Mining Association Rules from Tabular Data Guided by Maximal Frequent Itemsets

Q. Zou[1], Y. Chen[1], W.W. Chu[1], and X. Lu[2]

[1] Computer Science Department, University of California, Los Angeles, California, 90095
{zou,chenyu,wwc}@cs.ucla.edu
[2] Shandong University, Jinan, China
luxc@sdu.edu.cn

Summary. We propose the use of maximal frequent itemsets (MFIs) to derive association rules from tabular datasets. We first present an efficient method to derive MFIs directly from tabular data using the information from previous search, known as tail information. Then we utilize tabular format to derive MFI, which can reduce the search space and the time needed for support-counting. Tabular data allows us to use spreadsheet as a user interface. The spreadsheet functions enable users to conveniently search and sort rules. To effectively present large numbers of rules, we organize rules into hierarchical trees from general to specific on the spreadsheet Experimental results reveal that our proposed method of using tail information to generate MFI yields significant improvements over conventional methods. Using inverted indices to compute supports for itemsets is faster than the hash tree counting method. We have applied the proposed technique to a set of tabular data that was collected from surgery outcomes and that contains a large number of dependent attributes. The application of our technique was able to derive rules for physicians in assisting their clinical decisions.

1 Introduction

Many algorithms have been proposed on mining association rules in the past decade. Most of them are based on the transaction-type dataset. In the real world, a huge amount of data has been collected and stored in tabular datasets, which usually are dense datasets with a relatively small number of rows but a large number of columns. To mine a tabular dataset, previous approaches required transforming it into a transaction-type dataset in which column structures are removed. In contrast, we propose a new method that takes advantage of the tabular structure and that can mine association rules directly from tabular data.

Many previous approaches take four steps to generate association rules from a tabular dataset: (1) transforming a tabular dataset T into a transaction-type dataset D; (2) mining frequent itemsets (FI) or frequent closed itemsets (FCI) from D; (3) Generating rules from FI or FCI; (4) presenting rules to the user. This strategy achieves a certain degree of success but has three shortcomings:

First, transforming a tabular dataset into a transaction-type dataset increases the search space and the time required for support-counting, since column structures are removed. Figure 1a shows a table with five rows and five distinct columns, column A to E. The *Occ* column indicates the number of occurrences of a row. Thus, the *Occ* value can be used to count support of an itemset. For example, given a minimal support 3, "$A = 3$" is a frequent item since its support is 4, which can be obtained by adding the *Occ* values of rows 3 and 4. Likewise, both "$A = 1$" and "$A = 2$" have a support of 2, so they are infrequent. Figure 1b shows the corresponding transaction-type dataset where each column value is mapped to an item, e.g. $A = 1$ to a_1. There are 9 frequent items in Fig. 1b. Since any combination of the nine items can be a possible FI, FCI, or MFI, the search space is as large as $2^9 = 512$. In contrast, by keeping column structure, the search space in Fig. 1a can be significantly reduced to 162 (2*3*3*3*3) since the combinations of items on the same column can be excluded. Note that in Apriori-like level-wised approaches, keeping column structures reduces the number of candidates generated at all levels. For instance, in Fig. 1b, Apriori-like approaches would generate $b_1d_1d_2$ as a 3-item candidate since b_1d_1 and b_1d_2 are frequent. By using column constraints, d_1d_2 are in the same column, so any candidate containing d_1d_2 can be pruned. Furthermore, keeping column structure reduces the time needed for support-counting. For example, in vertical data representation, the supports of all items in one column can be counted in a single scan of the column.

Row No.	**A**	**B**	**C**	**D**	**E**	***Occ***
1	~~1~~	**2**	**1**	**1**	**1**	*2*
2	~~2~~	**1**	**2**	**1**	**1**	*1*
3	**3**	**2**	**2**	**2**	**1**	*2*
4	**3**	**1**	**2**	**2**	**2**	*2*
5	~~2~~	**1**	**1**	**2**	**2**	*1*
Frequent item#	*1*	*2*	*2*	*2*	*2*	

(a). A source tabulardata

TID	**Item set**	***Occ***
1	~~a_1~~ b_2 c_1 d_1 e_1	*2*
2	~~a_2~~ b_1 c_2 d_1 e_1	*1*
3	a_3 b_2 c_2 d_2 e_1	*2*
4	a_3 b_1 c_2 d_2 e_2	*2*
5	~~a_2~~ b_1 c_1 d_2 e_2	*1*
Totally 9 frequent items		

(b). A transaction-type dataset

Fig. 1. Tabular data *vs.* transaction data

Second, in certain situations, mining frequent itemsets (FI) or frequent closed itemsets (FCI) becomes difficult since the number of FIs or FCIs can be very large. Researchers have realized this problem and recently proposed a number of algorithms for mining maximal frequent itemsets (MFI) [3, 4, 6, 21], which achieve orders of magnitudes of improvement over mining FI or FCI.

When mining a dense tabular dataset, it is desirable to mine MFIs first and use them as a roadmap for rule mining.

Finally, a challenging problem is how to present a large number of rules effectively to domain experts. Some approaches prune and summarize rules into a small number of rules [10]; some cluster association rules into groups [14]. While reducing the number of rules or putting them into several clusters is desirable in many situations, these approaches are inconvenient when an expert needs to inspect more details or view the rules in different ways. Other approaches [11, 17] developed their own tools or a new query language similar to SQL to select rules, which provides great flexibility in studying rules. But, in some situations, domain experts are reluctant to learn such a query language.

In this Chapter, we present a new approach to address these problems.

