HIGH CONFIDENCE RELIABILITY METHOD FOR KNOWLEDGE DISCOVERY USING EVALUATION OF LEARNING ALGORITHMS IN DATA MINING

N.CHANDRA SEKHAR REDDY , DR.N.PEDAKOLMI , DR A.GOVARDHAN , N.VAMSI KRISHNA , N.L.MANASA

Abstract— : Association rule mining is the most popular technique in data mining. Mining association rules is a prototypical problem as the data are being generated and stored every day in corporate computer database systems. To manage this knowledge, rules have to be pruned and grouped, so that only reasonable numbers of rules have to be inspected and analyzed. In this paper we compare the standard association rule algorithms with the proposed HCRA algorithm. All these algorithms are compared according to the various factors like Type of dataset, support counting, rule generation, candidate generation, computational complexity and other factors .The conclusions drawn are based on the efficiency ,performance ,accuracy and scalability parameters of the algorithms.

Keywords— Association rule, Data Mining, Multidimensional dataset, Pruning, machine learning, Frequent itemset.

Introduction –

Mining association rules is particularly useful for discovering relationships among items from large databases. A standard association rule is a rule of the form X→ Y which says that if X is true of an instance in a database, so is Y true of the same instance, with a certain level of significance as measured by two indicators, support and confidence. The goal of standard association rule mining is to output all rules whose support and confidence are respectively above some given support and coverage thresholds. These rules encapsulate the relational associations between selected attributes in the database, for instance, coke → potato chips: 0.02 support; 0.70 coverage denotes that in the database, 70% of the people who buy coke also buy potato chips, and these buyers constitute 2% of the database. This rule signifies a positive (directional) relationship between buyers of coke and potatochips. The mining process of association rules can be divided into two steps.1. Frequent Itemset Generation: generate all sets of items that have support greater than a certain threshold, called minsupport.2. Association Rule Generation: from the frequent itemsets, generate all association rules that have confidence greater than a certain threshold called minconfidence.Apriori is a renowned algorithm for association rule mining primarily because of its effectiveness in knowledge discovery. However, there are two bottlenecks in the Apriori algorithm.The purpose of the association rules is to find correlations between the different processes of any application. Knowing the associations between these processes, it helps to take decisions and to use the process methods effectively. The various association rule mining algorithms were used to different applications to determine interesting frequent patterns. One of the association rule mining algorithm such as Apriori algorithm used the property of support and confidence to generate frequent patterns. Another measure is Predictive Accuracy , it is an indicator of a rule's accuracy in future over unseen data. Confidence of a rule is the ratio of the correct predictions over all records for which a prediction is made but it is measured with respect to the database that is used for training. This confidence on the training data is only an estimate of the rule's accuracy in the future, and since the user searches the space of association rules to maximize the confidence, the estimate is optimistically biased. Thus, the measure predictive accuracy is introduced. It gives for an association rule its probability of a correct prediction with respect to the process underlying the database.The paper consists of 6 sections as follows. We introduce description of some works in the literature concerning the improvement of association rule algorithms in Section 2.Section 3 parameters on which the algorithms are compared. Section 4 gives the experimental study. The conclusion and future scope are presented in sections 5 and 6 respectively.

1.Proposed method (HCRA) -

HCRA contains array which is processed with a simple split and merge scheme, which computes a conditional database, processes this conditional database recursively, and finally eliminates the split item from the original (conditional) database. HCRA preprocesses a given transaction database in a way that is very similar to the preprocessing used by many other frequent item set mining algorithms. The steps are illustrated in Figure 1 for a simple example transaction database. Step 1 shows the transaction database in its original form. In step 2 the frequencies of individual itemsare determined from this input in order to

be able to discard infrequent items immediately. If we assume a minimum support of three transactions for our example, there are no infrequent items, so all items are kept. In step 3 the (frequent) items in each transaction are sorted according to their frequency in the transaction database, since it is well known that processing the items in the order of increasing frequency usually leads to the shortest execution times. In step 4 the transactions are sorted lexicographically into descending order, with item comparisons again being decided by the item frequencies, although here the item with the higher frequency precedes the item with the lower frequency. In step 5 the data structure on which HCRA operates is built by combining equal transactions and setting up an array, in which each element consists of two fields: an

