Mining Association Rules for Classification Using Frequent Generator Itemsets in arules Package

Makhlouf Ledmi (ledmi.makhlouf@univ-khenchela.dz)
Abbas Laghrour University of Khenchela

Mohammed El Habib Souidi
Abbas Laghrour University of Khenchela

Michael Hahsler
Vienna University of Economics and Business

Chafia Kara-Mohamed
Ferhat Abbas University of Setif 1

Abdeldjalil Ledmi
Abbas Laghrour University of Khenchela

Research Article

Keywords: frequent generator itemsets, Classification, Association Rules, Data mining, R

Posted Date: May 17th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1650542/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
Mining Association Rules for Classification Using Frequent Generator Itemsets in arules Package

Makhlfouf Ledmi\textsuperscript{*}, Mohammed El Habib Souidi\textsuperscript{†}, Michael Hahsler\textsuperscript{‡}, Chafia Kara-Mohamed\textsuperscript{‡†} and Abdeldjalil Ledmi\textsuperscript{††}

\textsuperscript{*}Department of Computer Sciences - ICOSI Lab, Abbas Laghour University of Khenchela, 40000, Khenchela, Algeria.
\textsuperscript{‡}Institut fur Informationswirtschaft, Wirtschaftsuniversität Wien, Augasse 2–6, 1090, Wien, Austria.
\textsuperscript{‡†}Department of Computer Science - LRSD Lab, Ferhat Abbas University of Setif 1, 19000, Setif, Algeria.

\textsuperscript{*}Corresponding author(s). E-mail(s): ledmi.makhlfouf@univ-khenchela.dz;
Contributing authors: souidi.mohammed@univ-khenchela.dz;
Michael.Hahsler@wu-wien.ac.at; chafia.kara-mohamed@univ-setif.dz;
abdeldjalil.ledmi@univ-khenchela.dz;
\textsuperscript{††}These authors contributed equally to this work.

Abstract
Mining frequent itemsets is an attractive research activity in data mining whose main aim is to provide useful relationships among data. Consequently, several open-source development platforms are continuously developed to facilitate the users’ exploitation of new data mining tasks. Among these platforms, the R language is one of the most popular tools. In this paper, we defend the usefulness of providing a new version of arules package that can mine frequent generator itemsets, we describe the possibility of mining frequent generator itemsets in arules package, and we discuss what generators can be used for a classification task through an application example.

Keywords: frequent generator itemsets, Classification, Association Rules, Data mining, R

1 Introduction
In recent years, data mining [5] has been defined as a field of computer science where different techniques such as machine learning methods are utilized to find out previously unknown properties in large data sets [23]. Formally, data mining main goal is to analyze data sets in order to discover interesting, useful, and new patterns, models, and relationships. Data mining tasks take into consideration techniques combining artificial intelligence [2], machine learning [20], fuzzy systems [11], statistics [22], as well as mathematics [24]. The global practical objective of a data mining task is therefore to extract information from a data set and its transformation into a comprehensible structure to be reused.

Frequent pattern mining [15] is considered as a focused area in data mining research for many
years. These patterns are usually represented by itemsets, which can be defined as unordered sets of distinct items. An itemset is said to be frequent if it appears a number of times at least equal to a given minimum support in the transaction database. The support of an itemset is defined by the number of transactions containing this itemset. Among the different types of frequent itemsets, we can cite the generator frequent itemsets [6]. A generator can be defined as an itemset \( X \) such that there does not exist another itemset strictly included in \( X \) and having an identical support [9].

To mine frequent patterns, three primary methodologies are usually used, Apriori [3], FP-Growth [8], Eclat [28], as well as their extensions [4, 19]. Agrawal et al. [3] proposed an attractive downward closure property, known as Apriori. This methodology is based on the principle that a given \( k \)-itemset is frequent only when its sub-itemsets are also frequent. This fact involves that frequent itemsets can be extracted through a first database scanning in order to discover the frequent 1-itemsets. After that, the frequent 1-itemsets are utilised to generate frequent candidate 2-itemsets and verify against the database to get the frequent 2-itemsets. This process is repeated until no more frequent \( k \)-itemsets can be generated.

