School of IT
Technical Report

The University of Sydney

CCCS: A TOP-DOWN ASSOCIATIVE CLASSIFIER FOR IMBALANCED CLASS DISTRIBUTION

TECHNICAL REPORT NO 584

BAVANI ARUNASLAM AND SANJAY CHAWLA
SCHOOL OF INFORMATION TECHNOLOGIES
UNIVERSITY OF SYDNEY, AUSTRALIA

MARCH 2006

CCCS: A Top-down Associative Classifier for Imbalanced Class Distribution

Bavani Arunasalam
School of Information Technologies
University of Sydney
NSW, Australia
bavani@it.usyd.edu.au

Sanjay Chawla
School of Information Technologies
University of Sydney
NSW, Australia
chawla@it.usyd.edu.au

ABSTRACT

In this paper we propose CCCS, a new algorithm for classification based on association rule mining. The key innovation in CCCS is the use of a new measure, the "Complement Class Support (CCS)" whose application results in rules which are guaranteed to be positively correlated. Furthermore, the anti-monotonic property that CCS possesses has very different semantics *vis-a-vis* the traditional support measure. In particular, "good" rules have a low CCS value. This makes CCS an ideal measure to use in conjunction with a top-down algorithm. Finally, the nature of CCS allows the pruning of rules without the setting of any threshold parameter! To the best of our knowledge this is the first threshold-free algorithm in association rule mining for classification.

Keywords

Association Rules Mining, Classification, Skewed Data sets

1. INTRODUCTION

Classification is a core data mining and machine learning task. The objective in classification is to build a model (the classifier) which maps objects into pre-defined classes based on the attributes of objects. The methodology of evaluating a classifier consists of partitioning a data set into two subsets called the training and test data. The model is built on the training data and its accuracy is determined by measuring its ability to correctly classify the objects in the test data. The interested reader should consult a recent textbook on data mining (for example, [13]) for a survey of classification techniques.

In 1998, Liu, Tsa and Ma [12] introduced CBA, a new classification algorithm based on association rule mining [1]. Association rule mining is considered one of the major success stories of data mining research. Association Rules are traditionally described in the framework of market basket analysis. Given a set of items I and a set of transactions T consisting of subsets of I, an Association Rule is a relationship of the form $A \stackrel{s,c}{\rightarrow} B$, where A and B are subsets of I while s and c are the minimum support and confidence of the rule. A is called the antecedent and B the consequent of the rule. The support of a subset A of I is given by $\frac{\sigma(A)}{N}$, where $\sigma(A)$ is the number of transactions which contain A and N is the total number of transactions in the data set. The confidence of a rule $A \rightarrow B$ is given by $\frac{\sigma(A \cup B)}{\sigma(A)}$.

CBA mines association rules of the form $A \rightarrow C$, where A is a subset of the attribute set and C is an element of the set of pre-defined classes. The mining is performed on the training data. The set of mined rules are then used to classify instances in the test data. Several variations on the original CBA algorithm have emerged in the research literature [12, 14, 9] with the main challenges being:

1. To select an optimal subset of rules mined to build a classifier, and
2. Selecting an appropriate rule in order to classify new instances.

While several CBA-type algorithms are available (collectively called "Associative Classifiers") all of them use the traditional support measure to mine association rules. However using a minimum support threshold is inappropriate for data sets with highly imbalanced class distribution. For example in direct marketing applications, the response rate, the people who actually buy the product after responding to a promotion, is typically $1\% - 2\%$ [10]. Similarly in medical databases the percentage of people with actual disease is extremely small. In such applications setting the minimum support to even 1% is likely to prune most of the cases of interest. On the other hand using a low minimum support results in the generation of extremely large amount of rules.

While *support* is often used as a measure of significance of the rule (rules with high support intuitively capture an underlying process rather than being artifacts of the particular data set), the strength of the rule is captured by its *confidence*. However, even confidence can often be misleading measure in the case of imbalanced distribution. Consider the example, shown in Table.1 which shows the association between an item A and class C_1. A and $\bar{A}$ represent the presence and absence of item A respectively , C_1 represents a class and $\bar{C_1}$ represents the complement of C_1.

	C_1	$\bar{C_1}$	Total
A	20	5	25
$\bar{A}$	70	5	75
Total	90	10	100

Table 1: Example

Now consider the association rule $A \Rightarrow C_1$. The confidence of this rule is given by $conf(A \Rightarrow C_1) = \sigma(A \cup C_1)/\sigma(A)$ = 20/25= 80% and the support of the rule is $\sigma(A \cup C_1)/N$ =20%. Hence this rule has high confidence and support.

Now lets calculate the correlation between A and C_1. As given in [4], one way of calculating the correlation is to compute $P(A \cup C_1)/P(A) \times P(C_1) = 0.2/(0.25 \times 0.90) = 0.89$. The fact that this quantity is less than 1 indicates negative correlation between A and C_1. Such situations occur when the class distribution is imbalanced as in the example where class C_1 makes up 90% of the data.

