# Cross Validation Method in Frequent Itemset Mining

**Savo Tomović, Predrag Stanišić**
Faculty of Science
University of Montenegro
Cetinjski put bb, 81000 Podgorica, Montenegro
{savotom@rc.pmf, pedjas}@ac.me

**Abstract.** *We suggest a new method for frequent itemset mining which is based on well known cross validation method from artificial intelligence and machine learning. In non optimized version we partition the database into two subsets. First, we choose one of the subsets for training, and the other for testing. From the training subset we mine frequent itemsets and use testing subset to calculate itemsets' support in whole database. We then swap the roles of the subsets, so that the previous training set becomes the test set and vice versa. Again we mine all frequent itemsets from training subset and use the other set to calculate supports in whole database. In this approach each record is used exactly once for training and once for testing, which means that the database is read just twice. Optimized version is based on the idea to use all known information about itemsets from the first step when we run the second step. This will reduce the number of itemsets to be considered.*

**Keywords.** itemset mining, association analysis, Apriori algorithm

## 1 Introduction

Finding frequent itemsets in databases is the fundamental operation behind association rule mining. The problem of mining association rules over transactional databases was introduced in [1]. An example of such rule might be that "85% of customers who bought milk also bought bread". Discovering all such rules is important for planning marketing campaigns, designing catalogues, managing prices and stocks, customer relationships management etc. For example, a shop may decide to place bread close to milk because they are often bought together, to help shoppers finish their task faster. Or the shop may place them at opposite ends of a row, and place other associated items in between to tempt people to buy those items as well, as shoppers walk from one end of the row to the other.

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

1. Mining frequent itemsets. Frequent itemsets have support greater than the minimal support threshold;

2. Generating rules. The aim of this step is to derive rules with high confidence from large itemsets (confidence of association rule is defined in the next section). For each large itemset $l$ one finds all not empty subsets of $l$; for each $a \subset l \wedge a \neq \oslash$ one generates the rule $a \rightarrow l - a$, if $conf(a \rightarrow l - a) \geq minimalconfidence$.

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 presented in [2].

Many algorithms were proposed to solve frequent itemset mining but the most important is Apriori algorithm [1].

In this paper we propose Apriori based algorithm for frequent itemset mining which uses cross validation approach from machine learning. New algorithm is scalable and comparable to existing algorithms.

The paper is organized as follows. Section 2 provides formalization of frequent itemsets mining problem. Section 3 presents the new algorithm. Section 4 is conclusion.

## 2 Preliminaries

This section contains definitions that are necessary for further text. We primarily use notions from [3].

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

**Definition 1.** A transaction data set on $I$ is a function $T : \{1, ..., n\} \to P(I)$. The set $T(k)$ is the $k$th transaction of $T$. The numbers $1, ..., n$ 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.** Let $T : \{1, ..., n\} \to P(I)$ 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 $suppcount_T(K)$ given by: $suppcount_T(K) = |\{t | 1 \leq t \leq n \wedge K \subset T(t)\}|$. The support of an itemset $K$ is the number: $support_T(K) = \frac{suppcount_T(K)}{n}$.

**Definition 3.** An itemset $K$ is $\mu$-frequent relative to the transaction data set $T$ if $support_T(K) \geq \mu$. We denote by $F_T^\mu$ the collection of all $\mu$-frequent itemsets relative to the transaction data set $T$ and by $F_{T,r}^\mu$ the collection of all $\mu$-frequent itemsets that contain $r$ items for $r \geq 1$.

Note that $F_T^\mu = \bigcup_{r \geq 1} F_{T,r}^\mu$.

**Definition 4.** Frequent itemset mining problem consists of finding the set $F_T^\mu$ for given minimal support $\mu$ and transaction data set $T$.

The following rather straightforward statement is fundamental for the study of frequent itemsets. It is known as Apriori principle.

**Theorem 1.** Let $T : \{1, ..., n\} \to P(I)$ be a transaction data set on a set of items $I$. If $K$ and $K_1$ are two itemsets, then $K_1 \subset K$ implies $support_T(K_1) \geq support_T(K)$.

**Definition 5.** An association rule on an itemset $I$ is a pair of nonempty disjoint itemsets $(X, Y)$. An association rule $(X, Y)$ is denoted by $X \to Y$. The confidence of $X \to Y$ is the number $conf_T(X \to Y) = \frac{support_T(XY)}{support_T(X)}$.

## 3 Cross Validation Approach in Frequent Itemset Mining

### 3.1 Cross Validation

Cross validation method is developed in machine learning. In this approach each record from the input data set is used the same number of times for training and exactly once for testing. To illustrate this method, suppose we partition the data into two equal sized subsets. First, we choose one of the subsets for training and the other for testing. We then swap the roles of the subsets so that the previous training set becomes the test set and vice versa. This approach is called two-fold cross validation. In this example, each record is used exactly once for training and once for testing.

The $k$-fold cross validation method generalizes this approach by segmenting the data into $k$ equal sized partitions. During each run, one of the partition is chosen for testing, while the rest of them are used for training. This procedure is repeated $k$ times so that each partition is used for testing exactly once.

A special case of the $k$-fold cross validation method sets $k = n$, the size of the input data set. In this so-called leave-one-out approach each test set contains only one record. This approach has the advantage of utilizing as much as possible for training. In addition the test sets are mutually exclusive and they effectively cover the entire database. The drawback of this approach is that it is computationally expensive to repeat the procedure $n$ times.

#### 3.1.1 Apriori Two-Fold Cross Validation

We implemented the idea of two-fold cross validation for frequent itemset mining. Apriori Two-Fold Cross Validation algorithm is presented in Figure 1.

The first step is partitioning transaction data set $T : \{1, ..., n\} \to P(I)$ on a set of items $I$. Let $\pi$ be a partition of the set $\{1, ..., n\}$ of transaction identifiers, $\pi = \{B_1, ..., B_p\}$. Let $n_i = |B_i|$ for $1 \leq p \leq n$.

