Threshold Tuning for Improved Classification Association Rule Mining

Frans Coenen[1], Paul Leng[1], and Lu Zhang[2]

[1] Department of Computer Science, The University of Liverpool, Liverpool L69 3BX, UK
{frans, phl}@csc.liv.ac.uk

[2] Department of Computer Science and Technology, Peking University Beijing 100871, P.R. China
zhanglu@sei.pku.edu.cn

Abstract. One application of Association Rule Mining (ARM) is to identify Classification Association Rules (CARs) that can be used to classify future instances from the same population as the data being mined. Most CARM methods first mine the data for candidate rules, then prune these using coverage analysis of the training data. In this paper we describe a CARM algorithm that avoids the need for coverage analysis, and a technique for tuning its threshold parameters to obtain more accurate classification. We present results to show this approach can achieve better accuracy than comparable alternatives at lower cost.

1 Introduction

An Association Rule (AR) is a way of describing a relationship that can be observed between database attributes [1], of the form "if the set of attribute-values A is found together in a database record, then it is likely that the set B will be present also". A rule of this form, $A \to B$, is of interest only if it meets at least two threshold requirements: *support* and *confidence*. The support for the rule defines the number of database records within which the association can be observed. The confidence in the rule is the ratio of its support to that of its antecedent. Association Rule Mining (ARM) aims to uncover all such relationships that are present in a database, for specified thresholds of support and confidence.

One application of ARM is to define rules that will *classify* database records. A Classification Association Rule (CAR) is a rule of the form $X \to c$, where X is a set of attribute-values, and c is a class to which database records (instances) can be assigned. Mining of CARs usually proceeds in two steps. First, a training set of database records is mined to find all ARs for which one of the target classes is the consequent, and which satisfy specified thresholds of support and confidence. This stage is essentially similar to ARM in the more general case, with the classes c treated as attribute-values, and the restriction that the only rules we need consider are those for which the consequent is one of these. A second stage then sorts and reduces the set of rules found, with the aim of producing a consistent set that will enable efficient and reliable classification of future instances.

The CBA algorithm described in [6] exemplifies the approach. First, a version of the well-known Apriori algorithm [2] is used to generate a set of *ruleitems* that satisfy a required support threshold, where a ruleitem is a set of items (attribute-values) associated with a class label, which thus defines a potential CAR. The rules thus generated are pruned, using the calculated confidence to eliminate those that fail to meet a required confidence threshold or which conflict with higher-confidence rules. Finally, a classifier is built by selecting an ordered subset of the remaining CARs. This process involves *coverage analysis* in which each candidate CAR is examined in turn, to find a set of CARs that cover the dataset fully. The CMAR algorithm of [7] has a similar general form, using a version of the FP-growth algorithm [5] to generate the candidate CARs.

Results presented in [6] and[7] show that classification using CARs seems to offer greater accuracy, in many cases, than other methods such as C4.5 [8]. The problem with both CBA and CMAR, however, is that the cost of the coverage analysis is essentially a product of the size of the dataset and the number of candidate CARs being considered. The CPAR algorithm [10] has a different approach to generating rules, using a procedure derived from the FOIL algorithm [9] rather than a classical ARM method. Although this improves performance by generating a smaller set of rules, the performance is still $O(nmr)$, where n is the number of records, m the number of items, and r the number of candidate rules. The RIPPER algorithm [4] incorporates a pruning strategy that can be applied in a manner independent of the rule generation strategy used. All these methods follow an *overfit and prune* strategy that will be costly if used to construct classifiers from the very large and wide datasets that are characteristic of classical ARM.

Thus, we identify three general problems of CAR mining. First, the ARM task is inherently costly because of the exponential complexity of the search space. Second, it is very likely that this first stage will generate a very large number of candidate rules, and so the selection of a suitable subset for classification may also be computationally expensive. Finally, the reliability of the resulting classifier depends in some degree on the rather arbitrary choice of support and confidence thresholds used in the mining process.

