OBTAINING BEST PARAMETER VALUES FOR ACCURATE CLASSIFICATION

Frans Coenen and Paul Leng
Department of Computer Science, The University of Liverpool, Liverpool, L69 3BX
{frans,phl}@csc.liv.ac.uk

Abstract

A method of classification that has attracted recent interest is to apply an Association Rule Mining algorithm to obtain classification rules from a training set of previously-classified data. The rules thus generated will be influenced by the choice of parameters employed by the algorithm, especially the support and confidence threshold values. In this paper we examine the effect that this choice has on the predictive accuracy of classifiers obtained by some methods for Classification Association Rule Mining. We show that the accuracy can almost always be improved by a suitable choice of threshold values, and we describe a method for finding the best values. We present results that demonstrate this approach can obtain higher accuracy without the need for further coverage analysis of the training data. **Keywords***: Classification, Association Rule Mining*

1 INTRODUCTION

Systems for classification attempt to categorise records in a data set by applying a set of Classification Rules of the general form $A \rightarrow C$, where A, the antecedent, is the union of some set of values of attributes of the records involved, and the consequent, C, is the label of a class to which records can be assigned. Classification rules are typically derived from examination of a *training* set of records that have been previously annotated with appropriate class labels. Techniques for obtaining rules include methods that use decision trees [12], Bayesian networks [11], and Support Vector Machines [5]. A method that has attracted recent attention is to make use of Association Rule Mining (ARM) techniques. In general, an Association Rule defines a relationship between disjoint subsets of the overall set of attributes represented by the dataset. For the purpose of obtaining Classification Rules, however, we define binary attributes that represent class-labels, and search only for rules whose consequent is one of these. Examples of the use of ARM algorithms to generate classification rules (i.e. Classification Association Rule Mining or CARM) include PRM and CPAR [14], CMAR [8] and CBA [9].

In general, CARM algorithms begin by generating all rules that satisfy at least two user-defined threshold conditions. The *support* of a rule describes the number of instances in the training data for which the rule is found to apply. The *confidence* of the rule is the ratio of its support to the total number of instances of the rule's antecedent: i.e. it describes the proportion of these instances that were correctly classified by the rule. Minimum threshold values for support and confidence eliminate from consideration candidate rules that either describe too few cases or offer too low classification accuracy. Frequently, however, it will not be possible to obtain a complete classification using only a small number of high-confidence rules. Most methods, therefore, generate a relatively large number of rules which are pruned and ordered using other techniques, applying various other threshold values to determine rule selection. Typically, *coverage analysis* is employed, in which cases in the training data are examined to identify from the candidate rules a covering set that will classify all cases correctly.

Although it is clear that the accuracy of classification may be influenced significantly by the choice of appropriate values for whatever thresholds are used, most work in applying and evaluating methods of classification makes use of threshold values that are chosen arbitrarily. In previous work [3] we described an algorithm, TFPC, that obtains classification rules from an efficient CARM process without coverage analysis, and we showed that provided support and confidence values are well-chosen, the method can offer as good or better accuracy than comparable methods. In this paper we examine the effect of varying the support and confidence thresholds on the accuracy of both TFPC and other algorithms. We show that classification accuracy can be significantly improved, in most cases, by an appropriate choice of thresholds. We describe a *hill climbing* algorithm which aims to find the "best" thresholds from examination of the training data. We show that this procedure can lead to higher classification accuracy at lower cost than methods of coverage analysis.

In section 2 we summarise previous work on CARM. In section 3 we describe some experiments that demonstrate the effect of varying the support and confidence thresholds on the accuracy obtained from three CARM algorithms: TFPC, CBA, and CMAR. In section 4 we describe the hill-climbing algorithm, and in section 5 we present results of experiments applying the hill-climbing procedure to the three algorithms. We show the effect on classification accuracy that can be obtained by a best choice of thresholds, and discuss how the approach can be applied in practice. Our conclusions are presented in section 6.

2 BACKGROUND

Algorithms for generating Classification Rules can be broadly categorised into two groups. **Two stage algorithms** first produce a set of candidate CRs, by a CARM process or otherwise, and in a separate second stage these are pruned and ordered for use in the classifier. Examples of this approach include CMAR [8], CBA [9] and REP as used for example in [10]. **Integrated algorithms**, conversely, produce a classifier in a single processing step, i.e. generation and pruning is "closely coupled". Examples include rule induction systems such as FOIL [13], PRM and CPAR [14]. Finally, IREP [6] and RIPPER [4] are one-stage pruning algorithms which may, in general, be applied also as the second stage of a two-stage method.

