IMPACT: International Journal of Research in Engineering & Technology (IMPACT: IJRET)
ISSN(E): 2321-8843; ISSN(P): 2347-4599
Vol. 4, Issue 3, Mar. 2016, 27-34

ASSOCIATION RULE MINING FOR DIFFERENT MINIMUM SUPPORT

MUKESH SONI & TEJAS PATEL

Department of Computer Science, ITM Universe, Vadodara, India

ABSTRACT

In this paper, we describe the Data Mining and Association Rule Mining concepts. Data mining is classified as a knowledge discovery process that is specifically used to analyze data from the different mining principles and perspectives and combine it into a powerful tool that is used to pool this information to increase revenue in its totality and boost the general output of organizations.

One such principal of the data mining concept is the Association Rule Mining, which finds the entire rule that exists in the database which will satisfy an amalgamation of some minimum support and minimum confidence constraints.Although the target of discovery isn't pre-determined, this type of rule satisfies the Minsup(Minimum Support) and Minconf (Minimum confidence) controls which assumes every item in the data will behave in the same nature and will possess similar patterns in its frequencies in its data. Branching out further, the Minsupwill control the absolute minimum amount of data that it has to cover for the rule while the Minconf will regulate predictive analytics of the rule.

However these might not be a real-time scenario as in most applications, the items will rarely appear in some instances and would appear in an infrequent pattern in others in the data. So if the Minsupis set to a very high state, those rare items that these are rules are involved in will not be found. This in turn will cause a combinatorial explosion as the frequent items will be caught entangled with each other in all possible ways and outcomes.

We're going to introduce a niche technique to absolve of this issue as this specific technique will allow the user to quantify multiple Minsupso that it may reflect the true natures of its items and their own variance of frequencies found in the database.

KEYWORDS: CBA-RG Algorithm, Closure Property, MTS

INTRODUCTION

Categories and Subject Descriptors

D.3.3 [**Association Rule Mining]**: *Frequent Itemset Generation & Rule generation.*

Frequent Itemset Generation – to obtain all the itemsets that will satisfy the Minsup. They are called frequent itemsets.

Rule Generation –its primary objective is to extract all those rules the frequent itemsets found in the previous step that are high-confidence rules. These are called strong rules.

These frequent sets plays an integral role the Data Mining tasks as they constantly trying to find these patterns from the databases, such as their sequences, correlations, association rules, episodes, clusters and classifiers.

The identification of sets of the by-products, items and characteristics makes the mining of association rules the most outstandingly popular problematic equation of all.

These often will happen in a clumped mass within the database whereby they are viewed as the fundamental task in Data Mining. An urgent need to search and analyze large corporate franchise data which forms the basis of the intent for examining various customer behaviors and patterns.

Frequency of these sets of products will explain how are these items are bought together. Officiallylets make I be the set of items. Transaction over I is a couple F = (fid, I) where fid is the transactional identifier and I is the set of items from I. A database DB over I is considered as the set of a transactions over I which that means that every transaction has been bonded with a unique identifier.

So a transaction F = (fid, I) is said to support a set M, if M C I. The cover of a set M in DB consists of a set of transactional identifiers of transactions in DB that will support M. Support of a set M in DB is the number of transactions in the cover of Min DB. The frequency of a set M in DB is the probability that M occurs in a transaction, or in other words, the support of M divided by the total number of transactions in the database. We omit DB whenever it is clear from the context.

Algorithm

Algorithms of Associated rules discovery – From a possible of transactions *T*, the general idea is to find all rules having support >Minsupand the confidence > Minconf, where the exact Minsupand Minconf are the commanding support and confidence bearers.

$$\text{Support, } s(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{N};$$
$$\text{Confidence, } c(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}.$$

This rough approach is widely considered as the ultimate evidence when it comes to getting to compute the exact support and confidence required for each and every possible rule in the database.

As there are many rules exponentially that can be extracted from the data set, this type of computation is rigid and valued expensively.

Specifically the exact total of possible rules extracted from a data set that has *d* item is

$$R = 3^d - 2^{d+1} + 1.$$

The proposed algorithm is called algorithm CBA (Classifiers Based on Associations). It will be of two parts, a rule generator (called CBA-RG), which is based on the robust algorithm Apriori for finding association rules in (Agrawal and Srikant 1994), and a classifier builder (called CBA-CB). This section discusses CBA-RG. The next section discusses CBA-CB.

