A Fuzzy Close Algorithm for Mining Fuzzy Association Rules

Régis Pierrard, Jean-Philippe Poli, Céline Hudelot

A Fuzzy Close Algorithm for Mining Fuzzy Association Rules

Régis Pierrard[1,2], Jean-Philippe Poli[1], and Céline Hudelot[2]

[1] *CEA, LIST, Data Analysis and System Intelligence Laboratory, 91191 Gif-sur-Yvette cedex, France.*
[2] *Mathematics Interacting with Computer Science, CentraleSupélec, Paris-Saclay University, 91190, Gif-sur-Yvette, France.*
regis.pierrard@cea.fr, jean-philippe.poli@cea.fr, celine.hudelot@centralesupelec.fr

Abstract

Association rules allow to mine large datasets to automatically discover relations between variables. In order to take into account both qualitative and quantitative variables, fuzzy logic has been applied and many association rule extraction algorithms have been fuzzified.

In this paper, we propose a fuzzy adaptation of the well-known Close algorithm which relies on the closure of itemsets. The Close-algorithm needs less passes over the dataset and is suitable when variables are correlated. The algorithm is then compared to other on public datasets.

1 Introduction

Extracting association rules from data has been one of the main tasks in data mining for years. It relies on the extraction of frequent itemsets. In order to deal with both quantitative and qualitative variables, some algorithms have used the fuzzy set theory. Fuzzy logic provides tools to manage the vagueness inherent in both the natural language and the knowledge itself. Different fuzzy association rule mining algorithms have already been developed to handle this kind of data.

Because datasets are nowadays getting bigger and bigger, the way these fuzzy association rule mining algorithms manage huge databases is essential. Some algorithms store a big amount of data while some others need to perform many database passes.

There exist several crisp association rule mining algorithms that do not store a lot of data or need only a limited number of database passes. However, most of them do not have a fuzzy counterpart. In this paper, we propose an algorithm that uses the fuzzy set theory and the fuzzified version of the Close mining algorithm [1] to extract frequent itemsets from data with a reduced number of database passes.

The rest of the paper is organized as follows. Section 2 reviews related work. In section 3, we present the fuzzy set framework and we describe the algorithm. Section 4 presents the experimental results we got and section 5 concludes the paper.

2 Related Works

2.1 Fuzzy Association Rule Mining

The first fuzzy association rule mining algorithms were based on the Apriori algorithm [2]. It consists in two main steps. First, finding the frequent itemsets and second, generating

fuzzy rules based on the previously extracted frequent itemsets. In order to find the frequent itemsets, it first scans the whole database to extract frequent itemsets that contain only one item (1-itemsets). An itemset is said to be frequent when the support of this itemset in the database, i.e. the number of occurences, is larger than a user-specified minimum support threshold. After that first step, frequent 1-itemsets are used to generate candidate 2-itemsets. Frequent 2-itemsets are extracted computing their support. The process continues until no more candidate can be generated. It requires n database passes, where n is the size of the maximum length frequent itemset. Once frequent itemsets have been mined, every candidate association rule is generated. An association rule is valid when its confidence is larger than a user-specified minimum confidence threshold. For a frequent itemset I and an association rule $I_1 \Rightarrow I_2$ such as $I_1 \subset I$, $I_2 \subset I$ and $I_1 \cap I_2 = \varnothing$, the confidence of this association rule is its number of occurrences among the occurrences of I. All candidate association rules are generated to find the most confident ones. Many fuzzy association rule mining algorithms rely on the Apriori algorithm. The F-APACS algorithm [3] first converts data into linguistic terms using the fuzzy set theory. A statistical analysis is performed to automatically set both the minimum support threshold and the minimum confidence threshold. The FDTA algorithm [4] proposes another way of converting quantitative data into linguistic terms. Kuok et al. [5] proposed a different approach to handle quantitative databases for generating fuzzy association rules.