In comparison with frequent generator itemsets, frequent closed itemsets [18] are considered as the maximal itemsets among the itemsets that appear in the same set transactions’ set. On the other hand, the generators are the itemsets with the minimal apparition. The mining of frequent closed itemsets has been well studied, and many useful algorithms have been introduced for mining this kind of itemsets [25]. However, the development of algorithms for mining frequent generator itemsets got less attention from researchers in this field. In several application cases, generators are more appreciable in comparison with closed itemsets. In terms of classification, generators are more adequate than closed itemsets, knowing that closed itemsets include some repeated itemsets that are considered unusable for classification. Furthermore, these itemsets do not respect the minimum description length principle.

In this paper, we describe the possibility of mining frequent generator itemsets in \texttt{arules} package \footnote{https://cran.r-project.org/web/packages/arules/}, and we show how generators can help in solving a classification task. This new extension of \texttt{arules} package is also used in \texttt{R} language [21] in order to address the users’ needs regarding the lack of mining frequent generator itemsets through this platform.

The rest of the paper is organized as follows: Section 2 discusses the main related works regarding mining the frequent itemsets using different methodologies. Section 3 describes the frequent generator itemsets mining problem by giving specific definitions concerning the processed concepts. In section 4, we describe how generator itemsets are used in \texttt{arules} package. Section 5 discusses what generators can be used for a classification task through an application example. Finally, in section 6, we conclude this paper by showcasing the utility points regarding the proposed work.

## 2 Related work

Since the apparition of the primary methodologies aiming to mine frequent patterns, improved extensions of these lasts were proposed in the literature. After analyzing the first version of the apriori algorithm [3], the authors in [27] have found that it has two major limits. The first limit is that this method is based on a frequent scanning of the database. The second limit concerns the fact that this method generates a consequent number of candidate sets. Based on these limits, Yuan [27] proposed the use of a novel database mapping in order to escape the redundancy of database scanning. The aim is to improve joining efficiency and further prune candidates. Finally, they introduced an overlap strategy to calculate support for reaching high efficiency.

In [26], another ameliorated Apriori algorithm based on the innovative idea of weight matrix was presented. The algorithm builds a 0–1 transaction matrix through scanning a transaction database, in the case where the rows and columns of the matrix correspond to the transactions and items, respectively. The weight of items and transaction is done to showcase the importance in the transaction database. Finally, the frequent itemsets are
obtained via the computation of the items’ weight support.

Furthermore, Chang and Liu [10] have introduced a novel algorithm for mining all frequent itemsets, known as APRIORI-IMPROVE algorithm. This apriori extension algorithm is based on reading the transaction database through only one scanning and without generating candidate sets. Specifically, APRIORI-IMPROVE utilizes hash structure with the aim of generating $L_2$. Moreover, it is based on an efficient horizontal data representation as well as optimized strategy of storage in order to reduce time and space.

In relation to Apriori algorithm that mines frequent patterns from a set of transactions presented in horizontal data format, Eclat algorithm or Equivalence CLAss Transformation algorithm [28] performs frequent patterns’ mining with data presented in vertical data format. The main objective of Eclat algorithm is to deal with Transaction Id Sets (tidssets) intersections to calculate the support value of a candidate and to escape the subsets’ generation, knowing that, the avoided subsets do not exist in the prefix tree. At the first iteration of the algorithm, all single items are employed along with their specific tidsets. At the other recursive calls of the algorithm, each item-tidset pair is checked and mixed with other item-tidset pairs. This process does not stop until no candidate item-tidset pairs can be mixed.

In the same context of Apriori algorithm, interesting extensions and improvements regarding Eclat algorithm was proposed. Knowing that the inconvenient of Eclat algorithm is the exponential time complexity concerning the computation of the transactions’ intersection. BloomEclat [1] based on Bloom filter was introduced in order to enhance the Eclat algorithm complexity in discovering frequent itemsets. In other words, to improve intersection and union in the Eclat algorithm, they put the members of the first list into the Bloom Filter and verified if each second-list member is a part of the Bloom filter or not. In comparison with normal eclat algorithm, BloomEclat can intersect transactions with a minimal error rate in less time.

Regarding mining frequent generator itemsets (FGIs), Ledini et al [17] confirmed that the process of mining Frequent Closed Itemsets (FCIs) as well as mining FGIs unified with their full-support and extension items is an equivalent operation.

