TAR-IMF: Temporal Association Rule Mining and Improved Algorithm for Mining Frequent Elements

M. Sornalakshmi*, S. Balamurali, M. Venkatesulu, M. Navaneethakrishnan
Department of Computer Applications, Kalasalingam University, Krishnan-koil 626106, Tamilnadu, India
sorna.jesus@gmail.com*, sbmurali@rediffmail.com, venkatesulum2000@gmail.com

Abstract. Data mining involves the discovery of interesting patterns from the huge dataset for increasing the profit of the business enterprises. Association Rule Mining (ARM) is the most prevalent technique used in the field of data mining. It is used for mining the high frequent itemsets in the large dataset. The algorithms such as Apriori algorithm and FP-tree algorithm are developed for mining the frequent patterns. However, the performance of these algorithms reduces slowly with the increase in the number of total attributes in the dataset. By focusing on the drawbacks in ARM algorithms, a new method for mining the temporal association rules is developed and an improved algorithm for finding frequent elements over a sliding window. The Rough Set Theory (RST) is combined with the parallel computing technology to generate an attribute reduction based temporal association rule mining (ARTAR) algorithm that generates temporal association rules. The Classification Based on Association (CBA) classifier is used to classify the normal and abnormal items. The proposed TAR-IMF algorithm accomplishes a significant reduction in the number of items in each transaction, execution time, memory usage, number of datasets and better dataset utilization rate than the S-RWARM and SemanticRuleScorer algorithm. The throughput of the proposed TAR-IMF is higher than the LH and MC schema.

Keywords: Association Rule Mining (ARM), ARTAR, Classification Based on Association (CBA) Classifier, Frequent Pattern Mining, Rough Set Theory.

1 Introduction

Due to the increase in the computerization and usage of Internet services, the business enterprises collect and store a huge amount of data in the routine operations [1]. Hence, there is a need for mining the valuable information from a huge amount of data [2]. The data mining approaches enable to analyze large dataset and find the interesting data patterns from the dataset. Hence, the data mining is encouraged as a decision support problem for any business enterprises [3]. The data mining procedures are characterized as descriptive or predictive procedures [4]. The descriptive mining procedure finds the general characteristics of data in the dataset. The predictive data mining procedures execute the interpretation on the input data for discover-
ing the useful hidden knowledge for future prediction [5]. Therefore, the data mining approaches should be capable to the size of dataset [6]. The temporal semantic information in the real-world data should be considered during the knowledge discovery process. ARM [7] is an important technology used to discover the hidden frequent patterns from the transaction datasets. As the rules with temporal constraints describe the objective knowledge in a better way, they are found to be valuable [8].

Initially, the dataset with time series has non-linear and irregular features that generate a huge number of candidate itemsets and extended sub-itemset. This increased the effort of excavating the temporal rules. The temporal rules are mined in the large candidate itemsets with multi-attribute charges to generate an exponential number of sub-itemsets. Hence, the creation and storage of the frequent itemsets should be handled before mining of the temporal rules. Then, there is a need for detailed knowledge about setting the appropriate time constraint and integrating the constraint into the mining process.

In this paper, the ARTAR is applied for discovering the regularities among the temporal data. The number of items included in each transaction is reduced and redundancy items are deleted using an attribute reduction method based on the RST. An improved algorithm for finding the frequent items over a sliding window is presented. The CBA classification is applied to classify the normal and abnormal items in the input dataset. The main structure of the algorithm comprises only few maps and short queues. The overall running time is minimized by using the hash table and data structure to manage the counters. From the experimental results, it is observed that the proposed work requires minimum execution time, memory usage, number of datasets and better dataset utilization rate and throughput than the existing algorithms.

The remaining sections of the paper are organized in the following way: Section II describes the existing techniques for mining the frequent patterns. Section III explains the proposed method for mining the temporal association rules and an improved algorithm for finding frequent elements over a sliding window and Section IV illustrates the performance evaluation results including the comparative analysis of the proposed TAR-IMF algorithm with the existing algorithms. The proposed TAR-IMF work is concluded in Section V.