We have designed a new measure, which we call the Complement Class Support (CCS), which tightly captures the relationship between the antecedent of the rule and its class. Given a rule $A \rightarrow C$,

$$CCS(A \rightarrow C) = \frac{\sigma(A \cup \bar{C})}{\sigma(\bar{C})}$$

Intuitively, CCS captures the strength of the rule in the complement class. If a rule is *strong* then its CCS should be small. Infact in Section 3, we will prove the following properties which explain why CCS can be used to design an efficient and accurate classifier for imbalanced data sets.

LEMMA 1. *For a rule $A \rightarrow C$, the following are equivalent*

a) A and C are positively correlated.

b) $CCS(A \rightarrow C) < \frac{\sigma(A)}{N}$.

c) $conf(A \rightarrow C) > \frac{\sigma(C)}{N}$.

d) $conf(A \rightarrow C) > 1 - \frac{\sigma(\bar{C})}{N}$.

Lemma 1.(b) asserts that for a rule to be positively correlated it is necessary and sufficient that the CCS of the rule be bounded by the support of the antecedent. This explains why the rule $A \rightarrow C_1$ in Table 1 is not positively correlated even though it has a high confidence: $CCS(A \rightarrow C_1) = 5/10 = 0.5 > \frac{\sigma(A)}{N}$.

Lemma 1.(c) shows that when the classes are equally distributed, by setting the *minconf* to 0.5, we can make sure that all the rules discovered are positively correlated. However, when the class distribution is imbalanced, the *minconf* has to be set to the support of the majority class in order for the rules discovered to be positively correlated rules. This is likely to prune most of the rules from the minority class and hence affect the accuracy of classification.

Assuming that we are dealing with a binary classification, Lemma 1.(d) asserts that the *minconf* for each class should be set in terms of the support of the other class to get positively correlated rules. If the class distributions are equal the *minconf* for each class would be 0.5 and this would be the same as using confidence alone. However, if the class distributions are imbalanced such as 90% and 10%, for the majority class the *minconf*=1-0.1=0.9 and for the minority class, *minconf*=1-0.9=0.1. By setting a lower value for the minority class we avoid pruning its rules and by setting a higher value for the majority class we avoid generating large number of rules.

The above discussion leads us to conclude that CCS is a better measure for imbalanced class distribution. It automatically selects the right confidence level for each class in order to guarantee that the rules discovered are positively correlated. Finally, it should be noted that CCS does not have a prescribed a pre-defined threshold value. At each node in the lattice, the CCS value will dynamically adjust (based on the support of the antecedent at that node) in order to generate only positively correlated rules.

In many cases, for reasons of efficieny, we may want to set a threshold value for CCS. In such situations we can take advantage of the fact that CCS enjoys the anti-monotonic property.

LEMMA 2. *Given a rule $A_i \Rightarrow C_j$, if $CCS(A_i \Rightarrow C_j) > t$ then, $CCS(subsetOf(A_i) \Rightarrow C_j) > t$.*

It is important to note that the anti-monotonicity of CCS has different semantics *vis-a-vis* the traditional *support* measure. In particular, "good" rules have low CCS value. As we will show later, we can combine CCS with top-down row enumeration algorithms to efficiently discover class rules. Even though row enumeration algorithms [5, 6, 7] use the support measure for pruning they cannot exploit the anti-monotonic property as they descend down the tree. Instead at each node, an upper bound on the maximum support value of all nodes rooted at that node is derived. The subtree is pruned if the maximum is below the support threshold. For CCS, no such bound is required. If a node has a high CCS value, then it (and possibly its descendants) can be pruned.

2. RELATED WORK

Associative classifier is a classification model that has shown a lot of promise recently. The first associative classifier, CBA, was proposed in 1998 [12]. Since then many algorithms have been developed to improve the efficiency and accuracy of classifications. These methods differ in three aspects as follows:

1. The technique used to mine the association rules efficiently
2. The measures and method used to select the best set of rules for the classifier.

3. The measures used to effectively predict the class of a new instance.

CBA follows the traditional apriori method to mine the associations rules that satisfy *minsup* and *minconf*. Rule selection for the classifier is based on confidence and support. CBA uses the database coverage method for building the classifier. In this method a rule is selected if it can classify at least one instance in the training set correctly and the rules with higher confidence have higher priority. New instances are classified to the class of the rule that has the highest confidence among the rules that cover the new instance.

Wang and Karypis proposed an instance centric classifier called Harmony [14]. This method uses a projection-based enumeration framework for mining the association rules which increases the efficiency of the mining process. Instead of the database coverage method, Harmony selects the highest confidence rule for each transaction in the training set to build the classifier which proves to yield higher accuracy of classification.