**Definition 6.** A *partitioning* of $T$ is a sequence $T_1, T_2, ..., T_p$ of transaction data sets on $I$ such that $T_i : \{1, ..., n_i\} \to P(I)$ is defined by $T_i(l) = T(k_l)$, where $B_i = \{k_1, ..., k_{n_i}\}$ for $1 \leq p \leq n$.

Intuitively, this corresponds to splitting hori-

**Apriori Two-Fold Cross Validation (database $T$, minimal support $\mu$)**

1: divide database into two partitions $T_1$ and $T_2$, $T_1 \cup T_2 = T, T_1 \cap T_2 = \oslash$
2: $F^{\mu}_{T_1} = Apriori\,(T_1, \mu)$
3: $F^{\mu}_{T_1,T_2}$ =validate $F^{\mu}_{T_1}$ with $T_2$
4: $F^{\mu}_{T_2,T_1}$ =validate $F^{\mu}_{T_1}$ with $T_1$
5: $F^{\mu}_{T_2,T_1} = validate F^{\mu}_{T_2} with T_1$
6: $F^{\mu}_{T} = F^{\mu}_{T_1,T_2} \cup F^{\mu}_{T_2,T_1}$

Figure 1: Apriori Two-Fold Cross Validation Algorithm

zontally the table $T$ into $p$ tables that contain $n_1, ..., n_p$ consecutive rows.

We choose $p = 2$ which means that we have two partitions of approximately the same size, $\frac{n}{2}$ transactions. Because of that we have two-fold cross validation.

In the second step Apriori algorithm [1] is used to find the set $F^{\mu}_{T_1}$. It contains all $\mu$-frequent itemsets relative to $T_1$. It is possible that some of them are not $\mu$-frequent relative to the whole transaction data set $T$. Itemsets that are $\mu$-frequent in $T_1$, but not $\mu$-frequent in $T$ are eliminated in step three. In this step partition $T_2$ is used to calculate support of each itemset from $F^{\mu}_{T_1}$ relative to $T$. Those itemsets from $F^{\mu}_{T_1}$ that have $suppcount_T \geq \mu$ are $\mu$-frequent relative to $T$. They are stored in $F^{\mu}_{T_1,T_2}$.

The set $F^{\mu}_{T_1,T_2}$ contains all $\mu$-frequent itemsets relative to $T$ that appear and are also $\mu$-frequent in $T_1$. Itemsets that are $\mu$-frequent relative to $T$, but appear in $T_2$ will be found next.

In the step four Apriori algorithm is used to find $\mu$-frequent itemsets in $T_2$. Result is the set $F^{\mu}_{T_2}$.

In the step five all itemsets that are $\mu$-frequent in $T_2$, but not $\mu$-frequent relative to $T$ are eliminated. It this step $T_1$ is used to calculate support relative to $T$ for each itemset from $F^{\mu}_{T_2}$. After this step itemsets from $F^{\mu}_{T_2}$ that are also $\mu$-frequent relative to $T$ are stored in the set $F^{\mu}_{T_2,T_1}$.

Finally, in the step six we obtain the set $F^{\mu}_{T} = F^{\mu}_{T_1,T_2} \bigcup F^{\mu}_{T_2,T_1}$ with $\mu$-frequent itemsets in $T$. Generally, sets $F^{\mu}_{T_1,T_2}$ and $F^{\mu}_{T_2,T_1}$ are not disjoint.

Apriori Two-Fold Cross Validation finds $\mu$-frequent itemsets in $T$ in two steps; it finds $\mu$-frequent itemsets that appear in $T_1$ and after that $\mu$-frequent itemsets in $T_2$. The following theorem states that algorithm is complete. It is easy to extend theorem to more general case when data set $T$ is partitioned into $p$ data sets $T_1, T_2, ..., T_p$.

**Theorem 2.** Let transaction data sets $T_1, T_2$ on $I$ be a partitioning of transaction data set $T : \{1, ..., n\} \rightarrow P(I)$ as in definition 6. For any itemset $K$ we have $support_T(K) \geq \mu \Rightarrow support_{T_1}(K) \geq \mu \vee support_{T_2}(K) \geq \mu$.

*Proof.* Seeking a contradiction suppose that $support_{T_1}(K) < \mu \wedge support_{T_2}(K) < \mu$. According to definition 2 we have $support_{T_1}(K) = suppcount_{T_1}(K)/|T_1|$ and $support_{T_2}(K) = suppcount_{T_2}(K)/|T_2|$. It follows $suppcount_{T_1}(K) = support_{T_1}(K)\,|T_1| < \mu\,|T_1|$ and $suppcount_{T_2}(K) = support_{T_2}(K)\,|T_2| < \mu\,|T_2|$. Adding two previous inequalities we get $suppcount_{T_1}(K) + suppcount_{T_2}(K) = suppcount_T(K) < \mu\,(|T_1| + |T_2|) = \mu n$. We conclude that $support_T(K) = \frac{suppcount_T(K)}{n} < \mu$ contradicting our assumption that $support_T(K) \geq \mu$. $\nabla$

We will now explain main steps in Apriori

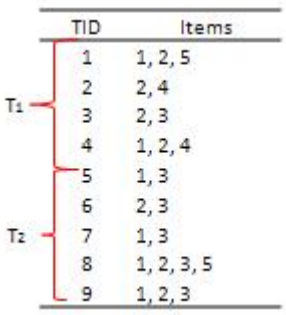


Figure 2: Transaction data set $T$

Two-Fold Cross Validation algorithm on simple example. Consider transaction data set $T$ from figure 2. It contains nine transactions. We partition $T$ into two transaction data sets $T_1$ and $T_2$; $T_1$ contains transaction 1-4; $T_2$ contains transactions 5-9. Let $\mu = \frac{2}{9} \approx 0.22$.