2 Existing Frequent Pattern Mining Algorithms

Elseidy et al. [9] presented an extension of the novel frequent subgraph mining framework for mining the frequent patterns. The proposed framework enabled faster discovery of interesting patterns than the existing approaches. Gee et al. [10] proposed a novel algorithm for mining the maximal frequent patterns that replicates the recent information about the data streams based on the sliding window model. The proposed mining algorithm achieved high scalability and lower runtime and memory usage. Pyun et al. [11] applied a novel Linear Prefix (LP) Tree for mining the frequent patterns. The LP tree structure comprises array forms and reduces the pointers between the nodes in the tree. Due to the reduction in the number of pointers, the proposed techniques required minimum runtime and memory consumption. The scala-
Proposed TAR-IMF Algorithm

Fig. 1 shows the overall flow diagram of the proposed work. Initially, data preprocessing is performed to remove the irrelevant and redundant items from the database. An improved algorithm is applied for mining the frequent items over the sliding window. The ARTAR algorithm is applied for mining the temporal association rules. Then, the temporal association rules are applied as input to the classifier to classify the normal and abnormal data in the dataset.

3.1 Preprocessing

Data preprocessing is performed in three stages such as data cleaning, selection and transformation. Data cleaning is the process of removing inaccurate items from a transaction database. The missing values in the bone marrow dataset are eliminated. The cleaned data is considered as input for the data selection. The attributes are selected for the generation of association rules and selected for further process. Then, the data is transformed into an appropriate form used for the mining process. The ranges of the attribute values are used to find whether the value is low, normal or high.
3.2 Identification of frequent items

Managed Counter

A managed counter is used to count the occurrence frequency of items in sliding windows. When an item slides during the existence of counter, the counter will be updated. Else, a new counter is created with initial value of zero and updated. If there are more than $1/\xi$ counters whose value is greater than 0, each counter will be decreased once. A dynamic hash table is used to access each counter in $O(1)$ time. In one query, the values of all counters are calculated and the counter whose value greater than $\theta - \xi n$ is returned [19].

The main components of a counter are double queue $Q$ and a variable $l$ for storing a $\lambda$-snapshot. Two auxiliary variables such as $curblk$ and $offset$ are introduced to maintain the track of the $\lambda$-blocks. The $curblk$ represents the index of the current blocks and $offset$ denotes the number of 1-bit in the current block after the
last sampled 1-bit. Some of the elements expire, when the sliding window is shifted. When a block is expired, some counters maintain an index of the expired block. These counters are to be updated on the expired block. A multi key value map is used for managing the counters. The head of the queue is the key of the map. The value of the map is a reference of the counter. Initially, the double queue and map are empty. The value of \( l \) and auxiliary variable \( curblk \) are equal to zero. The offset is equal to \( \lambda - 1 \). The C.shift() algorithm is executed when an item slides in the sliding window. When there are more than \( 1/\xi \) counters, the decrement operation should be performed. Suppose the counter \( C \) has executed some shift operations, the value of \( C \) is not zero. The double queue and \( l \) can only be modified. The algorithm for shift and decrement operation is described as follows:

**C.shift() Algorithm**

```python
offset ← (offset + 1) mod \lambda
if offset = 0
   curblk ← curblk + 1
   exprblk ← curblk - (n/\lambda)
if map.contains(exprblk)
   for all counter ∈ map.get(exprblk)
      if counter.Q.size > 0 then
         map.put(head(counter.Q), counter)
      end if
   end for
map.remove(exprblk)
end if
for all length ← (length + 1) mod \lambda
if length = 0
   if Q.size = 0 then
      map.put(curblk, C)
   end if
   Q.add(curblk)
end if
```

**C.decrement() Algorithm**

```python
if l > 0 then
   l ← l - 1
else
   pop_tail(Q)
   l ← \lambda - 1
end if
```

According to the \( \xi \)-approximate frequent items problem, a set ‘S’ of items should be produced. Let \( \lambda = \xi n/\delta \) and \( S = e_1 e_2 e_3 \ldots \ldots \) be the stream of items and \( f_e = b_1 b_2 b_3 \ldots \ldots \) be the bit stream for any \( i \geq 1 \), if \( e_i = e \), then \( b_i = 1 \). Otherwise, \( b_i = 0 \).
Thus, in the stream ‘S’ for any \( e_i \in E \), an adapted counter \( C_e \) is used to count the bit stream. The update and query operations are performed for managing the counter schema.