- We propose a method that can generate MFIs directly from tabular data, eliminating the need for conversion to transaction-type datasets. By taking advantage of column structures, our method can significantly reduce the search space and the support counting time.
- We introduce a framework that uses MFIs as a roadmap for rule mining. A user can select a subset of MFIs to including certain attributes known as targets (e.g., surgery outcomes) in rule generation.
- To derive rules from MFIs, we propose an efficient method for counting the supports for the subsets of user-selected MFIs. We first build inverted indices for the collection of itemsets and then use the indices for support counting. Experimental results show that our approach is notably faster than the conventional hash tree counting method.
- To handle the large number of rules generated, we hierarchically organize rules into trees and use spreadsheet to present the rule trees. In a rule tree, general rules can be extended into more specific rules. A user can first exam the general rules and then extend to specific rules in the same tree. Based on spreadsheet's rich functionality, domain experts can easily filter or extend branches in the rule trees to create the best view for their interest.

The organization of this chapter is as follows. Section 1 discusses related works. Section 2 presents the framework of our proposed algorithm called SmartRule and the method of mining MFIs directly from tabular data. A new support counting method is introduced in Sect. 3. Then rule trees are developed in Sect. 4 to represent a set of related rules and their representations on the spreadsheet user interface. Performance comparison with transaction datasets is given in Sect. 5. Finally, an application example of using this technique on a medical clinical dataset for surgery consultation is given.

1.1 Related Works

Let I be a set of items and D be a set of transactions, where each transaction is an itemset. The support of an itemset is the number of transactions containing the itemset. An itemset is frequent if its support is at least a user-specified minimum support. Let FI denote the set of all frequent itemsets. An itemset is closed if there is no superset with the same support. The set of all frequent closed itemsets is denoted by FCI. A frequent itemset is called maximal if it is not a subset of any other frequent itemset. Let MFI denote the set of all maximal frequent itemsets. Any maximal frequent itemset X is a frequent closed itemset since no nontrivial superset of X is frequent. Thus we have $MFI \subseteq FCI \subseteq FI$.

Many algorithms for association rule mining require discovering FI before forming rules. Most methods for generating FI can be classified into three groups. First is the candidate set generate-and-test approach [1, 7, 12], which finds FI in a bottom up fashion. Second, the sampling approach [16] reduces computation complexity but the results are incomplete. Third is the data transformation approach [8, 20], which transforms a dataset to a new form for efficient mining. Some algorithms [13, 19] use FCI to generate rules.

Recently many MFI mining algorithms have been proposed. MaxMiner [4] uses a breadth-first search and performs look-ahead pruning on tree branches. The developments in mining MFI, however, use a depth first search with dynamic reordering as in DepthProject [2], Mafia [3], GenMax [6], and SmartMiner [21]. All of these methods for mining MFI have reported orders of magnitudes faster than methods mining FI/FCI. SmartMiner uses tail information to prune the search space without superset checking. Little research has been done on using MFIs for mining association rules since no rule can be generated from MFIs. However, MFIs can serve as roadmaps for rule mining. For example, given the set of MFIs, we can analyze many interesting properties of the dataset such as the longest pattern, the distribution and the overlap of the MFIs.

Extensive research has been done on association rule mining. For example, there has been research on mining multi-level association rules [9], on selecting the right interestingness measure [15], on synthesizing high frequency rules [18], etc.

Association rule mining techniques often produce a large number of rules that are hard to comprehend without further processing. Many techniques [10, 11, 14, 17] are proposed to prune, cluster, or query rules. The rules are usually presented in free text format, where it is easy to read a single rule but difficult to compare multiple rules at the same time. In this chapter, we propose using spreadsheets to present hierarchically organized rules to remedy this problem.

2 Methodology

Figure 2 illustrates SmartRule using an Excel book (or other spreadsheet software) to store data and mining results. The Excel book also serves as an interface for interacting with users. There are three functions in the system:

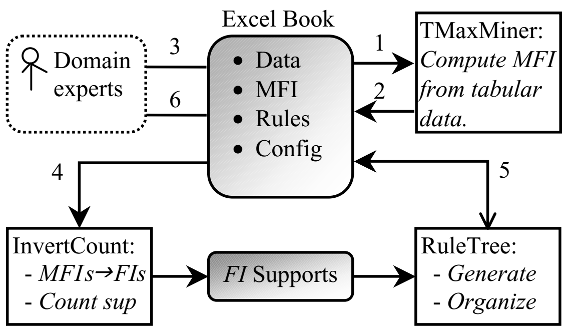

Excel Book
Domain experts
Data
MFI
Rules
Config
TMaxMiner: Compute MFI from tabular data.
InvertCount: - MFIs→FIs - Count sup
FI Supports
RuleTree: - Generate - Organize
1
2
3
4
5
6

Fig. 2. System overview of SmartRule

TMaxMiner, InvertCount, and RuleTree. TMaxMiner directly mines MFIs for a given minimal support from tabular data. InvertCount builds the FI list contained by the user-selected MFIs and counts the supports for the FIs (Sect. 3). RuleTree is used to generate rules and to organize them hierarchically (Sect. 4).

In this chapter, we assume that our data only contains categorical values. In the situation that a column contains continuous values, we use clustering technique (e.g., [5, 22]) to partition the values into several groups.

2.1 Tail Information

Let $N = X : Y$ be a node where X is the head of N and Y is the tail of N. Let M be known frequent itemsets and $N = X : Y$ be a node. The **tail information** of M to N is denoted as $Inf\ (N|M)$, and is defined as the tail parts of the frequent itemsets in $\{X : Y\}$ that can be inferred from M, that is, $Inf(N|M) = \{Y \cap Z | \forall Z \in M, X \subseteq Z\}$.

For example, $Inf(e : bcd|\{abcd, abe, ace\}) = \{b, c\}$, which means that eb and ec are frequent given $\{abcd, abe, ace\}$ frequent. For simplicity, we call tail information "information".

Since the tail of a node may contain many infrequent items, pure depth-first search is inefficient. Hence, dynamic reordering is used to prune away infrequent items from the tail of a node before exploring its sub nodes.