In the above two methods, the rule selection is purely based on confidence. However this method may not always give the correct classification especially when the class distribution is highly imbalanced. As discussed in Section 1, in imbalanced data sets, rules with high confidence could be actually negatively correlated.

CMAR [9] overcomes this problem by selecting rules for the classifier only if they pass the χ^2 test and ARC-PAN [2] by calculating the correlation coefficient. CMAR uses the FP-Growth method to mine the association rules and ARC-PAN uses the apriori method and then these rules are tested for positive correlation using the χ^2 test and correlation coefficient respectively by the two methods. We will show in Section 3 that by using Complement Class Support for pruning, we only generate positively correlated rules and this removes the need for further testing.

Further more, all of the above methods use *minsup* to prune rules and control the number of rules generated. However as we discussed in Section 1, this will prune most of the rules from the minority class in highly imbalanced databases. We overcome this problem by using Complement Class Support for pruning rules.

Recently the *row enumeration* method used in algorithms such as Farmer, RERII and RCBT [5, 6, 7] has proved to be very efficient in mining association rules from databases with extremely large number of attributes such as the micro array data sets. We adopt this method in our algorithm for two reasons.

First, imbalanced data sets such as medical databases typically consists of large number of attributes and relatively small number of transactions which makes it most suitable for row enumeration method.

Second, as the row enumeration tree grows the CCS of the nodes increase and we can prune all the nodes with CCS greater than the user-defined threshold. We will discuss this in detail in Section 4.

The remainder of the paper is arranged as follows: In Section 3 we present the basic definitions and notations of the concepts used in this paper and also present the theorems.In Section 4 we present our classification algorithm CCCS with an example. Finally in section 5 we present the experiments and results and Section 6 concludes the paper with a summary of the research.

3. BASIC DEFINITIONS AND NOTATION

Let I be a finite set of items and C a finite set of class labels. A row (or instance) is an element of the set $(2^I \times C)$. D is the set of all rows given. Assume $|D| = N$. A class rule is an implication of the form $A \rightarrow C_1$ where $A \subset I$ and $C_1 \in C$. Traditionally, the strength of of a class rule is defined in terms of *support* and *confidence*.

Definition 1. The Support of the rule $A_i \Rightarrow C_j$ is given by

$$Sup(A_i \Rightarrow C_j) = \frac{\sigma(A_i \cup C_j)}{N}$$

where $\sigma(A_i \cup C_j)$ the number of rows in D that contains A_i and C_j.

Definition 2. The confidence of the rule $A_i \Rightarrow C_j$ is a measure of conditional probability defined in terms of the support of the rule and the antecedent

$$Conf(A_i \Rightarrow C_j) = \frac{\sigma(A_i \cup C_j)}{\sigma(A_i)}$$

Definition 3. Given a rule $A \rightarrow C$, we measure the correlation between A and C based on the magnitude of the ratio $\frac{P(A \cup C)}{P(A)P(C)}$. In particular if

$$\frac{P(A \cup C)}{P(A)P(C)} \quad \begin{cases} > 1 & \text{then } A \text{ and } C \text{ are positively correlated} \\ < 1 & \text{then } A \text{ and } C \text{ are negatively correlated} \end{cases}$$

Definition 4. The Class Support of rule $A_i \Rightarrow C_j$ is the support of A_i in class C_j.

$$ClSup(A_i \Rightarrow C_j) = \frac{\sigma(A_i \cup C_j)}{\sigma(C_j)}$$

Definition 5. The Complement Class Support (CCS) of a rule $A_i \Rightarrow C_j$ is the support of A_i in the classes other than C_j. Let $\overline{C_j}$ denote all the classes in D other than C_j. Then

$$CCS(A_i \Rightarrow C_j) = \frac{\sigma(A_i \cup \overline{C_j})}{\sigma(\overline{C_j})}$$

Definition 6. The Strength Score of a rule $A_i \Rightarrow C_j$ is given by

$$SS(A_i \Rightarrow C_j) = \frac{Conf(A_i \Rightarrow C_i) \times ClSup(A_i \Rightarrow C_i)}{max(CCS(A_i \Rightarrow C_j), t)}$$

where t is set to a very low value such as 0.001 to avoid division by 0 when CCS =0.

We now show that CCS is anti-monotonic.

LEMMA 3. *Given a rule* $A_i \Rightarrow C_j$, *if* $CCS(A_i \Rightarrow C_j) > t$ *then,* $CCS(subsetOf(A$

4. CCCS ALGORITHM

We now introduce the Classification using Complement Class Support (CCCS) Algorithm.

The definition of CCS suggests that rules with lower CCS values are likely to be *stronger* as they are less frequently seen in other classes. This property along with the anti-monotonic property of CCS makes it possible for it to be integrated with a *row enumeration* method. As the tree grows, the CCS of the rules increase and hence the rules become less desirable and are candidates for being pruned. We show that CCS can be integrated with RERII [6] type algorithm. However, other row enumeration algorithms can be substituted in place of RERII. Row-enumeration algorithms were designed for data sets where the number of rows is much smaller than the number of columns. For example, microarray data sets fall in this category. However they can be used for small to moderate size data sets (like those in the UCI Repository[3]) even if the data sets do not match these characteristics.

