Finding Association Rules with some very frequent attributes

Frans Coenen and Paul Leng

Department of Computer Science, The University of Liverpool,
Chadwick Building, P.O. Box 147, Liverpool L69 3BX, England.
{frans,phl}@csc.liv.ac.uk

Abstract. A key stage in the discovery of Association Rules in binary databases involves the identification of the "frequent sets", i.e. those sets of attributes that occur together often enough to invite further attention. This stage is also the most computationally demanding, because of the exponential scale of the search space. Particular difficulty is encountered in dealing with very densely-populated data. A special case of this is that of, for example, demographic or epidemiological data, which includes some attributes with very frequent instances, because large numbers of sets involving these attributes will need to be considered. In this paper we describe methods to address this problem, using methods and heuristics applied to a previously-presented generic algorithm, Apriori-TFP. The results we present demonstrate significant performance improvements over the original Apriori-TFP in datasets which include subsets of very frequently-occurring attributes.
Keywords: Association Rules, Frequent sets, Dense data

1 Introduction

Association rules [2] are observed relationships between database attributes, of the form "if the set of attributes A is found in a record, then it is likely that B will be found also". More formally, an association rule R takes the form $A \rightarrow B$, where A, B are disjoint subsets of the attribute set. Usually, a rule is thought to be "interesting" if at least two properties apply. First, the *support* for the rule, that is, the number of records within which the association can be observed, must exceed some minimum threshold value. Then, if this is the case, the *confidence* in the rule, which is the ratio of its support to that of its antecedent, must also exceed a required threshold value. Other measures, (e.g. *lift* [6], or *conviction* [7]) have also been proposed to provide further definition of the interest in a potential rule. All work on association-rule mining, however, has recognised that the identification of the *frequent* sets, the support for which exceeds the required threshold, is a necessary stage, and also that this is computationally the most demanding, because of the inherently exponential nature of the search space.

Most work on association rule discovery has focused in particular on its application in supermarket shopping-basket analysis. This, however, is not the most demanding problem domain. In other applications, such as census data, there may be many attributes (for example "female", "married",etc.) which occur in a very high proportion of records. The correspondingly high frequency of combinations including these attributes gives rise to very large *candidate sets* of attributes that potentially exceed the support threshold, causing severe problems for methods such as Apriori [3]. The problem can be reduced by setting the support threshold at a sufficiently high level, but this will risk eliminating potentially interesting combinations of less common attributes.

In this kind of data, associations involving only the most common attributes are likely to be obvious and therefore not genuinely interesting. In this paper, therefore, we describe a method which seeks to identify only those frequent sets which involve at least one "uncommon" attribute. This reduction makes it possible to employ heuristics which can reduce the computational cost significantly. We describe the algorithms we have used, and present results demonstrating the performance gains achieved in dealing with data which includes a proportion of very frequent attributes.

2 Finding frequent sets

We will begin by reviewing the well-known and seminal "Apriori" algorithm of [3]. Apriori examines, on successive passes of the data, a candidate set C_k of attribute sets the members of which are all those sets of k attributes which remain in the search space. Initially, the set C_1 consists of the individual attributes. Then, the kth cycle proceeds as follows (for k=1,2,.. until C_k = empty set):

1. Perform a pass over the database to compute the support for all members of C_k.
2. From this, produce the set L_k of frequent sets of size k.
3. Derive from this the candidate set C_{k+1}, using the *downward closure* property, i.e. that all the k-subsets of any member of C_{k+1} must be members of L_k.

Apriori and related algorithms work reasonably well when the records being examined are relatively sparsely populated, i.e few items occur very frequently, and most records include only a small number of items. When this is not so, however, the time for the algorithm increases exponentially. The principal cost arises in step 1, above, which requires each database record to be examined, and all its subsets that are members of the current candidate set to be identified. Clearly, the time for this procedure will in general depend both on the number of attributes in the record and on the number of candidates in the current set C_k. If the database is densely-populated, the size of the candidate sets may become very large, especially in the early cycles of the algorithm, before the "downward closure" heuristic begins to take effect. Also, any record including a large number of attributes may require a large subset of the candidate set to be examined: for example, the extreme case of a record containing all attributes will require all candidates in the current set to be inspected.