2.2 Mining MFI from Tabular Data

We now present TmaxMiner, which computes MFIs directly from a tabular dataset. TMaxMiner uses tail information to eliminate superset checking. TMaxMiner uses tables as its data model and selects the column with the least entropy for the next search.

To process the tabular data more efficiently, we start with the column of the least entropy. The entropy of a column can be computed by the probability of each item in that column. For example, for column A in Fig. 4a,

```
TMaxMiner(table T, inf)
1  Count sup and remove infrequent items from T;
2  Find pep, remove pep's columns, update inf;;
3  while(select x on a column of least entropy){
4     Rows in T containing x →table T';
5     The part of inf relevant to T'→inf';
6     mfi'= TMaxMiner(T', inf');
7     Add (x + mfi') to mfi;
8     Remove x from T;
9     Update inf with mfi';
10 }
11 return (pep + mfi);
```

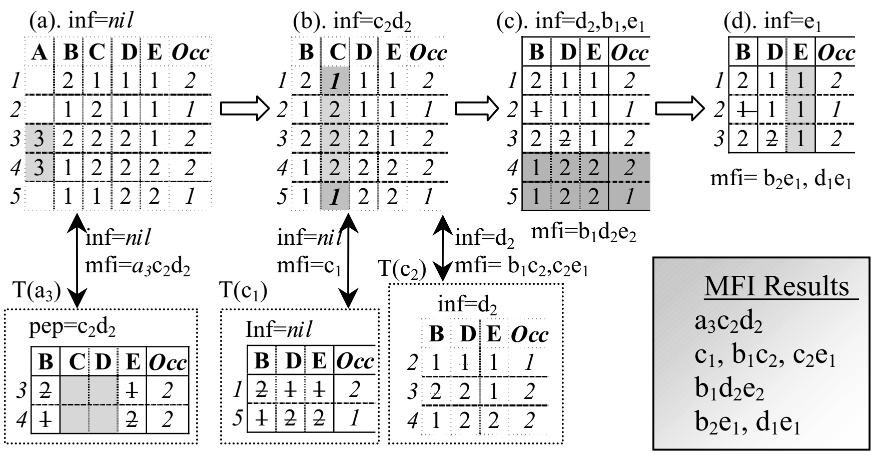
Fig. 3. TMaxMiner – a depth-first method that discovers MFI directly from a table guided by tail information

the probability of a_3 is $\{P(a_3) = 1\}$ where empty cells are ignored; thus the entropy of column A is $I(\{P(a_3) = 1\})$ is 0. The entropy of column B is $I(\{P(b_1) = 0.6, P(b_2) = 0.4\}) = -0.6 * \log_2 0.6 - 0.4 * \log_2 0.4 \approx 0.97$. So we start mining MFI from column A.

As shown in Fig. 3, TMaxMiner takes two parameters, a table T and information *inf*. It returns the discovered MFI in a depth-first fashion. The parameter T is the current table for processing and *inf* specifies the known frequent itemsets from the previous search. The TMaxMiner algorithm that uses tail information to derive MFIs is shown in Fig. 3. Line 1 computes the support for each item in T and then removes infrequent items. Parent equivalence pruning (PEP) [3] is defined as follows: Let x be a node's head and y be an element in its tail, if any transaction containing x also contains y, then move item y from the tail to the head. Line 2 finds the PEP items that appear in every row of T and remove the corresponding columns. Line 3 selects an item x in the column with the least entropy. Lines 4 to 10 find the MFIs containing x. Specifically, a new table T' is formed by selecting those rows of T containing x, and then removing the column of x. At Line 5, the part of *inf* mentioned x is assigned to *inf'*. Line 6 discovers MFI in the new table T', which is extended by the item x and added to *mfi*. Since we discovered the MFI containing x, we can remove x from T as in Line 8. Then *mfi'* is added into *inf* to inform the succeeding steps of the known frequent itemsets. Finally, the answer is returned at Line 11.

2.3 An Example of TMaxMiner

We shall show how TMaxMiner derives MFIs from the table as shown in Fig. 4a. Let minimum support threshold be equal to 3, we first select item a_3 as the first node since column A has the least entropy, and then yield the table $T(a_3)$, where items c_2 and d_2 appear in every row, i.e., $pep = c_2d_2$, and all other items are infrequent. Therefore we obtain MFI $a_3c_2d_2$ for $T(a_3)$.



(a). inf=*nil*

	A	B	C	D	E	*Occ*
1		2	1	1	1	*2*
2		1	2	1	1	*1*
3	3	2	2	2	1	*2*
4	3	1	2	2	2	*2*
5		1	1	2	2	*1*

(b). inf=c_2d_2

	B	C	D	E	*Occ*
1	2	***1***	1	1	*2*
2	1	2	1	1	*1*
3	2	2	2	1	*2*
4	1	2	2	2	*2*
5	1	***1***	2	2	*1*

(c). inf=d_2,b_1,e_1

	B	D	E	*Occ*
1	2	1	1	*2*
2	~~1~~	1	1	*1*
3	2	~~2~~	1	*2*
4	1	2	2	*2*
5	1	2	2	*1*

mfi=$b_1d_2e_2$

(d). inf=e_1

	B	D	E	*Occ*
1	2	1	1	*2*
2	~~1~~	1	1	*1*
3	2	~~2~~	1	*2*

mfi= b_2e_1, d_1e_1

T(a_3): pep=c_2d_2

	B	C	D	E	*Occ*
3	~~2~~			~~1~~	*2*
4	~~1~~			~~2~~	*2*

T(c_1): Inf=*nil*

	B	D	E	*Occ*
1	~~2~~	~~1~~	~~1~~	*2*
5	~~1~~	~~2~~	~~2~~	*1*

T(c_2): inf=d_2

	B	D	E	*Occ*
2	1	1	1	*1*
3	2	2	1	*2*
4	1	2	2	*2*

MFI Results
$a_3c_2d_2$
c_1, b_1c_2, c_2e_1
$b_1d_2e_2$
b_2e_1, d_1e_1

Fig. 4. An example of TMaxMiner: discover MFI directly from the tabular dataset

Removing column A from Fig. 4a, we have Fig. 4b with the tail information c_2d_2.