The set $F^{\mu}_{T_1}$ is generated by calling Apriori algorithm. The procedure is presented in figure 3. We did not explain Apriori algorithm in this paper; details are presented in [1]. Set of candidate $k$-itemsets (potentially $\mu$-frequent itemsets) is denoted by $C_k$. Set of $\mu$-frequent $k$-itemsets (those

with minimum support) is denoted by $F_k$. It holds that $F^{\mu}_{T_1} = F_1 \cup F_2 \cup F_3$. With $\mu = \frac{2}{9} \approx 0.22$ itemset $K$ is $\mu$-frequent in $T_1$ if it appears in at least $\lceil \mu \cdot |T_1| \rceil = \lceil 0.22... \cdot 4 \rceil = 1$ transaction (definition 2). Itemsets that are pruned according to Apriori principle (theorem 2) are denoted by red line.

The next step is to calculate support relative to

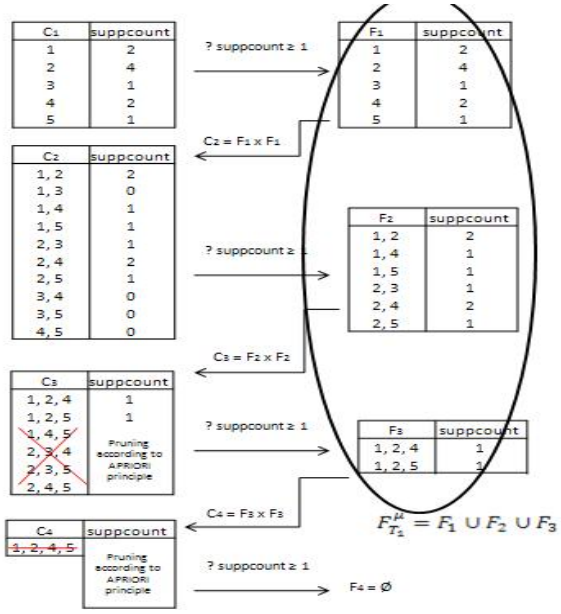


Figure 3: The second step of Apriori Two-Fold Cross Validation

$T$ for each $\mu$-frequent itemset in $T_1$ (the set $F^{\mu}_{T_1}$) and eliminate those that are not $\mu$-frequent relative to $T$. In other words the algorithm generates the set $F^{\mu}_{T_1,T_2}$ which contains $\mu$-frequent itemsets in $T_1$ that are also $\mu$-frequent relative to $T$. In this step algorithm reads $T_2$ and updates support for each itemset from $F^{\mu}_{T_1}$ that also appears in $T_2$. With $\mu = \frac{2}{9} \approx 0.22$ itemset $K$ is $\mu$-frequent in $T$ if it appears in at least $\lceil \mu \cdot n \rceil = \lceil 0.22... \cdot 9 \rceil = 2$ transactions (definition 2). Values for *suppcount* from $F^{\mu}_{T_1}$ that are changed in this step are presented with red color in figure 4. Generation of $F^{\mu}_{T_1,T_2}$ in the third step of Apriori Two-Fold Cross Validation is presented in figure 4. Note that itemsets $\{1,2,4\}$ and $\{1,4\}$ did not survive validation step.

Now we change roles for $T_1$ and $T_2$.

In the forth step Apriori Two-Fold Cross Validation generates the set $F^{\mu}_{T_2}$. It contains all $\mu = \frac{2}{9}$-frequent itemsets in $T_2$. An itemset $K$ is $\mu = \frac{2}{9}$-frequent in $T_2$ if it appears in at least $\lceil \mu \cdot |T_2| \rceil = \lceil 0.22... \cdot 5 \rceil = 2$ transactions. The procedure is presented in figure 5.

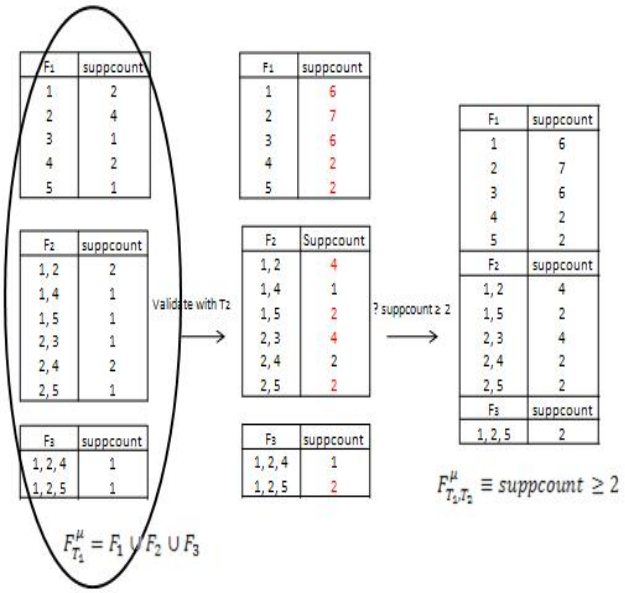


Figure 4: The third step of Apriori Two-Fold Cross Validation

In the sixth step Apriori Two-Fold Cross Validation generates the set $F^{\mu}_{T_2,T_1}$ which contains $\mu$-frequent itemsets in $T_2$ that are also $\mu$-frequent relative to $T$. The set $F^{\mu}_{T_2,T_1}$ is calculated from the set $F^{\mu}_{T_2}$ by eliminating those itemsets that are not $\mu$-frequent relative to $T$ (although they are $\mu$-frequent relative to $T_2$). In this step algorithm reads $T_1$ transaction by transaction and updates *suppcount* for each itemset in $F^{\mu}_{T_2}$ that appears in transactions from $T_1$. With $\mu = \frac{2}{9} \approx 0.22$ itemset $K$ is $\mu$-frequent in $T$ if it appears in at least $\lceil \mu \cdot n \rceil = \lceil 0.22... \cdot 9 \rceil = 2$ transactions (definition 2). Values for *suppcount* from $F^{\mu}_{T_2}$ that are changed in this step are presented with red color in figure 6. Generation of $F^{\mu}_{T_2,T_1}$ is presented in figure 6.