A number of strategies have been adopted to reduce some of the inherent performance costs of association-rule mining in large, dense, databases. These include methods which reduce the scale of the task by working initially with a subset or sample of the database [14],[15]; methods which look for *maximal* frequent sets without first finding all their frequent subsets [4],[5]; methods which redefine the candidate set dynamically [7], [10], and methods which are optimised for dealing with main-memory-resident data [1]. No method, however, offers a complete solution to the severe scaling of the problem for dense data.

A particular case that causes difficulty for Apriori and related methods occurs when the attribute set includes a possibly quite small subset of very frequently-occurring attributes. Suppose, for example, there is a subset F of attributes each of which is present in about 50% of all records. Then if the support threshold is set at 0.5%, which may be necessary to find interesting combinations of scarce attributes, it is likely that most of the combinations of attributes in F will still be found in L_7. Not only will this lead to a large number of database passes to complete the count, but also the candidate sets will be inflated by the continuing presence of these combinations. For example, if F contains only 20 attributes, C_8 is likely to include about 125,000 candidates which are combinations of these only, and for 40 attributes, the size of C_8 may exceed 76×10^6.

We have described previously [12] a method we have developed which reduces two of the performance problems of Apriori and related algorithms: the high cost of dealing with a record containing many attributes, and the cost of locating relevant candidates in a large candidate-set. In the following section we will briefly summarise this method, before going on to describe adaptations of this approach to address the problems of datasets such as the one outlined above.

3 Computing via Partial Support

The methods we use begin by using a single database pass to restructure the data into a form more useful for subsequent processing, while at the same time beginning the task of computing support-counts. Essentially, each record in the database takes the form of a set of attributes i that are represented in the record. We reorganise these sets to store them in lexicographic order in the form of a set-enumeration tree [13]. Records that are duplicated in the database occur only once in the tree, and are stored with an associated incidence-count. As the tree is being constructed, it is also easy and efficient to add to this count, for each set i stored, the number of times i occurs as a subset of a record which *follows* i in the set ordering. We use the term *P-tree* to refer to this set-enumeration tree with its associated partial support-counts. A detailed description of the algorithm for building the P-tree is given in [9]. The construction is simple and efficient, and both the tree size and construction time scale linearly with the database size and density.

The P-tree thus constructed contains all the sets present as records in the original data, linked into a structure in which each subtree contains all the lexicographically following supersets of its parent node, and the count stored at a parent node incorporates the counts of its subtree nodes. The concept is similar to that of the FP-tree of [10], with which it shares some properties, but the P-tree has a simpler and more general form which offers some advantages. In particular, it is easy and convenient to convert the P-tree to an equivalent tabular form, explained below. Thus, although for simplicity our experiments use data for which the P-tree is store-resident, the structures are simple enough also to enable straightforward and efficient implementations in cases when this is not so. Results presented in [12] also show the memory requirement for the P-tree to be significantly less than that for the FP-tree.