Next, column C in Fig. 4b is selected. We find a MFI c_1 from $T(c_1)$ and two MFIs b_1c_2 and c_2e_1 from $T(c_2)$. After removing column C, we have Fig. 4c with the tail information $\{d_2,\ b_1, e_1\}$. The rows 4 and 5 contain the same items and can be combined to form a MFI $b_1d_2e_2$. Then the two rows can be removed and the table shown in Fig. 4d is formed.

We find two more MFIs, b_2e_1 and d_1e_1, from Fig. 4d and the search process is now completed. The final results are the union of the MFIs in Fig. 4a–4d. They are $a_3c_2d_2$, c_1, b_1c_2, c_2e_1, $b_1d_2e_2$, b_2e_1, and d_1e_1.

3 Counting Support for Targeted Association Rules

Domain experts often wish to derive rules that contain a particular attribute. This can be accomplished by selecting a set of MFIs that contain such a target column. When the MFIs are very long, columns of less interest can be excluded.

The counting itemset C can be formed from the selected MFIs. For example, if the target column is column E in Fig. 1a, we can select the MFIs containing e_1 or e_2 and have $\{b_1d_2e_2,\ b_2e_1,\ c_2e_1,\ d_1e_1\}$. Then the counting itemsets C are all the subsets of the selected MFIs, $C = \{b_1,\ b_2,\ c_2,\ d_1,\ d_2,\ e_1,\ e_2,\ b_1d_2,\ b_1e_2,\ b_2e_1,\ c_2e_1,\ d_1e_1,\ d_2e_2,\ d_2b_1e_2\}$.

In order to generate rules, we need to count the support for the counting itemsets C from the source table T. There are two ways to determine which itemsets in C are contained in a row i in T: (1) for each subset s of i, check if s exists in C; (2) for each itemset c in C, check if c is contained in i. Clearly, both approaches are not efficient.

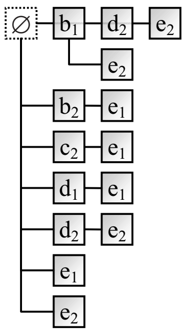


Fig. 5. Prefix tree for counting FIs

To solve this problem, some previous works build a hash tree from a prefix tree for the counting itemsets C. Figure 5 shows the prefix tree containing all the 14 itemsets in C. Then a hash table is added to each internal node to facilitate the search for a particular child. To find the itemset in C contained in a row i in T, we need to determine all the nodes in the prefix tree that match i. This approach is faster than the above two approaches, but the tree operations are relatively expensive.

We propose an algorithm InvertCount for counting supports that use only native CPU instructions such as "+1" and "*mov*".

Figure 6 illustrates that InvertCount employs inverted indices to count supports. Figure 6a shows the 14 itemsets for counting where the column *len* is the number of items in an itemset. The column *sup* stores the current counting results. The column *cnt* resets to 0 whenever the counting for one row is done. Figure 6b is the inverted indices *itm2ids* that map an *item* to a list of itemsets identifiers *ids*.

For a given row i, we can quickly determine the itemsets contained in i and increase their support by *Occ(i)*. For example, for the first row of Fig. 1a, $i = a_1b_2c_1d_1e_1$, we get the inverted indices (b_2 : 1, 9), (d_1 : 3, 11), and (e_1 : 5, 9, 10, 11). There is no inverted index for a_1 and c_1. Then we use the column *cnt* to count the occurrence of each *id* in the above three indices, e.g., 9 occurring twice. Finally we determine that itemsets 1, 3, 5, 9, 11 are contained in i since their occurrences are equal to the values on column *len* (itemsets' lengths). Therefore we increase their support by *Occ(i)*, i.e. 2, as shown on the column *sup*.

The algorithm InvertCount (shown in Fig. 7) counts the support for itemsets in C from source tabular data T without any expensive function calls. Line 1 builds the data structure as shown in Fig. 6. For each row i in T, we first find the itemsets in C that are contained in i and then increase their support by *Occ(i)*. Specifically, Lines 3 and 4 count the occurrence of *ids* in the inverted lists of the items in i. Line 7 increases the support of an itemset

id	itemset	sup	len	cnt
0	b_1	4	1	
1	b_2	4	1	1
2	c_2	5	1	
3	d_1	3	1	1
4	d_2	5	1	
5	e_1	5	1	1
6	e_2	3	1	
7	b_1d_2	3	2	
8	b_1e_2	3	2	
9	b_2e_1	4	2	2
10	c_2e_1	3	2	*1*
11	d_1e_1	3	2	2
12	d_2e_2	3	2	
13	$b_1d_2e_2$	3	3	

a. FI-table

item	ids
b_1	0, 7, 8, 13
b_2	1, 9
c_2	2, 10
d_1	3, 11
d_2	4, 7, 12, 13
e_1	5, 9, 10, 11
e_2	6, 8, 12, 13

b. itm2ids

Example:
Count from row 1:
$a_1b_2c_1d_1e_1$: *occ*=2
Get inverted index
b_2: 1, 9
d_1: 3, 11
e_1: 5, 9, 10, 11

Fig. 6. Example of InvertCount for efficient computing FI supports

```
InvertCount(table T, C)
1  build FI-table F and itm2ids from C;
2  foreach(row i in T) do
3    foreach(item x in i) do
4      foreach(id in itm2ids[x].ids) do
F.cnt[id]++;
5    foreach(item x in i) do
6      foreach(id in itm2ids[x].ids) do
7        if(F.cnt[id]>=F.len[id])
F.sup[id]+=Occ(i);
8        F.cnt[id]=0;
9  return F.sup;
```

Fig. 7. The InvertCount Algorithm – using inverted indices to count supports for the itemsets in C

by *Occ(i)* if the itemset is a subset of i. Line 8 clears the count for item *id* in FI-table for later use. We return the counting results at Line 9.