Most CARM algorithms are of the first type, exemplified by CBA ([9]), which was one of the first to make use of a general ARM algorithm for the first stage. CBA uses a version of the best-known ARM algorithm, Apriori [1], using user supplied support and confidence thresholds, to generate CARS which are then prioritised, using confidence, support, and rule-length (in that order) to determine the order of precedence. The ordered set of rules is then pruned (stage 2) by coverage analysis of the training data. In this process, each record in the training set is examined to find the first rule (the one with the highest precedence) that correctly classifies the record (the $cRule$) and the first rule that wrongly classifies the record (the $wRule$). If the $cRule$ has higher precedence than the $wRule$, the rule is included in the classifier. Otherwise, rules with lower precedence must be considered. CBA illustrates a general performance drawback of two-stage algorithms; the cost of the pruning stage is a product of the size of the data set and the number of candidate rules, both of which may in some cases be large. It is clear, also, that the choice of support and confidence thresholds can strongly influence the operation of CBA.

The CMAR algorithm ([8]) has a similar general structure to CBA, and uses the same CR prioritisation approach as that employed in CBA. CMAR differs in the method used in stage 1 to generate candidate rules, which makes use of the FP-tree data structure coupled with the FP-growth algorithm [7]; this makes it more computationally efficient than CBA. Like CBA, CMAR tends to generate a large number of candidate rules, which are pruned by removing all rules with a $\chi 2$ value below a user defined threshold or where a more general rule with higher precedence exists. Finally, a database coverage procedure is used to produce the final set of rules. This stage is similar to that of CBA, but whereas CBA finds only one rule to cover each case, CMAR uses a coverage threshold paramater to generate a larger number of rules. When classifying an "unseen" data record, CMAR groups rules that satisfy the record according to their class and determines the combined effect of the rules in each group using a Weighted χ Squared (WCS) measure.

The cost of coverage analysis, especially when dealing with large data sets with many attributes and multiple cases, motivated us to consider whether it is possible to generate an acceptably accurate set of Classification Rules directly from an ARM process, without further coverage analysis. In [3] we described an algorithm, TFPC, of this kind. The heuristic applied by TFPC is that once a general rule is found that satisfies the required thresholds of support and confidence, no more specific rules (rules with the same consequent, whose antecedent is a superset) will be considered. This provides a very efficient method for generating a relatively compact set of CRs. Giving precedence to more general rules also reduces the risk of overfitting. Because no coverage analysis is carried out, however, the choice of appropriate support and confidence thresholds is critical in determining the final rule set. In the next section we examine the effect of varying these thresholds on both TFPC and other algorithms.

3 The effect of varying threshold values

To examine the effect that may result from different choices of threshold values, we carried out a series of experiments using test data from the UCI Machine Learning Repository. The data sets chosen were discretized using the LUCS-KDD DN software [1]. In the discussion and illustrations that follow, we identify each data set with a label that describes its key characteristics, in the form in which we have discretized it. For example, the label **glass.D48.N214.C7** refers to the "glass" data set, which includes 214 records in 7 classes, with attributes which for our experiments have been discretised into 48 binary categories.

Using this data, we investigated the classification accuracy that can be achieved using the TFPC algorithm, and

[1]Available at $http://www.csc.liv.ac.uk \sim frans/KDD/Software/LUCS-KDD-DN/lucs-kdd\_DN.html$

also from CMAR and CBA, across the full range of values for the support and confidence thresholds. For each pair (support, confidence) of threshold values we obtained a classification accuracy from a division of the full data set into a 90% training set and 10% test set. All the implementations were written in Java 1.4 by the authors. The results for a selection of the data sets considered are illustrated in Figures 1 and 2 in the form of 3-D plots. For each plot the X and Y axes represent support and confidence threshold values from 100% to 0%, and the Z axis the corresponding percentage classification accuracy obtained.

In these figures,the first illustration in each row is the plot for the TFPC algorithm. These results demonstrate a range of different characteristics of the various data sets. In some cases, for example **flare**, the best accuracy is obtained for a very broad range of threshold values. These represent "easy" cases to classify: those in which all the necessary rules have high support and confidence in the training data. In other cases, conversely, such as **led7**, and **letRecog** the accuracy obtained is very sensitive to the choice of thresholds. Generally, the best accuracy is obtained using a low support threshold, but there are cases where this is not so. Usually, these arise when the training set is small so that rules with low support may represent very few cases: see, for example **iris** and **wine**. The choice of confidence threshold, however, is often more critical. Our experiments include cases (e.g. **ticTacToe**) where a high confidence threshold is required. These represent cases where TFPC can perform badly if a low confidence threshold is chosen, because a rule that meets this threshold may mean that a better high-confidence rule is never found. In other cases, however, a low confidence threshold is required, because otherwise too few candidate rules are identified: **letRecog** is an example. Usually these are data sets for which classification accuracy is low, and only low-confidence rules can be found.