Basic Concepts Used In the CBA-RG Algorithm

The keynote operation of the CBA-RG is to find all of its ruleitems that have the framework of support above

Minsup. A ruleitem is of the form: where condset is a set of items, s Î S is a class label. The support count of the condset (called Cond supCount) is the number of cases in DB that contain the condset. The exact support count of the ruleitem (called rulesupCount) is the number of cases in DB that contain the condset and are labeled with class s. Each ruleitem basically represents a rule: condset s, whose support is (rules upCount / |DB|) *100%, where |DB| is the size of the dataset, and whose confidence is (rules upCount / Cond supCount)*100%.

One of the most important class of regularities that will be found in the databases are Association rules.

Its problems has widely received some great feedback and attention throughout the data mining sphere. One of the classic ones is the market basket analysis.

This analyzes on how the items are being purchased by those customers that are associated. One example of an association rule is as follows, dairy → alcohol [sup = 10%, conf = 80%] This rule says that 10% of customers buy dairy and alcohol together, and those who buy dairy products also buy alcohol 80% of the time. The basic model of association rules is as follows: Let O = {o1,o2, …, om} be a set of items. Let G be a set of transactions (the database), where each transaction g (d data case) is a set of items such that g⊆ O. An association rule is an implication of the form, X → Y, where X ⊂ O, Y ⊂ O, and X ∩ Y = ∅. The rule X → Y holds in the transaction set G with confidence cf if cf% of transactions in G that support X also support Y. The rule has support s in G if s% of the transactions in G contains X ∪ Y. Given a set of transactions G (the database), the problem of mining association rules is to discover all association rules that has the confidence and support is definitely greater as it supersedes the user-specified of Minsupand Minconf.

MODEL EXTENSION

The extended model has been defined as the rulesof association rules remains the same. The definition of Minsuphas changed dramatically however because the Minsupof a rule is expanded in terms of the minimum item supports (MTS) of the items that will appear in the rule. This is so because, each item existing in the database can have a MTS that is specified by the user. Providing different MTS values for each and every item, the user basically expresses different support requirements for different rules.

Let MTS (i) denote the MTS value of item it. The minimum support of a rule U is the lowest MTS value among the items in the rule. That is, a rule U, a1 , a2 , …, ak → ak+1 , …, ar where aj ∈ IT, satisfies its minimum support if the rule's actual support in the data is greater than or equal to: min(MTS (a1), MTS (a2), …, MTS (ar)). This allows the Minimum item supports to achieve the goal of having higher minimum supports for rules that only involve those frequent items, and having lower minimum supports for rules that involve less frequent items.

Example 1: Consider the following items in a database, cheese, shorts, and clothes. The user-specified MTS values are as follows: MTS (cheese) = 2% MTS (shoes) = 0.1% MTS (clothes) = 0.2% the following rule doesn't satisfy its minimum support: clothes → cheese [sup = 0.15%, conf = 70%] because min (MTS (cheese), MTS (clothes)) = 0.2%. The following rule satisfies its minimum support: clothes → shorts [sup = 0.15%, conf = 70%] because min (MTS (clothes), MTS (shorts)) = 0.1%

3. THE MINING OF LARGE ITEMSETS WITH MULTIPLE MTSS

3 1 Downward Closure Property

The existing algorithms for mining association rules typically consists of two steps: (1) finding huge itemsets; and (2) generating association rules using the huge itemsets.

Nearly all research material for association rule mining algorithms are solely targeted on the first step since it is considered computationally more expensive. Also, the second step does not automatically extend itself as well to smart algorithms as confidence does not possess the closure property that is necessary. Support, on the other hand, is downwardly closed, which means that if a set of items satisfies the Minsup, then all of its subsets also will fiercely satisfy the Minsup. Downward closure property holds the key to reduce in all existing mining algorithms.