Our preliminary experimental results reveal that InvertCount is notably faster than previous hash tree counting.

4 Generating Rule Trees

Using the final support counting results, we are able to generate association rules. For example, Fig. 6a shows the final counting results in the column *sup*. Since we are looking for rules with e_1 or e_2 as rule head, we can build a rule for every itemset containing e_1 or e_2. For instance, for itemset $b_1d_2e_2$ containing

id	rule	Sup	Conf
5	$nil \rightarrow e_1$	5	1.0
6	$nil \rightarrow e_2$	3	1.0
8	$b_1 \rightarrow e_2$	3	0.75
9	$b_2 \rightarrow e_1$	4	1.0
10	$c_2 \rightarrow e_1$	3	0.6
11	$d_1 \rightarrow e_1$	3	1.0
12	$d_2 \rightarrow e_2$	3	0.6
13	$b_1d_2 \rightarrow e_2$	3	1.0

Fig. 8. A table of rules

e_2, a rule $b_1d_2 \rightarrow e_2$ is created. Figure 8 shows the list of rules created from Fig. 6a.

The support and confidence of a rule can be easily derived from the final counting result. For example, for the itemset d_1e_1 in Fig. 6a, we created a rule $d_1 \rightarrow e_1$ whose support is 3 and whose confidence is the support of d_1e_1 divided by that of d_1, which is equal to 1.0.

Association rule mining usually generates too many rules for the user to comprehend, so we need to organize rules into a hierarchical structure so that users can study the rules at different levels with varying degrees of specificity. Trees are built from the list of rules, where each tree represents rules sharing the same rule head. The hierarchical relationship on a tree is determined by the containment relationship among rules' bodies. A rule r_1 is an ancestor of r_2 if and only if the head of r_1 is equal to the head of r_2 and where the body of r_1 is a subset of the body of r_2.

Figure 9 shows the algorithm TreeRule that takes a list of rules R as input, sorted by increasing length, and returns trees with hierarchically organized rules. Specifically, it builds a tree for each target column value, as shown in Lines 2 and 3 where rules have empty bodies. For a rule with a non-empty body, we add it into the tree corresponding with its head, as shown in Lines 4 to 5.

Figure 10 shows a recursive method AddRule that adds a rule r into a tree. If a more specific rule r has similar support and confidence with the current

```
vector TreeRule(List R)
1  int i=0;
2  for(; R[i].body.len==0; i++) //nil→h
3    new Tree(R[i]) →gtr[R[i].head];
4  for(; i<R.Count; i++)
5    tr[R[i].head].AddRule(R[i]);
6  return tr;
```

Fig. 9. TreeRule – building a forest of trees from a list of rules

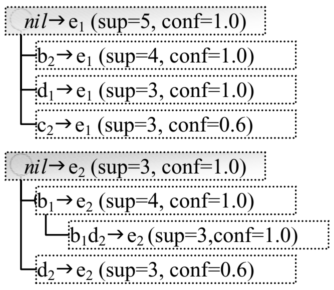
```
AddRule(rule r)
1  if(r.sup, r.conf)≈(this.sup, this.conf);
2    this.optItms += r.body-this.body;
3    return;
4  r.body -= this.optItms;
5  foreach(subtree t & t.body is a subset of
r.body)
6    t.AddRule(r);
7  if(no such a subtree)
8    t'=new Tree(r);
9    Add t' to this.subTrees;
10 return;
```

Fig. 10. AddRule – recursively adding a rule into a rule tree

rule node referred by *this*, then we simply add the extra items as optional items (optItms) without creating a new node as in Lines 1 to 3.

If the rule r is not similar to the current rule node, then we add r into sub-trees of *this* whose bodies are subsets of $r.body$ as in Lines 5 and 6. If no such sub tree exists, we create a new tree and add it into *this*.subTree as in Lines 7 to 9. Line 10 returns the updated tree r.

For example, Fig. 11a shows the rule trees for the rules in Fig. 8. Two rule trees are built: one for e_1 with 3 branches, another for e_2 with two levels of children. The second level node $b_1 \rightarrow e_2$ is the parent of $b_1 d_2 \rightarrow e_2$ because they both have the same head e_2, and b_1 is a subset of $b_1 d_2$. In general, a more specific rule gives higher confidence but lower support. Figure 11b represents the tree in table format, in which each node is represented as a row with its support, confidence, node depth and number information. To output trees into tabular format, we number nodes in the trees by their preorders. The node's number plus its depth can be used to represent the hierarchical relationship of the specific rules in a rule tree.


nil→ e_1 (sup=5, conf=1.0)
b_2→ e_1 (sup=4, conf=1.0)
d_1→ e_1 (sup=3, conf=1.0)
c_2→ e_1 (sup=3, conf=0.6)
nil→ e_2 (sup=3, conf=1.0)
b_1→ e_2 (sup=4, conf=1.0)
b_1d_2→ e_2 (sup=3,conf=1.0)
d_2→ e_2 (sup=3, conf=0.6)

a. Example of rule trees.

Rules	Sup	Conf	Depth	Num
***nil*→ e_1**	**5**	**1.0**	**0**	**0**
$b_2 \rightarrow e_1$	4	1.0	1	1
$d_1 \rightarrow e_1$	3	1.0	1	2
$c_2 \rightarrow e_1$	3	0.6	1	3
***nil*→ e_2**	**3**	**1.0**	**0**	**4**
$b_1 \rightarrow e_2$	3	0.75	1	5
$b_1d_2 \rightarrow e_2$	3	1.0	2	6
$d_2 \rightarrow e_2$	3	0.6	1	7

b. Rule trees represented by node *Depth* and *Number*

Fig. 11. Example of rule trees and their tabular representation

5 Hybrid Clustering Technique for Partitioning Continuous Attributes

The number of cells and the cell size will affect the clustering results and the data mining outcome, both in support and confidence.