Final result is $F^{\mu}_{T} = F^{\mu}_{T_1,T_2} \cup F^{\mu}_{T_2,T_1}$.

Note that Apriori Two-Fold Cross Validation calculates twice *suppcount* for some itemsets. In the previous example *suppcount* is calculated twice for itemsets: $\{1\}, \{2\}, \{3\}, \{5\}, \{1,2\}, \{1,3\}, \{2,3\}$.

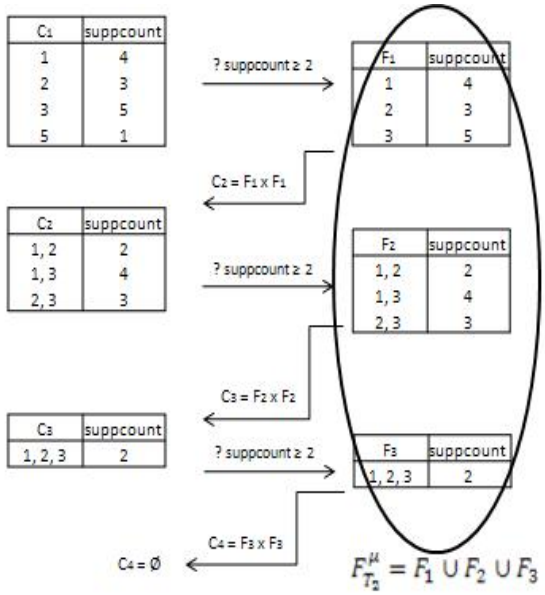


Figure 5: The forth step of Apriori Two-Fold Cross Validation

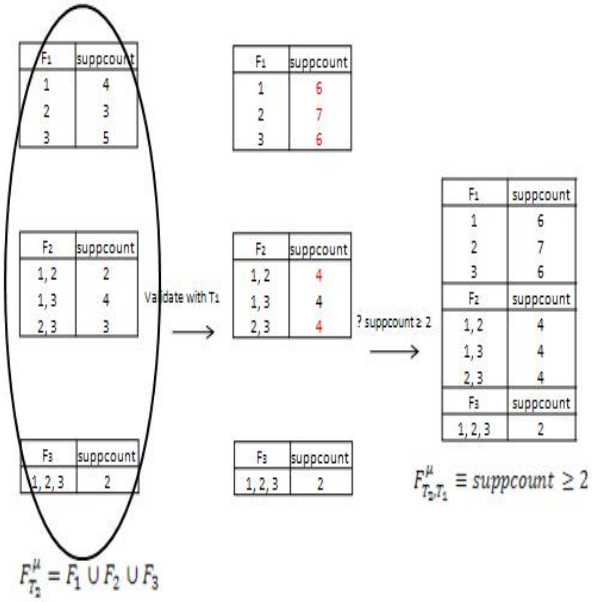


Figure 6: The sixth step of Apriori Two-Fold Cross Validation

The next section presents optimized version of Apriori Two-Fold Cross Validation in which *suppcount* for each itemset is calculated once.

### 3.2 Apriori Cross Validation Optimized

Apriori Cross Validation Optimized is based on special tree structure that is called TS-tree. TS-tree is presented in details in [4], [5], [6]. This paper contains just basic definitions.

**Definition 6.** 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 $P \subset S$, the $view(P, d)$ is the subset $view(P, d) = \{s \in S | d(s) > max_{p \in P} d(p)\}$.

**Definition 7.** Let $T : 1, ..., n \to P(I)$ be a transaction data set on a set of items $I$. TS-tree for database $T$ is special tree structure with the following properties:

- the root of TS-tree is labeled with *NULL*. It represents the empty set.
- every node $N$ in TS-tree contains four fields: *label, support, children* and *parent*. The field *label* registers which item this node represents. Path from the root to the node $N$ is unique in TS-tree and the node $N$ represents one candidate. If path $NULL \to N_{i_1} \to N_{i_2} \to ... \to N$ reaches the node $N$ then the node $N$ represents candidate set $A = \{N_{i_1}.label, ..., N.label\}$. The field *support* registers the number of transactions in database $T$ that contains the portion of the path reaching the node $N$. In other words, field *support* is $suppcount_T(A)$. The field *children* is array of pointers to TS-subtrees of the node $N$. Roots of these subtrees are labeled from the set $view(N)$. It implies that all paths in the TS-tree are lexicographically sorted. For leaves all pointers in array *children* are *null*. The field *parent* is pointer to the parent node. For the root this field is *null*.
- the root is at level 0. Nodes that represent candidate $k$-itemsets are at level $k$.

Consider candidate set $C_3 = \{145, 124, 457, 125, 458, 159, 136, 234, 567, 345, 356, 357, 689, 367,$

368}. TS-tree for $C_3$ is shown in figure 7. Null pointers, pointers to the parent nodes and values for support are not presented. The leaves of the tree represents candidate 3-itemsets. For example, the leave 4 from the path $NULL \rightarrow 1 \rightarrow 2 \rightarrow 4$ represents $124 \in C_3$. Support for each candidate will be calculated in support counting phase by "mapping" each transaction on the tree. In this phase every transaction is mapped on paths in the tree which reaches the leaves representing candidates contained in that transaction. At the end, candidate *suppcount* is stored in the field *support* in the leave that coresponds to that candidate. For example, *suppcount* for 124 is stored in the leave 4 from path $NULL \rightarrow 1 \rightarrow 2 \rightarrow 4$.

