generated by the method $C_{T,k} = F^{\mu}_{T,k-1} \times F^{\mu}_{T,k-2}$ for $k \geqslant 3$, it is enough $(k-3)$ equality comparisons to process.

The method $C_{T,k} = F^{\mu}_{T,k-1} \times F^{\mu}_{T,k-2}$ can be represented by the following SQL query:

INSERT INTO $C_{T,k}$
SELECT $R_1.item_1, \dots, R_1.item_{k-1}, R_2.item_{k-2}$
FROM $F^{\mu}_{T,k-1}$ **AS** $R_1, F^{\mu}_{T,k-2}$ **AS** R_2
WHERE $R_1.item_1 = R_2.item_1 \wedge \cdots \wedge R_1.item_{k-3} = R_2.item_{k-3} \wedge R_1.item_{k-1} < R_2.item_{k-2}$

For the implementation of the join $C_{T,k} = F^{\mu}_{T,k-1} \times F^{\mu}_{T,k-2}$ we suggest a modification of sort-merge-join algorithm (note that $F^{\mu}_{T,k-1}$ and $F^{\mu}_{T,k-2}$ are sorted because of the way they are constructed and lexicographic order of itemsets).

By the original sort-merge-join algorithm [**9**], it is possible to compute natural joins and equi-joins. Let $r(R)$ and $s(S)$ be the relations and $R \cap S$ denote their common attributes. The algorithm keeps one pointer on the current position in relation $r(R)$ and another one pointer on the current position in relation $s(S)$. As the algorithm proceeds, the pointers move through the relations. It is supposed that the relations are sorted according to joining attributes, so tuples with the same values on the joining attributes are in a consecutive order. Thereby, each tuple needs to be read only once, and, as a result, each relation is also read only once.

The number of block transfers is equal to the sum of the number of blocks in both sets $F^{\mu}_{T,k-1}$ and $F^{\mu}_{T,k-2}$, $n_{b1} + n_{b2}$. We have seen that nested-loop join requires $|F^{\mu}_{T,k-1}| * n_{b1} + n_{b1}$ block transfers and we can conclude that merge-join is more efficient ($|F^{\mu}_{T,k-1}| * n_{b1} \gg n_{b2}$).

The modification of sort-merge-join algorithm we suggest refers to the elimination of restrictions that a join must be natural or equi-join. First, we separate the condition (4.1):

$$x_i = y_i, \quad 1 \leqslant i \leqslant k-3 \tag{4.2}$$

$$x_{k-1} < y_{k-2}. \tag{4.3}$$

Joining $C_{T,k} = F^{\mu}_{T,k-1} \times F^{\mu}_{T,k-2}$ is calculated according to the condition (4.2), in other words we compute the natural join. For this, the described sort-merge-join algorithm is used, and our modification is: before $X = \{x_1, \ldots, x_{k-1}\}$ and $Y = \{y_1, \ldots, y_{k-2}\}$, for which $X \in F^{\mu}_{T,k-1}$ and $Y \in F^{\mu}_{T,k-2}$ and $x_i = y_i$, $1 \leqslant i \leqslant k-3$ is true, are joined, we check if condition (4.3) is satisfied, and after that we generate a candidate k-itemset $\{x_1, \ldots, x_{k-2}, x_{k-1}, y_{k-2}\}$.

