MINING ASSOCIATION RULES IN LARGE DATABASES

Shraddha Wakchaware
Assistant Professor, P. C. College Of Engineering
Department Of Information Technology, Goa University, Verna

Abstract- **Data mining uses a technique of Association Rule Mining to generate rules in an efficient way. Association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. The main problem that occurs is, handling the large databases. The paper presents the methodology to find the association rules from a large dataset. It also puts forward the ways to deal with the large datasets and determines the factors on which the generation of association rules is dependent. Thus, the association rules are generated by two ways considering the support count which is user defined and secondly, generated in accordance with the database with minimum memory usage. One such large database that is under consideration is census data set which is significant in the stock exchange system. The input data is preprocessed and the factors like memory usage and total time are accounted. Upon generation of the association rules the variance and the difference is recorded and they are compared for their accuracy.**

Index Terms- **Mining, Association Rule, Database, Support Count, Census, Variance and Accuracy.**

I. INTRODUCTION

Database mining is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Apart from the raw analysis step, it involves database and data management aspects, data pre-processing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating. The actual data mining task is the automatic or semi-automatic analysis of large quantities of data to extract previously unknown interesting patterns such as groups of data records (cluster analysis), unusual records (anomaly detection) and dependencies (association rule mining). Data mining is also the analysis step of the "Knowledge Discovery in Databases" process, or KDD.

The paper implementations is done mainly in three stages:

1. Pre-processing
2. Data mining
3. Results validation

Pre-processing – Before the mining algorithms are applied there is a need to gather data. Once the data is made available it should be large enough so that the relevant patterns can be reveled. Once the relevant patterns are found then the appropriate mining strategy could be applied to mine them. This section makes use of the partitioning algorithm to partition the large data into subsets.

Data Mining – The paper makes use of the following:

1. Association Rule Learning: It searches for the relationships between different variables.
2. Summarization: provides a more compact representation of the data set, including visualization and report generation.

Result Validation- It involves the depiction whether the patterns are really relevant or not. After applying the data mining algorithms the results generate the relevant patterns.

Thus, one of the main challenges in database mining is to have such techniques and procedures that can handle large volumes of data. This is because most of the mining algorithms perform their computations over entire database and the databases are very large. Thus, the generation of association rules from these large databases is a crucial task under consideration.

The main aim of the implementation of this paper is to compare various parameters based on the support count values which may be user defined or found accurately as per the nature of the dataset. The validation of the user defined values is done by using simple Apriori algorithm and the exact values of the support counts are found by the linear and polynomial strategies.

II. Importance Of Association Rule Mining

It is intended to identify strong rules discovered in databases using different measures of interestingness.Based on the concept of strong rules, introduction to association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets has been done.

For example, the rule $\{\mathrm{onions}, \mathrm{potatoes}\} \Rightarrow \{\mathrm{burger}\}$ found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, he or she is likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection, Continuous production and bioinformatics. Association rules are usually required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. Association rule generation is usually split up into two separate steps:

1. First, minimum support is applied to find all *frequent item sets* in a database.
2. Second, these frequent item sets and the minimum confidence constraint are used to form rules.

While the second step is straightforward, the first step needs more attention.

Finding all frequent item sets in a database is difficult since it involves searching all possible item sets (item combinations). The set of possible item sets is the power set over $I$and has size $2^n - 1$(excluding the empty set which is not a valid item

set). Although the size of the power set grows exponentially in the number of items n in I, efficient search is possible using the ***downward-closure property*** of support(also called *anti-monotonicity*) which guarantees that for a frequent item set, all its subsets are also frequent and thus for an infrequent item set, all its supersets must also be infrequent.

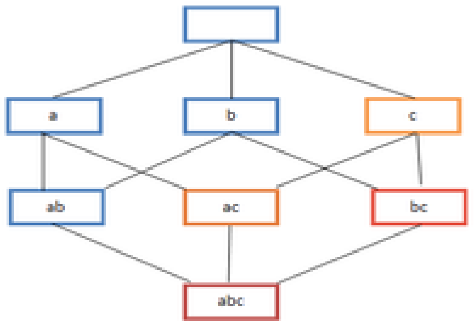


Figure 2.1: Downward Closure Property

In the above figure frequent item set is lattice, where the color of the box indicates how many transactions contain the combination of items. Note that the lower levels of the lattice can contain at most the minimum number of their parents items; eg: {ac} can have only at most min(a,c) items. This is called downward closure property.