The plots for CMAR (second illustration in each row) have a broadly similar pattern. The illustrations show, however, the way in which the coverage analysis smooths out some of the influence of the choice of threshold. **led7**, **iris** and **wine**, all cases that for TFPC are sensitive to the thresholds of support and/or confidence, are good examples of this. In these cases, provided sufficiently low thresholds are chosen, then CMAR will find all the necessary candidate rules, and the coverage analysis reliably selects the best ones. This smoothing effect is not perfect, however; note that for **iris**, CMAR still has a suboptimal result if a very low support threshold is chosen, and the examples of **ticTacToe** and **ionosphere** illustrate cases where a choice of a low confidence threshold will lead to poor rules being selected. Conversely, in cases such as **letRecog**, where few if any high-confidence rules can be found, a low confidence threshold is again needed to find candidate rules.

The illustrations for CBA also, in many cases, demonstrate the "smoothing" effect obtained as a result of applying coverage analysis to select rules from the initial candidates. This is especially to be seen in some of the larger data sets: **letRecog** and **ticTacToe** illustrate cases where CBA's accuracy is more stable than that of CMAR. In other cases, however, usually involving small data sets, CBA may be even more sensitive than TFPC to the choice of thresholds. The example of **ionosphere** is especially interesting: here a poor choice of thresholds (even values that appear reasonable) may lead to a dramatically worse result. This is partly because, unlike CMAR, CBA's coverage analysis may sometimes retain a rule that applies only to a single case. This makes the method liable to include spurious rules, especially if the data set is small enough for these to reach the required thresholds.

4 Finding best threshold values

It is clear from the experiments discussed above that the accuracy and performance of rule-generation algorithms may be very sensitive to the choice of threshold values used. The coverage analysis used by methods such as CMAR and CBA sometimes reduces this sensitivity, but does not eliminate it. It is apparent that the accuracy of the classifiers obtained using any of these methods may be improved by a careful selection of these thresholds. In this section we describe a procedure for identifying the "best" threshold values, i.e, those that lead to the highest classification accuracy, from a particular training set. The method applies a "hill-climbing" strategy, seeking to maximise accuracy while varying the thresholds concerned. We will describe this in relation to thresholds of support and confidence, although the method can be applied in general to other kinds of threshold also.

The hill climbing technique makes use of a 3-D playing area measuring $100 \times 100 \times 100$, as visualised in the illustrations discussed above. The axes represent percentage values for (1) support thresholds, (2) confidence thresholds and (3) accuracies. The procedure commences with initial support and confidence threshold values, describing a current location (cl) in the base plane of the playing area. Using these values, the chosen rule-generation algorithm is applied to the training data, and the resulting classifier applied to the test data, with appropriate cross-validation, to obtain a classification accuracy for cl.

The hill-climbing procedure then attempts to move round the playing area with the aim of improving the accuracy value. To do this it continuously generates data for a set of eight test locations. The test locations are defined by applying two values, δs (change in support threshold) and δc (change in confidence threshold), as positive and negative increments of the support and confidence threshold val-