In this paper we describe a new approach to the generation of CARs, that significantly reduces the cost of mining the training data by using both support and confidence thresholds in the first stage of mining, to produce a small set of rules without the need for coverage analysis. We present results to show that this approach achieves a classification accuracy that is comparable with other methods, but in many cases at much lower cost. We also describe a strategy for tuning support and confidence thresholds to obtain a best accuracy.

2 Generating Classification Association Rules Using TFPC

We begin with the observation that, if the support and confidence thresholds have been selected correctly, then the existence of a rule $X \rightarrow c1$ should make it unnecessary to consider any other rules whose antecedent is a superset of

X. In practice, however, we may still find a rule $Y \rightarrow c2$, say, where Y is a superset of X, which has higher confidence and to which we would wish to give higher precedence. It remains possible, also, that there will be a further rule $Z \rightarrow c1$, where Z is a superset of Y, with still higher confidence, and so on. This reasoning leads other methods to a process in which all possible rules are first generated and then evaluated. In this paper we adopt an alternative heuristic: *If we can identify a rule $X \rightarrow c$ which meets the required support and confidence thresholds, then it is not necessary to look for other rules whose antecedent is a superset of X and whose consequent is c.* It will still of course be necessary to continue to look for rules that select other classes. This heuristic both reduces the number of candidate rules to be considered, and the risk of overfitting.

We use a method derived from our TFP (Total From Partial) algorithm to generate a set of CARS. This method, described in [3], first builds a set-enumeration tree structure, the *P-tree*, that contains an incomplete summation of support-counts for relevant sets. Using the P-tree, the algorithm uses an Apriori-like procedure to build a second set enumeration tree, the *T-tree*, that finally contains all the frequent sets (i.e. those that meet the required threshold of support), with their support-counts. The T-tree is built level by level, the first level comprising all the single items (attribute-values) under consideration. In the first pass, the support of these items is counted, and any that fail to meet the required support threshold are removed from the tree. Candidate-pairs are then generated from remaining items, and appended as child nodes. The process continues, as with Apriori, until no more candidate sets can be generated.

Figure 1 illustrates the form of a T-tree, for the set of items $\{A, B, C, x, y\}$, where x and y are class identifiers. This tree is complete, i.e it includes all possible itemsets, except for those including both x and y which we will assume cannot occur. In practice, an actual T-tree would include only those nodes representing the frequent sets. For example, if the set $\{A, C\}$ fails to reach the required support threshold, then the node labelled AC would be pruned from the tree, and the nodes ABC, ACx, ACy, $ABCx$ and $ABCy$ would not be created. All the candidate itemsets that include the class-identifier x can be found in the subtree rooted at x, and thus all the rules that classify to x can be derived from this subtree (and likewise for y).

The algorithm used to build the T-tree in Figure 1 is a modification of the original TFP approach. As each pass is concluded, we first remove from the tree all those nodes representing sets that fail to meet the support threshold. The remaining (frequent) sets that are included within the class-identifier subtrees (x and y in this example) define possible classification rules: for example, the set Bx corresponds to a rule $B \rightarrow x$. We now calculate the confidence of all such rules, i.e. in the case of Bx, the ratio $(supportofBx)/(supportofB)$. If this rule exceeds the required confidence threshold, we add the rule to our target set, and remove the corresponding node (Bx) from the tree. The effect of this is that when the next level is generated, supersets of Bx (ie ABx and BCx) will not be added to the tree.