These effective algorithms for finding large itemsets are based on level-wise search [3]. Let l-itemset denote an itemset with l items. At level 1, all large 1-itemsets are generated. At level 2, all large 2-itemsets are generated and so on. If an itemset is not large at level l-1, it is discarded as any addition of items to the set cannot be large (downward closure property). All the potentially large itemsets at level k are generated from large itemsets at level l-1. However, in the proposed model, if we use an existing algorithm to find all large itemsets, the downward closure property no longer holds

Example 2: Consider these four items a, b, c and d in a database. Their minimum item supports are: MTS (a) = 10% MIS (b) = 20% MTS (c) = 5% MTS (d) = 6% If we find that itemset {a, b} has 9% of support at level 2, then it does not satisfy either MTS (a) or MTS (b). Using an existing algorithm, this itemset is discarded since it is not large. Then, the potentially large itemsets {a, b, c} and {a, b, d} will not be generated for level 3. Clearly, itemsets {a, b, c} and {a, b, d} may be large because MTS (c) is only 5% and MTS (d) is 6%. It is thus wrong to discard {a, b}. But if we do not discard {a, b}, the downward closure property is lost.

3 2: The Algorithm Specifics

The proposed algorithm generalizes the Apriori algorithm for finding huge itemsets given in [3]. We call the new algorithm, MTSapriori. When there is only one MTS value (for all items), it reduces to the Apriori algorithm. Like algorithm Apriori, our algorithm is also based on level wise search. It generates all large itemsets by making multiple passes over the data. In the first pass, it counts the supports of individual items and determines whether they are large. In each subsequent pass, it starts with the seed set of item sets found to be large in the previous pass. It uses this seed set to generate new possibly large itemsets, called candidate item sets. The actual supports for these candidate itemsets are computed during the pass over the data. At the end of the pass, it determines which of the candidate itemsets are actually large. However, there is an important exception in the second pass as we will see later.

A key operation in the proposed algorithm is the sorting of the items in I in ascending order of their MTS values. This ordering is used in all subsequent operations of the algorithm. The items in each itemset also follow this order. For example, in Example 2 of the four items a, b, c and d, and their given MTS values, the items are sorted as follows: c, d, a, b. this will help the ordering to solve the problem identified in Section 3.1.

Let Mk denote the set of large k-itemsets. Each itemset c is of the following form, , which consists of items, c[1], c[2], …, c[k], where MTS(c[1]) ≤ MTS(c[2]) ≤ … ≤ MTS(c[k]). The algorithm is given below:

Algorithm MtSapriori

- M = sort (I, MS); /* according to MIS (i)'s stored in MS */
- F = init-pass (M, T); /* make the first pass over T */
- L1 = {| f ∈ F, f.count ≥ MTS (f)};
- for (k = 2; Mk -1 ≠ ∅; k++) do
- if k = 2 then C2 = level2-candidate-gen (F)
- else Ck = candidate-gen (Mk -1)
- end
- for each transaction t ∈ T do
- Ct = subset (Ck, t);
- for each candidate c ∈ Ct do c.count++;
- end
- Lk = {c ∈ Ck | c.count ≥ MTS (c [1])}
- end
- Answer = ∪k Mk

Line 1 performs the sorting on I according to the MTS value of each item (stored in MS). Line 2 makes the first pass over the data using the function init-pass, which takes two arguments, the database T and the sorted items M to produce the seeds for generating the set of candidate large itemsets of length 2, i.e., C2. Init-pass has two steps: 1. it makes a pass over the data to record the actual support count of each item in M. 2. It then follows the sorted order to find the first item i in M that meets MTS (i). i is inserted into F. For each subsequent item j in M after i, if j.count ≥ MTS (i) then j is also inserted into F (j.count means the count of j). Note that for simplicity, we use the terms support and count interchangeably (actually, support = count/|T|, where |T| is the size of the database T)

3 3: Candidate Generation

The level2-candidate-gen takes as argument H (not L1), and returns a superset of the set of all large 2-itemsets. The algorithm is as follows: 1 for each item g in H in the same order do 2 if f.count ≥ MTS (h) then 3 for each item g in H that is after f do 4 if h.count ≥ MTS (h) then 5 insert into C2

Example 4: Let us continue with **Example 3**. We obtain, C2 = {,} is not a candidate 2-itemset because the support count of item 1 is only 9 (or 9%), which is less than MTS (1) (= 10%). Hence, cannot be large. Note that we must use H rather than L1 because L1 does not contain those items that may satisfy the MTS of an earlier item (in the sorted order) but not the MTS of itself (see the difference between H and L1 in Example 3). Using H, the problem discussed in Section 3.1 is solved for C2.