The generality of the P-tree makes it possible to apply variants of many existing methods to this to complete the summation of required support-totals. We have experimented with a method which applies the Apriori procedure outlined in the previous section to the tabulated P-tree nodes, rather than to the records in the original database. We store candidates whose support is to be counted in a second set-enumeration structure, the T-*tree*, arranged in the opposite order to that of the P-tree. Each subtree of the T-tree stores predecessor-supersets of its root node. The significance of this is that it localises efficiently those candidates which need to be considered when we examine the subsets of a P-tree node. The algorithm we use, which we call Apriori-TFP, is described in detail in [12]. Apriori-TFP follows the Apriori methodology of performing repeated passes of the data, in each of which the support for candidates of size k is counted. The candidates are stored on the T-tree, which is built level by level as each pass is performed, and pruned at the end of the pass to remove candidates found not to be frequent. Each pass involves a complete traversal of the P-tree. Because the order in which this happens is irrelevant, it is possible to store the P-tree in a node-by-node tabular form. An entry in this table representing a set $ABDFG$, say, present in the tree as a child of $ABDF$, would be stored in a form $ABDF.G$, (with an associated count). When this entry is examined in the second pass, for example, this count is added to the totals stored in the T-tree for the pairs AG, BG, DG and FG, all of which, assuming they remain in the candidate set, will be found in the branch of the T-tree rooted at G. Notice that, in contrast with Apriori, we do not need to count the other subsets AB, BD, etc., at this point, as the relevant totals will be included in the count for the parent set $ABDF$ and counted when that node and its ancestors are processed. It is these properties that lead to the performance gain from the method. The advantage gained increases with increasing density of the data.

4 Heuristics for dense data

Notwithstanding the gains achievable from using the P-tree, very dense data poses severe problems for this as for other methods. Its performance can be improved by ordering the tree by descending frequency of attributes [8]. The extent of the improvement depends on the nature of the data; for the "mushroom" dataset of [16], for example, the gain was about 50%. This heuristic points us to a strategy for dealing with data including some very frequent attributes. Suppose that there is a subset F of such attributes, and let us assume that we have no interest in deriving rules which include only members of F. We still, of course, wish to investigate rules which associate subsets of F with one or more of the less common attributes, and for this purpose will need to compute the support for at least some of the sets in the power set of F.

We begin by constructing the P-tree as before, with an ordering of attributes that places the members of F first. From this, we construct an *incomplete* T-tree, to include only candidate sets that contain at least one attribute not in F. The form of this tree is illustrated in Figure 1, for a set of attributes $\{A, B, C, D, E\}$, of which A, B and C are in the set F. Note again that the tree is ordered so that each subtree contains only the lexicographically preceding supersets of its parent node. The branches rooted at A, B and C, which would be present in the full T-tree, are in this case omitted.

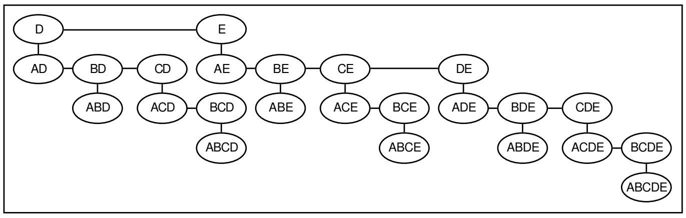

Fig 1: Incomplete T-tree for attributes $\{(A, B, C), D, E\}$

Apart from the omission of the sets derived only from F, the tree shown in Figure 1 is complete. In the actual implementation, however, it is constructed so as to contain, finally, only the frequent sets. The algorithm Apriori-TFP for doing this builds the tree level by level via successive passes of the P-tree table. In each pass, the level k currently being considered contains the candidate set C_k defined as for Apriori. The support for each set in C_k is counted and attached to the corresponding node of the T-tree. At the end of the pass, sets found not to be frequent are removed, leaving the sets in L_k on the tree, and the next level of the tree is built using the Apriori heuristic.

Although the tree we have illustrated will not contain the support-counts for the members of the "very frequent" subset F, we may still require to know these values. Firstly, when a level is added to the tree, we wish to include nodes only if all their subsets are frequent, including those subsets that contain only members of F and thus are not in the tree. Finally, also, we will need to know the support of all of the subsets of F that are included as subsets of sets in the T-tree so that we can compute the confidence for each possible association that may result from sets in the T-tree. Because of the way we have constructed the P-tree, however, there is a very efficient procedure for computing these support-totals exhaustively, provided F is small enough for all these counts to be contained in an array in main memory. Associated with each set i in the P-tree is an incomplete support-count Q_i. A "brute-force" algorithm to compute the final "total" support-counts T_i may be described thus:

```
Algorithm ETFP (Exhaustive Total- from Partial- supports)
        for each node j in P-tree do
        if j ⊆ F then
        begin k = j - parent (j);
              for each subset i of j with k ⊆ i do
                  add Q_j to T_i;
        end
```