The pseudocode of **apriori_gen** function comes next.

```
FUNCTION apriori_gen(F^μ_{T,k-1}, F^μ_{T,k-2})
    1.  i = 0
    2.  j = 0
    3.  while i ⩽ |F^μ_{T,k-1}| ∧ j ⩽ |F^μ_{T,k-2}|
          iset_1 = F^μ_{T,k-1}[i++]
          S = {iset_1}
          done = false
          while done = false ∧ i ⩽ F^μ_{T,k-1}
              iset_1a = F^μ_{T,k-1}[i++]
              if iset_1a[w] = iset_1[w], 1 ⩽ w ⩽ k-2 then
                  S = S ⋃ {iset_1a}
                  i++
              else
                  done = true
              end if
          end while
          iset_2 = F^μ_{T,k-2}[j]
          while j ⩽ |F^μ_{T,k-2}| ∧ iset_2[1,...,k-2] ≺ iset_1[1,...,k-2]
              iset_2 = F^μ_{T,k-2}[j++]
          end while
          while j ⩽ |F^μ_{T,k-2}| ∧ iset_1[w] = iset_2[w], 1 ⩽ w ⩽ k-2
              for each s ∈ S
                  if iset_1[k-1] ≺ iset_2[k-2] then
                      c = {iset_1[1],...,iset_1[k-1], iset_2[k-2]}
                      if c contains-not-frequnet-subset then
                          DELETE c
                      else
                          C_{T,k} = C_{T,k} ⋃ {c}
                      end if
              end for
              j++
              iset_2 = F^μ_{T,k-2}[j]
          end while
      end while
```

We implemented the original Apriori [**2**] and the algorithm proposed here. Algorithms are implemented in C in order to evaluate its performances. Experiments are performed on PC with a CPU Intel(R) Core(TM)2 clock rate of 2.66GHz and with 2GB of RAM. Also, run time used here means the total execution time, i.e., the period between input and output instead of CPU time measured in the experiments in some literature. In the experiments dataset which can be found on www.cs.uregina.ca is used. It contains 10000 binary transactions. The average length of transactions is 8.

Table 1 shows that the original Apriori algorithm from [**2**] is outperformed by the algorithm presented here.

TABLE 1. Execution time in seconds

min sup(%)	Apriori	Rymon tree based implementation
10.0	15.00	3.10
5.0	15.60	6.20
2.5	16.60	6.20
1.0	16.60	7.80
0.5	18.10	7.80

5. Validating frequent itemsets

In this section we use hypothesis testing to validate frequent itemsets which have been generated in the Apriori algorithm.

Hypothesis testing is a statistical inference procedure to determine whether a hypothesis should be accepted or rejected based on the evidence gathered from the data. Examples of hypothesis tests include verifying the quality of patterns extracted by many data mining algorithms and validating the significance of the performance difference between two classification models.

In hypothesis testing, we are usually presented with two contrasting hypothesis, which are known, respectively, as the null hypothesis and the alternative hypothesis. The general procedure for hypothesis testing consists of the following four steps:

- Formulate the null and alternative hypotheses to be tested.
- Define a test statistic θ that determines whether the null hypothesis should be accepted or rejected. The probability distribution associated with the test statistic should be known.
- Compute the value of θ from the observed data. Use the knowledge of the probability distribution to determine a quantity known as p-value.
- Define a significance level a which controls the range of θ values in which the null hypothesis should be rejected. The range of values for θ is known as the rejection region.

Now, we will back to mining frequent itemsets and we will evaluate the quality of the discovered frequent itemsets from a statistical perspective. The criterion for judging whether an itemset is frequent depends on the support of the itemset. Recall that support measures the number of transactions in which the itemset is actually observed. An itemset X is considered frequent in the data set T, if $\operatorname{supp}_T(X) > \min\sup$, where minsup is a user-specified threshold.

The problem can be formulated into the hypothesis testing framework in the following way. To validate if the itemset X is frequent in the data set T, we need to decide whether to accept the null hypothesis, $H_0 : \mathrm{supp}_T(X) = \min\sup$, or the alternative hypothesis $H_1 : \mathrm{supp}_T(X) > \min\sup$. If the null hypothesis is rejected, then X is considered as frequent itemset. To perform the test, the probability distribution for $\mathrm{supp}_T(X)$ must also be known.

THEOREM 5.1. *The measure* $\mathrm{supp}_T(X)$ *for the itemset* X *in transaction data set* T *has the binomial distribution with mean* $\mathrm{supp}_T(X)$ *and variance* $\frac{1}{n}(\mathrm{supp}_T(X)*(1-\mathrm{supp}_T(X)))$*, where* n *is the number of transactions in* T.