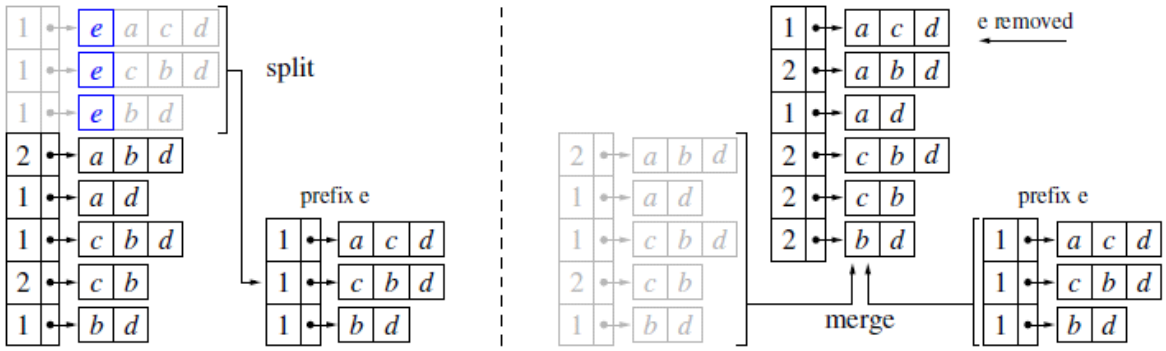


Fig. 2. The basic operation of the HCRA algorithm: split(left) and merge(right).

occurrence counter and a pointer to the sorted transacion (array of contained items). This data structure is then processed recursively to find the frequent item sets. The basic operations of the recursive processing, which follows the general depthfirst/ divide-and-conquer scheme reviewed in Section 2, are illustrated in Figure 2. In the split step (see the left part of Figure 2) the given array is split w.r.t. the leading item of the first transaction (item e in our example): all array elements referring to transactions starting with this item are transferred to a new array. In this process the pointer (in)to the transaction is advanced by one item, so that the common leading item is "removed" from all transactions. Obviously, this new array represents the conditional database of the first subproblem (see Section 2), which is then processed recursively to find all frequent items sets containing the split item (provided this item is frequent). The conditional database for frequent item sets not containing this item (needed for the second subproblem, see Section 2) is obtained with a simple merge step (see the right part of Figure 2). The new array created in the split step and the rest of the original array (which refers to transactions starting with a different item) are combined with a procedure that is almost identical to one phase of the well-known mergesort algorithm. Since both arrays are obviously lexicographically sorted, a single traversal suffices to create a lexicographically sorted merged array. The only difference to a mergesort phase is that equal transactions (or transaction suffixes) are combined. That is, there is always just one instance of each transaction (suffix), while its number of occurrences is kept in the occurrence counter. In our example this results in the merged array having two elements less than the input arrays together: the transaction (suffixes) cbd and bd, which occur in both arrays, are combined and their occurrence counters are increased to 2. Note that in both the split and the merge step only the array elements (that is, the occurrence counter and the (advanced) transaction pointer) are copied to a new array. There is no need to copy the transactions themselves (that is, the item arrays), since no changes are ever made to them. (In the split step the leading item is not actually removed, but only skipped by advancing the pointer (in)to the transaction.) Hence it suffices to have one global copy of all transactions, which is merely referred to in different ways from different arrays used in the recursive processing. Note also that the merge result may be created in the array that represented the original (conditional) database, since its front elements have been cleared in the split step. In addition, the array for the split database can be reused after the recursion for the