Based on this fact, they proposed a generator-based algorithm called GrAFCI + with the aim of mining FCIs. The experiment results proved that the proposed GrAFCI + is more efficient concerning the execution time in the majority of cases in comparison with recent algorithms. Knowing that GrAFCI + principal objective is to improve the runtime issue, it consumes a large memory part, especially in the case where the support is too small. However, this consumption has no great consequences since GrAFCI + is seconded by only one competitor out of four in memory use and for large support values.

In the same context [16], the authors introduced an efficient bit-vector method based on the exploitation of dual properties, tidset and superset information of an itemset with the aim of mining FCIs with their lattice structure. Specifically, they proposed a new memory efficient data structure known as dynamic superset bit-vector in order to develop the relationship among FCIs in a lattice. The innovative contribution of this approach is the efficient utilization of dual data structures knows as a dynamic bit-vector and a dynamic superset bit-vector jointly with the aim of reducing the search space and removing the generators involving non-closed itemsets.

### 3 Mining frequent generator itemsets

Consider a set of items $I = \{a_1, a_2, \ldots, a_M\}$ and a transaction database $T = \{t_1, t_2, \ldots, t_N\}$, where $t_i \in \{1, N\} \subseteq I$ is a transaction. Each subset of $I$ is called an itemset. A $k$-itemset is an itemset that contains $k$ items. The support of an itemset $x$ in $D$ is defined as the number of transactions where it appears, i.e., $supp(x) = |\{t| t \in D, x \subseteq t\}|$.

Given a prescribed minimum support threshold, denoted as $\text{min}\_s$

**Definition 3.1 (frequent itemset)**

An itemset $x$ is called frequent itemset if its support is no less than $\text{min}\_s$, i.e.,

$x$ is frequent itemset $\iff$ supp($x$) $\geq \text{min}\_s$

Table 1 shows an example of a database containing seven transactions. With $\text{min}\_s = 2$. There are 83 frequent itemsets such as $\{a\}$, $\{b\}$,
\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Transaction ID & Items \\
\hline
\(t_1\) & \(a, b, c, d, e, g\) \\
\(t_2\) & \(a, b, c, d, e, f\) \\
\(t_3\) & \(a, b, c, d, e, h, i\) \\
\(t_4\) & \(a, b, d, e\) \\
\(t_5\) & \(a, c, d, e, h, i\) \\
\(t_6\) & \(b, c, e, i\) \\
\(t_7\) & \(d, h\) \\
\hline
\end{tabular}
\caption{Transaction database}
\end{table}

\(\min_s = 2\)

\(\{a, b, c\}\), for example, \(\{f\}\) and \(\{g\}\) are not frequent since its support is less than \(\min_s\).

Like [18] and [17], we define a frequent generator itemset and the anti-monotone property of generators below.

**Definition 3.2 (generator itemset)**
An itemset \(x\) is a generator itemset if none of its subsets has a support equal to the support of \(x\). In other words:
\[x\text{ is a generator itemset } \iff (\exists x' / x' \subseteq x \text{ and supp}(x') = \text{supp}(x))\]

**Definition 3.3 (frequent generator itemset)**
An itemset \(x\) is a frequent generator if it is a generator whose support is not less than \(\min_s\). In other words:
\[x\text{ is a frequent generator itemset } \iff (x \text{ is generator } / \text{supp}(x) \geq \min_s)\]

Consequently from definition 3.3, the empty set \(\emptyset\) is a frequent generator in any database.

**Example:** From database of Table 1 and with \(\min_s = 2\): \(\emptyset\), \(\{b\}\), \(\{c\}\), \(\{b\}\), \(\{b, c\}\) are frequent generator itemsets. \(\{d\}\) is not generator since it has the same support as \(\emptyset\), \(\{a, b, e\}\) is not generator since it has the same support as its subset \(\{a, b\}\).

**Definition 3.4 (anti-monotone property of generators)**
If itemset \(x\) is not a generator, then none of its supersets is a generator. In other words:
\[(x \text{ is not a generator} ) \Rightarrow (\forall x' \supset x, x' \text{ is not a generator})\]

Conversely, if only one of the subsets of \(x\) is not a generator, then \(x\) cannot be a generator.

The Apriori algorithm [3] uses the depth-first search strategy for mining frequent generator itemsets in the same way as frequent itemsets. Algorithm 1, which we called Apriori\(_G\), shows a standard Apriori version adapted for mining frequent generator itemsets.