Notice again that because the incomplete support-count Q_i for a parent node i incorporates the counts for its children, we need only consider subsets of a node that are not subsets of its parent. If the counts T_i can be contained in a simple array (initialised to zero), the count-update step is trivial. Also note that because of the ordering of the P-tree, the sets to be counted are clustered at the start, so not all the tree need be traversed to find them.

We can now describe methods for computing all the support-totals we need to determine the frequent sets. In every case, we begin by constructing the P-tree.

Method 1

1. Use Algorithm ETFP to count the support for all subsets of F.
2. Convert the P-tree to tabular form, omitting subsets of F.
3. Use Algorithm Apriori-TFP to find all the frequent sets that include at least one attribute not in F, storing these in an incomplete T-tree of the form illustrated in Figure 1.

As each new level is added to the T-tree during step 3, we include candidates only if all their subsets are frequent, which can be established by examining the support counts stored at the current level of the T-tree and, for the subsets of F, in the array created in step 1. However, because of the high probability that the latter are indeed frequent, it may be more efficient to assume this; thus:

Method 2

As Method 1, but in step 3, assume that all subsets of F are frequent. This may occasionally result in candidates being added unnecessarily to the tree, but will reduce the number of checks required as each new level is added.

Notice that in fact these methods do find all the frequent sets, including those that contain only attributes from F. In comparison with the original Apriori-TFP, the cost of building the T-tree is reduced both because of the smaller number of candidates in the reduced T-tree, and because when examining sets in the P-tree table, only those subsets containing a "scarce" attribute need be considered (and subsets of F are left out of the table). These gains more than compensate for the cost of exhaustively counting support for the very frequent attributes by the efficient ETFP algorithm, provided the size of F is relatively small. If there are more than about 20 very frequent attributes, however, the number of combinations of these becomes too great for exhaustive counting to be feasible. In this case, a third method may be applied — method 3.