We now explain our algorithm with the help of an example given in Table 2 and Figure 1. Table 2 is a transaction table with two classes C1 and C2 and 5 transactions. Figure 1 shows the enumeration tree for Table 2. Each node X in the tree is represented as X={itemlist, childlist, classlist}, where itemlist is the item set at X, childlist is the list of childnodes of X and classlist is the list of transaction ids in each class at X. For example, consider node(6), the itemlist of this node is EF, the childlist is {(10),(11),(12)} and the classlist is {[01],[02]} where transaction 01 belongs to C1 and 02 belongs to C2. Hence node(6) can be represented as {{EF},{(10),(11),(12)},{[01],[02]}}. In Figure 1, we show the ids of the transactions that belong to C1 above the itemlist and those belonging to C2 below the itemlist. The representation of a node in CCCS differs from RERII in using classlist instead of the support of the itemset. We need to track the transaction ids in each class to calculate the CCS and to build the classifier.

Basically CCCS takes each transaction and generates positively correlated rules between its items and class. Initially each transaction is added to the root of the enumeration tree. Then item sets are generated from a transaction by performing intersection with sibling nodes in the enumeration tree and these itemsets are added as child nodes of the transaction and the class of the transaction is passed on as the class of all its child nodes for the purpose of calculating the CCS. The tree is grown in a depth-first fashion by recursively performing intersection of each node with its sibling node. Each intersection will result in a child node whose itemlist is the intersection of the two nodes, whose child will be available in the next level and whose classlist will be the union of the classlists of the two nodes. The intersection at each node is performed in the same way as RERII according to Lemma 4.

LEMMA 4. *Let Xr_i and Xr_j be two sibling nodes, where $Xr_i < Xr_j$. The following properties hold:*

1. *If $Xr_i.itemlist \cap Xr_j.itemlist = 0$ nothing needs to be done.*
2. *If $Xr_i.itemlist = Xr_j.itemlist$, Xr_j will be integrated into Xr_i.*

   i.e. $Xr_i.classlist = Xr_i.classlist \cup Xr_j.classlist$ and Xr_j will be pruned.
3. *If $Xr_i.itemlist \subset Xr_j.itemlist$, then $Xr_i.classlist = Xr_i.classlist \cup Xr_j.classlist$.*
4. *If $Xr_i.itemlist \supset Xr_j.itemlist$, Xr_j will be pruned and Xr_j will become a child node of Xr_i.*
5. *If $Xr_i.itemlist \neq Xr_j.itemlist$, intersection of the nodes will become a new child node n' of Xr_i such that $n'.itemlist = Xr_i.itemlist \cap Xr_j.itemlist$ and $n'.classlist = Xr_i.classlist \cup Xr_j.classlist$.*

After intersection with all the siblings on the right of the enumeration tree, if a node is found to be negatively correlated ($CCS > support$), that rule is not considered as potential candidate for the class of the corresponding transaction. However it could be a potential candidate for the complement class. So the node is added to a potential candidate set and is included as a child node for the next transaction of the complement class.

In the following section we explain the algorithm in detail.

4.1 Mining Rules

Figures 2 and 3 present the CCCS algorithm. In line 5 the Rule base is initialized to 0. In line 6 the set of potential candidates for the complement class PC is initialized to 0. Line 7 adds all the transactions to the root node. These are nodes (1)-(5) in Figure 1.

In lines 8-17, each transaction is intersected with the transactions on its right in the enumeration tree by calling the MineRule procedure. The class of a transaction is assigned as the class of all its child nodes for the purpose of calculating the CCS. In lines 11-14, potential candidates belonging to the class of the current transaction are added to the node with other transaction.

Lines 18-48 presents the MineRules procedure which performs the intersection of the nodes according to Lemma 4. The child nodes are created depending on the type of intersection. For example in Figure 1, node (6) is the intersection of nodes (1) and (2). According to line 39-45, since the itemlists are not equal, the items in common, namely EF, are added as a child node (6) and the transaction ids of node (1) and (2) are added to class 1 (above) and class 2 (below) respectively. Consider the intersection between nodes (1) and (5). Since node (5) is a subset of node (1), according to line 31-37, node (5) is removed and DE is added as a child node(9) with the transaction ids of node (1) and (5).

The tree is grown in a depth-first fashion. When a node completes the intersection with all the nodes to its right, if the CCS of the node is less than the support, it is positively correlated and hence considered as a potential candidate for the classifier. For example, the class of node(14) is C1 since this is its parent transaction's class. Therefore CCS of node(14) is 0 as there are no transactions in C2 at this

node. The support at this node is 3/5= 0.6. Hence it is considered as a potential candidate for the classifier. However, node(6) has CCS of 1/2 and support = 2/5. Since the CCS of this node is greater than support it is not used to build the classifier.