When the sample size is very small and number of attributes is large, conventional statistical classification techniques such as CART [22] fail to classify the continuous value of an attribute into cells. Using unsupervised clustering techniques [5] has the problem of not knowing the optimal number of cells to represent the variables. Therefore, we developed a hybrid technique that combines both statistical and data mining techniques iteratively in determining the optimal number of cells as well as the cell sizes.

The basic idea is to use data mining technique to select a small set of key attributes, and then use a statistical classification technique such as CART to determine the cell sizes and number of cells from the training set. Then we use the partitioning result for data mining. The procedure works as follows:

Perform the mining on the training set, and select a set of attributes with high confidence and support for statistical classification.

Perform statistical classification based on the training set for the selected attributes set from step (1). Since the attribute set is greatly reduced, statistical classification techniques such as CART [22] can be used to determine the optimal number of cells and their corresponding cell sizes for attribute.

Based on the optimal cell sizes for each attribute from step (2), data mining algorithms can then be used to generate the rules for this set of attributes.

With this hybrid clustering approach, we are able to generate optimal partitioning for the set of continuous attributes. Our experimental results reveal that deriving rules based on such partitioning yield better mining results than conventional unsupervised clustering techniques [5].

6 Performance Comparisons

SmartRule was implemented using Microsoft .Net C# and the Office XP primary interop assemblies so that our program could directly read from and write to Excel workbooks. Experimental results have shown that SmartMiner is close to one order of magnitude faster than Mafia and Genmax in generating MFI from transaction dataset [21]. Note SmartMiner, Mafia and GenMax do not keep column constraints during generating MFIs. By taking advantage of column constraints in tabular data format, TmaxMiner achieves performance gains over SmartMiner as shown in Fig. 12. Further, we note that the gain increases as the support decreases. The dataset Mushroom used for performance comparison is in tabular format and was downloaded from the UCI machine learning repository [23].

To evaluate the performance of InvertCount, we used the Mushroom dataset and selected the MFIs containing the class attributes (*edible* or

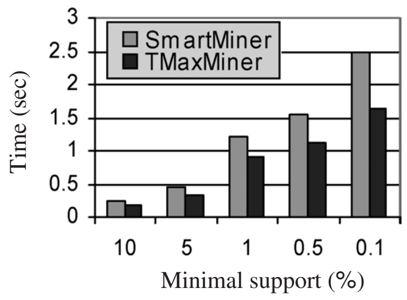


Fig. 12. Comparison of TMaxMiner with SmartMiner

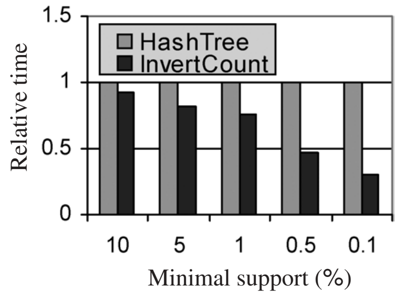


Fig. 13. Comparison of InvertCount with Hashtree

poisonous) to generate counting itemsets. Before testing, we build a hash tree and inverted indices for these itemsets. Then we compared the counting time of InvertCount and HashTree in the source Mushroom table.

Figure 13 shows the relative time of HashTree and InvertCount for the tests at varying minimal supports. When minimal support is 10%, they have similar performances because of the small number of counting itemsets. As the minimal support decreases, MFIs become longer, which results in an increase in the number of counting itemsets. In such a case, InvertCount is notably faster than HashTree.

7 An Application Example

We have applied SmartRule to mine a set of clinical data that was collected from urology surgeries during 1995 to 2002 at the UCLA Pediatric Urology Clinic. The dataset contains 130 rows (each row represents a patient) and 28 columns, which describe patient pre-operative conditions, type of surgery performed, post-op complications and final outcome of the surgeries. The pre-operative conditions include patient ambulatory status (A), catheterizing skills (CS), amount of creatinine in the blood (SerumCrPre), leak point pressure (LPP), and urodynamics, such as minimum volume infused into bladder

Table 1. Four types of surgeries

Operation Type	Operation Description
Op-1	Bladder Neck Reconstruction with Augmentation
Op-2	Bladder Neck Reconstruction without Augmentation
Op-3	Bladder Neck Closure without Augmentation
Op-4	Bladder Neck Closure with Augmentation

when pressure reached 20 cm of water (20%min). The data mining goal is to derive a set of rules from the clinical data set (training set) that summarize the outcome based on patients' pre-op data. This knowledge base can then be used to examine a given patient pre-op profile and decide which operation should be performed to achieve the best outcome.

This set of clinical data represents four types of surgery operations as shown in Table 1. We separate the patients into four groups based on the type of surgery they were treated with. For a given surgery type, we partitioned the continuous value attributes, e.g., patient urodynamics data, into discrete intervals or cells. To achieve best mining results, an attribute may partition into different cell sizes for different types of operations, as shown in Table 2. Since our sample size is very small, especially after subgrouping the dataset into different operations, we used the hybrid technique that combines both statistical and data mining techniques iteratively in determining the optimal number of cells as well as the cell sizes.

For the training set, we are able to generate optimal partitioning for the set of continuous attributes from each operation type as shown in Table 2. Sets of rules are generated based on the discretized variables for each type of operation, which can be viewed as the knowledge base for this type of operation. Our experimental results reveal that deriving rules based on such partitioning yield better mining results than conventional unsupervised clustering techniques [5].

For a given patient with a specific set of pre-op conditions, the generated rules from the training set can be used to predict success or failure rate for a specific operation.

To provide the user with a family of rules, SmartRule can organize matched rules into rule trees from general to specific rules. The rules closer to the root are more general, contain fewer constraints and yield higher support; the rules closer to the leaves are more specific, which contain more constraints and yield lower support. In case of multiple match rules, the quality of the rules in terms of confidence and support may be used in rule selection.