**Update() Algorithm**

```plaintext
C_e.shift()
if counters.size > 4/ξ
    forall counter ∈ counters do
        counter.decrement()
    end for
end if
```

**Query() Algorithm**

```plaintext
forall \( e \in E \) do
    if \( v(C_e) - 2λ \geq (θ - ξ)n \)
        output \( v(C_e) - 2λ \)
    endif
end for
```

To count the items in \( f = e_1 e_2 e_3 \ldots \) the stream is fed into the sliding window. The update operation is executed, during sliding of each item. At any time \( t \), after executing a sequence of update operations \( update(e_1), update(e_2), \ldots \), if the items in the sliding window \( W_t \) is required. The query() is executed to obtain the set of items and their frequencies [19].

**Selection of Important attributes**

Let \( A = \{a_1, a_2, \ldots, a_n\} \) is a collection of attributes in the databank DB and \( P = \{p_1, p_2, \ldots, p_n\} \) are important attributes that are selected from the DB. It is mathematically written as follows [20]

\[
P = \{p_1, p_2, \ldots, p_n\}(DB)
\]

This will reduce the computing time for finding the frequent patterns, as the significant frequent itemsets can be found out using this logic.

**Dataset size reduction**

Normally, there are duplicate transactions in the dataset for finding frequent mining. These transactions are combined as a single transaction with the count. This process reduces the number of transactions in the dataset less than or equal to \( 2^p - 1 \) transactions. Here, ‘\( p \)’ represents the total number of unique attributes in the databank. This is done in the pre-processing stage. The combined transactions are stored in the Transaction Joining Table (TJT) and memory. This reduces the computing time for finding frequent itemsets.

**Frequency Gathering Table (FGT)**
The FGT table has two fields that store the combinations of all itemsets for the selected attributes having corresponding direct in the dataset. This is stored in the memory until the total frequencies of all itemsets are computed \[20\].

**Proposed IMF Algorithm**

Input: A database DB and Least Support Value (LSV)
Output: Frequent itemsets FS

Begin

Step 1: Read the important attributes stored in a file.
Step 2: Store these attributes as a set.
Step 3: Construct TJT with two fields such as itemset and count.
Step 4: Read the transactions in raw database one by one until it reaches the end of transaction in the file.

Step 5: 

Step 6: Convert transaction as set.
Step 7: Remove the unwanted attributes in the transaction by comparing with the selected attributes set.
Step 8: If the new transaction entry is available in the TJT

Step 9: 

Step 10: Update the corresponding entry in TJT.
Step 11: }
Step 12: Else
Step 13: 
Step 14: Insert the new transaction entry to the TJT.
Step 15: }

Step 16: Construct the FGT with two fields for storing itemset and recording its direct occurrence count in the transaction.
Step 17: Find all combination for selective attributes and insert those with initial value 0 for finding direct occurrence of each itemset

Step 18: Read TJT and update matching entry in FGT
Step 19: \(FS = \emptyset\)

Step 20: For \((m = 1; m < 2^i; m + +)\)  
//Each itemset \(Y_m\) in FGT, perform the following steps

Step 21: 

Step 22: \(AC_{ym} = 0\)
Step 23: For \((n = m + 1; n < 2^i; n + +)\)
Step 24: 

Step 25: If \(Y_m \subseteq Y_n\)

Step 26: 

Step 27: \(AC_{ym} = AC_{ym} + FGT \cdot FGC(n)\)
Step 28: }
Step 29: \} //end of inner loop

Step 30: If \((AC_{ym} \geq LSV)\)
Step 31: 

Step 32: \(FS = FS \cup Y_m\)
Step 33: }

1959
Step 34: }//end of outer loop
Step 35: End

Consider a dataset with ten transactions, minimum support value is 4, itemset \( I = \{ A1, B1, C1, D1, E1 \} \) and important attributes \( IM = \{ A1, B1, C1 \} \). Table I shows the transaction set.