Pseudo-code:

```
function HCRA (a: array of transactions ,          //conditional database to process//
               p: set of items,                    //prefix of the conditional database a//
               smin: int) : int                    // minimum support of an item set//
var i: item;                                       //buffer for the split item//
    s: int;                                        //support of the current split item//
    n: int;                                        // number of found frequent item sets//
    b; c; d: array of transactions;                //split result and merged database//
begin                                              //split and merge recursion//
  n := 0;                                          // initialize the number of found item sets //
while a is not empty do                            //while conditional database is not empty //
    b := empty; s := 0;                            // initialize split result and item support //
    i := a[0].items[0];                            //get leading item of the first transaction //
```

```
while a is not empty and a[0].items[0] = i do   //and split database w.r.t. this item //
      s := s + a[0].wgt;                          // sum the occurrences (compute support) //
remove i from a[0].items;                  //remove the split item from the transaction //
if      a[0].items is not empty                  //if the transaction has not become empty //
then   remove a[0] from a and append it to b;
else    remove a[0] from a; end ;                //move it to the conditional database, //
end;                                             //otherwise simply remove it//
c := b; d := empty;                        // note split result, init. the output array//
while a and b are both not empty do              // merge split and rest of database//
if      a[0].items > b[0].items            //copy lex. smaller transaction from a//
then    remove a[0] from a and append it to d;
else if  a[0].items < b[0].items                 //copy lex. smaller transaction from b //
then    remove b[0] from b and append it to d;
else    b[0].wgt := b[0].wgt +a[0].wgt;     //sum the occurrence counters/weights//
        remove b[0] from b and append it to d;
        remove a[0] from a;                      //move combined transaction and //
  end ;                                    //delete the other, equal transaction: //
end;                                             //keep only one instance per transaction //
while  a is not empty do                   //copy the rest of the transactions in a //
    remove a[0] from a and append it to d; end;
while b is not empty do                          //copy the rest of the transactions in b//
    remove b[0] from b and append it to d; end;
a := d;                                  //second recursion is executed by the loop//
if s >= smin then                                // if the split item is frequent: //
   p := p ∪ {i};                         // extend the prefix item set and //
   report p with support s;                      // report the found frequent item set //
   n := n + 1 + HCRA(c; p; smin);        // process the created database recursively //
   p := p – {i};                                 //and sum the found frequent item sets, //
  end ;                                  // then restore the original item set prefix //
end;
return n;                                  //return the number of frequent item sets //
end;                                     //function HCRA() //
```

Fig. 3. Pseudo-code of the HCRA algorithm.

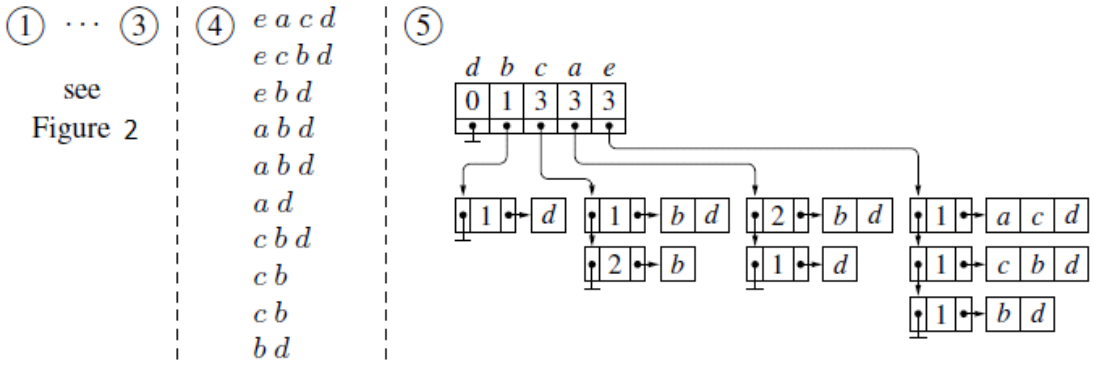


Fig. 4. Preprocessing the example database Structure.

split w.r.t. to the next item. As a consequence, each recursion step, which expands the prefix of the conditional database, only needs to allocate one new array, with a size that is limited to the size of the input array of that recursion step. This makes the algorithm not only simple in structure, but also very efficient in terms of memory consumption. Finally, note that due to the fact that only a simple array is used as the data structure, the algorithm can fairly easily be implemented to work on external storage or a (relational) database system. There is, in principle, no need to load the transactions into main memory and even the array may easily be stored as a simple (relational) table. The split operation can then be implemented as an SQL select statement, the merge operation is very similar to a join, even though it may require a more sophisticated comparison of transactions (depending on how the transactions are actually stored).