PROOF. We will use measure $\mathrm{suppcount}_T(X)$ and calculate the mean and the variance for it and later derive mean and variance for the measure $\mathrm{supp}_T(X)$. The measure $\mathrm{suppcount}_T(X) = X_n$ presents the number of transactions in T that contain itemset X, and $\mathrm{supp}_T(X) = \mathrm{suppcount}_T(X)/n$ (Definition 2.2).

The measure X_n is analogous to determining the number of heads that shows up when tossing n coins. Let us calculate $E(X_n)$ and $D(X_n)$.

Mean is $E(X_n) = n * p$, where p is the probability of success, which means (in our case) the itemset X appears in one transaction. According to the Bernoulli law it is: $\forall \epsilon > 0,\ \lim_{N\to\infty} P\{|X_n/n - p| \leqslant \epsilon\} = 1$. Freely speaking, for large n (we work with large databases so n can be considered large), we can use relative frequency instead of probability. So, we now have:

$$E(X_n) = np \approx n\frac{X_n}{n} = X_n$$

For variance we compute:

$$D(X_n) = np(1-p) \approx n\frac{X_n}{n}\Big(1 - \frac{X_n}{n}\Big) = \frac{X_n(n-X_n)}{n}$$

Now we compute $E(\mathrm{supp}_T(X))$ and $D(\mathrm{supp}_T(X))$. Recall that $\mathrm{supp}_T(X) = X_n/n$. We have:

$$\begin{aligned}
E(\mathrm{supp}_T(X)) &= E\Big(\frac{X_n}{n}\Big) = \frac{1}{n}E(X_n) = \frac{X_n}{n} = \mathrm{supp}_T(X).\\
D(\mathrm{supp}_T(X)) &= D\Big(\frac{X_n}{n}\Big) = \frac{1}{n^2}D(X_n) = \frac{1}{n^2}\frac{X_n(n-X_n)}{n}\\
&= \frac{1}{n}\,\mathrm{supp}_T(X)(1-\mathrm{supp}_T(X)) \qquad \square
\end{aligned}$$

The binomial distribution can be further approximated using a normal distribution if n is sufficiently large, which is typically the case in association analysis.

Regarding the previous paragraph and Theorem 5.1, under the null hypothesis $\mathrm{supp}_T(X)$ is assumed to be normally distributed with mean $\min\sup$ and variance $\frac{1}{n}(\min\sup(1-\min\sup))$. To test whether the null hypothesis should be accepted or rejected, the following statistic can be used:

$$W_N = \frac{\mathrm{supp}_T(X) - \min\sup}{\sqrt{(\min\sup(1-\min\sup))/n}}. \tag{5.1}$$

The previous statistic, according to the Central Limit Theorem, has the distribution $N(0,1)$. The statistic essentially measures the difference between the observed support $\mathrm{supp}_T(X)$ and the min sup threshold in units of standard deviation.

Let $N = 10000$, $\mathrm{supp}_T(X) = 0.11$, $\min\sup = 0.1$ and $\alpha = 0.001$. The last parameter is the desired significance level. It controls Type 1 error which is rejecting the null hypothesis even though the hypothesis is true.

In the Apriori algorithm we compare $\mathrm{supp}_T(X) = 0.11 > 0.1 = \min\sup$ and we declare X as frequent itemset. Is this validation procedure statistically correct?

Under the hypothesis H_1 statistics W_{10000} is positive and for the rejection region we choose $R = \{(x_1, \dots, x_{10000}) | w_{10000} > k\}$, $k > 0$. Let us find k.

$$0.001 = P_{H_0}\{W_{10000} > k\}$$
$$P_{H_0}\{W_{10000} > k\} = 0.499$$
$$k = 3.09$$

Now we compute $w_{10000} = (0.11 - 0.1)\left(\frac{0.1*(1-0.1)}{10000}\right)^{-1/2} = 3.33\dots$.

We can see that $w_{10000} > k$, so we are within the rejection region and H_1 is accepted, which means the itemset X is considered statistically significant.

6. Appendix

Rymon tree was introduced in [**8**] in order to provide a unified search-based framework for several problems in artificial intelligence; the Rymon tree is also useful for data mining algorithms.