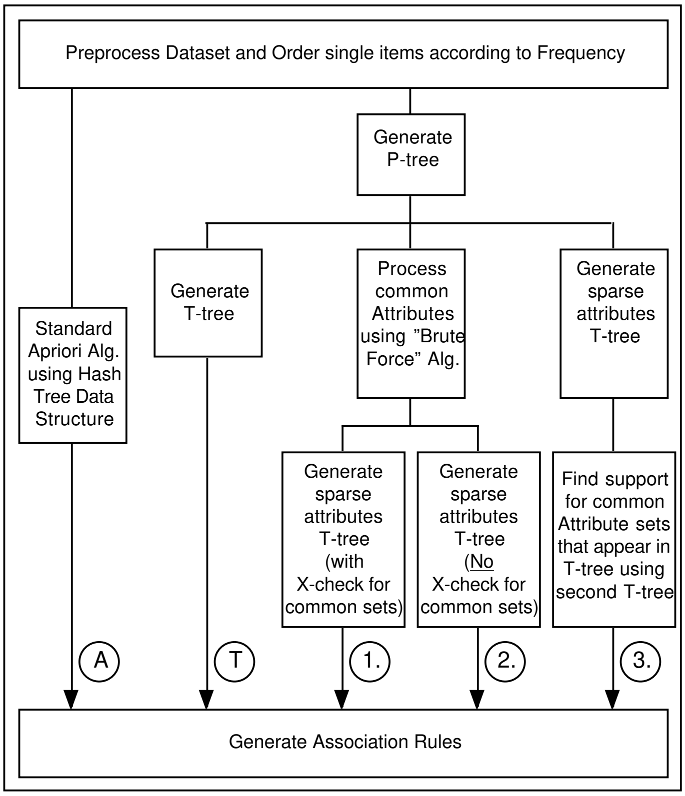

Fig 2: Summary of methods used

Method 3

1. Convert the P-tree to tabular form.
2. Use Algorithm Apriori-TFP, as before, to find all the frequent sets that include at least one attribute not in F, storing these in a T-tree. As with Method 2, we assume all subsets of F are (probably) frequent.
3. Traverse the T-tree to extract all the subsets of F that are included in frequent sets with scarce attributes, storing these on a second T-tree.
4. Perform a single pass of the part of the P-tree table that contains the subsets of F, to count the support for those that are left in the second T-tree.

The effect of this is to count the support of subsets of F only if they are associated in frequent sets with the "scarce" attributes. Because this will not be so for most of the larger subsets, this is likely to be feasible even when F is too large to be exhaustively enumerated. The result at step 2 of this method is similar to what would be achieved using the multiple support threshold algorithm MSapriori [11], with the higher support threshold set at 100%. In this case, however, MSapriori would not count the support for the subsets of F needed to complete the derivation of rules. In other respects, also, MSapriori shares the problems of the original Apriori, with further overheads added by the need to process multiple support thresholds.

The three methods above are summarised in Figure 2, in comparison with the "standard" Apriori algorithm (labelled A), and our original Apriori-TFP algorithm (labelled T). In the following section we will compare the performance of the 5 methods outlined in Figure 2.

5 Results

To examine the performance of the methods we have described, we have applied them to datasets generated using the QUEST generator described in [3]. This defines parameters N, the number of attributes of the data, T, the average number of attributes present in a record, and I, the largest number of attributes expected to be found in a frequent set. For the purpose of these experiments, we began by generating a dataset of 250,000 records with $N = 500$, $T = 10$, and $I = 5$. This gives rise to a relatively sparse dataset, typical of that used in experiments on shopping-basket data. To create a more challenging dataset, we also performed a second generation of 250,000 records, with $N = 20$, $T = 10$, and $I = 5$. The two sets were merged, record-by-record, to create a final dataset of 250,000 records with 520 attributes, within which the first 20 attributes are each likely to occur in about 50% of all records. In a real case, of course, the distinction between the "very common" and "less common" is likely to be less clear-cut, and may be influenced by a subjective assessment of which attributes are of most interest. For methods 1 and 2, however, the number of attributes which can be included in the set F is limited by the size of the array needed to store all their combinations, a practical limit of about 20. We examine below the case in which F is larger than this.

Figure 3 illustrates the performance of the basic Apriori-TFP algorithm on this dataset, in comparison with the original Apriori and with the FP-growth algorithm of [10], for varying support thresholds down to 1%. All methods are our own implementations (in Java), intended to give as fair a comparison as we are able of the approaches. The graphs show the total time required in each case to determine the frequent sets, for Apriori (labelled A), FP-growth (labelled F) and Apriori-TFP (labelled T). In the latter case, this includes the time required to construct the P-tree. As we have shown in earlier work [12], Apriori-TFP strongly outperforms the original Apriori. In the present case, the difference is particularly extreme, because of the higher density of data produced as a result of the inclusion of the 20 "very frequent" attributes. The consequent large candidate sets, with the typical record-length of 20 attributes, requires in Apriori large numbers of subsets to be counted and a high level of expensive hash-tree traversal. In our implementation, the time required for this becomes prohibitive for the lower support thresholds. As we have shown in [8], the improvement shown by Apriori-TFP is maximised when, as in this case, the most common attributes are placed at the start of the set order.

The comparison with FP-growth is much closer, as we would expect from methods which share similar advantages. This is shown more clearly in Figure 4, which also illustrates the performance of the three variants we described in the previous section, for the dataset described above, (this time with a linear scale on the graph). All methods show the relatively sharp increase in time required as the support threshold is dropped and the number of candidates to be considered increases correspondingly. As can be seen, FP-growth matches the basic Apriori-TFP method for higher support thresholds, but the latter begins to perform better at lower thresholds. We believe this is because of the overheads of the relatively complex recursive tree-generation required by FP-growth, compared with the rather simple iteration of Apriori-TFP.