2. Which algorithms are compared?

In this section we describe the software implementations of the association rule algorithms used in our experiments. The four algorithms evaluated were Apriori, FP-growth, Scaled association rule algorithm and HCRA algorithm. We provide references to articles describing the details of the algorithm when available and also specify the algorithms' parameter settings used in our experiments (if any).We started the experiments several months ago and published pHCRAinary results to the authors of the algorithms. Several authors provided us with an updated version of their code to fix bugs and/or improve the performance. We reran our experiments with the new versions and noted below when updated versions were received. The reasons for comparing these algorithms are :

i)Flexibility
ii)Popularity
iii)Applicability
iv)Types of dataset used

2.1 Aprori algorithm :

The apriori algorithm [1] is one of the earliest algorithms for mining association rules and has became the standard approach in this area. The search for association rules is guided by two parameters: support and confidence.Apriori returns an association rule if its support and confidence values are above user defined threshold values. The output is ordered by confidence. If several rules have the same confidence, then they are ordered by support. Thus apriori favors more confident rules and characterises these rules as more interesting.The apriori Mining process is composed of two major steps. The first one (generating frequent item sets) was discussed briefly in the last section. This step can be seen as supportbased pruning, because only item sets with at least minimum support were considered. The second step is the generation of rules out of the frequent item sets. In this step confidencebased pruning is applied.Rule discovery is straightforward. For every frequent item set f and every non-empty subset s of f, apriori outputs a rule of the form $s \Rightarrow (f - s)$ if and only if the confidence of that rule is above the user specified threshold. e task of discovering association rules consists in finding all the association rules having a minimum support minsup and a minimum condence minconf . The task of discovering association rules consists in finding all the association rules having a minimum support minsup and a minimum confidence minconf.

Apriori is Christian Borgelt's implementation of the wellknown Apriori association rule algorithm [1][2]. The source code in C for this implementation is available under the "GNU Lesser General Public License" from http://fuzzy.cs.unimagdeburg.de/~borgelt/. Apriori takes transactional data in the form of one row for each pair of transaction and item identifiers. It first generates frequent itemsets and then creates association rules from these itemsets. It can generate both association rules and frequent itemsets. Apriori supports many different configuration settings. In our experiments, we used the percentage of transactions that satisfy both the LHS and the RHS of a rule as the support. We also specified that Apriori should load the entire dataset into memory rather than making multiple database scans.The running Apriori using multiple database scans would be significantly slower. Apriori is the first algorithm to use Apriori-gen for candidate generation. As mentioned in the previous paragraph, Apriorigen is separate from the counting step that determines the frequency of each current candidate. This means that each pass of Apriori consists of a call to Apriori-gen togenerate all candidates of a given size (size k in pass k) and a counting phase that determines the support for all these candidates. Each counting phase scans the entire database. Upon reading a transaction T in the counting phase of pass k, Apriori has to determine all the kcandidates supported by T and increment the support counters associated with these candidates.In order to perform this operation efficiently,Apriori stores candidate item-sets in a tree. The actual itemsets are stored in the leaves of the tree, and edges are labeled with items. To find the proper location for a candidate, starting from the root, traverse the edge with the first item in the set. Reaching an internal node, choose the edge labeled with the the next item in the set, until a leaf is reached. The path to locate set is marked with thickened arrows in the figure. Note that by virtue of ordering the items, each set has its unique place in the tree. The smallest items in a set that are used along the path to the leaf need not be stored. Inserting item-sets into the tree can cause a leaf node to overflow, in which case it is split and the tree grows. To count all candidates for transaction T, all leaves that could contain a candidate have to be searched, and to reach all these leaves, Apriori tries all possible combinations of the items in T as paths to a leaf. Once a leaf with a set of candidates is located in this fashion it remains to be checked which are actually supported by the transaction. As far as the implementation is concerned, this test for set inclusion can be optimized by storing the item-sets as bitmaps, one bit for each item. As observed in [1], these bitmaps can become quite large for many items (128 Bytes for 1000 items) and cause considerable overhead. Internal nodes are implemented as hash tables to allow fast selection of the next node. To reach the leaf for a set, start with the root and hash on the first item of the set. Reaching the next internal node, hash on the second item and so on until a leaf is found. Item-lists The major problem for Apriori (and for AIS as well) is that it always has to read the entire database in every pass, although many items and many transactions are no longer needed in later passes of the algorithm.