A completely different way of mining fuzzy frequent itemsets relies on a frequent-pattern tree structure. The generic framework is as follows. The first step consists in fuzzifying data, if necessary. Then, the tree is constructed and the final step is the mining of fuzzy frequent itemsets based on the previously constructed tree. Papadimitriou et al. [6] proposed an algorithm called fuzzy frequent pattern tree (FFPT). Non frequent 1-itemsets are removed from the database and each transaction is sorted according to the membership value of its frequent 1-itemsets. Then, the tree is constructed by handling each transaction one by one. Since transactions are sorted by membership values, several different paths may represent the same itemset. As a consequence, a few useless tree nodes are generated. The compressed fuzzy frequent pattern tree (CFFPT) algorithm solves this problem by using a global sorting strategy [7]. However, this solution leads to attaching an array to each node. Lin et al. [8] proposed the upper bound fuzzy frequent pattern tree algorithm (UBFFPT). It estimates the upper bound membership values of frequent itemsets to avoid attaching an array to each node. This algorithm requires four database passes to build the tree. Then, the tree is parsed several times to generate all candidate frequent itemsets. Depending on the database, the tree can be long and have a large amount of nodes. An ultimate database pass is performed to compute the support of every candidate frequent itemset.

2.2 The Close Algorithm

Pasquier et al. [1] proposed the Close algorithm. This algorithm handles non-fuzzy databases. It uses a closure operator to find closed itemsets. Those itemsets have interesting properties that benefit the mining of frequent itemsets. Since there are often less frequent closed itemsets than frequent itemsets, the search space is smaller, the computation is less costly and the number of database passes is reduced. The algorithm relies on the following properties [1]:

1. all subsets of a frequent itemset are frequent;
2. all supersets of an infrequent itemset are infrequent;
3. all closed subsets of a frequent closed itemset are frequent;
4. all closed supersets of an infrequent closed itemset are infrequent;

5. the set of maximal frequent itemsets is identical to the set of maximal frequent closed itemsets;

6. the support of a frequent itemset I which is not closed is equal to the support of the smallest frequent closed itemset containing I.

The algorithm goes through three phases to generate association rules. First, it generates all frequent closed itemsets from the database. Then, it derives all frequent itemsets from the previously generated frequent closed itemsets. The final step consists in generating all confident association rules.

3 Fuzzified Close Algorithm

3.1 Fuzzy Sets

Zadeh introduced the fuzzy set theory [9]. In a universe X, a fuzzy set F is characterized by a mapping $\mu_F : X \to [0, 1]$. This mapping specifies in what extent each $x \in X$ belongs to F and it is called *the membership function of* F. If F is a non-fuzzy set, $\mu_F(x)$ is either 0, i.e. x is not a member of F, or 1, i.e. x is a member of F.

The *kernel* of a fuzzy set F is a non-fuzzy set defined as

$$\ker(F) = \{x \in X | \mu_F(x) = 1\} \ . \tag{1}$$

A binary fuzzy relation can be defined the same way as a fuzzy set. Given two universes X and Y, a binary fuzzy relation $\mathcal{R}$ is a mapping defined as

$$\mathcal{R} : X \times Y \to [0, 1] \ . \tag{2}$$

It assigns a degree of relationship to any $(x, y) \in X \times Y$.

3.2 Formal Concept Analysis

Formal Concept Analysis (FCA) [10, 11, 12] provides a framework for analyzing the relationship between a set of objects and a set of attributes. A database with fuzzy values can be represented by a triplet $\langle \mathcal{O}, \mathcal{A}, \mathcal{R} \rangle$ with $\mathcal{O}$ a finite set of objects, $\mathcal{A}$ a finite set of attributes and $\mathcal{R}$ a binary fuzzy relation defined as $\mathcal{R} : \mathcal{O} \times \mathcal{A} \to [0, 1]$. This triplet is called a formal fuzzy context.