III. Dataset Used

The dataset was collected from the site “UCI repositories” (http://archive.ics.uci.edu/ml/datasets.html). This is a large repository of data where in numerous datasets are available. The UCI Machine Learning Repository is a collection of databases, domain theories, and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. The archive was created as an ftp archive in 1987 by David Aha and fellow graduate students at UC Irvine. Since that time, it has been widely used by students, educators, and researchers all over the world as a primary source of machine learning data sets. As an indication of the impact of the archive, it has been cited over 1000 times, making it one of the top 100 most cited "papers" in all of computer science.

Note:

- The dataset that is used for this project is extracted from the census bureau database found at http://www.census.gov/ftp/pub/DES/www/welcome.html donated by Ronny Kohavi and Barry Becker.
- It is split into train-test using MLC++ GenCVFiles (2/3, 1/3 random).
- There are 48842 instances, mix of continuous and discrete. Also, 45222 if instances with unknown values are removed.

- The Extraction was done by Barry Becker from the 1994 Census database.
- A set of reasonably clean records were extracted using the following conditions:

((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

The census dataset is basically considered because of the requirement of large dataset. Thus the dataset is preprocessed for generating association rules.

The choice of dataset is significantly important. Similar to census dataset other datasets such as bank datasets, stock exchange datasets etc. can be used. One of the key point in choosing the dataset is it should have large size and relevant entries. The entities in the datasets should be significantly different and distinguished from one another. Any limitations if cited is to be taken care of before preceeding to preprocessing step.

IV. Methodology And Procedure

A. *Block Diagram*

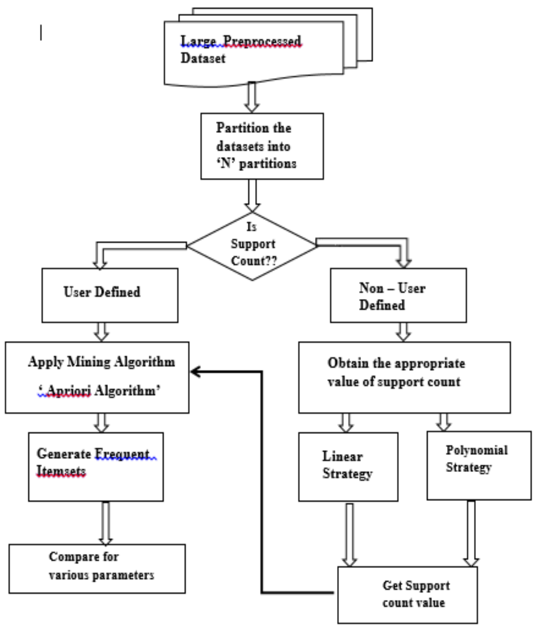

Large Preprocessed Dataset
Partition the datasets into 'N' partitions
Is Support Count??
User Defined
Non – User Defined
Apply Mining Algorithm 'Apriori Algorithm'
Obtain the appropriate value of support count
Generate Frequent Itemsets
Linear Strategy
Polynomial Strategy
Compare for various parameters
Get Support count value

Figure 4.1 Block diagram of the proposed System

- The user will select the large dataset as an input.
- The partitioning algorithm is applied to obtain the partitions.
- Decision is made depending on the value of support count whether it is user defined or non-user defined.
- If the support count is user defined then directly apply Apriori algorithm and find the frequent item sets.
- If the support count is not user defined then its value is found out by either of the two methods i.e. linear strategy and/or polynomial strategy.
- With the determined support count value as per the nature of the dataset again the Apriori algorithm is executed and frequent item sets are found out.
- The generated item sets depending on the value of support count is compared for various parameters.

V. Data Preprocessing Module

For the generation of the frequent item sets the dataset must be in a format suitable for finding the associativity. The transactions in the dataset must be converted into a suitable format so that their comparison can be done and the associativity between various attributes can be found. Therefore, the transactions are in the form (TID, ij, ik,……in). The items in a transaction are kept in lexicographic order. The algorithm can be straightforward applied when transactions are kept normalized in (TID, item) form. Also TIDs are monotonically increasing.