In particular, the items that are not frequent and the transactions that contain less items than the current candidates are not necessary. Removing them would obviate the expensive effort to try to count item-sets that cannot possibly be candidates. Apriori does not include these optimizations, moreover they would be hard to add to Apriori (andAIS as well). The reason

stems from the item-list data representation used by both algorithms. At before, transactions are stored as a sequence of sorted item-lists in this representation. While item- lists are the most common representation and the one that is usually assumed as input format, they make it difficult to remove unnecessary parts of the data. Let's assume we want to remove all items that are not part of any frequent set. Unfortunately, the knowledge of which items to keep and which to discard is only available and applicable after scanning the database to count the support for the candidates. Therefore, we can eliminate items only in the subsequent pass over the data, that is they have to be read once more, although this is not really necessary. As we will see later, the other two representations remove these items instantly, which leads to much smaller data sizes in later passes; unfortunately this is not the case for early passes, where the volume of intermediate data representations can exceed the original data size. The advantage of item-lists is therefore that the size of the data does not grow in the course of the algorithms.

2.2 FPGrowth algorithm :

The FPGrowth method constructs FP-tree which is a highly compact form of transaction database. Thus both the size and the cost of computation of conditional pattern bases, which corresponds roughly to the compact form of projected transaction databases, are substantially reduced. Hence, FPGrowth mines frequent itemsets by (1) constructing highly compact FPtrees which share numerous "projected" transactions and hide (or carry) numerous frequent patterns, and (2) applying progressive pattern growth of frequent 1-itemsets which avoids the generation of any potential combinations of candidate itemsets implicitly or explicitly, whereas Apriori must generate more number of candidate itemsets for each projected database. Therefore, FPGrowth is more efficient and more scalable than Apriori, especially when the number of frequent itemsets becomes really large. FP-growth is an algorithm for generating frequent itemsets for association rules from Jiawei Han's research group at Simon Fraser University. It generates all frequent itemsets satisfying a given minimum support by growing a frequent pattern tree structure that stores compressed information about the frequent patterns. In this way, FP growth can avoid repeated database scans and also avoid the generation of a large number of candidate itemsets[4].The FP tree algorithm addresses these issues and scans the data in a depth-first way. The data is only scanned twice.In the first scan, the frequent items (or 1-itemsets) are determined. The data items are then ordered based on their frequency and the infrequent items are removed. In the second scan, the data base is mapped onto a tree structure. The FP tree does never break a long pattern into smaller patterns the way the Apriori algorithm does. Long patterns can be directly retrieved from the FP tree. The FP tree also contains the full relevant information about the data base. It is compact, as all infrequent items are removed and the highly frequent items share nodes in the tree. The number of nodes is never less than the size of the data base measured in the sum of the sizes of the records but there is anecdotal evidence that compression rates can be over 100. FP-Growth: allows frequent itemset discovery without candidate itemset generation. Two step approach:

Step 1: Build a compact data structure called the FP-tree.I Built using 2 passes over the data-set.

Step 2: Extracts frequent itemsets directly from the FP-tree Traversal through FP-Tree.

2.3 Scaled Association Rules algorithm

Step 1: Input Phase The distribution of attribute values in the clusters was used for making the discretization according to the following procedure:

1. The number of intervals for each attribute is the same of the number of clusters where m is the mean value of the attribute in the in the clusters.

2. When two adjacent intervals overlap, the cut point (superior boundary of the first and inferior boundary of the next) is placed in the middle point of the overlapping region. These intervals are merged into a unique interval 3. When two adjacent intervals are separated, the cut point is placed in the middle point of the separation region. This procedure was applied for creating intervals of values for every one of the
attributes in order to generate the association rules.

Step 2: Candidate Generation Given Lk -1, the set of all frequent -itemsets, the candidate
generation procedure must return a superset of the set of all frequent k-itemsets. The k-means clustering helps in finding the appropriate and definite cluster with partitioning. This procedure has three parts:

a) Join Phase. Lk - 1 is joined with itself, the join condition being that the lexicographically ordered first k – 2 items are the same, and that the attributes of the last two items are different.

b) Subset Prune Phase. In this phase all itemsets from the join result which have some (k – 1)-subset that is not in Lk-1 are deleted.

c) Interest Prune Phase. If the user specifies an interest level and wants only itemsets whose support and confidence is greater than expected, the interest measure is used to prune the candidates further.