It should be noted that if a node is negatively correlated, it does not mean that its child nodes also will be negatively correlated. For example, even if $\sigma(ABC_1)/\sigma(C_1) > \sigma(AB)/N$, it is possible to have $\sigma(AC_1)/\sigma(C_1) < \sigma(A)/N$.

CCCS has three variations depending on the way in which the nodes are pruned.

method 1 (Lines 49-58 in Figure 3).In this method we grow the tree till the last node, which will be the item set of size 1. At each node, after intersection with all its siblings on the right, if the CCS of the node is less than support, we take it as a potential candidate for the classifier. This method guarantees that we generate the complete set of positively correlated rules in the data set.

method 2 (Lines 59-70 in Figure 3). In this method we fix a user-defined threshold (*maxccs*) as the maximum value of CCS . We grow the tree until the CCS at a node is greater than *maxccs*. The anti-monotonic property of CCS enables us to prune all its child nodes because the CCS of the child nodes will definitely be greater than *maxccs*. At each node, if the CCS of the node is less than support, we take it as a potential candidate for the classifier. Though this method will not guarantee the completeness of the positively correlated rules, it will increase the efficiency of the algorithm.

method 3 (Lines 71-80 in Figure 3).In this method the tree is grown until the CCS of a node is greater than support. This eliminates the need for checking whether a rule is positively correlated, as all the rules we generate will have CCS less than support. This method also will not guarantee the completeness of the positively correlated rules. However it improves the accuracy of classification because of the following reason. Consider the two rules R1: $AB \Rightarrow C_1$ and R2: $A \Rightarrow C_1$. Let $CCS(AB \Rightarrow C_1) > \sigma(AB)/N$ and $CCS(A \Rightarrow C_1) < \sigma(A)/N$. Assume that we choose the second rule for the classifier. If a new instance, ABCD is to be classified, since the items in the rule R2 are contained in the new instance, it could be classified as C_1. But according to the first rule, AB and C_1 are negatively correlated. This may lead to error in classification. Therefore we choose a rule $R : A \Rightarrow C$ for the classifier only if every rule of the form $B \Rightarrow C$ is positively correlated, where $B \supset A$. Moreover, due to the anti-monotonic property of CCS, as the tree grows the CCS increases and the items will be more frequently seen in other classes which makes it less desirable for the classifier. We used this method in all our experiments.

In each of these methods, if the CCS of an item set is greater than the support of the antecedence we don't use it to build the classifier. However, the item set can have class support less than the support of the item. This means that the item set is a potential candidate for the complement class. So we add the node to PC and grow it with the next transaction of the complement class.

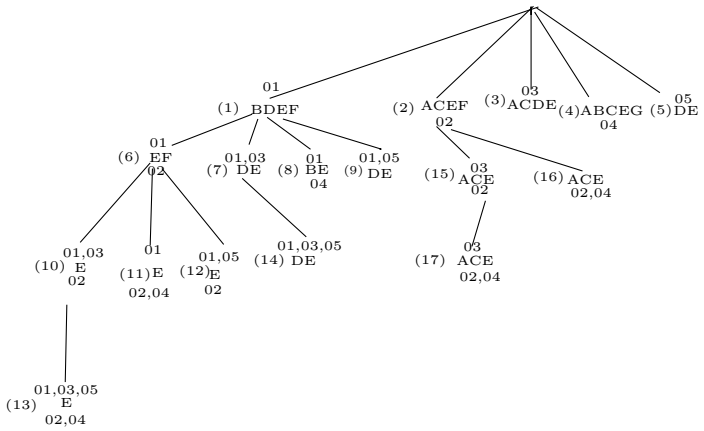

Figure 1: Row Enumeration Tree

Id	Transaction	Class
01	BDEF	C1
02	ACEF	C2
03	ACDE	C1
04	ABCEG	C2
05	DE	C1

Table 2: Transaction Table

4.2 Building the classifier

Lines 81-85 in Figure 3 show our rule selection method. Our rule selection is based on the *Score Strength (SS)*of a node. Our algorithm finds the best rule for each transaction in the training set and uses them for the classifier similar to Harmony [14]. In our case the best rule is considered as the rule with the highest *SS*. From Definition 6, it could be seen that the *SS* gives an overall measure of the rule as opposed to using only the confidence alone.

When a node has $CCS < support$, its score strength is calculated. If the transactions corresponding to the transaction ids in the class under consideration have no previous rules assigned, then this rule is assigned as the best rule. Otherwise the *SS* of the current and the pervious rules are compared and the rule with higher *SS* is assigned to the transaction.