Operators $\uparrow$ and $\downarrow$ can then be defined [13]. Let X be a fuzzy set of objects and Y be a fuzzy set of attributes. $\uparrow$ and $\downarrow$ are defined as follows:

$$\forall a \in \mathcal{A}, \mu_{X^\uparrow}(a) = \bigwedge_{o \in O} \big(\mu_X(o) \to \mathcal{R}(o, a)\big) \ , \tag{3}$$

$$\forall o \in \mathcal{O}, \mu_{Y^\downarrow}(o) = \bigwedge_{a \in A} \big(\mu_Y(a) \to \mathcal{R}(o, a)\big) \ . \tag{4}$$

$X^\uparrow$ is a fuzzy set of attributes and $Y^\downarrow$ is a fuzzy set of objects.

We use the Lukasiewicz implication operator defined as

$$a \to b = \min(1 - a + b, 1) \ . \tag{5}$$

The Lukasiewicz implication is compatible with the implication from classical logic.

3.3 Closure Operator

The closure operator cannot be the same in the fuzzified version of the algorithm. It still takes as an argument a crisp set, which we call a *generator*, and also returns a crisp set. However, the relation $\mathcal{R}$ between objects and attributes is no longer crisp. That is why this operator needs to be modified.

Definition 3.1. Let E be a set and $\mathcal{P}(E)$ its power set. A closure operator on E is defined as $h : \mathcal{P}(E) \to \mathcal{P}(E)$ and satisfies the following conditions:

$$\forall X \subset \mathcal{P}(E), X \subset h(X) \ , \tag{6}$$

$$\forall X \subset \mathcal{P}(E), h(h(X)) = h(X) \ , \tag{7}$$

$$\forall X, Y \subset \mathcal{P}(E), X \subset Y \Rightarrow h(X) \subset h(Y) \ . \tag{8}$$

A fuzzy closure operator is defined the same way. For any formal fuzzy context $\langle \mathcal{O}, \mathcal{A}, \mathcal{R} \rangle$, for a fuzzy set of attributes Y, $\uparrow\downarrow$ is a fuzzy closure operator [12, 14]. The fuzzy closure of Y by $\uparrow\downarrow$ is $Y^{\uparrow\downarrow}$, which is a fuzzy set of attributes.

In our case, the closure operator takes a crisp set of items (or attributes) as a generator. Let I be a crisp set of items. It can be turn into a fuzzy set as follows:

$$\forall a \in \mathcal{A}, \mu_I(a) = \begin{cases} 1, & \text{if } a \in I \\ 0, & \text{otherwise} \end{cases} \ . \tag{9}$$

As for the set the closure operators returns, it also has to be a crisp set. The fuzzy closure operator $\uparrow\downarrow$ returns a fuzzy set F. We can get a crisp set of items I using the kernel function as follows:

$$I = \ker(F) \ . \tag{10}$$

This operator is still a closure operator. In the following, this closure operator is written h such as $h : \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$. One can interpret the result of this closure operator as the set of attributes that are shared by all the objects that have all the attributes from the generator.

The algorithm also relies on the fact that a generator and its closure have the same support.

Proposition 1. $\forall I \in \mathcal{P}(\mathcal{A}), \text{support}(h(I)) = \text{support}(I)$
with $\text{support}(I) = \sum_{o \in \mathcal{O}} (\min_{a \in I} \mathcal{R}(o, a)) \quad .$

3.4 Algorithm Description

The proposed fuzzy association rule mining approach integrates concepts from both the fuzzy set theory and the Close algorithm [1]. It does not tackle the fuzzification of the database. This task has been addressed in the previously mentioned articles [3, 4, 5]. Besides, the generation of all confident association rules is the same as in the Apriori algorithm [2].

FCC_i refers to the set of triplets associated with all the frequent closed candidate itemsets whose generator's size is i. FC_i refers to the set of triplet associated with all the frequent closed itemsets whose generator's size is i. Each triplet is under the following form:

$$\text{(generator, closure, support)} \quad .$$

Thus, in the remainder of this article, for any $p \in FCC_i$ or FC_i, p.generator refers to the generator linked to p, p.closure is its closure and p.support is its support. FCC_i.generators refers to the set of all generators in FCC_i. FCC_i.closures and FCC_i.supports are defined the same way.