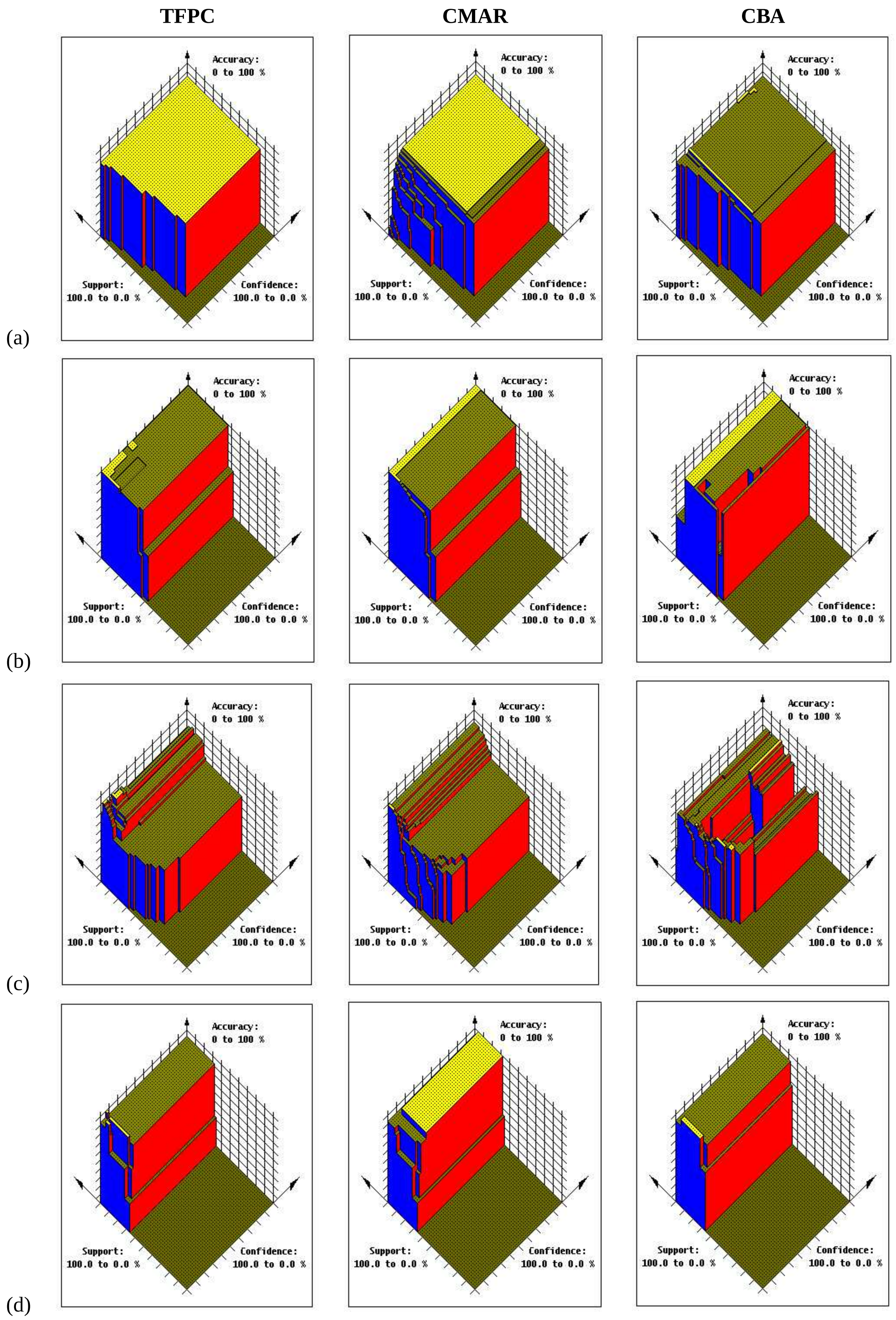

Figure 1 *3-D Plots I: (a) flare.D39.N1389.C9, (b) mushroom.D90.N8124.C2 (c) ionosphere.D157.N351.C2 and (d) iris.D19.N150.C3*