4.3 Classifying New Instance

We classify new instances based on the CCS. Given a new instance, we select the rules that cover the new instance from each class of the classifier. The we compute the minimum CCS in each class. The class that has the minimum value for the minimum CCS chosen as the predicted class.

5. EXPERIMENTAL EVALUATION

In this section we report on the experiments that we have carried out to measure and assess the accuracy of CCCS.

Algorithm: CCCS

Input: Transaction table, t
Output: Classification Rules

1. Let D be the set of n transactions Tr_1,Tr_2,.....,Tr_n.
2. Let C be the set of class labels $c_1, c_2, c_3, ...c_m$ in D.
3. Let $Tr_i.items$ be the set of non-class attributes of Tr_i.
4. Let $Tr_i.class$ be the class label of Tr_i.

5. $Rules \leftarrow 0$
6. $PC \leftarrow 0$
7. Add the transactions to the root node
8. For each $Tr_i \epsilon D$
9. Let c_k be the class attribute of Tr_i.
10. Let N be the set of transactions $Tr_{i+1...n}$
11. For each node n in PC
12. if n.class = c_k
13. $N = N \cup n$
14. }
15. End For
16. MineRules[Tr_i, N, c_k,Rules]
17. End for

MineRules[Tr, N, c,Rules]
18. For each $n \epsilon N$
19. $N_i \leftarrow 0$ //set of intersections is set to 0//
20. $d = Tr.items \cap n.items$
21. if $|d| > 0$
22. if $Tr.items = n.items$
23. remove n from N
24. add n.id to Trn and $n'(n' \epsilon N)$
25. corresponding to n.class
26. }
27. if $Tr.items \subset n.items$
28. add n.id to Tr and $n'(n' \epsilon N)$
29. corresponding to n.class
30. }
31. if $Tr.items \supset n.items$
32. remove n from N
33. if d is not discovered before
34. add n' to N_i
35. $n'.tems = d$
36. add ids of Tr and n to n'
37. }
38. }
39. if $Tr.items \neq n.items$
40. if d is not discovered before
41. add n' to N_i
42. $n'.tems = d$
43. add ids of Tr and n to n'
44. }
45. }
46. }
47. End For
48.SelectRules[Rules,D,Tr,c]

Figure 2: Algorithm CCCS.

SelectRules: The three methods

SelectRules[Rules,D,Tr,c] - Method 1
49. If $Tr.CCS < sup(Tr.itemlist)$
50. BuildClassifier[Rules,D,Tr,c]
51. else if $Tr.ClSup < sup(Tr.itemlist)$
52. $PC = PC \cup Tr$
53. }
54. If $|N_i| > 0$
55. For each $n'_i \epsilon N_i$
56. MineRules[n'_i, N_i, c,Rules]
57. End For
58. }

SelectRules[Rules,D,Tr,c] - Method 2
59. If $Tr.CCS < maxccs$
60. If $Tr.CCS < sup(Tr.itemlist)$
61. BuildClassifier[Rules,D,Tr,c]
62. else if $Tr.ClSup < sup(Tr.itemlist)$
63. $PC = PC \cup Tr$
64 }
65. if $|N_i| > 0$
66. For each $n'_i \epsilon N_i$
67. MineRules[n'_i, N_i, c,Rules]
68. End For
69. }
70.}

SelectRules[Rules,D,Tr,c] - Method 3
71. If $Tr.CCS < sup(Tr.itemlist)$
72. BuildClassifier[Rules,D,Tr,c]
73. if $|N_i| > 0$
74. For each $n'_i \epsilon N_i$
75. MineRules[n'_i, N_i, c,Rules]
76. End For
77. }
78. else if $Tr.ClSup < sup(Tr.itemlist)$
79. $PC = PC \cup Tr$
80. }

BuildClassifier[Rules,D,Tr, c]
81. Calculatethe *Score Strength* of the rule
$R_i : Tr.items \Rightarrow c$
82. For each id in Tr
83. If current $SS >$ prev SS
84. set R_i as the best Rule
85. End For

Figure 3: The three Variations of the SelectRules procedure

Data set	# attributes	# rows	CBA	CCCS
Breast	10	699	4.2	3.1
Heart	14	270	18.5	18.1
Diabetes	9	768	25.3	23.9
Pima	9	768	27.6	27.9
Cleve	12	303	16.7	16.4
Australian	15	690	13.4	14.4
Average			17.6	17.3

Table 3: Comparison of Error Rates in UCI Data sets

All our comparisons are with CBA. The executable of the CBA program was downloaded from [11]. Our objective is to experimentally validate the hypothesis:

For data sets with an imbalanced (skewed) class distribution, CCCS will be more accurate compared to CBA

5.1 Environment

The CCCS algorithm was implemented in Java and all our experiments were performed on a 2GHz Intel Pentium machine with 1GB memory running Windows XP.

5.2 Data Sets and Preparation

Eight data sets were obtained from the UCI ML Repository[3]. For a fair comparison the continuous variables were discretization using the same techniques as described in [12].