First, the Apriori\(_G\) algorithm mines all frequent items in the database whose support is less than the number of database transactions and puts them in \(FG_1\). Then, according to the anti-monotone property of generators, the Apriori\(_G\) algorithm prunes the generated itemsets of size \(k \geq 2\) in two ways: the first one, there is at least a non-generator subset of the generated itemset. The second way is if the candidate itemset is infrequent or has the same support as at least one of its immediate subsets. The process of mining frequent generator itemsets ends if there is no itemset to be generated.

### 4 Mining frequent generator itemsets in arules package

The arules package [14] for R provides the essential infrastructure for mining association rules by including a simple and intuitive interface that manipulates and analyzes rules, sets of itemsets, and transaction data. The package also provides an interface to two fast mining algorithms: Apriori and Eclat, based on efficient C implementations of Christian Borgelt’s [7], which includes some improvements (e.g., a prefix tree and item sorting). The algorithms are used to mine different kinds of itemsets, such as frequent itemsets, maximal, and closed frequent itemsets. Moreover, the implementation of Apriori can be used to generate association rules [13].

Several open-source platforms are developed for mining frequent generator itemsets in different languages such as Java [12], Python (PyFIM²) and C (Apriori³, FP-growth⁴, and Eclat⁵). However, there are only very few implementations available in language R such as (FIM4R⁶). Moreover, there is no package in R that supports the mining of frequent generator itemsets at the time of writing these lines. For these reasons, we proposed to integrate the mining frequent generator itemsets as a

²https://borgelt.net/pyfim.html
³https://borgelt.net/apriori.html
⁴https://borgelt.net/fgrowth.html
⁵https://borgelt.net/eclat.html
⁶https://borgelt.net/eclat.html
Algorithm 1 The standart Apriori adapted version for mining frequent generator itemsets

Require: Dataset $D$, minimum support threshold $min_s$ 

Ensure: $FG$: frequent generator itemsets

1: $FG_1 \leftarrow$ frequent generator 1-itemsets 
2: $k \leftarrow 2$
3: while $FG_{k-1} \neq \emptyset$ do 
4: \hspace{1em} $C_k \leftarrow \text{apriori}_\text{gen}(FG_{k-1})$
5: \hspace{1.5em} while $t \in D$ do 
6: \hspace{2.5em} $C_t \leftarrow \text{subset}(C_k, t)$ 
7: \hspace{2em} for candidate $c \in C_t$ do 
8: \hspace{3em} $c$.count ++ 
9: \hspace{1.5em} end for 
10: \hspace{1em} end while 
11: $FG_k = \{c \in C_k | (c$.count $\geq min_s \land (\exists s \in \text{subset}(c).c$.count $= s$.count))\}$ 
12: $k++$
13: end while 
14: $FG \leftarrow \bigcup_k FG_k$

15: function $\text{apriori}_\text{gen}(FG_{k-1}) \supset FG_{k-1}$: frequent generator itemsets of size $(k-1)$ 
16: \hspace{1em} $C_k \leftarrow \emptyset$
17: \hspace{1em} for itemset $l_1 \in FG_{k-1}$ do 
18: \hspace{2em} for itemset $l_2 \in FG_{k-1}$ do 
19: \hspace{3em} if $(l_1[1] = l_2[1]) \land \ldots \land (l_1[k-2] = l_2[k-2]) \land (l_1[k-1] < l_2[k-1])$ then 
20: \hspace{4em} $c \leftarrow l_1 \times l_2$
21: \hspace{4em} if is_generator($c, FG_{k-1}$) then 
22: \hspace{5em} Add $c$ to $C_k$
23: \hspace{4em} end if 
24: \hspace{3em} end if 
25: \hspace{2em} end for 
26: \hspace{1em} end for 
27: \hspace{1em} return $C_k$
28: end function

19: function is_generator($c, FG_{k-1}$) \supset c: candidate itemset of size $(k)$ 
20: \hspace{1em} for $(k-1)$-subset $s \in c$ \supset all subsets of size $(k-1)$ must be generators 
21: \hspace{2em} if $s \notin FG_{k-1}$ then 
22: \hspace{3em} return $false$
23: \hspace{2em} end if 
24: \hspace{1em} end for 
25: \hspace{1em} return $true$
26: end function

The frequent generator itemsets obtained can be inspected using inspect().

feature in the arules package ($< 1.7 - 3$) instead of developing a new package in order to take advantage of benefits offered by the arules package and ensuring the mining of frequent generator itemsets with other kinds of frequent itemsets in a coherent and comparable way.