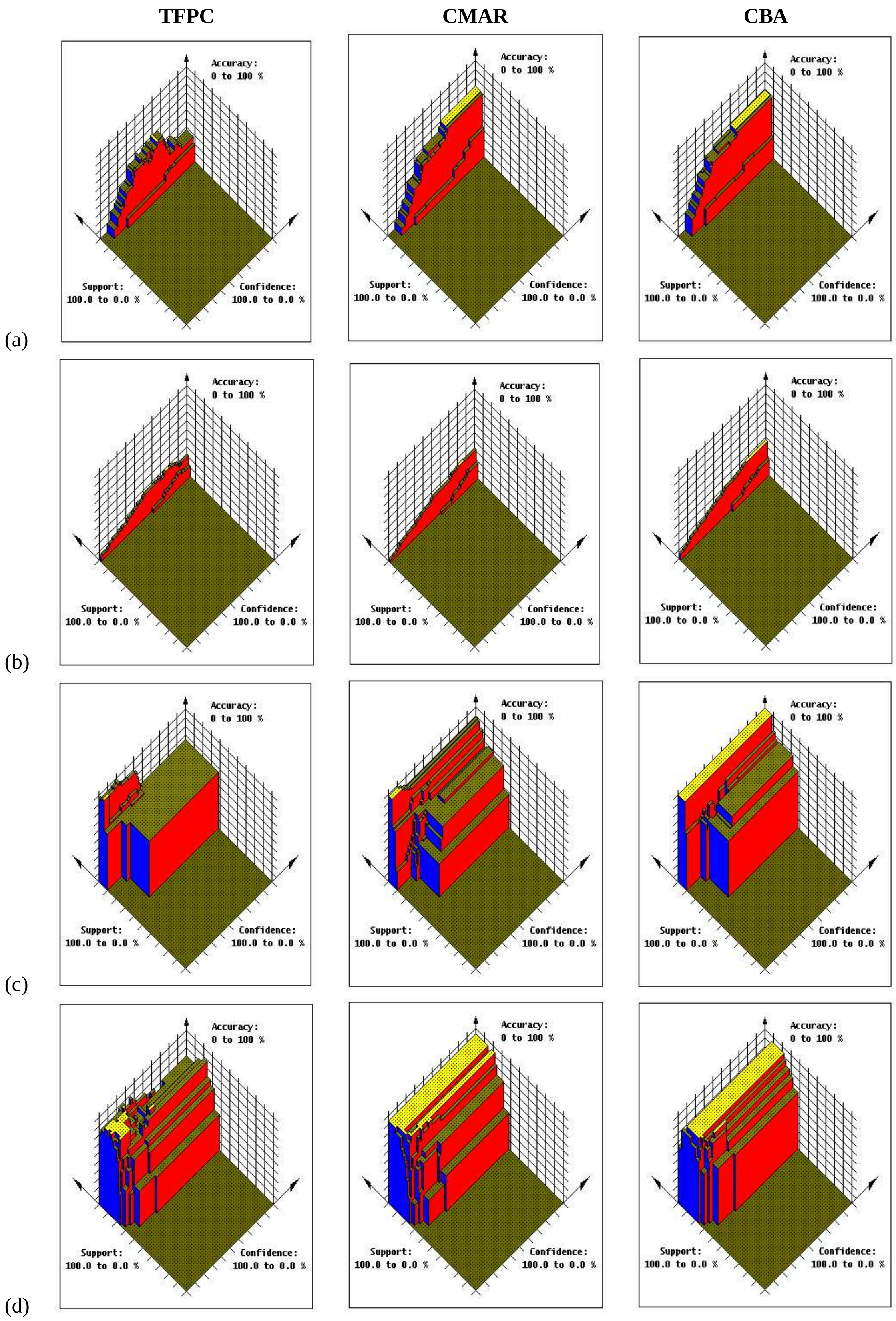

Figure 2 *3-D Plots III: (a) led7.D24.N3200.C10, (b) letRecog.D106.N20000.C26, (c) ticTacToe.D29.N958.C2 and (d) wine.D68.N178.C3*

ues associated with cl. The current and test locations form a 3×3 location grid with cl at the center (figure 3). The test locations are labeled: n, ne, e, se, s, sw, w, nw and n, with the obvious interpretations.

The rule-generation algorithm is applied to each of the test locations which is inside the playing area and for which no accuracy value has previously been calculated. A classification accuracy is thus obtained for each location. If this stage identifies a point with a superior accuracy to the current cl, the procedure continues with cl as this point. Between candidate points of equal accuracy, the algorithm uses a weighting procedure to select a "best" point from which to continue. If the current cl has the best accuracy, then the threshold increments are reduced and a further iteration of test locations takes place. The process concludes when either: (i) a best accuracy is obtained or (ii) a lower limit on the threshold-increments is reached at which point the current pair of threshold values which leads to a "best" classification accuracy for the chosen training and test data are selected. This will not necessarily be a true best value, depending on the choice of the initial cl and other parameters, but will at worst be a local optimum. The hill climbing process is summarised in Table 1.

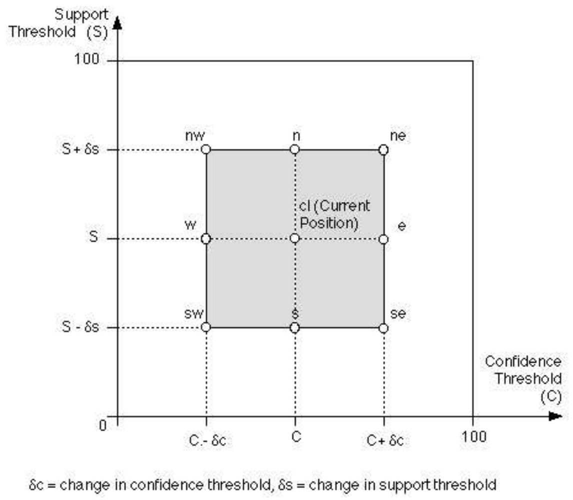

Figure 3: 8 point location grid within hill climbing playing area on plane of support and confidence axes