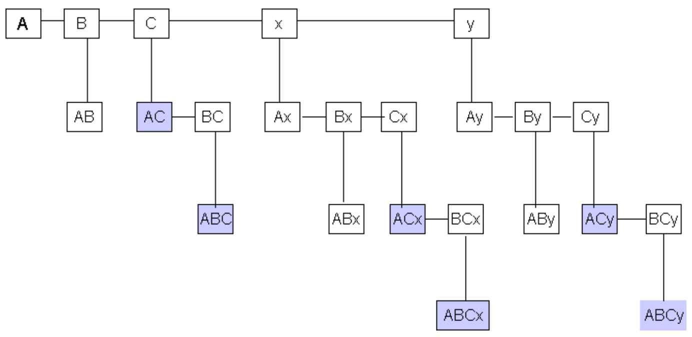


Fig. 1. Form of a T-tree for $\{A, B, C, x, y\}$

This algorithm, which we call TFPC (Total From Partial Classification), generates far fewer frequent sets than would be produced from a generic ARM algorithm such as Apriori or TFP. Moreover, the sets left on the tree can immediately be used to define classification rules which meet our threshold requirements for support and confidence. To classify using these rules, the rule set is sorted by confidence. Between two rules of equal confidence, the more specific takes precedence: i.e. if $B \rightarrow x$, say, and $BD \rightarrow y$, then the latter will be selected.

3 Experimental Results

To investigate the performance of TFPC, we carried out experiments using a number of data sets taken from the UCI Machine Learning Repository. The implementation of TFPC was as a Java program, and for comparison purposes we have used our own (Java) implementations of the published algorithms for CMAR [7] and CPAR [10]. In the first set of experiments, as in [6] and [7], we have assumed a support threshold of 1% and a confidence threshold of 50%. For the implementation of CPAR, we used the same parameters used in [10], i.e. minimum gain threshold = 0.7, total weight threshold = 0.05, decay factor = 2/3, and similarity ratio 1 : 0.99. We tried to use the same data sets as those used to analyse CMAR and CPAR. However in many cases the data sets that were used in [7] and [10] appear to be no longer available, and in others were found not have identical parameters to those reported.

The data sets chosen therefore comprise a subset of those used in [7] and [10], augmented by a further selection from the UCI repository. The choice of additional data sets concentrated on larger/denser data sets (2000+ records) because the majority of the data sets used in the reported analysis of CMAR and CPAR were relatively small (less than 1000 records). The sets chosen were discretized using the LUCS-KDD DN software [1], where appropriate continuous attributes

[1] Available at http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/lucs-kdd_DN.html

were ranged using five sub-ranges. The programs were run on a 1.2 GHz Intel Celeron CPU with 512 Mbyte of RAM running under Red Hat Linux 7.3.

The results from these experiments are shown in Table 1. The row labels describe the key characteristics of the data, in the form into which it was discretized. For example, the label **anneal.D106.N798.C6** denotes the 'anneal' data set, which includes 798 records in 6 classes, with attributes that for our experiments have been discretized into 106 binary categories.

The last three columns in Table 1 tabulate the accuracy of the classification obtained by the three methods. In all cases, the figure shown is the average obtained from a *10-fold cross-validation* using the full data set. The accuracy obtained using the TFPC method is in most cases comparable with that obtained from the other methods investigated. Although the average accuracy of the method was slightly lower than the others, in 5 cases TFPC gave an accuracy as high as or higher than both other methods, and only in two cases (ionosphere and wine) was it markedly worse than both.

The significance of these results is that this level of accuracy was obtained from a very efficient rule-generation process. The first three columns in Table 1 show the execution times for the three methods. In each case, the time shown is that obtained for the full experimental evaluation using 10-fold cross-validation. As can be seen, the performance of TFPC (in comparable implementations) is