In this example, we show how frequent generator itemsets can be mined with the new version of arules package. First, we load arules and testthat to compare the results obtained by the two interfaces.

```r
> install.packages("arules")
> install.packages("testthat")
> library("arules")
> library("testthat")
```

Then, we create a transactions list for data given in Table 1.

```r
> data <- list(c("a", "b", "c", "d", "e", "g"), c("a", "b", "c", "d", "e", "f"), c("a", "b", "c", "d", "e", "h", "i"), c("a", "b", "c", "d", "e", "h", "i"), c("a", "b", "c", "d", "e", "h", "i"), c("a", "b", "c", "d", "e", "h", "i")
> names(data) <- paste("Tr", c(1:7), sep = "")
> trans <- transactions(data)
```

Next, we perform the apriori() function by initializing the target parameter with one of these values: "generator", "generator itemset" or "frequent generator itemset".

```r
> is_a_gen <- apriori(trans, parameter = list(target = "generator", supp = 0.2))
```

The same is true with Eclat interface, we perform the eclat() function by initializing the target parameter with one of these values: "generator", "generator itemset" or "frequent generator itemset".

```r
> is_a_gen <- eclat(trans, parameter = list(target = "generator", supp = 0.2))
```

The frequent generator itemsets obtained can be inspected using inspect().

```r
> inspect(is_a_gen)
```

<table>
<thead>
<tr>
<th>items</th>
<th>support</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{h}</td>
<td>0.43</td>
<td>3</td>
</tr>
<tr>
<td>{i}</td>
<td>0.43</td>
<td>3</td>
</tr>
<tr>
<td>{b}</td>
<td>0.71</td>
<td>5</td>
</tr>
<tr>
<td>{c}</td>
<td>0.71</td>
<td>5</td>
</tr>
<tr>
<td>{a}</td>
<td>0.71</td>
<td>5</td>
</tr>
<tr>
<td>{d}</td>
<td>0.86</td>
<td>6</td>
</tr>
<tr>
<td>{e}</td>
<td>0.86</td>
<td>6</td>
</tr>
<tr>
<td>{h, i}</td>
<td>0.29</td>
<td>2</td>
</tr>
<tr>
<td>{c, h}</td>
<td>0.29</td>
<td>2</td>
</tr>
<tr>
<td>{a, h}</td>
<td>0.29</td>
<td>2</td>
</tr>
<tr>
<td>{s, h}</td>
<td>0.29</td>
<td>2</td>
</tr>
</tbody>
</table>
First, we load the packages \texttt{arules}, \texttt{arulesCBA}, and the COVID dataset to create transactions:

\begin{verbatim}
> install.packages("arules")
> install.packages("arulesCBA")
> library(arules)
> library(arulesCBA)
> df <- read.csv("Covid Dataset.csv", header = TRUE, sep = ",", stringsAsFactors = TRUE)
> trans <- as(df, "transactions")
\end{verbatim}

Next, we find all frequent generator itemsets, and we use rule induction to produce all generator association rules that surpass the specified minimum confidence threshold:

\begin{verbatim}
> Supp=0.5; Conf=0.95;
> gen_is <- apriori(trans, target = "generator", support = Supp, maxlen=100);
> cl_rules <- ruleInduction(gen_is, transactions = trans, confidence = Conf);
> cl_rules <- sort(cl_rules, by = "confidence");
\end{verbatim}

Finally, we easily create an association rule-based classifier from generator association rules using the package \texttt{arulesCBA}:

\begin{verbatim}
> c1 <- CBA_ruleset(COVID.19 ~ ., cl_rules);
> prediction <- predict(c1, trans);
> tab_cl<--table(prediction, response(COVID.19 ~ ., trans));
\end{verbatim}

To make a comparison with frequent closed itemsets, we repeated the same process by varying the support minimum threshold ($Min_{Supp}$) from 0.05 to 0.5 and the confidence minimum threshold ($Min_{Conf}$) from 0.7 to 0.95 for both generators and closed itemsets.