<table>
<thead>
<tr>
<th>Tid</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>( A1, B1, C1, D1, E1 )</td>
</tr>
<tr>
<td>T2</td>
<td>( B1, C1, E1 )</td>
</tr>
<tr>
<td>T3</td>
<td>( D1, E1 )</td>
</tr>
<tr>
<td>T4</td>
<td>( C1, D1, E1 )</td>
</tr>
<tr>
<td>T5</td>
<td>( A1, C1, D1 )</td>
</tr>
<tr>
<td>T6</td>
<td>( A1, B1 )</td>
</tr>
<tr>
<td>T7</td>
<td>( A1, B1, C1 )</td>
</tr>
<tr>
<td>T8</td>
<td>( A1, B1, E1 )</td>
</tr>
<tr>
<td>T9</td>
<td>( B1, C1, D1, E1 )</td>
</tr>
<tr>
<td>T10</td>
<td>( A1, B1, C1, D1, E1 )</td>
</tr>
</tbody>
</table>

Table 1. Transaction set.

The proposed method scans the transactions one by one and removes unwanted attributes. The reduced transaction is stored in TJT. Table II shows the dataset after removing unwanted attributes and Table III illustrates the combined identical transactions.

Table 2. Dataset after removing unwanted attributes.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>( A1, B1, C1, D1, E1 )</td>
</tr>
<tr>
<td>T2</td>
<td>( B1, C1 )</td>
</tr>
<tr>
<td>T3</td>
<td>-</td>
</tr>
<tr>
<td>T4</td>
<td>( C1 )</td>
</tr>
<tr>
<td>T5</td>
<td>( A1, C1 )</td>
</tr>
<tr>
<td>T6</td>
<td>( A1, B1 )</td>
</tr>
<tr>
<td>T7</td>
<td>( A1, B1, C1 )</td>
</tr>
<tr>
<td>T8</td>
<td>( A1, B1 )</td>
</tr>
<tr>
<td>T9</td>
<td>( B1, C1 )</td>
</tr>
<tr>
<td>T10</td>
<td>( A1, B1, C1 )</td>
</tr>
</tbody>
</table>

Table 2. Dataset after removing unwanted attributes.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( A1, C1 )</td>
</tr>
<tr>
<td>2</td>
<td>( B1, C1 )</td>
</tr>
<tr>
<td>3</td>
<td>( C1 )</td>
</tr>
<tr>
<td>4</td>
<td>( B1, C1 )</td>
</tr>
<tr>
<td>5</td>
<td>( A1, B1, C1 )</td>
</tr>
</tbody>
</table>

Table 3. Combined identical transactions.
Table 4. FGT.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Itemset</th>
<th>FGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>B1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>C1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>A1, B1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>A1, C1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>B1, C1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>A1, B1, C1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5. Actual support value for each itemset.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Itemset</th>
<th>Frequency Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>B1</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>C1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>A1, B1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>A1, C1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>B1, C1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>A1, B1, C1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table V proves that the execution time of the proposed technique is lesser than the FP-tree. The frequent itemsets for a given set of transactions as per the entries in the actual support value table are

\[ FS = \{\{A1\}, \{B1\}, \{C1\}, \{A1, B1\}, \{A1, C1\}, \{A1, B1, C1\}\}. \]

Attribute reduction

The reduced attributes is a subset of attributes that are adequate and required to maintain a specific property of the information table. Rough set is an approximation process that is defined using the current attributes. The approximation is described in the terms of the lower approximation and upper approximation of the original dataset.