Table 1. Results for support= 1%, confidence= 50%

Data Set	Time			Number of rules				Accuracy		
	CMAR	CPAR	TFPC	CMAR	CPAR	TFP	TFPC	CMAR	CPAR	TFPC
adult.D131.N48842.C2	2088.6	809	80	3063	183	20530	82	71.2	76.7	76.1
anneal.D106.N798.C6	150.5	1.8	12.1	319	34	20525	61	83.5	90.2	84.6
breast.D48.N699.C2	7.5	0.7	1	191	16	4008	43	85.2	94.8	95.9
connect4.D129.N67557.C3	1449.2	24047	206.4	821	816	3600	106	66.9	54.3	65.9
heart.D53.N303.C5	36.2	1	11.4	305	53	13429	193	56.8	51.1	57.1
hepatitus.D58.N155.C2	160	0.3	15.2	99	14	31126	58	76.3	76.5	77.3
horseColic.D94.D368.C2	40.3	0.6	6.3	387	18	17219	99	74.5	82.3	78.7
ionosphere.D104.N351.C2	28.4	1.1	8.4	214	26	8581	380	96	92.9	83.8
iris.D23.N150.C3	0.2	0.2	0.1	69	11	200	20	94.7	94.7	94.7
led7.D24.N3200.C10	1.6	5.7	0.7	229	31	464	29	73.7	71.2	69
mushroom.D127.N8124.C2	1045.5	15.4	31.6	171	31	25253	116	99.1	98.8	96.1
nursery.D32.N12960.C5	22.2	51.7	5.2	487	84	1709	30	91.4	78.5	77.9
pageBlocks.D55.N5473.C5	25.3	15.5	2.4	243	56	7882	19	89.8	76.2	89.8
penDigits.D90.N10992.C10	222.7	101.9	56.8	2052	167	7667	3015	79.1	83	79.7
pimaIndians.D42.N768.C2	3.2	1	1.3	540	23	3693	34	77.4	75.6	74.9
waveform.D108.N5000.C3	1149	38.1	86.9	1186	114	22876	662	71.2	75.4	71.7
wine.D68.N178.C3	1018.7	0.3	8.7	110	18	31961	164	97.1	92.5	86.3
zoo.D43.N101.C7	780.5	0.2	6	31	19	25050	250	92	96	93
Average	457.2	1394.0	30.0	584.3	95	15361	298	82.0	81.2	80.7

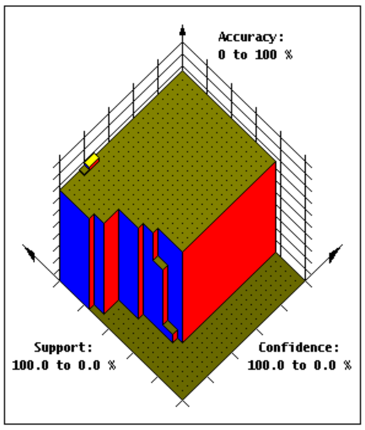
markedly superior to that of CMAR (in all cases) and CPAR (in many). The improvement over CMAR results from the smaller number of rules that tend to be produced using TFPC and because this approach eliminates the expensive coverage analysis carried out in CMAR. The columns headed 'Number of rules' show the average number of rules included in the final classifiers generated, for the three methods. For comparison, the column headed TFP shows the total number of classification rules generated by the TFP algorithm, which defines the total number of candidate classification rules produced by a straightforward ARM algorithm using the given support and confidence thresholds. A comparison of this column with that for TFPC shows the advantage gained during rule generation from the heuristics used in the latter. In most (although not all) cases, TFPC also produces fewer rules than CMAR.

CPAR, conversely, usually produces fewer rules than TFPC, and this is reflected in the faster execution times it achieves in some cases. The advantage of TFPC, however, is that by dispensing with coverage analysis the performance of the method scales better for larger data sets. In all the cases where the data included more than 10000 records (adult, connect4, nursery, pendigits), TFPC is much faster than CPAR, sometimes by an order of magnitude or more. CPAR also performed relatively poorly in the cases (pageblocks, led) in which there were both a moderately large number of records and a relatively large number of classes. In all these cases, TFPC achieves a classification accuracy close to or superior to CPAR at much lower cost.