In Definitions 6.1 and 6.2 we define necessary concepts and in Definition 6.3 we define the Rymon tree.

DEFINITION 6.1. Let S be a set and let $d : S \to N$ be an injective function. The number $d(x)$ is the index of $x \in S$. If $P \subseteq S$, *the view of* P is the subset $view(d, P) = \{s \in S \mid d(s) > \max_{p \in P} d(p)\}$.

DEFINITION 6.2. A collection of sets C is *hereditary* if $U \in C$ and $W \subseteq U$ implies $W \in C$.

DEFINITION 6.3. Let C be a hereditary collection of subsets of a set S. The graph $G = (C, E)$ is a Rymon tree for C and the indexing function d if:

- the root of the G is $\emptyset$
- the children of a node P are the sets of the form $P \cup \{s\}$, where $s \in \mathrm{view}(d, P)$

If $S = \{s_1, \dots, s_n\}$ and $d(s_i) = i$ for $1 \leqslant i \leqslant n$, we will omit the indexing function from the definition of the Rymon tree for $P(S)$.

A key property of a Rymon tree is stated next.

THEOREM 6.1. *Let G be a Rymon tree for a hereditary collection C of subsets of a set S and an indexing function d. Every set P of C occurs exactly once in the tree.*

PROOF. The argument is by induction on $p = |P|$. If $p = 0$, then P is the root of the tree and the theorem obviously holds.

Suppose that the theorem holds for sets having fewer than p elements, and let $P \in C$ be such that $|P| = p$. Since C is hereditary, every set of the form $P - \{x\}$ with $x \in P$ belongs to C and, by the inductive hypothesis, occurs exactly once in the tree.

Let z be the element of P that has the largest value of the index function d. Then $view(P - \{z\})$ contains z and P is a child of the vertex $P - \{z\}$. Since the parent of P is unique, it follows that P occurs exactly once in the tree. □

Note that in the Rymon tree of a collection of the form $P(S)$, the collection of sets of S_r that consists of sets located at distance r from the root denotes all subsets of the size r of S.

References

[1] R. Agrawal, R. Srikant, *Fast algorithms for mining association rules*, Proc. VLDB-94 (1994), 487–499.

[2] F. P. Coenen, P. Leng, S. Ahmed, *T-trees, vertical partitioni-ng and distributed association rule mining*, Proc. Int. Conf. Discr. Math. 2003 (2003), 513–516.

[3] F. P. Coenen, P. Leng, S. Ahmed, *Data structures for association rule mining: T-trees and P-trees*, IEEE Trans. Data Knowledge Engin. 16(6), (2004), 774–778.

[4] F. P. Coenen, P. Leng, G. Goulbourne, *Tree structures for mining association rules*, J. Data Min. Knowledge Discovery 8(1) (2004), 25–51.

[5] G. Goulbourne, F. P. Coenen, P. Leng, *Algorithms for computing association rules using a partial-support tree*, J. Knowledge-based Syst. 13 (1999), 141–149.

[6] G. Grahne, J. Zhu, *Efficiently using prefix-trees in mining frequent itemsets*, Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (2003).

[7] J. Han, J. Pei, P.S. Yu, *Mining frequent patterns without candidate generation*, Proc. ACM SIGMOD Conf. Management of Data, (2000), 1–12.

[8] R. Rymon, *Search through systematic set enumeration*, Proc. 3rd Int. Conf. Principles of Knowledge Representation and Reasoning, (1992), 539–550.

[9] A. Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts*, McGraw Hill, New York (2006)

[10] A. D. Simovici, C. Djeraba, *Mathematical tools for data mining (Set Theory, Partial Orders, Combinatorics)*, Springer-Verlag, London, 2008.

[11] P. Stanišić, S. Tomović, *Apriori multiple algorithm for mining association rules*, Information Technology and Control 37(4) (2008), 311–320.

[12] P. N. Tan., M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison Wesley, 2006.

Department of Mathematics and Computer Science
University of Montenegro
Podgorica
Montenegro
pedjas@ac.me
savotom@gmail.com

(Received 06 03 2009)
(Revised 22 11 2009)