3 4: Function Subsets

The subset function checks to see which itemsets in Ck are in transaction t. Item sets in Ck are stored in a tree similar to that in [3]. Each tree node contains an item (except the root). By depth first traversing of the tree against t, we can find if an item set is in t. At each node, we check whether the item in the node is in t. If so, we go down the tree. If not, we backtrack. When a leaf node is reached, we know that the item set represented by the path is in t. This method for finding Ct is different from that in [3]. The method in [3] uses each item in t to traverse the tree. In our extended model, this, however, requires the items in each transaction t to be sorted according to their MTS values in ascending order in order to achieve the sorted closure property. This computation can be substantial if the database is large and resides on hard disk. Most databases for association rule mining are very large. (This is, however, considered an alternative implementation).

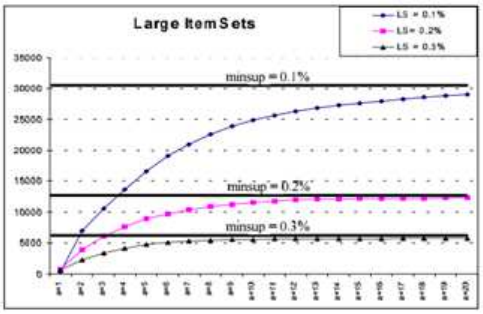

Figure 1: Number of Large Itemsets Found

3 5: Evaluation

This is the section that evaluates the extended modelso that we can show how the model allows us to find rules with very low supports (involving rare items) yet without generating a large number of meaningless rules with frequent items.

3 6: Synthetic Data Experimentation

The synthetic test data is generated with the data generator in [3], which is widely used for evaluating association rule mining algorithms. For our experiments, we need a method to assign MTS values to items in the data set. We use the actual frequencies (or the supports) of the items in the data as the basis for MTS assignments. Specifically, we use the following formulas:

$$MIS(i) = \begin{cases} M(i) & M(i) > LS \\ LS & Otherwise \end{cases}$$

$$M(i) = \beta f(i)$$

g (i) is the actual frequency (or the support expressed in percentage of the data set size) of item i in the data. US is the user-specified lowest minimum item support allowed. β ($0 \leq \beta \leq 1$) is a parameter that controls how the MTS values for items should be related to their frequencies. Thus, to set MTS `values for items we use two parameters, β and US. If β = 0, we have only one minimum support, US, which is the same as the traditional association rule mining. If β = 1 and $g(i) \geq$ US, g(i) is the MTS value for i.

4 APPLICATION TO REAL-LIFE DATA

We tested the algorithm using a number of real-life data sets. Here, we only use one application data set. The results with the others are similar. Due to confidentiality agreement, we are unable to provide the details of the application. Here, we only give the characteristics of the data. The data set has 55 items and 700 transactions. Each transaction has 14-16 items. Some items can appear in 500 transactions, while some may only appear in 30 transactions. The standard deviation of item frequencies in the data is 25.4% (the mean is 24.3%). For this application, the user sets LS = 1%. The results are shown in Figure 3, which include both the numbers of candidate itemsets and large itemsets found. The two thick lines show the number of candidate itemsets and the number of large itemsets found respectively by the single Minsup (= 1%) method. Our new method reduces the numbers dramatically. For this application, the user is happy with the large itemsets found at $\alpha = 4$. The number of large item sets found by our method at $\alpha = 4$ is only 8.5% of that found by the existing single Minsupmethod. The drop in the number of candidate item sets as it's even more drastic.