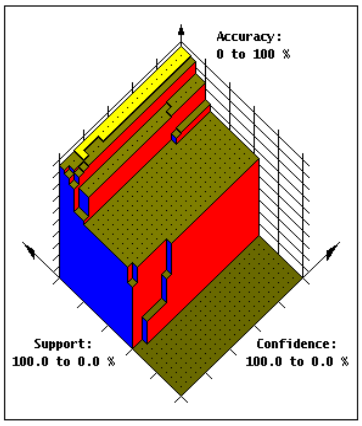
4 Finding Best Support and Confidence

The results presented above show that it is possible to obtain a set of rules that will in most cases provide acceptable classification accuracy, using CARM techniques, without further coverage analysis. However, because the tree generation heuristic used in TFPC stops looking for more specific rules once a general rule with satisfactory confidence has been found, it may sometimes fail to find high-confidence rules that could be significant in special cases. The method may therefore be more sensitive to the choice of support and confidence thresholds used than is the case for methods that rely on coverage analysis to derive the final ruleset. To investigate this further, we performed a series of experiments to identify the combination of support and confidence threshold that would lead to the highest classification accuracy in each of the data sets studied. Figure 2 shows the results obtained, in the form of 3-D plots, for a number of example data sets selected to demonstrate the variety of the results obtained. For each plot the X and Y axes represent support and confidence thresholds ranging from 100 to 0%, and the Z axis the classification accuracy obtained.

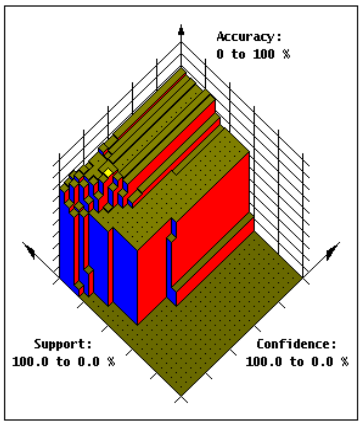
From Figure 2 it can be seen how the classification accuracy produced using TFPC may vary significantly depending on the choice of support and confidence thresholds. The extent of the variability depends on the characteristics of the data. For example, the 'adult' data set (Figure 2(a)) shows a substantial plateau of support-confidence values within which accuracy is constant (although there is



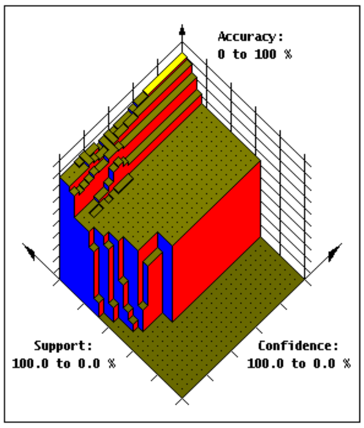
a) adult.D131.N48842.C2



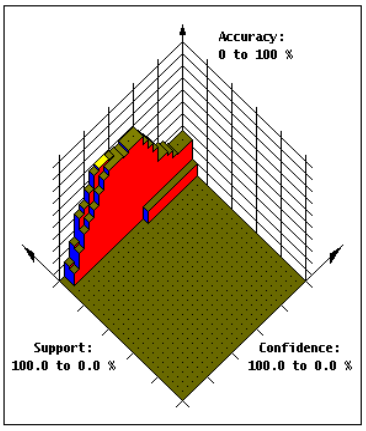
b) breast.D48.N699.C2



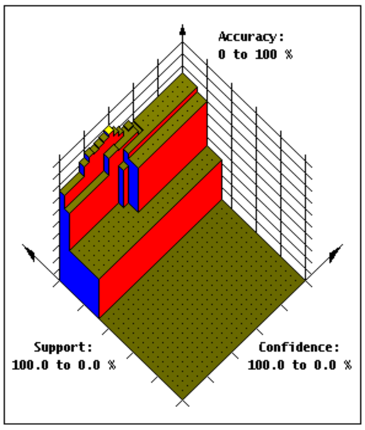
c) horseColic.D95.N300.C3



d) ionosphere.D172.N351.C2



e) led7.D24.N3200.C10



f) nursery.D32.N12960.C5

Fig. 2. 3-D plots of classification accuracy v. support/confidence thresholds a small peak, highlighted in white, at which a higher accuracy can be obtained). Conversely, the 'led7' data set (e) illustrates a case in which the method is highly sensitive to the choice of support and confidence threshold. In this case, the values 1% and 50% used for the experiments of Table 1 were, fortuitously, close to the peak, so reasonably good results were obtained. This was also the case for the 'breast' set (b) and 'horsecolic (c). In the cases of 'ionosphere' (d) and nursery (f) conversely, the relatively poor accuracy obtained using TFPC is a consequence of a poor choice of support and confidence thresholds.

