A REVIEW ON ASSOCIATION RULE MINING ALGORITHMS

Jyoti Arora¹, Nidhi Bhalla², Sanjeev Rao³

Pursuing M.Tech, Dept. Of CSE, Swami Vivekanand Institute of Engineering & Technology, Banur, India¹
Associate Professor, Dept. Of CSE, Swami Vivekanand Institute of Engineering & Technology, Banur, India²
Assistant Professor, Dept. Of CSE, RIMT Institute of Engineering & Technology, Mandi Gobindgarh, India³

ABSTRACT: In this paper, a review of four different association rule mining algorithms Apriori, AprioriTid, Apriori hybrid and tertius algorithms and their drawbacks which would be helpful to find new solution for the Problems found in these algorithms and also presents a comparison between different association mining algorithms. Association rule mining is the one of the most important technique of the data mining. Its aim is to extract interesting correlations, frequent patterns and association among set of items in the transaction database.

KEYWORDS: Data mining, Association rule algorithms, Apriori, AprioriTid, Apriori hybrid and Tertius algorithms.

I. INTRODUCTION

The science of extracting useful information from large data sets or databases is named as data mining[4]. Though data mining concepts have an extensive history, the term “Data Mining“, is introduced relatively new, in mid 90’s. Data mining covers areas of statistics, machine learning, data management and databases, pattern recognition, artificial intelligence, and other areas.

II. ASSOCIATION RULE MINING

In data mining, association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness[2]. Based on the concept of strong rules, Rakesh Agrawal et al [3]. A typical and widely-used example of association rule mining is Market Basket Analysis. The problem is to generate all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence.

\[
\text{Support (S)} = \frac{\text{Support count of (XY)}}{\text{Total number of transaction in D}}
\]

\[
\text{Confidence} = \frac{\text{Support (XY)}}{\text{Support (X)}}
\]

Support(S)

Support(S) of an association rule is defined as the percentage/fraction of records that contain X ∪ Y to the total number of records in the database. Suppose the support of an item is 0.1%, it means only 0.1 percent of the transaction contain purchasing of this item.

Support (XY) = Support count of (XY) / Total number of transaction in D
Confidence(C)
Confidence(C) of an association rule is defined as the percentage/fraction of the number of transactions that contain \(X \cup Y\) to the total number of records that contain \(X\). Confidence is a measure of strength of the association rules, suppose the confidence of the association rule \(X \Rightarrow Y\) is 80%, it means that 80% of the transactions that contain \(X\) also contain \(Y\) together.

**Confidence \((X|Y) = \text{Support (XY)} / \text{Support (X)}**

**A. Association Rules Goals**

- Find all sets of items (item-sets) that have support (number of transactions) greater than the minimum support (large item-sets).
- Use the large item-sets to generate the desired rules that have confidence greater than the minimum confidence.

**B. General AR Algorithm**

- In the first pass, the support of each individual item is counted, and the large ones are determined.
- In each subsequent pass, the large item-sets determined in the previous pass is used to generate new item-sets called candidate item-sets.
- The support of each candidate item-set is counted, and the large ones are determined.
- This process continues until no new large item-sets are found.

### III. APRIORI ALGORITHM

Apriori was proposed by Agrawal and Srikant in 1994[1]. The algorithm finds the frequent set \(L\) in the database \(D\). It makes use of the downward closure property. The algorithm is a bottom search, moving upward level-wise in the lattice. However, before reading the database at every level, it prunes many of the sets which are unlikely to be frequent sets, thus saving any extra efforts.

**Candidate Generation:** Given the set of all frequent (k-1) item-sets. We want to generate superset of the set of all frequent k-item-sets. The intuition behind the apriori candidates generation procedure is that if an item-set \(X\) has minimum support, so do all subsets of \(X\). after all the (l+1)- candidate sequences have been generated, a new scan of the transactions is started (they are read one-by-one) and the support of these new candidates is determined.

**Pruning:** The pruning step eliminates the extensions of (k-1) item-sets which are not found to be frequent, from being considered for counting support. For each transaction \(t\), the algorithm checks which candidates are contained in \(t\) and after the last transaction are processed; those with support less than the minimum support are discarded.

Discovering Large Item-sets
- Multiple passes over the data
- First pass – count the support of individual items.
- Subsequent pass
  - Generate Candidates using previous pass’s large item-set.
  - Go over the data and check the actual support of the candidates.
- Stop when no new large item-sets are found.

Any subset of large item-set is large, therefore

To find large k-item-set
- Create candidates by combining large k-1 item-sets.
Delete those that contain any subset that is not large[1].

---

**AR – Statement of Problem**

- $I = \{ i_1, i_2, \ldots, i_m \}$ is a set of items.
- $D$ is a set of transactions $T$.
- Each transaction $T$ is a set of items.
- $TID$ is a unique identifier that is associated with each transaction.

**DRAWBACKS**

The main drawbacks of Apriori algorithm are

- It takes more time, space and memory for candidate generation process.
- To generate the candidate set it requires multiple scan over the database.

**IV. APRIORITID ALGORITHM**

- The database is not used at all for counting the support of candidate item-sets after the first pass.
- The candidate item-sets are generated the same way as in Apriori algorithm.
- Another set $C'$ is generated of which each member has the TID of each transaction and the large item-sets present in this transaction. This set is used to count the support of each candidate item-sets[1].
a) For small problems, AprioriTid did about as well as Apriori, but performance degraded to about twice as slow for large problems.

b) During the initial passes the candidate item sets generated are very large equivalent to the size of the database. Hence the time taken will be equal to that of Apriori. And also it might incur an additional cost if it cannot completely fit into the memory.