5 RESULTS

Table 2 summarises the results of applying the hill-climbing procedure described above to datasets from the UCI repository, for the algorithms TFPC, CMAR and CBA. For each algorithm, the first two columns in the table show the average accuracy obtained from applying the algorithm to (90%, 10%) divisions of the dataset with ten-fold cross-validation. The first of the two columns shows the result for a support threshold of 1% and a confidence threshold of 50% (the values usually chosen in analysis of classification algorithms), and the second after applying the hill-climbing procedure to identify the "best" threshold values. Note that, with respect to the experiments δS and δC were set to 6.4 and 8.0 respectively, and $min\delta S$ and $min\delta C$ to 0.1 and 1.0 as these parameters were found to give the most effective operational results. In each case the support and confidence threshold values that produced the best accuracy are also tabulated.

```
Algorithm: 8-Point Hill Climbing
input A Training and Test data set $D$, and
        a Rule Generation Algorithm $RGA$
output A list of rules $R$
-----------------------------------------------
$cl.S \leftarrow initial\ support$
$cl.C \leftarrow initial\ confidence$
$\delta S \leftarrow initial\ support\ increment$
$\delta C \leftarrow initial\ confidence\ increment$
$min\delta S \leftarrow minimum\ support\ increment$
$min\delta C \leftarrow minimum\ confidence\ increment$

Ruleset $\leftarrow$ Apply $RGA$ to $D$ using $cl.S$ and $cl.C$
Evaluate ruleset to obtain accuracy $cl.accuracy$
Start loop
        Apply $\delta S, \delta C$ to $cl.S, cl.C$ to get test points $P$
        Apply $RGA$ to get accuracy for all $P$ not
                previously evaluated
        $bl \leftarrow location\ with\ best\ accuracy\ in\ P$
        If $bl$ is unique then
                If $bl = cl$ Then $\delta S \leftarrow \delta S/2, \delta C \leftarrow \delta C/2$
                Else $cl \leftarrow bl$
        Else If majority of $P$ have best accuracy Then
                $cl \leftarrow bl, Break$
        Else If $cl.accuracy = bl.accuracy$ Then
                $\delta S \leftarrow \delta S/2, \delta C \leftarrow \delta C/2$
        Else $cl \leftarrow bl$
        End If Else
        If $\delta C < min\delta C$ and $\delta S < min\delta S$ break
End loop
$R \leftarrow Rulesgeneratedatpointcl$
```
Table 1 *Hill-climbing algorithm*

Table 2 confirms the picture suggested by the illustrations in Figures 1 and 2 (although note the correspondence is not always exact, as cross-validation was not used in obtaining the graphical representations). In almost all cases, an improved accuracy can be obtained from a pair of thresholds different from the default (1%, 50%) choice. As would be expected, the greatest gain from the hill-climbing procedure is in the case of TFPC, but a better accuracy is also obtained for CMAR in 21 of the 25 sets, and for CBA in 20. In a number of cases the improvement is substantial.

It is apparent that CBA, especially, can give very poor results with the default threshold values. In the cases of **ionosphere** and **wine**, the illustrations reveal the reason to be that a 1% support threshold leads, for these small data sets, to the selection of spurious rules. This is also the case for **zoo** and **hepatitis**, and for **mushroom**, where even a much larger data set includes misleading instances if a small support threshold is chosen. In the latter case the hill-climbing procedure has been ineffective in climbing out of the deep trough shown in the illustration for CBA. Notice that here the coverage analysis used in CMAR is much more successful in identifying the best rules, although TFPC also does relatively well.

As we observed from the illustrations, and as results reported in [8] also show, CMAR is generally less sensitive to the choice of thresholds. Both CMAR and CBA, however, give very poor results when, as in the cases of **chess** and **letrecog**, the chosen confidence threshold is too high, and CMAR performs relatively poorly for **led7** for the same reason. The extreme case is **chess**, where both CMAR and CBA (and TFPC) find no rules at the 50% confidence threshold. Notice, also, that for the largest data sets (those with more than 5000 cases) a support threshold lower than 1% almost always produces better results, although the additional candidate rules generated at this level will make coverage analysis more expensive.

In general, the results show that coverage analysis, especially in CMAR, is usually (although not always) effective in minimising any adverse effect from a poor choice of thresholds. Although TFPC with the default threshold values produces reasonably high accuracy in most cases, the lack of coverage analysis generally leads to somewhat lower accuracy than one or both of the other methods. Interestingly, however, the results when the hill-climbing procedure is applied to TFPC show that high accuracy can be obtained without coverage analysis if a good choice of thresholds is made. In 18 of the 25 cases, the accuracy of TFPC after hill-climbing is as good or better than that of CMAR with the default thresholds, and in only one case (**wine**) is it substantially worse, the hill-climbing in this case failing to find the peak of the rather irregular terrain shown in the illustration. Conversely, the result for **penDig** demonstrates a case in which the hill-climbing procedure of TFPC works better than the coverage analysis of CMAR in identifying important low-support, high-confidence rules. The results also improve on CBA in 14 cases, often by a large margin. Overall this suggests that a good choice of thresholds can eliminate the need for coverage analysis procedures.