Figure 1, Figure 2 and Figure 3 show the number of rules, the number of classification-based associations (CBA), and the classification accuracy obtained by each method for all $Min_{Supp}$ and $Min_{Conf}$ values, respectively. Furthermore, Figure 4, Figure 5 and Figure 6 show a comparison between the performance measures of rules obtained with FGIs and FCIs in terms of support, confidence, and rule size, respectively. The results obtained show that the two methods achieve similar performances in terms of accuracy, support and confidence in most cases especially when $Min_{Supp}$ ranges from 15\% to 40\%. However, the number of rules obtained from frequent generator itemsets (FGIs) is significantly less than the number of rules obtained from frequent closed itemsets (FCIs). The same goes for the rule size and number of classification-based associations (CBA). For

5 Classification application example using generators

In this example, we show how to create rules from frequent generator itemsets, demonstrating that generators are more appreciable than closed itemsets. To accomplish that, we used the COVID-19 dataset available from Kaggle entitled “Symptoms and COVID Presence” \footnote{https://www.kaggle.com/hemanthhari/symptoms-and-covid-presence}. This dataset has 20 attributes that highlight the presence of various symptoms related to virus infection, and the class attribute, i.e. whether the person has COVID or not.

[12] \{b, i\} 0.29 2
[13] \{a, i\} 0.29 2
[14] \{d, i\} 0.29 2
[15] \{b, c\} 0.57 4
[16] \{a, b\} 0.57 4
[17] \{b, d\} 0.57 4
[18] \{a, c\} 0.57 4
[19] \{c, d\} 0.57 4
[20] \{d, e\} 0.71 5
[21] \{a, b, c\} 0.43 3
[22] \{b, c, d\} 0.43 3

\begin{verbatim}
> inspect(is_e_gen)
\end{verbatim}

\begin{verbatim}
> expect_true(setequal(is_a_gen, is_e_gen))
\end{verbatim}

Finally, The result obtained by the two interfaces can be compared with each other by \texttt{expect_true()}. It can be observed that the two sets \texttt{is_a_gen} and \texttt{is_e_gen} are equals.

\footnote{https://cran.r-project.org/web/packages/arulesCBA/}
example, with $Min_{Conf} = 95\%$ and $Min_{Supp} = 15\%$, the number of rules obtained from FGI is 843 which is $\approx 6\%$ of rules obtained from FCIs, and the number of CBA obtained from FGI is 825 which constitutes $\approx 58\%$ of CBA obtained from FCIs as shown in Figure 2f. The average size of CBA obtained from FGI is 5 compared with 6.65 for the average size of CBA obtained from FCIs as shown in Figure 6f.

Based on the experimental results, the rules obtained from FGI are more predictive on independent test data and therefore can better tolerate noise errors than the rules obtained from FCIs. This is due to the number of items to satisfy, which is less than the number of items in rules obtained from FCIs, in the case of a true test.

6 Summary

In this paper, we reviewed the mining of frequent generator itemsets. While many algorithms are proposed for mining frequent generator itemsets, there are only very few implementations available in R language. We have presented in this paper the opportunity to discover this kind of patterns through the improvement of the arules package. Furthermore, we have discussed the usefulness of generators in terms of the classification task. This new contribution, as well as the given examples, will certainly be beneficial to users regarding the frequent generator itemsets further investigation and improvement.

Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Research involving Human Participants and/or Animals

Not applicable

Informed consent

Not applicable

Funding

This research has not received any funding supports.

Author contributions

Makhlouf Ledmi, Mohammed El Habib Soudi, Michael Hahsler, Chafia Kara-Mohamed, and Abdeldjalil Ledmi contributed to writing, review, and editing. Makhlouf Ledmi and Michael Hahsler contributed to implementation. Mohammed El Habib Soudi and Makhlouf Ledmi contributed to supervision.

References


Fig. 1: Comparison between the number of rules obtained with FGIs and FCIs

Fig. 2: Comparison between the number of CBAs obtained with FGIs and FCIs
Fig. 3: Comparison between the accuracy of rules obtained with FGIs and FCIs

Fig. 4: Comparison between the performance of support measure of rules obtained with FGIs and FCIs
Fig. 5: Comparison between the performance of confidence measures of rules obtained with FGIls and FCIs.

Fig. 6: Comparison between the performance of size measure of rules obtained with FGIls and FCIs.
ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90, p 90