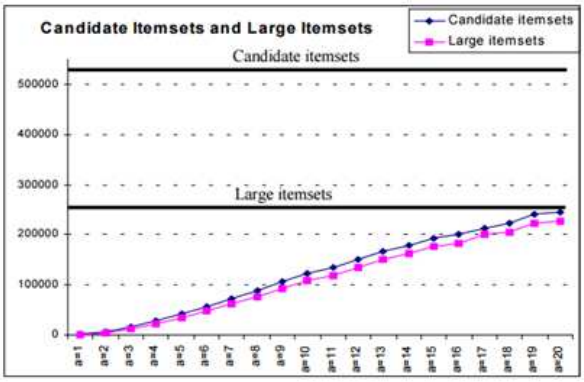

Figure 2: Number of Candidates Itemsets and Large Itemsets

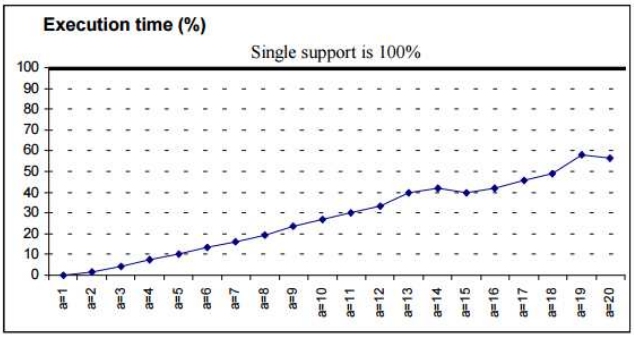

Figure 3: Comparison of Execution Times in Percentage

Figure 3 shows the execution time comparison in percentage. The execution time used by the single Minsup method is set to 100%. We can see that the proposed method also reduces the execution time significantly (since this data set is small, the itemsets generation dominates the whole computation). Note that for applications, the user can also assign MTS values manually rather than using the formulas in Section 3.1

4.1 Relative Works

Association rule mining has been studied extensively in the past [e.g., 2, 3, 5, 11, 4, 14, 10, 12, 1]. However, the model used in all these works is the same, i.e., with only one user-specified minimum support threshold [2]. Multiple-level

association rule mining in [5] can use different minimum supports at different levels of hierarchy. However, at the same level it uses only one Minsup. For example, we have the taxonomy: milk and cheese are Dairy product; and pork and beef are Meat. At the level of Dairy product and Meat, association rules can have one Minsup, and at the level of milk, cheese, pork and beef, there can be a different Minsup. This model is essentially the same as the original model in [2] because each level has its own association rules involving items of that level. Our proposed model is more flexible as we can assign a MTS value for each item. [13] Presents a generalized multiple-level association rule mining technique, where an association rule can involve items at any level of the hierarchy. However, the model still uses only one Minsup. It is easy to see that our algorithm MTSapriori is a generalization of the Apriori algorithm [3] for single Minsup mining. That is, when all MTS values are the same as LS, it reduces to the Apriori algorithm. A key idea of our algorithm MTSapriori is the sorting of items in I according to their MTS values in order to achieve the closure property. Although we still use level-wise search, each step of our algorithm is different from that of algorithm Apriori, from initialization, candidate item sets generation to pruning of candidate item sets.

REFERENCES

1. Aggarwal, C., and Yu, P. "Online generation of association rules." ICDE-98, 1998, pp. 402-411.
2. Agrawal, R., Imielinski, T., Swami, A. “Mining association rules between sets of items in large databases.” SIGMOD- 1993, 1993, pp. 207-216.
3. Agrawal, R. and Srikant, R. “Fast algorithms for mining association rules.” VLDB-94, 1994.
4. Brin, S. Motwani, R. Ullman, J. and Tsur, S. “Dynamic Itemset counting and implication rules for market basket data.” SIGMOD-97, 1997, pp. 255-264.
5. Han, J. and Fu, Y. “Discovery of multiple-level association rules from large databases.” VLDB-95.
6. Lee, W., Stolfo, S. J., and Mok, K. W. “Mining audit data to build intrusion detection models.” KDD-98.
7. Liu, B., Hsu, W. and Ma, Y. Mining association rules with multiple minimum supports. SoC technical report, 1999.
8. Liu, B., Hsu, W. and Ma, Y. "Pruning and Summarizing the Discovered Associations" KDD-99, 1999.
9. Mannila, H. "Database methods for data mining.” KDD-98 tutorial, 1998.
10. Ng. R. T. Lakshmanan, L. Han, J. “Exploratory mining and pruning optimizations of constrained association rules.” SIGMOD-98, 1998.
11. Park, J. S. Chen, M. S. and Yu, P. S. “An effective hash based algorithm for mining association rules.” SIGMOD-95, 1995, pp. 175-186.
12. Rastogi, R. and Shim, K. “Mining optimized association rules with categorical and numeric attributes.” ICDE – 98.
13. Srikant, R. and Agrawal, R. “Mining generalized association rules.” VLDB-1995, 1995.
14. Srikant, R., Vu, Q. and Agrawal, R. “Mining association rules with item constraints.” KDD-97, 1997, pp. 67-73