In [4], [5] and [6] TS-tree is used for:

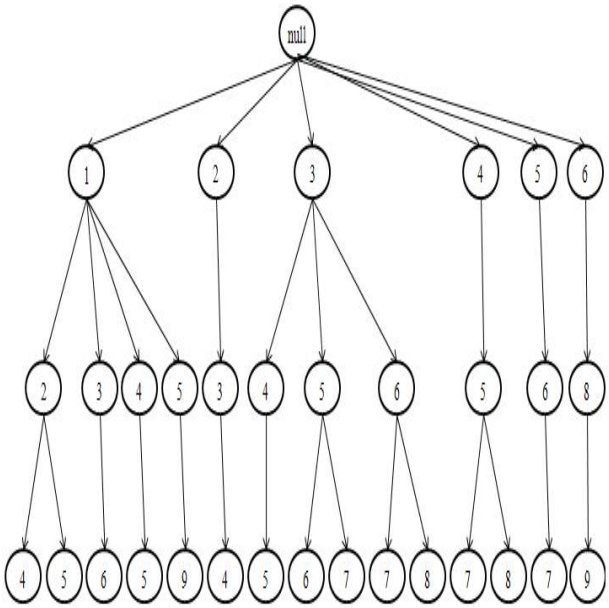


Figure 7: TS-tree example

- generating and storing candidate itemsets in candidate generation phase
- candidate pruning according to Apriori principle [1] in candidate generation phase
- calculating candidate supports in support counting phase

In addition, Apriori Cross Validation Optimized uses TS-tree for efficient implementation of validation steps (steps 3 and 5 in figure 1). The algorithm is presented in figure 8.

In the step 3, algorithm calls AprioriTS-tree procedure from [4]. It creates TS-tree for partition $T_1$. The tree contains the set $F^{\mu}_{T_1}$, i.e. all $\mu$-frequent itemsets in partition $T_1$. The next, forth step corresponds to the validation step 3 of the algorithm in figure 1. In this step itemsets that are in $F^{\mu}_{T_1}$, but not $\mu$-frequent in the whole database $T$ are eliminated. We implemented this step as follows. In the previous step algorithm generates TS-tree that contains all $\mu$-frequent itemsets in partition $T_1$. Each itemset is presented with one leave in the TS-tree. We use partiton $T_2$ to calculate support with respect to the whole database. After that we delete all leaves that corresponds to itemsets that are not $\mu$-frequent in $T$. In other words, TS-tree will contain $\mu$-frequent itemsets in $T$ that appear in $T_1$, i.e. the set $F^{\mu}_{T_1T_2}$.

**AprioriTS-tree Two-Fold Cross Validation (database $T$, minimal support $\mu$)**

1: divide database into two partitions $T_1$ and $T_2$, $T_1 \cup T_2 = T, T_1 \cap T_2 = \oslash$
2: TS-tree $root = NULL$
3: $F^{\mu}_{T_1} = AprioriTS - tree\,(T_1, \mu, root)$
4: $F^{\mu}_{T_1,T_2}$ =prune $root$ with $T_2$
5: $F^{\mu}_{T_2} = AprioriTS - tree\,(T_2, \mu, root)$
6: $F^{\mu}_{T_2,T_1}$ =prune $root$ with $T_1$
7: $F^{\mu}_{T} = F^{\mu}_{T_1,T_2} \cup F^{\mu}_{T_2,T_1}$

Figure 8: AprioriTS-tree Cross Validation Optimized Algorithm

The same TS-tree is used in the following steps. In the step 5 AprioriTS-tree procedure [4] is called to produce the set $F^{\mu}_{T_2}$, i.e. all $\mu$-frequent itemsets in partition $T_2$. Using TS-tree from the previous phase allows to make this phase more efficient: it is necessary to generate and calculate support just for itemsets that are not contained in the tree but are present in $T_2$. In this way we eliminate many candidates that are generated in the algorithm from figure 1. After this step TS-tree will contain:

- the set $F^{\mu}_{T_1T_2}$, i.e. all $\mu$-frequent itemsets in $T_1$ that are also $\mu$-frequent in $T$
- the set $F^{\mu}_{T_2}$, i.e. all $\mu$-frequent itemsets in $T_2$

It remains to validate the set $F^{\mu}_{T_2}$. We use the partition $T_1$. Transactions from $T_1$ are mapped onto paths in TS-tree that are created in step 5. These paths represent $\mu$-frequent itemsets in $T_2$. All other paths are not visited because they are processed in the step 4 and represent the set $F^{\mu}_{T_1T_2}$.

Finally, the tree contain the set $F^{\mu}_{T} = F^{\mu}_{T_1T_2} \cup F^{\mu}_{T_2T_1}$. Actually, all $\mu$-frequent itemsets are represented by leaves in the TS-tree along with their $suppcount_T$. With respect to memory usage previously described procedure can be charactetrized as mining at place, because no additional space is necessary for storing frequent itemsets.

We will illustrate the previous explanations by the following example. Consider transaction dataset from figure 2. Partitions $T_1$ and $T_2$ are indicated on the same figure. Let $\mu = \frac{2}{9} \approx 0.22$.

In step 3 algorithm creates TS-tree with $\mu$-frequent itemsets in $T_1$. The tree is presented in figure 9. For each itemset $X$ the number $suppcount_{T_1}(X)$ is shown above the node in the tree that represents that itemset. For example, itemset $\{1,2,5\}$ is represented by the last node in the path $NULL \rightarrow 1 \rightarrow 2 \rightarrow 5$ and its $suppcount_{T_1}$ is 1.