Fig 3: Comparison of Apriori (A), FP-growth (F) and Apriori-TFP (T)
($T10.I5.D250000.N20$ merged with $T10.I5.D250000.N500$)

Curve 1 in Figure 4 shows, in comparison, the time taken when the combinations of the 20 most common attributes are counted exhaustively (Method 1). For high support thresholds, this is slower than Apriori-TFP, because we are counting many combinations which are not in fact frequent. However, for support thresholds below about 3%, in this data, this is more than compensated by the more efficient counting method of ETFP, leading to a significant performance gain. The performance of Method 2 (Curve 2 in Figure 4) is similar; in fact, Method 2 slightly outperformed Method 1, although the difference was not significant and is not apparent in the graph. At these support thresholds, the assumption that all relevant combinations of the "very frequent" attributes are frequent is almost always correct, but the gain from this is slight because, with relatively small candidate sets, the cost of cross-checking is not a major factor.

Curve 3 in Figure 4 shows the performance of Method 3, in which the support for combinations of the 20 most frequent attributes was counted only when they appeared in frequent sets with less common attributes. Here, the results show a consistent performance gain over all the other methods.

Finally, we examine the performance of our methods in cases where the number of "very frequent" attributes is greater. For this purpose, we again began with the dataset of 250,000 records with $N = 500$, $T = 10$, and $I = 5$. In this case, we merged this with a second set of 250,000 records with $N = 40$, $T = 20$, and $I = 10$. The result is to create a set of records with 540 attributes, the first 40 of which have a frequency of about 50%, and for which the average record-length is about 30 attributes. This relatively dense data is far more demanding for all methods. Figure 5 shows the results of our experiments with this data. Again, the curve labelled T illustrates the performance of the basic Apriori-TFP (in all cases, we have continued the curves only as far as is feasible within the time-frame of the graph).

Fig 4: Performance for $T10.I5.D250000.N20$ merged with $T10.I5.D250000.N500$

For more than about 20 attributes, the "Brute Force" algorithm ETFP becomes infeasible, so Curve 1 shows the performance of Method 1, in which only 20 of the 40 most common attributes are counted in this way. This still shows a performance improvement over Apriori-TFP. However, because another 20 of the very frequent attributes are counted in the T-tree, the size of the candidate sets leads, as with other methods, to severe performance scaling as the support threshold is reduced. The same applies to Method 2, which offers a further small advantage as the support threshold is lowered, because with large candidate sets the amount of checking involved becomes significant. Even so, the gain from this is very slight (less than 2% improvement at the 7% support threshold).

Curve 3 illustrates Method 3, also (for comparison) with only 20 of the 40 most common attributes excluded from the initial T-tree. As we would expect, the curve is similar to that of Figure 4; although the method outperforms the others, the time taken increases rapidly for low support thresholds. This is, of course, because of the very large number of combinations of the very common attributes which are being counted. The final curve, 4, however, shows the results of excluding all 40 very frequent attributes from the initial count. In this case, the initial construction of the T-tree counts only those frequent sets which include at least one of the 500 less common attributes. For support thresholds down to about 3%, there are relatively few of these, so the time to count these sets is low, as is the time required finally to count the support of the combinations of common attributes which are subsets of these frequent sets. Only at a support threshold of 1% does the time start to rise rapidly (although this will still be far less than for the other methods described).

Fig 5: Performance for $T20.I10.D250000.N40$ merged with $T10.I5.D250000.N500$

6 Conclusions

Much of the work reported in the literature on the discovery of Association rules has focused on the case of shopping-basket analysis, characterised by very large numbers of database attributes (i.e. items available for purchase) but relatively low data density (frequency of individual items and sets of items). It is well-understood that all methods find increasing difficulty in coping with more densely-populated data. We have previously described a method, Apriori-TFP, which performs well in comparison with others, but is also subject to this adverse performance scaling when dealing with high-density data.