VI. Apriori Algorithm

In computer science and data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions. Other algorithms are designed for finding association rules in data having no transactions.

As in common in association rule mining, given a set of item sets, the algorithm attempts to find subsets which are common to at least a minimum number C of the item sets. Apriori uses a “bottom up” approach, where frequent subset are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breath- first search and a tree structure to count candidate item sets efficiently, generates candidate item sets of length k from item sets of length k-1. Then it prunes the candidates which have an infrequent such pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

A. Key Concepts

- Frequent Item sets : The sets of item which has minimum support (denoted by L, for the i^{th} item set)
- AprioriProperty : Any subset of frequent item set must be frequent.
- Join Operation : To find L_k, a set of candidate k-item sets is generated by joining L_{k-1} with self.

B. A brief description of the algorithm

1. Find all frequent item sets:
   - Get frequent items:
     - Items whose occurrence in database is greater than or equal to min_support threshold.
   - Get frequent item sets:
     - Generate candidates from frequent items.
     - Prune the results to find the frequent item sets.
2. Generate strong association rules from frequent item sets
   - Rules which satisfy the min_support and min_confidence threshold.

C. Steps of Apriori Algorithm

```
1.  $L_1$= {large 1-itemsets};
2.  For (k=2; $L_{k-1}\neq\Phi$; k++) do begin
3.  Ck= apriori-gen($L_{k-1}$); //New candidate
4.  Forall transactions t∊D do begin
5.  $C_t$= subset($C_k$,t); // Candidates contained in t
6.  Forallcandidiates c ∊ $C_t$ do
7.  c.count ++;
8.  end
9.  $L_k$= {c ∊$C_k$ | c.count ≥ minsup}
10. end
11. Answer = $U_k L_k$;
```

In Apriori algorithm the first pass of the algorithm simply counts item occurrences to determine the large 1-itemsets. A subsequent pass, say pass k, consists of two phases. First, the large item sets L_{k-1} found in the (k-1)th pass are used to generate the candidate item sets C_k, using the apriori–gen function. Next, the database is scanned and the support of candidates in Ck is counted. For fast counting, we need to efficiently determine the candidates in Ck that are contained in a given transaction t.

VII. PARTITIONING ALGORITHM

Partitioning algorithm works in two scans of the database. In one scan it generates a set of all potentially large item sets by scanning the database once. This is a superset of all large item sets. During the second scan, counters for each of these item sets are set up and their actual support is measured in one scan of the database.

The algorithm executes in two phases. In the first phase, the partition algorithm logically divides the database into a number of non- overlapping partitions. The partitions are considered one at a time and all the large item sets for that partition are generated. At the end of phase I, these large item sets are merged to generate a set of all potential large item sets. In phase II, the actual supports for these item sets are generated and the large item sets are identified. The partition sizes are chosen such that each partition can be accommodated in the main memory so that the partitions are read only once in each phase.

Definition: A partition p € D of the database refers to any subset of the transactions contained in the database D. Any two different partitions are non-overlapping i.e., $pi \cap pj = \Phi$, $i \neq j$. We define *local-support* for an item set as a fraction of transaction containing that item set in a partition. We define *local candidateitem set* to be an item set that is being tested for minimum support within a given partition. A *local large item set* to be an item set that is being tested for minimum support within a given partition. A *local large* item set is an item set whose local support in a partition is at least the user defined support. A local large item set may or may not be large in the context of entire database. We define *global support, global large item set, and global candidate item set* as above except they are in the context of entire database D. Our goal is to find all global large item sets.

Table : Notation

C_k^p	Set of local candidate k-item sets in partition p
L_k^p	Set of local large k-item sets in partition p
L^p	Set of all local large item sets in partition p
C_k^G	Set of global candidate k- item sets
C^G	Set of all global candidate item sets
L_k^G	Set of global large k- item sets

We use the notation shown in Table. Individual item sets are represented by small letters and sets of item sets are represented by capital letters. When there is no ambiguity we omit the partition number when referring to local item set. We use the notation c[1],c[2],…,c[k] to represent a k-item set c consisting of items c[1], c[2],….,c[k].