The information system is defined as follows [8]

\[ IS = (U, R, V_\alpha | \alpha \in At, f) \]

Where \( E(R) = \{(x, y) \in U \times U | \forall \alpha \in R, f(x, \alpha) = f(y, \alpha) \} \) is represented as an equivalence relation. In the actual application, for the dataset with huge information, the reductive information table should be decided. The significance of each attribute in the table is calculated. Therefore, the calculation process is simplified.

Itemset Reduction

Each core of the itemset in the transaction must be computed and added to the reduction set. Finally, the minimum reduction set is obtained. The algorithm for itemset reduction is described as follows
Temporal association rule

While generating the temporal association rule, the TFP-growth is used to find the frequent itemsets that satisfy the minimum support for a specific time. The attribute reduction procedure is described below.

**Itemset reduction algorithm**

Input: Dataset $D = \{T_1, T_2, \ldots, T_n\}$ and dependency threshold $\alpha$
Output: Reduction itemset Red;
Step 1: Initialize $core(I) = \emptyset$
Step 2: Calculate $POS_{t'-t}(d)$
Step 3: Delete $I$, when $I \in D$
Step 4: if $\left( POS_{t'-t}(d) \neq POS_{t}(d) \right)$
Step 5: $core(I) = core(I) + \{I\}$
Step 6: end if
Step 7: else
Step 8: Calculate dependency $k_{R(d)}$
Step 9: if $k_{R(d)} \geq \alpha$
Step 10: $Red = core(I)$
Step 11: else for each $I, I \in I - Red$
Step 12: get the maximum dependency
Step 13: end for
Step 14: end if
Step 15: Output Red;

**Attribute reduction procedure**

Input: Reduction itemset Red and time constraint $[\text{min}_T, \text{min}_t]$
Output: Temporal association mset $\cup R$
Step 1: Scan the reduction itemset and generate the matrix of support count $\mathcal{V}_r$ according to each element in $T'_r$
Step 2: Initialize a node $null(I)$ as root node of pattern tree and insert $I$ to $PT$ in the descending order. Obtain the pattern at the top ‘l’ having similar length as the length of frequent pattern.
Step 3: Obtain the candidate itemsets with all length by executing $\mathcal{G}_k = F_{k-1} \land F_{k-1}$.
Step 4: Run above steps iteratively until all the candidate itemsets are found.
Step 5: Add the frequent pattern $F_k$ to the set of temporal frequent patterns when $T_p, T_q$ in $F_k$ satisfy the constraint $T_p \geq \text{min}_T$ and $T_q \geq \text{min}_t$. Else, the frequent pattern will be deleted.
Step 6: Calculate all weighted confidence of $i \rightarrow j$, where $i, j \in (1, n)$ in each frequent pattern, if $\text{Conf} (i \rightarrow j) = \frac{\text{sup}(\{i\}) \cdot \text{sup}(\{j\})}{\text{sup}(\{i, j\})} \geq \text{min}_\text{conf}$, so $R = \cup R + [i \rightarrow j]$. Or else, the frequent pattern will be deleted [8].
3.3 CBA Classification

The CBA algorithm is built using the temporal association rules. A total order on the generated rules is defined before presenting the algorithm [21].

Definition: If two temporal association rules \( t_i \) and \( t_j \) are given, \( t_i > t_j \)
1. The rule \( t_i \) has higher priority than \( t_j \).
2. The confidence of the rule \( t_i \) is greater than \( t_j \).
3. The confidence of the rules \( t_i \) and \( t_j \) is same, but the support of the rule \( t_i \) is greater than \( t_j \).
4. The confidence and support of the rules \( t_i \) and \( t_j \) are same, but the rule \( t_i \) is generated earlier than \( t_j \).

Let ‘\( T \)’ be the set of temporal association rules and ‘\( D \)’ be the training data. The basic notion of the classification algorithm is to select a set of high priority rules. The classifier is described in the format as mentioned below:
\[
(t_{i1}, t_{i2}, \ldots, t_{in}, \text{default class})
\]
Where \( t_i \in T, t_a > t_b \) if \( b > a \). The steps for building the classifier is given as follows:

Step 1: Sort the set of rules ‘\( T \)’ according to the priority relationship. This enables the selection of high priority rules for the classifier.