Given patient Matt's pre-op conditions as shown in Table 3(a), since the attributes in the rules are represented in discrete values, the continuous pre-op conditions are transformed into discrete values (Table 3(b)) based on the partitioning done on the attributes of operation types as shown in Table 2.

Table 2. Partition of continuous variables into optimal number of discrete intervals (cells) and cell sizes for four types of operations. Each table presents the partitioning for a specific operation type. The optimal number of cells for an attribute is represented by the number of rows in each table. The size for each cell is represented in the column. Since different attributes have different optimal number of cells, certain attributes may contain no values in certain rows, and we use n/a to designate such a undefined cell sizes

(a) Operation Type 1

Cell#	LPP	SerumCrPre
1	[0, 19]	[0, 0.75]
2	(19, 33.5]	[0.75, 2.2]
3	(33.5,40]	n/a
4	normal	n/a

(b) Operation Type 2

Cell#	20% min	20% mean	30% min	30% mean	LPP	SerumCrPre
1	[80, 118]	[50, 77]	[100, 170]	[51, 51]	[12, 20]	[0, 0.5]
2	[145, 178]	[88, 104]	[206, 241]	[94, 113]	[24, 36]	[0.7, 1.4]
3	[221, 264]	[135, 135]	n/a	[135, 135]	normal	n/a

(c) Operation Type 3

Cell#	20% min	20% mean	30% min	30% mean	LPP	SerumCrPre
1	[103,130]	[57, 75]	[129, 157]	[86, 93]	[6, 29]	[0.3, 0.7]
2	[156,225]	[92, 105]	[188, 223]	[100,121]	[30,40]	[1.0, 1.5]

(d) Operation Type 4

Cell#	LPP	20% mean
1	[0, 19]	[0, 33.37]
2	(19, 69]	(33.37, 37.5]
3	normal	(37.5, 52]
4	n/a	(52, 110]

Based on these attributes values of a given patient such as "Ambulatory Status=4" and "CathSkills=1", we can search for rules from the knowledge base that were generated from the training set to match Matt's pre-op profile as shown in Table 4.

Table 3.(a). Patient Matt's pre-operative conditions

Ambulatory Status (A)	Cath Skills (CS)	Serum CrPre	20% min	20% mean(M)	30% min	30% mean	LPP	UPP
4	1	0.5	31	20	50	33	27	unknown

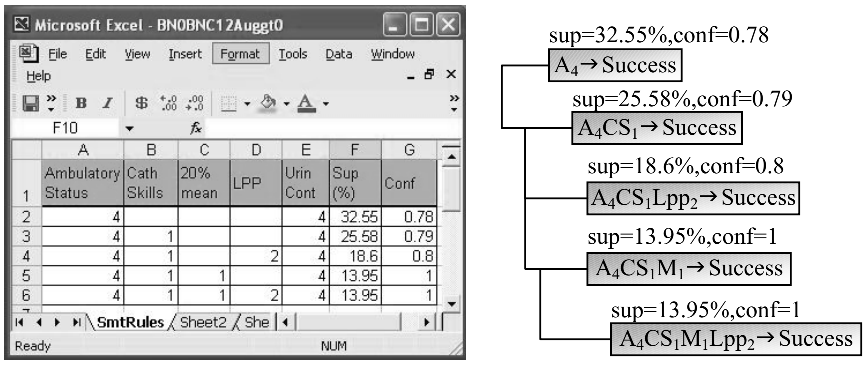
Table 3.(b). Discretized pre-operative conditions of patient Matt's pre-op conditions. The attributes not used in rule generation are denoted as n/a

	Ambulatory Status (A)	Cath Skills (CS)	Serum CrPre	20% min	20% mean(M)	30% min	30% mean	LPP
Op-1	4	1	1	n/a	n/a	n/a	n/a	2
Op-2	4	1	1	<1	<1	<1	<1	2
Op-3	4	1	1	<1	<1	<1	<1	1
Op-4	4	1	n/a	n/a	1	n/a	n/a	2

Table 4. Rule trees selected from the knowledge base (derived form the training set) that match patient Matt's pre-op profile

Surgery	Conditions	Outcome	Support	Support(%)	Confidence
Op-1	CS = 1	Success	10	41.67	0.77
	CS = 1 and LPP = 2	Success	3	12.5	0.75
Op-2	CS = 1 and LPP = 2	Fail	2	16.67	0.67
	20%min = 1 and LPP = 2	Fail	2	16.67	0.67
Op-3	CS = 1 and SerumCrPre = 1	Success	5	50	0.83
	CS = 1, SerumCrPre = 1 and LPP = 1	Success	2	20	1
Op-4	$A = 4$	Success	14	32.55	0.78
	$A = 4$ and CS = 1	Success	11	25.58	0.79
	$A = 4$, CS = 1 and LPP = 2	Success	8	18.6	0.8
	$A = 4$, CS = 1 and $M = 1$	Success	6	13.95	1
	$A = 4$, CS = 1, $M = 1$ and LPP = 2	Success	6	13.95	1

Based on the rule tree, we note that Operations 3 and 4 both match patient Matt's pre-op conditions. However, Operation 4 matches more attributes in Matt's pre-op conditions than Operation 3. Thus, Operation 4 is more desirable for patient Matt. A screen shot of the corresponding spreadsheet user interface is shown in Fig. 14a and the corresponding rule tree representation is shown in Fig. 14b. We have received favorable user feedback in using the spreadsheet interface because of its ease in rule searching and sorting.