A. *Pseudocode of Partitioning Algorithm*

```
1.  P = partition_database (D)
2.  n = Number of partitions
3.  For i= 1 to n begin // Phase I
4.  read_in_partition($p_i \epsilon P$)
5.      $L^i$ = gen_large_itemsets(pi)
6.  end
7.  for (i=2;  $L_i^j \neq \Phi$, j=1,2,………n; i++) do
8.  $C_i^G = U_{j=1,2,\ldots,n} L_i^j$ //Merge Phase
9.  for i=1 to n begin // Phase II
10. read_in_partition ($p_i \epsilon$ P)
11.     for all candidates c $\epsilon$ $C^G$gen_count (c, $p_i$)
12. end
13. LG = { c $\epsilon$ $C^G$| c.count $\geq$ minSup}
```

Algorithm: Initially the database D is logically partitioned into n partitions. Phase I of the algorithm takes n iterations. During iteration i only partition p_i is considered. The function gen_large_itemsets takes a partition p_i as input and generates local large item sets of all lengths, L_1^i,L_2^i,……L_l^i as the output. In the merge phase the local large item sets of same lengths from all n partitions are combined to generate the global candidate item sets. In phase II, the algorithm sets up counters for each global candidate item set and counts their support for the entire database and generates the global large item sets. The algorithm reads the entire database once during phase I and once during phase II.

VIII. Estimating Support Count Strategies

Suppose that the users specify a minimum-support *r_minsupp* with respect to the interval [0,1]. We need to determine the

desired*minsupp* for mining database D for which the support interval is [a,b], implemented by the mapping *f*: [a,b] → [0,1]. Very often, such a mapping *f* is hidden. Therefore, we should find an approximate polynomial function *f'* for *f*. Here is a strategy for constructing the mapping.

Let X in [a,b] and Y in [0,1]be x_1, x_2,……….,x_n, and y_1, y_2,………..,y_n as listed below:

X	x_1	x_2	………	x_n
Y	y_1	y_2	….….	y_n

A method for finding an approximate polynomial function *f'* for*f* between X and y can be performed by the following theorem:

Theorem 1 : For X and Y, the approximate polynomial function for fitting the above data can be constructed as

$$F(x) = F1(x) + \sum_{i=1}^{N}(Fi + 1(x)(\prod_{j=1}^{2i}(x - xj))) \ldots\ldots\ldots\ldots(8.1)$$

Where,

$$F_k(x)=((x- x_{2k})/(x_{2k-1} - x_{2k}))(G_k(x_{2k-1}) + ((x-x_{2k-1})/(x_{2k} - x_{2k-1}))(G_k(x_{2k}))$$

k= 1,2, …………..N; N is the number of fitting times; and G_k is the fitted data.

F(x) is the approximation function of *f* that we desire.It is a polynomial function for which the order is not over 2N+1. Using this approximation function, we can generate an approximate *minsupp*from the given *r_minsupp*.

A. *Simplifying the polynomial function*

It is unrealistic to obtain so many point pairs for constructing the approximation function of *f* using the above theorem. The following subsection describes the simple and useful approximation function of *f*.

1. Linear Strategy

   f(x) = (1/(b-a))x + (a/(a-b)) ……………………..(8.2)

2. Polynomial Strategy

   f(x) = (1/ $(b^n-a^n))x^n$ + $(a^n/(a^n-b^n))$………………….(8.3)

These equations would give the value of the support count as per the dataset. This value would be used to generate association rules.

B. *Identifying frequent item sets by polynomial approximation*

Based on the polynomial approximation and support- confidence framework, we can define that J is a frequent item set of potential interest, denoted by fipi(J)^2, if and only if