Algorithm 1 below describes the process. On line 1, FCC_1 is initialized with every item from the set of attributes $\mathcal{A}$. On line 5, for each generator in FCC_i, the *generateClosures* function provides the corresponding closure and support. This function is detailed below. Then, on lines 6 to 9, the set of candidate closed itemsets FCC_i is pruned to get the set of frequent closed itemsets FC_i. New generators, whose size is $i+1$, are generated on line 11 using the *genrateGenerators* function. This function is described below. The whole process will last until no new generators can be generated. The output is the set of all frequent closed itemsets that will be used to generate all frequent itemsets.

The *generateClosures* function is stated as shown in Algorithm 2 below. This function has been designed to compute the closures and the supports of the generators in FCC_i performing only one database pass. For each object $o \in \mathcal{O}$, for each element $p \in FCC_i$, the contribution k to the support and $\mu_{p^{\downarrow}}(o)$ are computed looping over the items in p.generator (from line 10 to line 13). Then, for each attribute $a \in \mathcal{A}$, the membership function $\mu_{p^{\uparrow\downarrow}}$ of the fuzzy closure is updated (line 15). When the last object is reached and there is no more update to the membership function, the kernel of the fuzzy closure is computed (from line 16 to line 20).

This *generateGenerators* is exactly the same as in the Close algorithm. This function generates all the potential generators of size $i+1$ from the generators in FC_i. In order to get one potential generator, two generators from FC_i that have the same $i-1$ first elements are combined. Then, this set of potential generators is pruned to avoid useless computations. In particular, if one of the new generators is included in the closure of one of the former generators, then it is pruned.

Overall, the whole algorithm, i.e. Algorithm 1, needs one database pass per iteration. That is the same as the algorithms based on the Apriori algorithm. However, the total number of iterations is usually smaller with the close algorithm because there are often less frequent closed itemsets than frequent itemsets.

After this phase, all the frequent closed itemsets are used to find all the frequent itemsets. This new phase is exactly the same as in the original Close algorithm. The first step consists in splitting the set of all frequent closed itemsets according to their size. Then, these new sets L_i are browsed in descending order of size to generate all frequent itemsets of size $i-1$. The process will finish when the set of frequent 1-itemsets is completed.

3.5 Example

For the sake of comprehension, we apply in this section the algorithm on a small database D, shown in Table 1. D contains five objects (1 to 5) and five items (A to E). The minimum support is equal to 0.4 (40%).

The pruning of FCC_1 leads to removing {B} since its support is smaller than the minimum support threshold. The other elements from FCC_1 are kept to generate FC_1. This corresponds to line 5 to line 10 in Algorithm 1. FCC_1 and FC_1 are shown in Table 2.

Then, on line 11, FCC_2 is generated. $\{AD\}$ is not a generator in FCC_2 because it is included in the closure of $\{A\}$. FC_2 is then generated. $\{CD\}$ and $\{AC\}$ have the same closure, so only one of them is kept. FCC_2 and FC_2 are shown in Table 3.

FC_2 contains only one element, that is why FCC_3 is empty. That is the end of the first phase, which corresponds to algorithm 1. FC is returned. It is shown in Table 4.

The second phase consists in deriving frequent itemsets from frequent closed itemsets. The longest closed itemset contains three items. That is why three different sets are

Table 1: The fuzzy database D

Objects	Items				
	A	B	C	D	E
1	0.8	0.1	0.9	0.8	0
2	0	0.3	0.2	0	0.9
3	1	0.7	0.7	1	0.6
4	0	0.2	0	0.2	1
5	0.9	0.6	0.8	1	0.9

generated for deriving frequent itemsets: L_3, L_2 et L_1. Bold itemsets are itemsets which have been derived from a bigger closed itemset. These three sets are shown in Table 5.

4 Experimental Results

In order to compare our algorithm to the fuzzy version of Apriori and to UBFFPT, we have implemented these algorithms. As our implementations of the algorithms may not be fully optimized, our results do not show any execution time. The metric that we used is the number of database passes. It allows to directly compare the fuzzy version of Apriori to our algorithm.