To obtain a best classification the TFPC algorithm was applied in an iterative manner in conjunction with a "hill-climbing" procedure. The hill climbing technique makes use of a 3-D playing area measuring $100x100x100$, as visualised in the plots of Figure 2. The axes of the playing area represent percentage values for support thresholds, confidence thresholds and accuracies. The technique commences with initial support and confidence thresholds, with an associated accuracy, that describes a current location in the playing area. The procedure then traverses the playing area with the aim of maximising the accuracy value. To do this it continuously generates data for a "grid" of eight test locations, obtained by applying to the current location positive and negative increments to the support and confidence thresholds. The current and test locations thus form a $3x3$ location grid centred on the current location. The threshold increments are reduced progressively as the procedure converges.

Table 2 shows the results obtained using TFPC-HC (columns headed THC) for all the data sets included in Table 1. For comparison, the results obtained from TFPC (without hill climbing), and from CMAR and CPAR, with 1% support and 50% confidence, are included.

The results confirm the theory demonstrated by the plots presented in Figure 2. In those cases (e.g adult, breast, led7, horsecolic) where, by chance, the (1%, 50%) pair gives a near-best result, little improvement is obtained from the hill-climbing procedure. In the cases of ionosphere and nursery, conversely, and also anneal, hepatitis and pendigits, a very substantial increase in accuracy is obtained. It is noteworthy, also, that both overall and in many particular cases, TFPC-HC achieves a higher classification accuracy than either CMAR or CPAR.

Although these experiments show the gain that can be obtained by tuning the selection of support and confidence thresholds, the hill-climbing procedure is, of course, more time-consuming than CMAR, CPAR or TFPC without hill climbing. To reduce the cost of this the algorithm was further adapted, TFPC-HC+, so that hill-climbing was only carried out using a single division of the data set into a (90%) training set and (10%) test set. The hill-climbing procedure was used to find the support and confidence thresholds that led to the most accurate classification of this test set. These thresholds were then used to apply the TFPC method to the full data set, with ten cross-validation. The results are included in the columns of Table 2 headed HC+.

As can be seen, this more restricted hill-climbing (essentially, on a sample of the data) is still relatively effective in obtaining a near-optimal support and confidence pairing for use in the method. Overall, the results from this show

Table 2. Comparison of methods of classification

Data Set	Accuracy					Execution times				
	CMAR	CPAR	TFPC	THC	HC+	CMAR	CPAR	TFPC	THC	HC+
adult.D131.N48842.C2	71.2	76.7	76.1	76.9	76.8	2089	809	80	1187	347
Anneal.D106.N798.C6	83.5	90.2	84.6	91.3	90.9	150	2	12	374	52
breast.D48.N699.C2	85.2	94.8	95.9	96.4	96.1	8	1	1	9	2
Connect4.D129.N67557.C3	66.9	54.3	65.9	65.9	65.9	1449	24047	206	11816	906
heart.D53.N303.C5	56.8	51.1	57.1	61.0	57.4	36	1	11	285	43
hepatitus.D58.N155.C2	76.3	76.5	77.3	86,0	77.3	160	1	15	249	24
horseColic.D94.D368.C2	74.5	82.3	78.7	80.9	79.5	40	1	6	186	41
ionosphere.D104.N351.C2	96.0	92.9	83.8	91.5	88.9	28	1	8	674	135
Iris.D23.N150.C3	94.7	94.7	94.7	94.7	94.7	1	1	1	6	1
Led7.D24.N3200.C10	73.7	71.2	69.0	71.3	68.8	2	6	1	7	3
mushroom.D127.N8124.C2	99.1	98.8	96.1	99.3	96.1	1046	15	32	1313	127
Nursery.D32.N12960.C5	91.4	78.5	77.9	92.2	90.5	22	52	5	263	22
pageBlocks.D55.N5473.C5	89.8	76.2	89.8	89.8	89.8	25	16	2	6	3
penDigits.D90.N10992.C10	79.1	83.0	79.7	89.0	88.1	223	102	57	2546	303
pimaIndians.D42.N768.C2	77.4	75.6	74.9	78.9	74.9	3	1	1	28	2
waveform.D108.N5000.C3	71.2	75.4	71.7	75.1	75.6	1149	38	87	5670	611
wine.D68.N178.C3	97.1	92.5	86.3	95.1	92.1	1019	1	9	55	6
Zoo.D43.N101.C7	92.0	96.0	93.0	96.0	93.0	781	1	6	148	14
Average	82.0	81.2	80.7	85.1	83.1	457	1394	30	1379	147