5.3 Imbalanced Data Sets

The eight data sets were modified to create versions with different degrees of imbalance. These data sets were created as follows. For a data set D, let $D = D_1 \cup D_2$, where D_1 are D_2 are the instances with the majority and minority class respectively. To create an imbalanced class distribution of ratio $100 - x : x$, all the instances of the majority class were scaled to be $(100 - x)\%$ of the data. The size of minority class was then calculated as $m = \frac{|D_1| \times 100}{100-x} - |D_1|$. A random sample of size m was then selected from D_2.

For example, the Diabetes data set has 500 and 268 instances of the majority and minority class respectively. To create a set with 5% minority class, all the majority class instances were scaled to be 95% of the data. In this case $m = \frac{500x100}{95} - 500 = 26$. A random sample of size 26 was selected from the minority instances.

For each original data set, three versions of minority class size, 5%, 10% and 15% were created.

5.4 Methodology and Parameters

All the error rates reported are the average values of 10-fold cross validation. For CBA, the *minconf* was set to 50% and *minsup* was set to 1%.

5.5 Results

A comparision of the error rates of six of the eight data sets is shown in Table 3. While the error rates of CCCS is lower than CBA, a bootstrapping analysis (Section 5.6) shows that the differences are not significant.

Table 4 shows the results of CCCS and CBA on three separate versions each of the eight data sets. For the 5% data sets, CCCS outperforms CBA except on the Diabetes data. For the 10% data sets, CCCS again outperforms CBA except on the Pima data. At 15%, the accuracy of CCCS and CBA begins to converge. This confirms our hypothesis that CCCS is more suitable than CBA for imbalanced data sets.

5.6 Bootstrap Analysis

To determine if the differences between the error rates of CCCS and CBA are real (as opposed to being artifacts of this particular instances of the data) we carried out a bootstrapping analysis. The advantage of bootstrapping (as opposed to a paired t-test), is that no distributional assumption about the error rates is required.

In bootstrapping, a resampling method is used to determine the confidence bounds of statistical estimators [8]. In our case we have two vectors, *cbaerror* and *cccserror* of length eight (the number of data sets). We want an estimate on the average of vector difference between them.

In order to create multiple samples of the vector difference, *sampling with replacement* is carried out and the average of the difference is computed for each sample. In sampling with replacement the data point sampled is returned to the data set and made available to be selected again. This way several samples can be created and "histogrammed". For our particular case if the zero point (0) lies in the bulk of the histogram then one can conclude that there is no significant difference between the two error vectors.

Figure 4 shows the bootstrapping results (on 1000 samples) on the four different versions of the data set. For the original data, the zero point lies in the bulk of the histogram and we can conclude that there is no siginificant difference between the CBA and CCCS error rate. For the 5% data, the zero point lies on the edge of the histogram which allows us to conclude that the differences betwen the CBA and CCCS errors are different. Similarly for the 10% data, the zero point is far removed from the bulk of the histogram. We again conclude that the differences between the CBA and CCCS errors are significant. Finally, for the the 15% data set, the zero point again returns to the middle of the histogram - the error differences are not significant.

Note that from Figure 4, it appears that CCCS does much better than CBA for the 10% data compared to the 5% data. This is because, for the 5% data, the chance to create errors is limited to 5% for the positive class.

6. CONCLUSION

In the last decade extensive research has been carried out in the mining of association rules. In 1998, Liu, Tsa and Ma [12] introduced the CBA algorithm for classification using association rules. Since then several variations on the original algorithm have been introduced. However, till date all algorithms have used the traditional support measure to mine association rules. We have shown that for imbalanced class data sets, the support/confidence framework is inadequate. In order to address this problem we have introduced a new measure, the Complement Class Support (CCS). CCS has several properties which make it extremely suitable for

Data Set		TN		TP		FP		FN		Error Rate	
Breast	pos%	CCCS	CBA	CCCS	CBA	CCCS	CBA	CCCS	CBA	CCCS	CBA
	5%	94.17	94.37	3.21	2.41	0.83	0.63	1.80	2.58	2.63	3.21
	10%	88.10	88.73	7.43	5.92	2.02	1.38	2.45	4.03	4.47	5.41
	15%	83.43	83.94	12.95	11.34	1.64	1.11	1.98	3.58	3.62	4.69
Average diff		0.45		1.3		0.46		1.32		0.86	