4.1 Datasets

We used three different datasets. The first one is the *mushroom* dataset [15]. It contains 8124 examples (objects). The number of attributes is 22. Those are all categorical attributes, so the final binary dataset contains 119 attributes. To fuzzify it, zeros were replace by a uniform random number in $[0, 0.5[$ and ones were replace by a uniform random number in $[0.5, 1]$.

The two other datasets come from the 2017 Civil Service People Survey [16]. Those are surveys that only contain numbers in $[0, 1]$. One dataset, that is called *benchmark scores*, contains 9 examples. Attributes have been pruned to avoid missing values for a final amount of 87 attributes. The other dataset is called *all organisation scores*. After filtering missing values, the dataset contains 93 examples and 84 attributes.

4.2 Results and Discussion

Results are shown in Fig.1. For the mushroom dataset, we can observe that our algorithm makes at best one less database pass than the fuzzy version of Apriori. This is due to the fact that data are not highly correlated and are sparse. That means that most frequent itemsets are closed. As a consequence, with the cost of computing closures, our algorithm should not be expected to outperform Apriori and UBFFPT on such a dataset.

Observations are different with the two other datasets. We can see that the lower the minimum support threshold, the larger the difference between the number of database passes of both algorithms. These data come from surveys, whose data are usually highly correlated and dense. Our algorithm takes advantage of this using the closure operator. Thus, most generators are much shorter than their closures. That explains the lower amount of database passes.

The UBFFPT algorithm needs 4 database passes to construct its tree and to extract frequent itemsets. Besides, frequent pattern mining algorithms, such as UBFFPT, spend most of their time traversing the tree. For highly correlated data, as in the benchmark

Algorithm 1: Close algorithm

input : A fuzzy formal context $\langle \mathcal{O}, \mathcal{A}, \mathcal{R} \rangle$
A minimum support threshold $S \in [0; 1]$
output: All frequent closed itemsets and their support

generators in $FCC_1 \leftarrow \{1\text{-itemsets}\}$
for $(i \leftarrow 1;\ FCC_i.\text{generators} \neq \varnothing;\ i{+}{+})$ **do**
  closures in $FCC_i \leftarrow \varnothing$
  supports in $FCC_i \leftarrow 0$
  $FCC_i \leftarrow \text{generateClosures}(FCC_i)$
  forall candidate closed itemsets $c \in FCC_i$ **do**
    if $c.\text{support} \geq minsupport$ **then**
      $FC_i \leftarrow FC_i \cup \{c\}$
    end
  end
  $FCC_{i+1} \leftarrow \text{generateGenerators}(FC_i)$
end
$FC \leftarrow \bigcup_{j=1}^{i-1} \{FC_j.\text{closures}, FC_j.\text{supports}\}$
return FC

Table 2: FCC_1 on the left and FC_1 on the right. $\{B\}$ is pruned from FCC_1 to FC_1 because it is not frequent.

Generator	Closure	Support
{A}	{AD}	2.7
{B}	{B}	1.9
{C}	{C}	2.6
{D}	{D}	3
{E}	{E}	3.4

Generator	Closure	Support
{A}	{AD}	2.7
{C}	{C}	2.6
{D}	{D}	3
{E}	{E}	3.4

Table 3: FCC_2 on the left and FC_2 on the right.

Generator	Closure	Support
{AC}	{ACD}	2.3
{AE}	{ADE}	1.5
{CD}	{ACD}	2.3
{CE}	{CE}	1.6
{DE}	{DE}	1.7

Generator	Closure	Support
{AC}	{ACD}	2.3

Table 4: FC

Closure	Support
{AD}	2.7
{C}	2.6
{D}	3
{E}	3.4
{ACD}	2.3

Algorithm 2: generateClosures function

input : The set of candidate closed itemsets FCC_i
output: Updated FCC_i after the computation of closures and supports