greater accuracy than either CMAR or CPAR, as well as improvements on TFPC with the (1%, 50%) thresholds. The times obtained show that TFPC-HC+ is relatively efficient, especially with respect to large datasets such as the adult and connect4 sets.

5 Conclusions

Previous research has shown that ARM can be an effective route to accurate classification. A drawback of existing methods is that they involve detailed coverage analysis of candidate rules to obtain a final set of classification rules. This procedure is expensive: prohibitively so when very large training sets are involved. We have here introduced an algorithm, TFPC, which obtains a set of classification rules directly from an ARM procedure, without further coverage analysis. Our results show that the classification accuracy obtained is comparable with, or close to, that obtained from established methods. The TFPC method, however, is very much faster than alternatives that involve coverage analysis, especially when dealing with large data sets. We believe this method offers a realistic approach to deriving classifiers from extremely large data sets, for which existing methods would be inapplicable.

The accuracy of classification obtained using TFPC is, however, relatively sensitive to the choice of support and confidence thresholds used when mining the classification rules. We have described a 'hill-climbing' procedure we have used to select the best values for these thresholds. Our results show that, using these optimal values, the accuracy of classification of TFPC is improved very substantially, in most cases improving on the best current methods. The significance of these results is that they demonstrate how performance can be improved by careful selection of these threshold values. Although the hill-climbing process is, in general, too time-consuming to be used routinely for this purpose, we have shown that a more limited version of this (TFPC-HC+) can give results that are almost as good with a speed that, for large data sets, improves on methods requiring coverage analysis.

It may also be the case that the tuning of support and confidence thresholds with respect to accuracy, may improve the performance of other methods. There is scope for further research in this direction, and also to investigate other ways to perform this optimisation efficiently.

References

1. Agrawal, R., Imielinski, T. and Swami, A.: Mining Association Rules between Sets of Items in Large Databases. Proc. ACM SIGMOD Conference on Management of Data, 207-216, 1993.
2. Agrawal, R. and Srikant, R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th VLDB Conference, Santiago, Chile, 487-499, 1994.
3. Coenen, F., Goulbourne, G., and Leng, P.: Computing Association Rules using Partial Totals. PKDD 2001, pages 54-66, 2001.
4. Cohen, W.W.: Fast Effective Rule Induction. Proc. of the 12th Int. Conf. on Machine Learning, pages115-123, 1995.
5. Han, J., Pei, J. and Yin, Y.: Mining Frequent Patterns without Candidate Generation. In Proc. of the ACM SIGMOD Conference on Management of Data, Dallas, pages 1-12, 2000.
6. Liu, B., Hsu, W. and Ma, Y.: Integrating Classification and Association Rule Mining. Proc KDD 1998, 80-86
7. Li, W., Han, J. and Pei, J.: CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. Proc ICDM'01, 2001, 369-376
8. Quinlan, J.R. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.
9. Quinlan, J.R. and Cameron-Jones, R.M.: FOIL: A midterm report. Proc ECML 1993, 3-20.
10. Yin, X. and Han, J.: CPAR: Classification Based on Predictive Association Rules. Proc SIAM Int Conf on Data Mining (SDM'03), 2003, 331-335