The step 4 is to validate $\mu$-frequent itemsets in

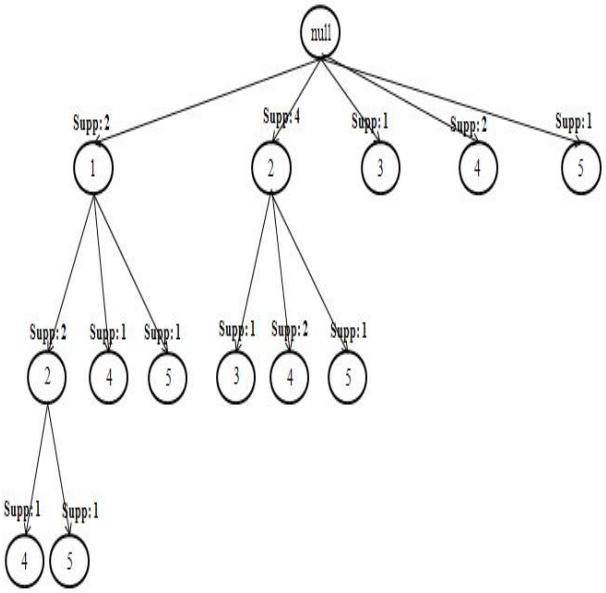


Figure 9: TS-tree for the first partition

$T_1$. The partition $T_2$ is used. Each transaction from $T_2$ is mapped onto paths from TS-tree. Procedure for mapping transaction onto paths in TS-tree is presented in [4]. The main idea is to extract those itemsets from transaction that are also contained in the tree. In this way just paths that represents itemsets contained in the transactions are visited, not all paths in the tree. The TS-tree after validation step is presented in figure 10. With red font we indicate $suppcount_{T_2}$ for $\mu$-frequent itemsets in $T_1$. For example, $suppcount_T(\{1,2,5\}) = suppcount_{T_1}(\{1,2,5\}) + suppcount_{T_2}(\{1,2,5\}) = 1 + 1$. Itemsets that are not $\mu$-frequent in $T$ (although they are in $T_1$) are pruned from the tree. For example, $suppcount_T(\{1,2,4\}) = suppcount_{T_1}(\{1,2,4\}) + suppcount_{T_2}(\{1,2,4\}) = 1 + 0 < 2$. Pruning is indicated in figure 10 by red cross on incoming edge for the node which is pruned.

In the step 5 AprioriTS-tree procedure is called

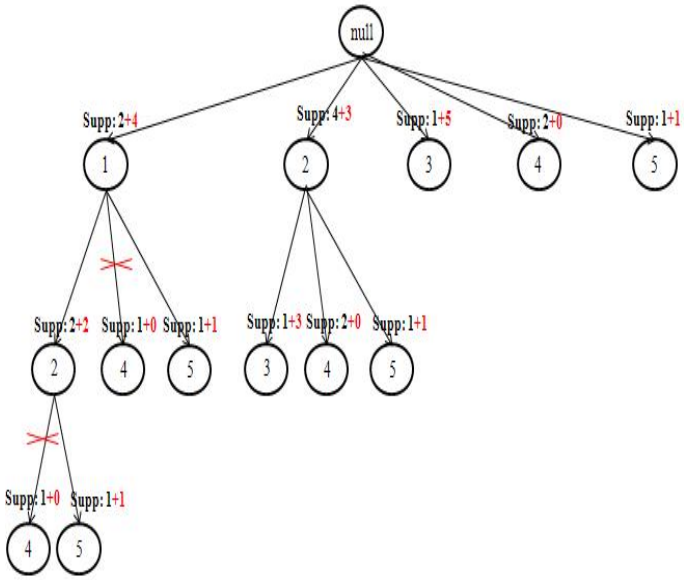


Figure 10: TS-tree after validation

for partition $T_2$. It is to find $\mu$-frequent itemsets in $T_2$. The previous TS-tree is used. We remind that this tree contains the set $F^{\mu}_{T_1T_2}$, i.e. $\mu$-frequent itemsets in $T_1$ that are also $\mu$-frequent in $T$. The tree also contains $suppcount_T$ for each itemset from $F^{\mu}_{T_1T_2}$.

The main benefit from using the same TS-tree is

in that it allows less candidate to be generated.

Let us explain in more details. In figure 5 presented is the corresponding step of Apriori Cross Validation algorithm. The algorithm generates and counts support for the following candidate itemsets: $C_1 = \{\{1\},\{2\},\{3\},\{5\}\}$, $C_2 = \{\{1,2\},\{1,3\},\{2,3\}\}$ and $C_3 = \{\{1,2,3\}\}$. Note that, except $\{1,2,3\}$ all other candidates are already generated and their support is known (see figure 4). On the contrary, this algorithm uses all information obtained so far and contained in the tree. It generates candidate itemset $X$ only if $X$ is not present in the tree and only in that case $suppcount_{T_2}(X)$ is calculated.

In our example, all candidates from $C_1$ are contained in the tree (level 1) as well as $suppcount_{T_2}$ and $suppcount_T$ for each of them (figure 10). An itemset $X$ is $\mu = \frac{2}{9}$-frequent in $T_2$ if it appears in at least $\lceil \mu \cdot |T_2| \rceil = \lceil 0.22... \cdot 5 \rceil = 2$ transactions. It means that $\mu$-frequent 1-itemsets in $T_2$ are $F_1 = \{\{1\},\{2\},\{3\}\}$. The only candidate in $C_2 = F_1 \times F_1$ that is not present in the tree is $\{1,3\}$. For it new path $NULL \to 1 \to 3$ is created and $suppcount_{T_2}$ is calculated. The set $F_2 = \{\{1,2\},\{1,3\},\{2,3\}\}$ is obtained. The set $C_3 = F_2 \times F_2 = \{\{1,2,3\}\}$ consists of one candidate for which new path $NULL \to 1 \to 2 \to 3$ is created. The algorithm counts $suppcount_{T_2}(\{1,2,3\}) = 2$ and generates $F_3 = \{\{1,2,3\}\}$. This is additionally illustrated in figure 11. With dash line presented are nodes that represent candidates created in this step. Above is indicated $suppcount_{T_2}$.

Note that Apriori Cross Validation Optimized generates and counts support for just two candidates while Apriori Two-Fold Cross Validation generates and counts support for eight candidates.