$n \leftarrow 0$
forall $p \in FCC_i$ **do**
  numbers in $\mu_{p^{\uparrow\downarrow}}$ [a] $\leftarrow 1$
end
forall objects $o \in \mathcal{O}$ **do**
  n++
  forall $p \in FCC_i$ **do**
    $k \leftarrow 1$
    $\mu_{p^{\downarrow}}$ [b] $\leftarrow 1$
    forall attributes $i \in p$.generator **do**
      $k \leftarrow \min\left(k, \mathcal{R}(o,i)\right)$
      $\mu_{p^{\downarrow}} \leftarrow \min\left(\mu_{p^{\downarrow}}, 1, \mathcal{R}(o,i)\right)$
    end
    forall attributes $i \in \mathcal{A}$ **do**
      $\mu_{p^{\uparrow\downarrow},i} \leftarrow \min\left(\mu_{p^{\uparrow\downarrow},i}, 1, 1 + \mathcal{R}(o,i) - \mu_{p^{\downarrow}}\right)$
      if $n = Card(\mathcal{O})$ **then**
        if $\mu_{p^{\uparrow\downarrow},i} = 1$ **then**
          p.closure $\leftarrow p$.closure $\cup \{i\}$
        end
      end
    end
    p.support $\leftarrow p$.support $+ k$
  end
end
return FCC_i

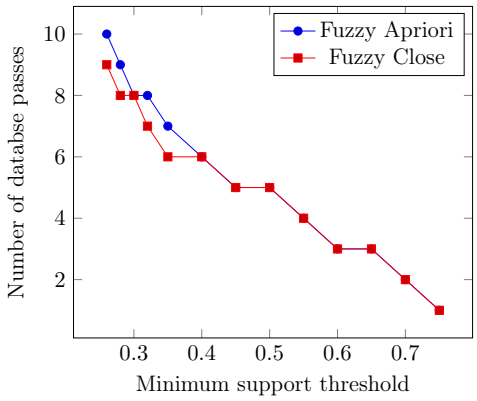
[a] $\mu_{p^{\uparrow\downarrow}}$ is a vector corresponding to the membership function of the fuzzy closure $p^{\uparrow\downarrow}$.
[b] $\mu_{p^{\downarrow}}$ is a fuzzy number that corresponds to $\mu_{p^{\downarrow}}(o)$.

Table 5: Deriving frequent itemsets. Bold lines refer to derived itemsets. From left to right: L_3, L_2 and L_1.

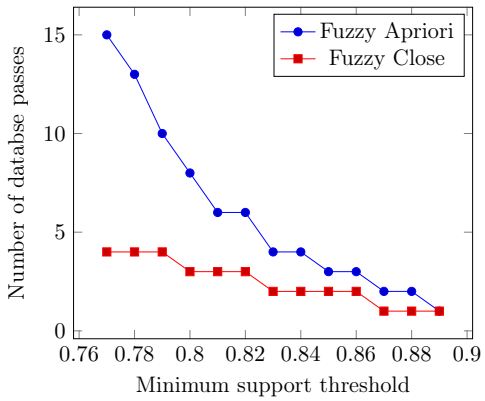
Itemset	Support
{ACD}	2.3

Itemset	Support
{AD}	2.7
{AC}	**2.3**
{CD}	**2.3**

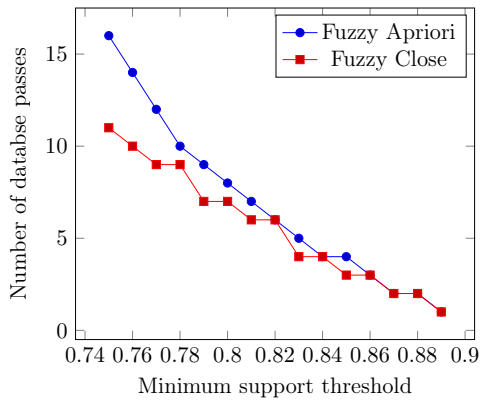
Itemset	Support
{C}	2.6
{D}	3
{E}	3.4
{A}	**2.7**



(a) Mushroom dataset



(b) Benchmark dataset



(c) All organisation scores dataset

Figure 1: Plots showing the number of database passes relatively to the minimum support threshold for the three datasets.

dataset, our algorithm has an edge on these algorithms. Moreover, it consumes less memory than Apriori, which generates many candidates at each iteration, and than UBFFPT, which browses all the paths to the currently studied item[1] to generate candidates.