The significance of this is that coverage analysis is relatively expensive, especially if the data set and/or the number of candidate rules is large, as is likely to be the case if a low support threshold is chosen. The final four columns of Table 2 give a comparison of the total execution times to construct a classifier with ten-fold cross validation, using TFPC, with or without the hill-climbing procedure, and for CMAR and CBA (for the 1%, 50% thresholds). These figures were obtained using our Java implementations on a single Celeron 1.2 Ghz CPU with 512 MBytes of RAM.

Data set	**TFPC**				**CMAR**				**CBA**				**Execution Time**			
	Def. val.	**HC**	**“best” t’hold**		**Def. val.**	**HC**	**“best” t’hold**		**Def. val.**	**HC**	**“best” t’hold**		TFPC	TFPC HC	CMAR	CPAR
			S	**C**			**S**	**C**			**S**	**C**				
adult.D97.N48842.C2	80.8	81.0	0.2	50.1	80.1	80.9	0.7	50.0	84.2	84.6	0.1	48.4	2.9	20.0	78.0	230.0
anneal.D73.N898.C6	88.3	90.1	0.4	49.1	90.7	91.8	0.4	50.0	94.7	96.5	0.8	46.8	0.5	2.7	2.3	5.8
auto.D137.N205.C7	70.6	75.1	2.0	52.4	79.5	80.0	1.2	50.0	45.5	77.5	2.7	50.8	3.3	61.7	703.9	536.3
breast.D20.N699.C2	90.0	90.0	1.0	50.0	91.2	91.2	1.0	50.0	94.1	94.1	1.0	50.0	0.3	0.3	0.4	0.6
chess.D58.N28056.C18	0.0	38.0	0.1	25.2	0.0	34.6	0.1	11.0	0.0	39.8	0.1	24.0	2.1	46.7	2.0	2.0
cylBds.D124.N540.C2	68.3	74.4	1.2	49.8	75.7	77.8	1.3	49.9	75.7	78.0	1.9	50.0	4.0	163.9	206.9	923.6
flare.D39.N1389.C9	84.3	84.3	1.0	50.0	84.3	84.3	1.0	50.0	84.2	84.2	1.0	50.0	0.4	0.5	1.0	2.4
glass.D48.N214.C7	64.5	76.2	2.6	45.6	75.0	75.0	1.0	50.0	68.3	70.7	3.0	51.6	0.4	1.1	0.8	0.8
heart.D52.N303.C5	51.4	56.0	4.2	52.4	54.4	54.8	1.6	50.0	57.3	60.0	4.2	49.2	0.6	2.7	0.9	1.4
hepatitis.D56.N155.C2	81.2	83.8	1.6	51.6	81.0	82.8	2.9	50.0	57.8	83.8	7.1	48.4	0.6	2.4	2.4	10.0
horseCol.D85.D368.C2	79.1	79.9	1.2	50.2	81.1	81.9	2.9	50.0	79.2	83.9	5.5	49.2	0.5	2.1	10.5	66.5
ion’sph.D157.N351.C2	85.2	92.9	9.8	50.0	90.6	91.5	2.6	50.0	31.6	89.5	10.0	49.2	2.3	16.3	3066.8	2361.1
iris.D19.N150.C3	95.3	95.3	1.0	50.0	93.3	94.7	2.3	50.0	94.0	94.0	1.0	50.0	0.3	0.4	0.3	0.3
led7.D24.N3200.C10	57.3	62.7	2.2	49.4	62.2	67.4	1.3	40.4	66.6	68.0	1.0	46.0	0.4	1.4	0.6	0.7
letRc.D106.N20K.C26	26.4	47.6	0.1	32.3	25.5	45.5	0.1	31.8	28.6	58.9	0.1	13.7	3.7	196.2	17.2	20.5
mush’m.D90.N8124.C2	99.0	99.7	1.8	69.2	100.0	100.0	1.0	50.0	46.7	46.7	1.0	50.0	1.4	30.6	269.0	366.2
nurs’ry.D32.N12960.C5	77.8	89.9	1.0	73.2	88.3	90.1	0.8	62.6	90.1	91.2	1.5	50.0	1.3	21.3	5.8	6.9
pgBlks.D46.N5473.C5	90.0	90.0	1.0	50.0	90.0	90.3	0.2	50.0	90.9	91.0	1.6	50.0	0.3	0.7	0.8	2.2
penD.D89.N10992.C10	81.7	88.5	0.1	62.3	83.5	85.2	0.8	50.0	87.4	91.4	0.1	50.9	3.7	227.8	39.3	43.6
pima.D38.N768.C2	74.4	74.9	2.3	50.0	74.4	74.5	1.6	50.0	75.0	75.7	2.8	50.0	0.3	0.4	0.4	0.7
soyLrg.D118.N683.C19	89.1	91.4	1.1	49.1	90.8	91.8	0.8	51.6	91.0	92.9	0.6	52.2	9.8	644.3	405.6	273.8
ticTacToe.D29.N958.C2	67.1	96.5	1.5	74.2	93.5	94.4	1.6	50.0	100.0	100.0	1.0	50.0	0.4	4.5	1.0	1.6
wavef’m.D101.N5K.C3	66.7	76.6	3.2	64.3	76.2	77.2	0.6	50.0	77.6	78.2	2.6	50.0	3.7	210.8	167.3	93.4
wine.D68.N178.C3	72.1	81.9	4.5	51.2	93.1	94.3	2.3	50.0	53.2	65.5	4.8	50.0	0.3	1.0	7.3	11.7
zoo.D42.N101.C7	93.0	94.0	1.0	49.2	94.0	95.0	1.6	50.0	40.4	93.1	7.4	50.0	0.5	2.5	3.1	3.2