In the step 6, algorithm validates $\mu$-frequent itemsets in $T_2$. The partition $T_1$ is used to calculate $suppcount_T = suppcount_{T_2} + suppcount_{T_1}$. Note that $suppcount_{T_1}$ is calculated just for candidates generated in the previous step (they are presented with dash line in figure 11). This step is presented in figure 12; $suppcount_{T_1}$ is written with red font. Because both itemsets $\{1,3\}$ and $\{1,2,3\}$ have sufficient $suppcount_T$ there is no pruning of the tree.

Finally, all $\mu$-frequent itemsets in $T$ are contained in the tree from figure 12: $\mu$-frequent $k$-itemsets are represented by nodes at level $k$.

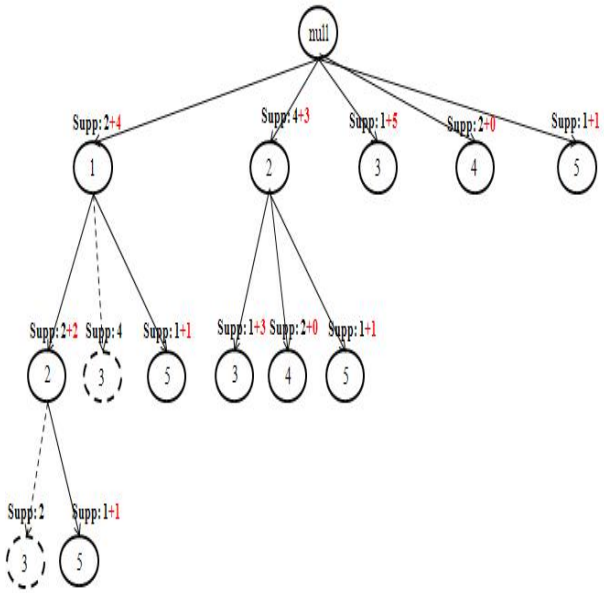


Figure 11: TS-tree for the second partition

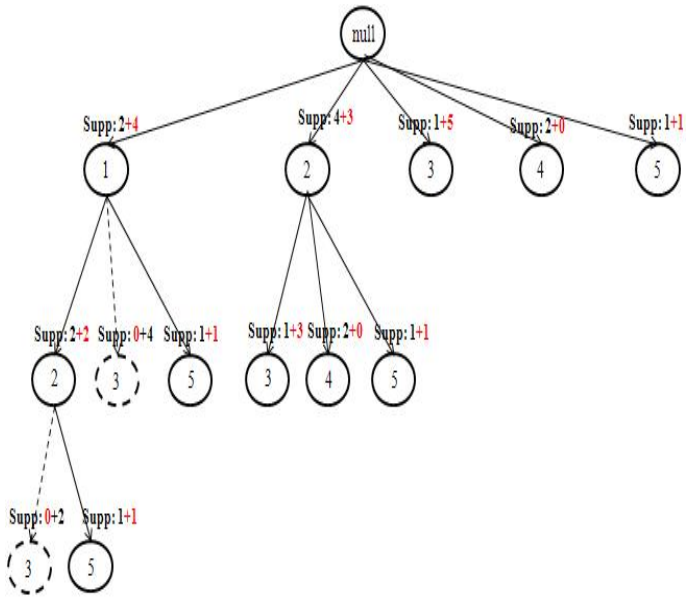


Figure 12: TS-tree with all frequent itemsets

# 4 Conclusion

In this paper we present new approach for mining frequent itemsets from database of transactions. It is based on modification of well-known Apriori algorithm from [4] and cross validation method.

We have implemented Apriori Cross Validation Optimized and one of the best modifications of the original Apriori algorithm from [7] (to the best of our knowledge). For experiments we used datasets T10I4D100K and T20I6D100K. Datasets contain 100000 transactions with 1000 items. Average size of transactions is 10 for T10I4D100K and 20 for T20I6D100K. Average size of the maximal potentially frequent itemset is 4 for T10I4D100K and 6 for T20I6D100K. We measured total execution time in seconds. The experiments showed that these algorithms are comparable. Actually, Apriori Cross Validation Optimized is better for higher values for *minsup*, while original Apriori is better for lower values, which can be seen from figure 13 and figure 14. On x-axis values for *minsup* (in percentages) are displayed. Execution time (in seconds) is displayed on y-axis.

As future work we plan to incorporate the following theorems in Apriori Cross Validation Optimal.

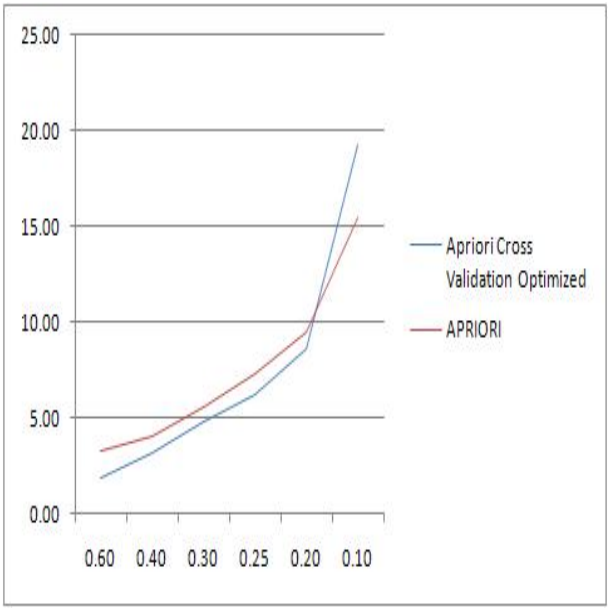


Figure 13: Execution time for Apriori Cross Validation and Apriori, dataset T10I4D100K