Also, the first iteration of generating closures in our algorithm can bring valuable insight. Indeed, if most 1-itemsets are closed, then the data is likely to be weakly correlated and another algorithm may perform better. However, if the proportion of closed 1-itemsets is low, the data is likely to be highly correlated and our algorithm will then compute all the frequent itemsets in few database passes.

5 Conclusion

In this paper, we introduced a new fuzzy association rule mining algorithm inspired by the Close algorithm. Our goal was to make it able to mine frequent itemsets from data in a reduced number of database passes and without storing too much data.

It relies on a closure operator that is able to process fuzzy data while both taking as an argument and returning a crisp set. This new closure operator is based on a fuzzy closure operator of whom we take the kernel. The closure is the set of items that are shared by all the objects that include the generator. That is why it is very efficient with highly correlated data.

The algorithm finds the set of all the closed frequent itemsets. This set is sufficient to extract all the frequent itemsets. As it is usually a smaller set than the set of all the frequent itemsets, the search space is also smaller.

We have tested our algorithm on three different datasets. We have shown that this approach outperforms other algorithms when dealing with correlated and dense data, which are the kind of data that can be found in surveys, census dataset or in some classification datasets. It needs less database passes and stores a small amount of data to extract all the frequent itemsets.

References

[1] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient mining of association rules using closed itemset lattices. Information Systems, 24, 1, 25-46 (1999)

[2] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases, pages 487-499 (1994)

[3] Chan, K.C.C., Au, W.H.: An Effective Algorithm for Discovering Fuzzy Rules in Relational Databases. Proceedings of the 1998 IEEE International Conference on Fuzzy Systems, pages 1314-1319 (1998)

[4] Hong, T.P., Kuo, C.S., Chi, S.C.: Mining association rules from quantitative data. Intelligent Data Analysis, 3, pages 363-376 (1999)

[5] Kuok, C.M., Fu, A., Wong, M.H.: Mining Fuzzy Association Rules in Databases. SIGMOD Rec., 27 (1), pages 41-46 (1998)

[6] Papadimitriou, S., Mavroudi, S.: The Fuzzy Frequent Pattern Tree. Proceedings of the 9th WSEAS International Conference on Computers, pages 3:1-3:7 (2005)

[7] Lin, C.W., Hong, T.P., Lu, W.H.: An efficient tree-based fuzzy data mining approach. International Journal of Fuzzy Systems, 12, 2, pages 150-157 (2010)

[1] One item is usually represented by several nodes in the tree.

[8] Lin, C.W., Hong, T.P., Lu, W.H.: A Two-Phase Fuzzy Mining Approach. International Conference on Fuzzy Systems (2010)

[9] Zadeh, L.A.: Fuzzy sets Information and Control, vol. 8, no. 3, pp. 338-353 (1965)

[10] Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. Oredered Sets, I. Rival, Ed. Dordrecht, Reidel, pp. 445-470 (1982)

[11] Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical foundations. Berlin, Germany: Springer-Verlag (1999)

[12] Belohlavek, R.: Fuzzy Relational Systems: Foundations and Principles. New York: Kluwer/Plenum (2002)

[13] Belohlavek, R.: Concept lattices and order in fuzzy logic. Annals of Pure and Applied Logic, vol. 128, no. 1, pp. 277-298 (2004)

[14] Gerla, G.: Fuzzy Logic. Mathematical Tools for Approximate Reasoning. Dordrecht, The Netherlands: Kluwer (2001)

[15] Lichman, M.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2013)

[16] Contains public sector information licensed under the Open Government Licence v3.0 (2017)