In this paper we have described some developments of our approach, optimised to deal with data that includes a proportion of very frequent attributes. This kind of data may be typical, for example, of demographic survey data and epidemiological data, in which some categorical attributes relating to age, gender, etc, have very frequent instances, while others which may be very relevant to the epidemiology are rather infrequent. In this case, we require support thresholds to be set low enough to identify the interesting sets including these scarce attributes.

Our results show, unsurprisingly, that both Apriori-TFP and other methods face problems with this data at low support thresholds. We have shown that the performance of Apriori-TFP can be improved significantly, although not dramatically, by the use of a heuristic which employs exhaustive counting for the most frequent attributes. However, if we make the (not unreasonable) assumption that the only sets that are of interest are those including at least one scarce attribute, then a much more effective adaptation of Apriori-TFP becomes possible, We have shown that this method strongly outperforms Apriori-TFP, enabling identification if the interesting sets in this kind of dense data even at low thresholds.

References

1. Agarwal, R., Aggarwal, C. and Prasad, V. Depth First Generation of Long Patterns. Proc ACM KDD 2000 Conference, Boston, 108-118, 2000.
2. Agrawal, R. Imielinski, T. Swami, A. Mining Association Rules Between Sets of Items in Large Databases. SIGMOD-93, 207-216. May 1993.
3. Agrawal, R. and Srikant, R. Fast Algorithms for Mining Association Rules. Proc 20th VLDB Conference, Santiago, 487-499. 1994
4. Bayardo, R.J. Efficiently Mining Long Patterns from Databases. Proc ACM-SIGMOD Int Conf on Management of Data, 85-93, 1998
5. Bayardo, R.J., Agrawal, R. and Gunopolos, D. Constraint-based rule mining in large, dense databases. Proc 15th Int Conf on Data Engineering, 1999
6. Berry, M.J. and Linoff, G.S. Data Mining Techniques for Marketing, Sales and Customer Support. John Wiley and sons, 1997
7. Brin, S., Motwani. R., Ullman, J.D. and Tsur, S. Dynamic itemset counting and implication rules for market basket data. Proc ACM SIGMOD Conference, 255-256, 1997

8. Coenen, F. and Leng, P. Optimising Association Rule Algorithms Using Itemset Ordering. In Research and Development in Intelligent Systems XVIII, (Proc ES2001), eds M. Bramer, F Coenen and A Preece, Springer, Dec 2001, 53-66
9. Goulbourne, G., Coenen, F. and Leng, P. Algorithms for Computing Association Rules using a Partial-Support Tree. J. Knowledge-Based Systems 13 (2000), 141-149. (also Proc ES'99.)
10. Han, J., Pei, J. and Yin, Y. Mining Frequent Patterns without Candidate Generation. Proc ACM SIGMOD 2000 Conference, 1-12, 2000.
11. Liu, B., Hsu, W. and Ma, Y. Mining association rules with multiple minimum supports. Proc. KDD-99, ACM, 1999, 337-341
12. Coenen, F., Goulbourne, G. and Leng, P. Computing Association Rules Using Partial Totals. Proc PKDD 2001, eds L. De Raedt and A Siebes, LNAI 2168, August 2001, 54-66
13. Rymon, R. Search Through Systematic Set Enumeration. Proc. 3rd Int'l Conf. on Principles of Knowledge Representation and Reasoning, 1992, 539-550.
14. Savasere, A., Omiecinski, E. and Navathe, S. An efficient algorithm for mining association rules in large databases. Proc 21st VLDB Conference, Zurich, 432-444. 1995.
15. Toivonen, H. Sampling large databases for association rules. Proc 22nd VLDB Conference, 134-145. Bombay, 1996.
16. UCI Machine Learning Repository Content Summary. http://www.ics.uci.edu/mlearn/MLSummary.html