Step 3: Counting Support of Candidates. In the process of counting support of candidates when we make a pass, we read one record at a time and increment the support count of candidates supported by the record. Thus, given a set of candidate itemsets C and a record t, we need to find all itemsets in C that are supported by t. We partition candidates into groups such that candidates in each group have the same attributes and the same values for their categorical attributes.

Step 4: Generating Rules. We use the frequent itemsets to generate association rules. The general idea is that RTYZ and RT are frequent itemsets, then we can determine if the rule RTYZ holds by computing the ratio conf = support (RTYZ)/support (YZ) . If conf >= supconf, then the rule will have minimum support because RTYZ is frequent. The clusters are created with a weight for the output. This is a supervised way of producing the most suitable clusters for the prediction of the output variables, which appear in the consequent part of the rules generation.

3. How algorithms are compared ?

The aim of the work is to obtain an associative model that allows studying the influence of the input variables related to the project management policy on the output variables related to the software product and the software process. The rules generated were created with a weight for the output variables three times greater than for input attributes. This is a supervised way of producing the most suitable clusters for the prediction of the output variables, which appear in the consequent part of the rules.

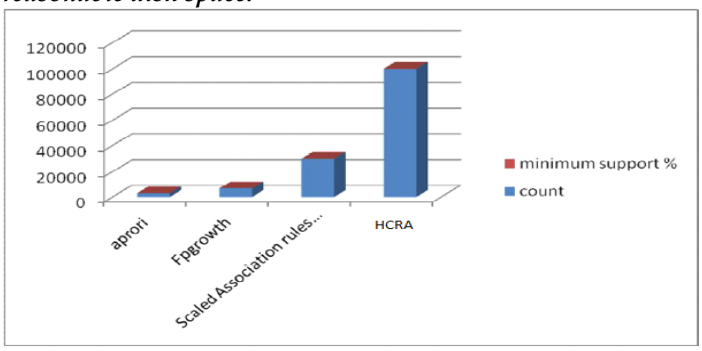
The four algorithms are compared on the following factors :
Candidate Generation
Support Counting
Frequent itemset generation
Computational Complexity
Rule generation
Type of dataset used

	Size of dataset	*Type of rules generated*	*Type of dataset*
Aprori	*Small and large dataset*	*Minimum support*	*One dimensional*
FP-Growth	*Small and large dataset*	*High confidence*	*One dimensional*
Scaled Association rules algorithm	*Large quantitative dataset*	*High support and confidence*	*Multidimensional*
HCRA	*Large quantitative dataset*	*High support and confidence*	*Multidimensional*

4. How algorithms are implemented ?

Each of the four algorithms described in Section 3 was tested on the four datasets described in Section 4. The performance measure was the execution time (seconds) of the algorithms on the datasets with the following minimum support settings 5.00%,0.80%, 0.60%, 0.70%, 0.20%, 0.50%, 0.08%, 0.06%, 0.04%,0.02%, and 0.01%. The minimum confidence was always set to zero. That is, we required no minimum confidence for the generated association rules. Since some of the algorithms could only generate frequent itemsets, and some others could directly generate association rules, we measured the execution time for both creating the frequent itemsets and for creating the association rules whenever possible. Note that time for generating the association rules includes the computation for generating the frequent itemsets. The experimental study is carried on dynamic simulation environment CBA which is a data mining tool. Its main algorithm was presented as a plenary paper "Integrating Classification and Association Rule Mining" in the 4th International Conference on

Knowledge Discovery and Data Mining. However, it turns out that it is more powerful than simply producing an accurate classifier for prediction. It can also be used for mining various forms of association rules, and for text categorization or classification. This environment manages data from real projects developed in local companies and simulates different scenarios. It works with more than 18 input parameters and more than 10 output variables and generates 1569091 rules. The number of records generated fornthis work is 400 and the variables used are sepal length, sepalwidth from the Iris dataset. We also found that the choice of algorithm only matters at support levels that generate more rules than would be useful in practice.For a support level that generates a small enough number of rules that a human could understand, Apriori finishes on all datasets in under a minute, so performance improvements are not very interesting. Even for support levels that generate around 2,000,000 rules, which is far more than humans can handle and is typically sufficient for prediction purposes, Apriori finishes processing in less than 10 minutes. Beyond this level of support,the number of frequent itemsets and association rules grows extremely quickly on the real-world datasets, and most algorithms quickly run out of either memory or reasonable disk space.