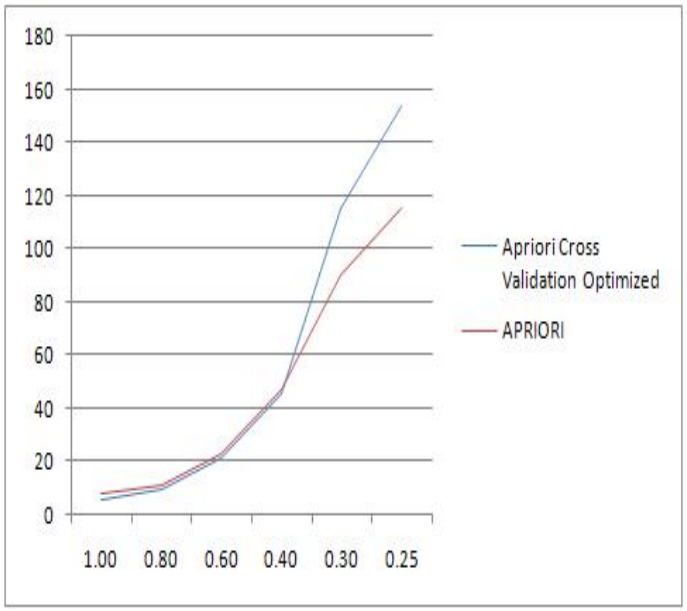


Figure 14: Execution time for Apriori Cross Validation and Apriori, dataset T20I6D100K

**Theorem 3.** Let transaction data sets $T_1, T_2$ on $I$ be a partitioning of transaction data set $T : \{1, ..., n\} \rightarrow P(I)$ as in definition 6. It is possible that an infrequent itemset $X$ in partition $T_1$ ($support_{T_1}(X) < \mu$) will become a frequent itemset in $T$ only if $support_{T_2}(X) > \mu$.

*Proof.* If an infrequent itemset $X$ in partition $T_1$ is to become frequent in the whole database, the following formula must hold:

$$\frac{suppcount_{T_1}(X) + suppcount_{T_2}(X)}{|T_1| + |T_2|} \geq \mu \qquad (1)$$

. That is, $\frac{|T_1| \cdot support_{T_1}(X) + |T_2| \cdot support_{T_2}(X)}{|T_1| + |T_2|} \geq \mu$. So, $|T_2| \cdot support_{T_2}(X) \geq |T_1| \cdot \mu + |T_2| \cdot \mu - |T_1| \cdot support_{T_1}(X)$. Or, $|T_2| \cdot (support_{T_2}(X) - \mu) \geq |T_1| \cdot (\mu - support_{T_1}(X))$. Because, $|T_2| > 0$, and $support_{T_1}(X) < \mu$ or $\mu - support_{T_1}(X) > 0$, the following condition must hold: $support_{T_2}(X) - \mu > 0 \Longleftrightarrow support_{T_2}(X) > \mu$. $\nabla$

**Theorem 4.** An infrequent itemset $X$ is kept if $support_{T_1}(X) \geq \mu + \frac{|T_2|}{|T_1|} \cdot (\mu - 1)$.

*Proof.* According to the previous theorem, an infrequent itemset in $T_1$ will become frequent in the whole database $T$ if equation (1) hold. The mini-

mum condition is:

$$\frac{|T_1| \cdot support_{T_1}(X) + |T_2| \cdot support_{T_2}(X)}{|T_1| + |T_2|} = \mu \quad (2)$$

So, $|T_1| \cdot support_{T_1}(X) = |T_1| \cdot \mu + |T_2| \cdot \mu - |T_2| \cdot support_{T_2}(X)$. Or, $support_{T_1}(X) = \mu + \frac{|T_2|}{|T_1|} \cdot (\mu - support_{T_2}(X)) > \mu + \frac{|T_2|}{|T_1|} \cdot (\mu - 1).\nabla$

The main idea we want to implement is in the following. Consider itemset $X$ that is infrequent in $T_1$ but frequent in the whole database. According to theorem 3 we have $support_{T_2}(X) > \mu$. After counting $support_{T_1}(X)$ in the step 3, algorithm from figure 8 will remove this itemset. Because $X$ is frequent in $T$, it will be generated again and $support_{T_2}(X)$ will be counted in the step 5 (figure 8). Additionally, in the step 6 $support_{T_1}(X)$ is counted again, although it has been counted in the step 3.

Instead of removing itemset $X$ in the step 3 if $support_{T_1}(X) < \mu$, the condition from theorem 4 should be checked. If it is not satisfied, itemset $X$ can be removed because it can not be frequent in $T$. On the other hand, if condition from theorem 4 is satisfied, itemset $X$ is kept because it is "promising" itemset. It remains to validate $X$ in the step 4, after which this itemset is not considered. We consider that it is possible to significantly reduce number of candidates and execution time in Apriori Cross Validation Optimized algorithm using this idea.

# References


[1] R. Agrawal, T. Imielinski, A. N. Swami, Mining association rules between sets of items in large databases, Proceedings of the ACM International Conference on Management of Data, pp. 207-216, 1993.

[2] P. Tan, M. Steinbach, V. Kumar, Intorduction to Data Mining, Addison Wesley, 2006.

[3] D. A. Simovici, C. Djeraba, Mathematical Tools for Data Mining - Set Theory, Partial Orders, Combinatorics, London, Springer, 2008.

[4] P. Stanišić, S. Tomović, A New Data Structure for Frequent Itemsets Mining: TS-tree, Proceedings of the 4th South East European Doctoral Student Conference, 2009.

[5] P. Stanišić, S. Tomović, A New Rymon Tree Based Procedure for Mining Statistically Significant Frequent Itemsets, International Journal of Computers Communications & Control, Vol. 5(4), pp. 567-577, 2010.

[6] S. Tomović, P. Stanišić, Mining the Most k-Frequent Itemsets with TS-tree, Proceedings of the IADIS International Conference WWW/Internet 2009, 2009.

[7] F. Bodon, A Fast Apriori Implementation, IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), Melbourne, Florida, USA, 2003.