fipi (J) = supp(J) ≥ minsupp X, Y: X U Y = J ^ X ∩ Y = Φ ^

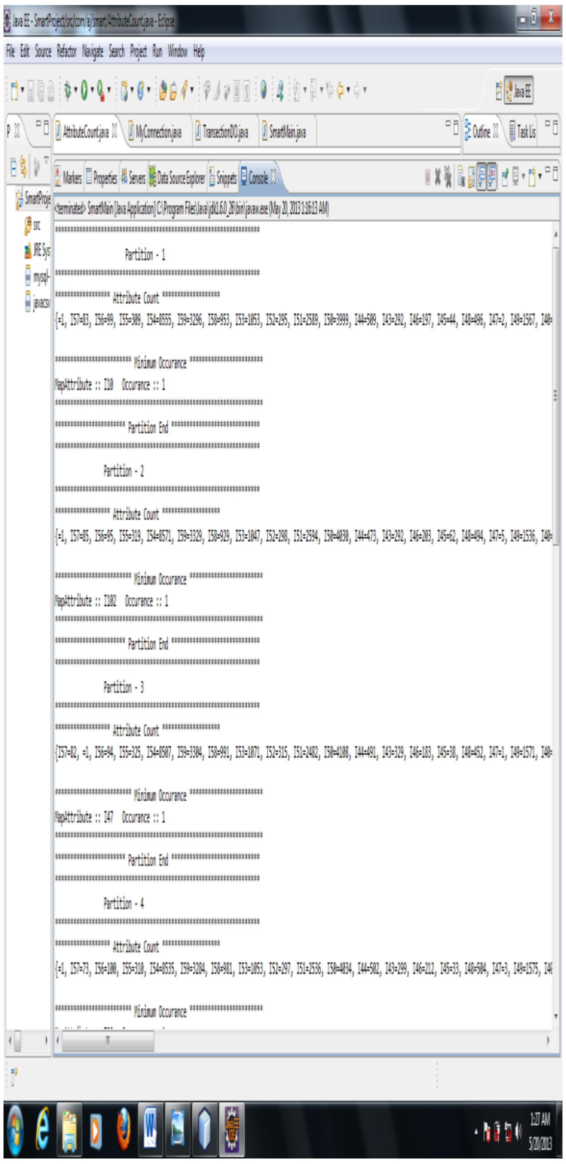
supp (X U Y)/ sup (X) ≥ minconf supp

(X U Y) – supp (X) supp(Y) ≥ mininterest

whereminsupp, minconf, and mininterest are the thresholds of minimum-support, minimum confidence (for the purpose of association rule analysis), and minimum interest, respectively.

IX. Implementation Details

A. *Partitioning The Database*



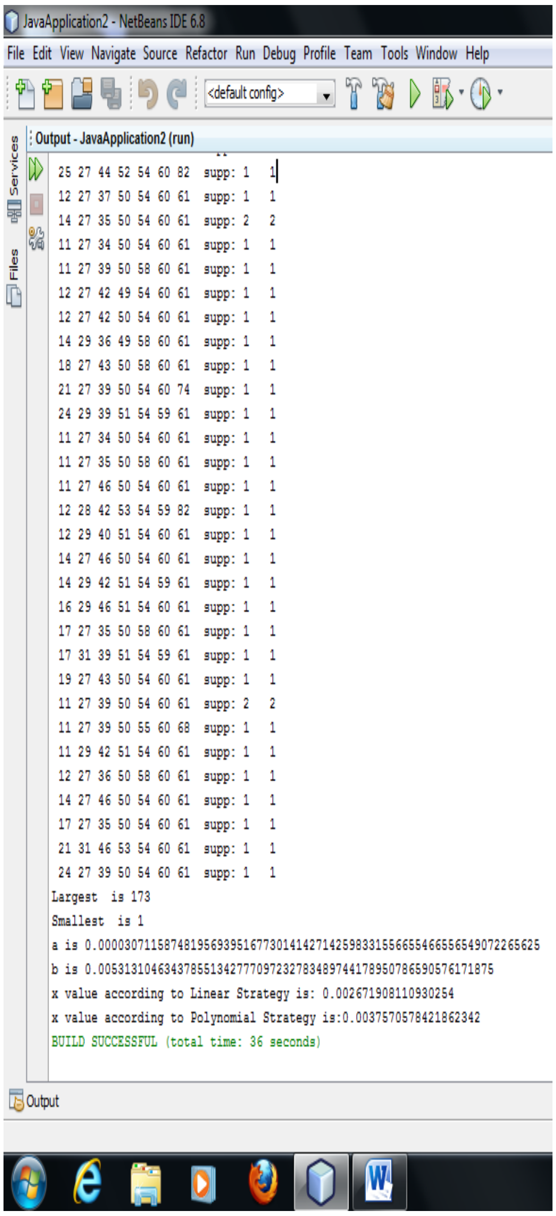
```
<terminated> SmartMain [Java Application] C:\Program Files\Java\jdk1.6.0_26\bin\javaw.exe (May 20, 2013 1:16:13 AM)
************************************************************

                    Partition - 1
************************************************************

**************** Attribute Count *****************
{=1, I57=83, I56=99, I55=309, I54=8555, I59=3296, I58=953, I53=1063, I52=295, I51=2589, I50=3999, I44=509, I43=292, I46=197, I45=44, I48=496, I47=2, I49=1567, I40=


********************** Minimum Occurance **********************
MapAttribute :: I18   Occurance :: 1
************************************************************

******************** Partition End ************************
************************************************************

              Partition - 2
************************************************************

**************** Attribute Count *****************
{=1, I57=85, I56=95, I55=319, I54=8571, I59=3329, I58=929, I53=1047, I52=298, I51=2594, I50=4030, I44=473, I43=292, I46=203, I45=62, I48=494, I47=5, I49=1536, I40=


********************** Minimum Occurance **********************
MapAttribute :: I102   Occurance :: 1
************************************************************

******************** Partition End ************************
************************************************************

              Partition - 3
************************************************************

**************** Attribute Count *****************
{I57=82, =1, I56=94, I55=325, I54=8587, I59=3304, I58=991, I53=1071, I52=315, I51=2482, I50=4108, I44=491, I43=329, I46=183, I45=38, I48=452, I47=1, I49=1571, I40=


********************** Minimum Occurance **********************
MapAttribute :: I47   Occurance :: 1
************************************************************

******************** Partition End ************************
************************************************************

              Partition - 4
************************************************************

**************** Attribute Count *****************
{=1, I57=73, I56=100, I55=310, I54=8535, I59=3284, I58=981, I53=1053, I52=297, I51=2536, I50=4034, I44=502, I43=299, I46=212, I45=33, I48=504, I47=3, I49=1575, I4


********************** Minimum Occurance **********************
```