HCRA generated 350,000 and 500,000 with minimum support of 0.40% respectively.HCRA algorithm terms of the number of closed frequent itemsets in some experiments, and the difference is large in some cases. For example, for the IBM-Artificial dataset, with a minimum support of 0.40%, However, with a minimum support of 0.01% they generated 303,610 and 283,397 closed frequent items respectively, a difference of 20,213.

5. Conclusion

We studied the algorithms for mining quantitative association rules. Our study showed that the algorithm scales linearly with the number of records. In addition, the proposed method avoids three of the main drawbacks presented by the rule mining algorithms: production of a high number of rules, discovery of uninteresting patterns and low performance. The results show that the association rule algorithms that we evaluated perform differently on our real-world datasets than they do on the artificial dataset. The performance improvements reported by previous authors can be seen on the artificial dataset, but some of these gains do not carry over to the real datasets, indicating that these algorithms overfit the artificial dataset.The primary reason for this seems to be that the artificial dataset has very different characteristics, suggesting the need for researchers to improve the artificial datasets used for association rule research or use more realworld datasets.

6.Future Scope

This paper was intented to compare between the standard association rule algorithms with the proposed HCRA algorithm.As a future work,comparisons can be made according to different factors other than those considered in the paper.

REFERENCES

[1] J. P. Bigus., "Data Mining with Neural Networks", McGraw-Hill, 1996.
[2] T. M. Mitchell., "Machine Learning", McGraw-Hill, 1997.
[3] Fayyad U, "Data Mining and Knowledge Discovery in Databases: Implications from scientific databases," In Proc. of the 9th Int. Conf. on Scientific and Statistical Database Management, Olympia, Washington, USA, pp. 2-11,1997.
[4] Tsau Young Lin, "Sampling in association rule mining", Conference on Data mining and knowledge discovery: Theory, Tools, and Technology VI,vol.
[5] F. Bodon, "A Fast Apriori Implementation", In B. Goethals and M. J.Zaki, editors, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Vol. 90 of CEUR Workshop Proceedings, 2003.
[6] Basel A. Mahafzah, Amer F. Al-Badarneh and Mohammed Z. Zakaria "A new sampling technique for association rule mining," in Journal Of Information Science, Vol.35, pp. 358-376, 2009.
[7] Wang, C., Tjortjis, C., Prices: An Efficient Algorithm for Mining Association Rules, Lecture Notes in Computer Science, Volume 2447, 2002. pp. 77-83.
[8] Wojciechowski, M., Zakrzewiez, M., Dataset filtering Techniques in Constraint based Frequent pattern Mining, Lecture Notes in Computer Science, Volume,2447, 2002, pp77-83.

[9] V.Umarani and M.Punithavalli," Developing a Novel and Effective Approach for Association Rule Mining Using Progressive Sampling" In the proc of 2nd Int'l Conference on Computer and Electrical Engineering (ICCEE 2009), vol.1, pp610-614.
[10] V.Umarani and M.Punithavalli," On Developing an Effectual Progressive Sampling Based Approach for Association Rule Discovery" In the proc of 2nd IEEE Int'l Conference on Information and data Engineering (2nd IEEEICIME 2010), Chengdu ,China April 2010.

N.CHANDRA SEKHAR REDDY is working as Professor & HOD Dept of CSE , SPEC HYDERABAD
He received his B.E and M.E in Computer Science Engineering. Currently a research scholar
[naguchinni@gmail.com]

DR.PEDAKOLMI , Working as Professor of CSE at NIT-M , Hyderabad , received his Ph.D in Computer Science Engineering[dr.pelakolmi@gmail.com]

DR A.GOVARDHAN , Working as Professor CSE and DE – JNTUH , received his Ph.D in Computer Science Engineering[govardhan_cse@yahoo.co.in]

N.VAMSI KRISHNA , completed B.TECH and M.TECH from JNTUH and currently working as Asst.professor in dept of CSE HITS COE , HYDERABAD [vkcse@outlook.com]

N.L.MANASA completed B.TECH from JNTUH and M.S from University of Houston , U.S.A . Currently a research scholar and working as Asst.professor in CSE at MJCET Hyderabad [nadipally.m@gmail.com]