Table 2. *Accuracy and performance results (Default values: confidence = 50%, support = 1%)*

As would be expected, the execution times for TFPC (with default threshold values) are almost always far lower than for either of the other two methods. Less obviously, performing hill-climbing with TFPC is in many cases faster than coverage analysis with CMAR or CBA. In 13 of the 25 cases, this was the fastest procedure to obtain classification rules, and it is only markedly worse in cases such as **chess** and **letRecog**, where the other methods have failed to identify the rules necessary for good classification accuracy. These results suggest that TFPC with hill-climbing is an effective way of generating an accurate classifier which is often less costly than other methods.

6 CONCLUSIONS

In this paper we have shown that the choice of appropriate values for the support and confidence thresholds can have a significant effect on the accuracy of classifiers obtained by CARM algorithms. The coverage analysis performed by methods such as CMAR and CBA reduces this effect, but does not eliminate it. CMAR appears to be less sensitive than CBA to the choice of threshold values, but for both methods better accuracy can almost always be obtained by a good choice. We have also shown that, if threshold values are selected well, it is possible to obtain good classification rules using a simple and fast algorithm, TFPC, without the need for coverage analysis. We describe a procedure for finding these threshold values that will lead to good classification accuracy. Our results demonstrate that this approach can lead to improved classification accuracy, at a cost that is comparable to or lower than that of coverage analysis.

References

[1] Agrawal, R. and Srikant, R. (1994). *Fast algorithms for mining association rules*. Proc. 20th VLDB Conference, Morgan Kaufman, pp487-499.

[2] Coenen, F., Leng, P., Goulbourne, G. (2004). *Tree Structures for Mining Association Rules* Journal of Data Mining and Knowledge Discovery, Vol 8(1), pp.25-51

[3] Coenen, F.,Leng, P. and Zhang, L (2005). *Threshold Tuning for Improved Classification Association Rule Mining* Proc PAKDD 2005: LNCS 3518, Springer, pp216-225

[4] Cohen, W.W. (1995). *Fast Effective Rule Induction* Proc. of the 12th Int. Conf. on Machine Learning, p115-123.

[5] Christianin, N. and Shawe-Taylor. J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press.

[6] Fürnkranz, J. and Widme, F. (1994) *Incremental reduced error pruning.* Proc. 11th Int. Conf. on Machine Learning, Morgan Kaufmann, pp70-77.

[7] Han., J., Pei, J. and Yin, Y. (2000). *Mining Frequent Patterns without Candidate generation* Proc ACM Sigmod conf, Dallas, pp1-12.

[8] Li W., Han, J. and Pei, J. (2001). *CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules.* Proc ICDM 2001, pp369-376.

[9] Liu, B. Hsu, W. and Ma, Y (1998). *Integrating Classification and Association Rule Mining.* Proceedings KDD-98, New York, 27-31 August. AAAI. pp80-86.

[10] Pagallo, G. and Haussler, D. (1990) *Boolean feature discovery in Empirical data.* Machine Learning, Vol 5, No 1. pp71-99

[11] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo.

[12] Quinlan, J.R. (1986). *Induction of decision trees.* Machine Learning, No 1, pp81-106.

[13] Quinlan, J. R. and Cameron-Jones, R. M. (1993). *FOIL: A Midterm Report.* Proc. ECML, Vienna, Austria, pp3-20.

[14] Yin, X. and Han, J. (2003). *CPAR: Classification based on Predictive Association Rules.* Proc. SIAM Int. Conf. on Data Mining (SDM'03), San Francisco, CA, pp. 331-335.