After following the partitioning algorithm we get the records in the database being partitioned into sub partitions. It also shows the minimum occurrence of the attribute. This is the first stage where in we have to just divide the records into 'n' number of sub

records so the number of scans is just restricted to two. Thus by doing this the large database would be scanned only twice reducing the large CPU overheads.

B. Determination of Exact Support Count Value

As per the dataset this snapshot finds the frequent item set and displays the exact value of the support count as per two strategies viz. Linear Strategy and Polynomial Strategy. By using this value of the support we can find the number of candidate sets. The results produced by these values differ from any value from the users.


JavaApplication2 - NetBeans IDE 6.8
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Output - JavaApplication2 (run)
Services
Files
Output

```
 25 27 44 52 54 60 82  supp: 1   1
 12 27 37 50 54 60 61  supp: 1   1
 14 27 35 50 54 60 61  supp: 2   2
 11 27 34 50 54 60 61  supp: 1   1
 11 27 39 50 58 60 61  supp: 1   1
 12 27 42 49 54 60 61  supp: 1   1
 12 27 42 50 54 60 61  supp: 1   1
 14 29 36 49 58 60 61  supp: 1   1
 18 27 43 50 58 60 61  supp: 1   1
 21 27 39 50 54 60 74  supp: 1   1
 24 29 39 51 54 59 61  supp: 1   1
 11 27 34 50 54 60 61  supp: 1   1
 11 27 35 50 58 60 61  supp: 1   1
 11 27 46 50 54 60 61  supp: 1   1
 12 28 42 53 54 59 82  supp: 1   1
 12 29 40 51 54 60 61  supp: 1   1
 14 27 46 50 54 60 61  supp: 1   1
 14 29 42 51 54 59 61  supp: 1   1
 16 29 46 51 54 60 61  supp: 1   1
 17 27 35 50 58 60 61  supp: 1   1
 17 31 39 51 54 59 61  supp: 1   1
 19 27 43 50 54 60 61  supp: 1   1
 11 27 39 50 54 60 61  supp: 2   2
 11 27 39 50 55 60 68  supp: 1   1
 11 29 42 51 54 60 61  supp: 1   1
 12 27 36 50 58 60 61  supp: 1   1
 14 27 46 50 54 60 61  supp: 1   1
 17 27 35 50 54 60 61  supp: 1   1
 21 31 46 53 54 60 61  supp: 1   1
 24 27 39 50 54 60 61  supp: 1   1
Largest  is 173
Smallest  is 1
a is 0.0000307115874819569395167730141427142598331556655466556549072265625
b is 0.005313104634378551342777097232783489744178950786590576171875
x value according to Linear Strategy is: 0.002671908110930254
x value according to Polynomial Strategy is:0.0037570578421862342
BUILD SUCCESSFUL (total time: 36 seconds)
```