V. APRIORI HYBRID ALGORITHM

Apriori performs better than AprioriTid in the initial passes but in the later passes AprioriTid has better performance than Apriori. Due to this reason we can use another algorithm called Apriori Hybrid algorithm[1].

In which Apriori is used in the initial passes but we switch to AprioriTid in the later passes. The switch takes time, but it is still better in most cases.

Estimate the size of $C'_k$

$$\sum_{\text{transactions}} \text{support(c)} + \text{number of transactions}$$

**DRAWBACKS**

a) An extra cost is incurred when shifting from Apriori to AprioriTid.

b) Suppose at the end of K th pass we decide to switch from Apriori to AprioriTid. Then in the (k+1) pass, after having generated the candidate sets we also have to add the Tids to $C'_k+1$. 

---

Fig 1.2 AprioriTid Diagram
VI. TERTIUS ALGORITHM

This algorithm finds the rule according to the confirmation measures (P. A. Flach, N. Lachiche 2001). It uses first order logic representation. It includes various option like class Index, classification, confirmation Threshold, confirmation Values, frequency Threshold, horn Clauses, missing Values, negation, noise Threshold, number Literals, repeat Literals, roc Analysis, values Output etc[6].

DRAWBACK

Tertius is its relatively long runtime, which is largely dependent on the number of literals in the rules. Increasing the allowed number of literals increases the runtime exponentially, so we want to keep the maximum to three. Even with an allowed maximum of three literals, the runtime is still quite long - running Tertius can take up to several hours for some of our larger tests.

Comparison of Apriori, AprioriTid, AprioriHybrid and tertius.

<table>
<thead>
<tr>
<th>SNO</th>
<th>PROPERTIES</th>
<th>APRIOR</th>
<th>APRIORITID</th>
<th>APRIORIHYBRID</th>
<th>TERTIUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Candidate generation</td>
<td>Candidate item-sets are generated using only the large item-sets of the previous pass without considering the transactions in the database.</td>
<td>The database is not used at all for counting the support of candidate item-sets after the first pass.</td>
<td>Hybrid algorithm can be designed that uses Apriori in the initial passes and switches to AprioriTid in the later passes.</td>
<td>The Tertius algorithm builds rules out of the attribute pair values in the training data.</td>
</tr>
<tr>
<td>2</td>
<td>Methodology used</td>
<td>Uses join &amp; prune step</td>
<td>Uses join &amp; prune as well as Tids</td>
<td>Uses Apriori + AprioriTid</td>
<td>Uses first order logic representation.</td>
</tr>
<tr>
<td>3</td>
<td>Database scan</td>
<td>Multiple scan over the database.</td>
<td>Uses the database only once.</td>
<td>Uses Apriori + AprioriTid</td>
<td>It is depend on the number of literals in the rules.</td>
</tr>
<tr>
<td>4</td>
<td>Memory usage</td>
<td>It takes more space and memory for candidate generation process.</td>
<td>In the kth pass, AprioriTid needs memory for $L_{k-1}$ and $C_{k-2}$ during candidate generation. An additional cost is incurred if it cannot completely fit into the memory.</td>
<td>An extra cost is incurred when shifting from Apriori to AprioriTid.</td>
<td>When the program runs out of memory, the best rules found so far are printed, and a message indicates that the search was interrupted.</td>
</tr>
<tr>
<td>5</td>
<td>Execution Time</td>
<td>It takes more execution time for candidate generation process.</td>
<td>For small problem, it’s better than Apriori but it takes more time for large problem.</td>
<td>It’s better than Apriori and AprioriTid</td>
<td>It’s relatively long runtime. Tertius can take up to several hours for some of our larger tests.</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

In this article provided an overview on four different association rule mining algorithms Apriori, AprioriTid, Apriori hybrid and tertius algorithms and their drawbacks which would be helpful to find new solution for the Problems found in these algorithms and also presents a comparison between different association mining algorithms.
REFERENCES


BIOGRAPHY

Er. Jyoti Arora, pursuing M.Tech-CSE from SVIET, Banur, India and received her B.Tech-CSE degree from DBEC, Mandi Gobindgarh, India. Her area of interest is Data Mining. She has published and presented Two papers in national conference.

Nidhi Bhalla is Lecturer in the department of Computer Science and Engineering at swami Vivekanand engineering college. She has almost six years of academic experience. She has done her B.Tech in information technology from Punjab technical university and M.Tech. From lovely professional university.

Er. Sanjeev Rao presently working as Assistant Professor-CSE at RIMTMAE, Mandi Gobindgarh, Punjab, India. He has done B.Tech-IT degree from SUCSET, Mohali, India, M.Tech-CSE degree from SVIET, Banur, India. He is Oracle Certified Professional from Oracle University. His areas of interest are Databases (Oracle), Data warehousing & Data Mining, Cloud Computing and Software Engineering. He has published and presented many papers in International Conferences and Journals and attended various FDP/Workshop programs. He has both Industry and Teaching experience of about 5 years. Also he has delivered trainings in Oracle and expert lectures to Engineering students and to Corporate.