Step 2: Select the rules from \( T \) for the classifier by following the sorted sequence. There is a need to go through the database to find the cases covered by the rule ‘\( T \)’. The rule is marked if it classifies the case ‘\( d \)’ correctly. \( d.id \) is the unique identification number of ‘\( d \)’. If the rule ‘\( T \)’ can classify at least one case correctly, it is a potential rule in the classifier. The cases covered by these rules are removed from the training data. A default class is also chosen as majority class in the remaining data. This indicates that if selection of more rules for the classifier ‘\( C \)’ is stopped, this class will remain as the default class of the classifier. The total number of errors committed by the current classifier and default class is computed and recorded. This is the sum of number of errors committed by all selected rules in the classifier and default class in the training data. The rule selection process is completed, when there is no rule. The rules that do not increase the accuracy of the classifier are discarded. The first rule with the minimum number of errors recorded on the data is called as the cutoff rule. All the rules after the first rule can be rejected as they create more errors. The non-rejected rules and default class of the last rule form the classifier.
This algorithm satisfies two main conditions

- Each training case is covered by the rule with highest priority among the rules that can cover the case.
- Every rule correctly classifies at least one remaining training case when it is chosen.

4 Results

The proposed Temporal Association Rule Mining and Improved Algorithm for Mining Frequent Elements (TAR-IMF) is compared with the Rank-based Weighted Association Rule Mining technique (S-RWARM) for secure cloud computing environment [22] and SemanticRuleScorer algorithm. Fig.2 shows the graph illustrating the variation in the execution time with respect to the threshold value. The execution time decreases with respect to the increase in the threshold value. This indicates that the proposed TAR-IMF requires minimum execution time than the S-RWARM technique and SemanticRuleScorer algorithm. There is a steep decrease in the execution time with respect to the increase in the threshold value. The proposed TAR-IMF requires minimum time to predict the best data corresponding to the user requirement.
Fig. 2. Execution time analysis of S-RWARM, SemanticRuleScorer and proposed TAR-IMF.

Fig. 3. Comparative analysis of memory usage for the proposed TAR-IMF and existing S-RWARM and SemanticRuleScorer.
Fig. 3 depicts the memory usage analysis of the proposed TAR-IMF and existing SemanticRuleScorer model and S-RWARM technique. The memory usage decreases with the increase in the threshold value. The proposed work requires minimum memory space than the S-RWARM and SemanticRuleScorer. Hence, the storage space of...
the server is utilized efficiently for the execution of other tasks. Fig. 4 illustrates the comparative analysis of the number of datasets for the proposed TAR-IMF and existing SemanticRuleScorer and S-RWARM with respect to the support value. The proposed TAR-IMF requires minimum number of datasets than the SemanticRuleScorer and S-RWARM. Fig. 5 shows the utilization rate of the dataset by the users. The proposed TAR-IMF yields better dataset utilization rate than the S-RWARM technique and SemanticRuleScorer. The throughput of the proposed TAR-IMF is compared with the LH schema [23] and Managed Counter (MC) schema[24]. The proposed TAR-IMF yields high throughput with the LH schema and MC schema.

Fig. 6. Throughput analysis of the proposed TAR-IMF and existing LH and MC schema

5 Conclusion

In this proposed work, a new method for mining the temporal association rules in a transaction database is developed and an improved algorithm for finding frequent elements over a sliding window. The ARTAR algorithm generates temporal association rules to yield higher accuracy. Also, an improved scheme for identifying the frequent items is presented. The proposed scheme achieves improvement in finding the frequent items over a sliding window. The multi-value hash map is used to reduce the loop over counters. The constant running time is achieved irrespective of the number of counters. The proposed TAR-IMF method achieves a significant reduction in the execution time, memory usage, number of datasets and better dataset utilization rate than the S-RWARM and SemanticRuleScorer algorithm. The throughput of the proposed TAR-IMF is higher than the LH and MC schema.
References