Table 5: Comparison of Confusion Matrix for the Breast Cancer Data Set

Data set	positive%	#Pos	#Neg	CBA	CCCS
Breast	5%	25	458	3.21	2.63
	10%	51	458	5.41	4.47
	15%	81	458	4.69	3.62
Heart	5%	8	150	6.50	5.33
	10%	17	150	14.10	12.50
	15%	26	150	12.50	12.94
diabetes	5%	26	500	5.10	5.95
	10%	56	500	10.50	8.91
	15%	88	500	14.40	15.48
Cleve	5%	9	165	7.20	4.71
	10%	18	165	13.70	12.78
	15%	29	165	16.10	15.79
Pima	5%	26	500	5.10	5.00
	10%	56	500	9.80	10.00
	15%	88	500	14.40	14.53
Mushroom	5%	11	201	5.20	1.90
	10%	22	201	1.8	1.52
	15%	35	201	3.00	2.17
Horse	5%	12	232	8.50	7.53
	10%	26	232	11.20	10.42
	15%	41	232	13.50	13.30
Australian	5%	20	383	6.00	5.25
	10%	43	383	8.08	6.19
	15%	68	383	10.40	10.00
Average				8.77	8.04

Table 4: Comparison of Error Rates in Imbalanced Data sets

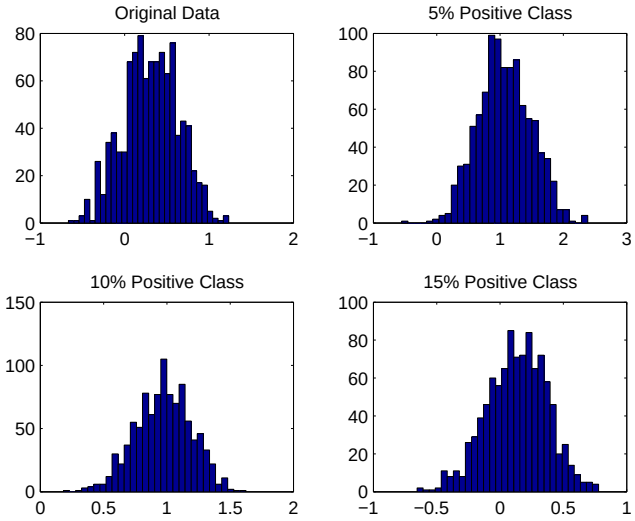

Figure 4: 1000 bootstrapped samples of the difference between the error rate of CBA and CCCS for the four different types of data sets. For the 5% and 10% data sets the zero point lies outside the bulk of the histogram. This provides quantitative evidence that the differences between error rates of CBA and CCCS are significant.

mining imbalanced data sets. The nature of CCS makes it an ideal candidate to be used in conjunction with a top-down row enumeration type algorithm. This is the essence of CCCS - a row enumeration algorithm with CCS guaranteed to generate positively correlated rules.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th International Conference on Very Large Data Bases VLDB Proceedings*, pages 487–499, 1994.

[2] M.-L. Antonie and O. R. Zaiane. An associative classifier based on positive and negative rules. In *9th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery(DMKD-04) Proceedings*, pages 64–69, 2004.

[3] C. Blake and C. Merz. *UCI KDD Archive*. http://kdd.ics.uci.edu/, 2000.

[4] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *ACM SIGMOD International Conference on Management of Data Proceedings*, pages 265–276, 1997.

[5] G. Cong, A. K.H.Tung, X. Xu, F. Pan, and J. Yang. Farmer: Finiding interesting rule groups in microarray datasets. In *23rd ACM SIGMOD International Conference on Management of Data Proceedings*, pages 145–154, 2004.

[6] G. Cong, K.-L. Tan, A. K.H.Tung, and F. Pan. Mining frequent closed patterns in microarray data. In *2004 IEEE International Conference on Data Mining (ICDM'04) Proceedings*, pages 363–366, 2004.

[7] G. Cong, K.-L. Tan, A. K.H.Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In *ACM SIGMOD/PODS 2005 Proceedings*, pages 670–681, 2005.

[8] M. Inc. *Matlab Statistical Toolbox*. Mathworks, 2005.

[9] W. Li, J. Han, and J. Pei. Cmar:accurate and efficient classification based on multiple class-association rules.

In *2001 IEEE International Conference on Data Mining (ICDM'01) Proceedings*, pages 369–376, 2001.

[10] C. X. Ling and C. Li. Data mining for direct marketing:problems and solutions. In *International Conference on Knowledge Discovery and Data Mining(KDD'98) Proceedings*, pages 73–79, 1998.

[11] B. Liu, W. Hsu, and Y. Ma. *Data Mining II*. http://www.comp.nus.edu.sg/ dm2/index.html, 1998.

[12] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *International Conference on Knowledge Discovery and Data Mining(KDD'98) Proceedings*, pages 80–86, 1998.

[13] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[14] J. Wang and G. Karypis. Harmony: Efficiently mining the best rules for classification. In *2005 SIAM International Conference on Data Mining(SDM'05) Proceedings*, 2005.

School of Information Technologies
Madsen Building F09
University of Sydney NSW 2006 AUSTRALIA
T: +61 2 9351 4917 F: +61 2 9351 3838
W: www.alumni.it.usyd.edu.au

ISBN 1 86487 823 1