Microsoft Excel - BN0BNC12Auggt0
File Edit View Insert Format Tools Data Window Help
F10
SmtRules Sheet2 She
Ready
NUM
sup=32.55%,conf=0.78
$A_4 \rightarrow$ Success
sup=25.58%,conf=0.79
$A_4CS_1 \rightarrow$ Success
sup=18.6%,conf=0.8
$A_4CS_1Lpp_2 \rightarrow$ Success
sup=13.95%,conf=1
$A_4CS_1M_1 \rightarrow$ Success
sup=13.95%,conf=1
$A_4CS_1M_1Lpp_2 \rightarrow$ Success

	A	B	C	D	E	F	G
1	Ambulatory Status	Cath Skills	20% mean	LPP	Urin Cont	Sup (%)	Conf
2	4				4	32.55	0.78
3	4	1			4	25.58	0.79
4	4	1		2	4	18.6	0.8
5	4	1	1		4	13.95	1
6	4	1	1	2	4	13.95	1

(a) Represent rule trees for Op-4 by spreadsheet (b) Rule tree for Op-4

Fig. 14. Representing rules in a hierarchical structure for the example

8 Conclusion

In this chapter, we have proposed a method to derive association rules directly from tabular data with a large number of dependent variables. The SmartRule algorithm is able to use table structures to reduce the search space and the counting time for mining maximal frequent itemsets (MFI). Our experimental results reveal that using tabular data rather than transforming to transaction-type data can significantly improve the performance of mining MFIs. Using simple data structures and native CPU instructions, the proposed InvertCount is faster than hash tree for support counting. Finally, SmartRule organizes rules into hierarchical rule trees and uses spreadsheet as a user interface to sort, filter and select rules so that users can browse only a small number of interesting rules that they wish to study. We have successfully applied SmartRule to a set of medical clinical data and have derived useful rules for recommending the type of surgical operation for patients based on their pre-operative conditions and demography information.

Acknowledgements

The authors wish to thank Dr. B. Churchill and Dr. Andy Chen for providing the clinical data as well as collaborating in formulating the data mining model, and Professor James W. Sayre for his stimulating discussion in developing the statistical and data mining approach in determining the cell size. This research is supported by the NIH PPG Grant #4442511-33780 and the NSF IIS Grant #97438.

References

1. R. Agrawal and R. Srikant: Fast algorithms for mining association rules. In Proceedings of the 20th VLDB Conference, Santiago, Chile, 1994. 166
2. R. Agarwal, C. Aggarwal, and V. Prasad: A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing, 2001. 166
3. D. Burdick, M. Calimlim, and J. Gehrke: MAFIA: a maximal frequent itemset algorithm for transactional databases. In Intl. Conf. on Data Engineering, Apr. 2001. 164, 166, 168
4. R. Bayardo: Efficiently mining long patterns from databases. In ACM SIGMOD Conference, 1998. 164, 166
5. W.W. Chu, K. Chiang, C. Hsu, and H. Yau: An Error-based Conceptual Clustering Method for Providing Approximate Query Answers Communications of ACM. 1996. 167, 174, 176
6. K. Gouda and M.J. Zaki: Efficiently Mining Maximal Frequent Itemsets. Proc. of the IEEE Int. Conference on Data Mining, San Jose, 2001. 164, 166
7. H. Mannila, H. Toivonen, and A.I. Verkamo: Efficient algorithms for discovering association rules. In KDD-94: AAAI Workshop on Knowledge Discovery in Databases, pp. 181–192, Seattle, Washington, July 1994. 166
8. J. Han, J. Pei, and Y. Yin: Mining Frequent Patterns without Candidate Generation, Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000. 166
9. J. Han and Y. Fu: Discovery of Multiple-Level Association Rules from Large Databases. In Proc. of the 21th Int. Conf. on Very Large Databases, Zurich, Swizerland, 1995. 166
10. B. Liu, W. Hsu, and Y. Ma: Pruning and summarizing the discovered associations. In Proc. of the Fifth Int'l Conference on Knowledge Discovery and Data Mining, pp. 125–134, San Diego, CA, August 1999. 165, 166
11. B. Liu, M. Hu, and W. Hsu: Multi-level organization and summarization of the discovered rules. Proc. ACM SIGKDD, 208–217, 2000. 165, 166
12. J.S. Park, M. Chen, and P.S. Yu: An effective hash based algorithm for mining association rules. In Proc. ACM SIGMOD Intl. Conf. Management of Data, May 1995. 166
13. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal: Discovering frequent closed itemsets for association rules. In 7th Intl. Conf. on Database Theory, January 1999. 166
14. B. Lent, A.N. Swami, and J. Widom: Clustering association rules. In Proceedings of International Conference on Data Engineering, 1997. 165, 166
15. P. Tan, V. Kumar, and J. Srivastava: Selecting the Right Interestingness Measure for Association Patterns (2002). Proc of the Eighth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-2002). 166
16. Hannu Toivonen. Sampling large databases for association rules. In Proc. of the VLDB Conference, Bombay, India, September 1996. 166
17. A. Tuzhilin and B. Liu: Querying multiple sets of discovered rules. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Edmonton, Canada, July 23–26, 2002. 165, 166
18. X. Wu and S. Zhang, Synthesizing High-Frequency Rules from Different Data Sources, IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 2, March/April 2003, 353–367. 166

19. M.J. Zaki and C. Hsiao: Charm: An efficient algorithm for closed association rule mining. In Technical Report 99–10, Computer Science, Rensselaer Polytechnic Institute, 1999. 166
20. Q. Zou, W. Chu, D. Johnson, and H. Chiu: Pattern Decomposition Algorithm for Data Mining of Frequent Patterns. Journal of Knowledge and Information System, 2002. 166
21. Q. Zou, W. Chu, B. Lu: SmartMiner: A Depth First Algorithm Guided by Tail Information for Mining Maximal Frequent Itemsets. Proc. of the IEEE Int. Conference on Data Mining, Japan, 2002. 164, 166, 174
22. http://www.salford-systems.com/products-cart.html 167, 174
23. http://www.ics.uci.edu/∼mlearn/MLRepository.html 174