X. Results And Observations

A	B	C	D	E	F	G
User Defi n-ed	10	362	7	166	5.24	735
	20	139	6	78	6.43	1493
	35	45	4	25	6.56	711
	50	17	4	11	6.45	1592
	65	15	3	6	7.84	703
	80	3	2	2	7.60	731
Linear	0.002671908110930254	20311	9	17849	8.19	6386
Poly no-mial	0.0037570578421862342	20311	9	17849	8.76	5933

Notations:

A= Strategies

B= Support (%)

C= Candidate Count

D= Algorithm Stopping Size

E= Frequent Item Set Count

F=Maximum Memory Usage (mb)

G= Total Time (ms)

XI. Conclusion

In this paper, the support count was a major factor focused on when finding the frequent item sets. In most of the algorithms on mining association rules the support count was user defined. But when it takes the user defined values the results are not accurate and vary from user to user. Thus, here an attempt is made to find the exact value of the support as per the nature of the database using linear strategy and polynomial strategy. Apart from the support count various other factors like candidate count, algorithm stopping size, frequent item set count, maximum memory usage and total time are also accounted.

References

[1] R.Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Proceedings of 20th International Conference on Very Large Data Bases, Santiago, Chile, August 29 –September 1 1994

[2] H. Mannila and H. Toivonen and A.I Verkamo. Efficient Algorithms for discovering Association Rules. In KDD-94: AAAI Workshop on Knowledge Discovery in Databases, July 1994

[3] Agrawal R, Shafer J (1996) Parallel mining of Association Rules. IEEE Trans Knowl Data Eng.

[4] A. Savasere, E. Omiecinski, S. Navathe. An efficient Algorithm for mining Association Rules In Large Databases. In the proceedings of the 21st VLDB Conference Zurich, Swizerland, 1995

[5] Bayardo B (1998) Efficiently Mining Long Patterns from Databases. In : Proceedings of ACM international conference on management of data.

[6] Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the ACM SIGMOD international conference on management of data.

[7] Han J, Wang J, Lu Y, Tzvetkov P (2002) Mining Top K-frequent closed

[8] patterns without minimum support. In : Proceedings of the 2002 IEEE international conference on data mining.

[9] http://en.wikipedia.org/wiki/Association_rule_learning

[10] www.cs.uic.edu/~liub/teach/cs583-fall-05/CS583-association-rules.ppt

[11] www.cs.uic.edu/~liub/teach/.../CS583-association-sequential-patterns.ppt

[12] http://archive.ics.uci.edu/ml/datasets.html

[13] http://archive.ics.uci.edu/ml/datasets/Census+Income

[14] https://www.webyog.com/

[15] en.wikipedia.org/wiki/SQLyog

[16] https://www.draw.io/

[17] en.wikipedia.org/wiki/Eclipse_(software)

[18] Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms - Jean-Marc Adamo,Springer

Author's Biography

Shraddha Wakchaware is born in Paratwada, District Amravati, Maharashtra, India. She received Masters Degree in Internet Technology from Goa University, Goa, India in 2013. She is working as Assistant Professor at P. C . College Of Engineering, Verna, Goa-403001, India. She received Bachelor's Degree in Computers, Mumbai University, Maharashtra in 2010. She has worked as Lecturer in SSPM college for one year. Her research interests include data mining, data analysis, association rule mining. She is publishing several papers in international journals and conferences.

Ms. Wakchaware is also an active member of ISTE students chapter. She is also the second rank holder of the state in Goa University for her excellence in Master's. She has also been awarded for securing the 1st position for her Bachelor's. She is also the rank holder in Mumbai University and has been awarded with excellence certificate.