Mining Progressive Confident Rules

Minghua Zhang, Wynne Hsu, Mong Li Lee
School of Computing, National University of Singapore
Singapore 117543, Singapore
{zhangmh, whsu, leeml}@comp.nus.edu.sg

ABSTRACT

Many real world objects have states that change over time. By tracking the state sequences of these objects, we can study their behavior and take preventive measures before they reach some undesirable states. In this paper, we propose a new kind of pattern called progressive confident rules to describe sequences of states with an increasing confidence that lead to a particular end state. We give a formal definition of progressive confident rules and their concise set. We devise pruning strategies to reduce the enormous search space. Experiment result shows that the proposed algorithm is efficient and scalable. We also demonstrate the application of progressive confident rules in classification.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: DATABASE MANAGEMENT—*Database Applications, Data Mining*

General Terms

Algorithms, Experimentation, Performance

Keywords

Progressive confident, sequence, classification

1. INTRODUCTION

Real life objects can be described by their attribute values. Consider a person with attributes gender and job title. While the gender of a person does not change, the job title may change with time. If we denote the set of attribute values of an object as its state, then the state of an object changes as the attribute values change with time. The states of an object at different time stamps form a state sequence. In many applications, an object's state sequence over time is usually more interesting than its current state because the state sequence depicts the object's behavior characteristics. The state sequence of an object also captures more information than the current state for classification. For example, it is hard to determine whether a patient has chronic lymphocytic leukemia if we only knows that he currently has anemia. However, if we track the patient's medical history over a period of time, then the doctor will be able to make a better judgement.

Table 1 shows a sample database that records the symptom sequences of patients and whether they have Chronic Lymphocytic Leukemia (CLL). For example, patient ID 1 first has night sweat and hypodynamia, followed by fever, then achroacytosis, and finally anemia. We regard the set of symptoms at a time point as the state of a patient at that time. Thus, the first state of patient 1 is {night sweat, hypodynamia}, and his last state is anemia. The last column in Table 1 is the doctor's diagnosis. The first four patients have CLL, while the last four do not.

ID	Symptom Sequence	CLL
1	{Night sweat, hypodynamia } →Fever →Achroacytosis→Anemia	Y
2	Night sweat→Fever→Achroacytosis→Anemia	Y
3	Night sweat→Fever→Achroacytosis→Anemia	Y
4	Night sweat→Fever→Achroacytosis→Splenomegalia	Y
5	Night sweat→Fever→Achroacytosis	N
6	Night sweat→Fever→Anemia	N
7	Night sweat→Splenomegalia→Anemia	N
8	Night sweat→Sleepy→Anemia	N

Table 1: Example Diagnosis on Chronic Lymphocytic Leukemia

If we examine the last state of each patient, or their last symptom, 3 people with anemia have CLL (IDs 1–3), while the other 3 with anemia do not (IDs 6–8). Thus it is not clear whether a patient with anemia will have CLL. However, if we study the patients' state sequences, or their sequence of symptoms, we see that most CLL patients (IDs 1–3) have $night\ sweat \rightarrow fever \rightarrow achroacytosis \rightarrow anemia$. This symptom sequence does not occur in non-CLL patients.

Suppose a new patient comes in with symptom $night\ sweat$. The doctor knows the current probability of the patient catching CLL is not high ($conf(night\ sweat, CLL) = 0.5$). If this is followed by fever and subsequently achroacytosis, the doctor could advise on preventive measures because the chance of having CLL has increased ($conf(night\ sweat \rightarrow fever \rightarrow achroacytosis, CLL) = 0.8$). However, if the patient becomes sleepy, then we have $conf(night\ sweat \rightarrow sleepy, CLL) = 0$ and we can conclude that the chance of having CLL is small.

In this paper, we propose a new kind of pattern called *progressive confident rules* that describes the changing states of objects with increasing probabilities that lead to some end state. The general form of a progressive confident rule is $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n$, where X_i is a state, and a state in the left hand side of "→" occurs earlier in time than a state in the right hand side of "→". By progressive confident, we mean the probability of an object achieving state C in the future increases as state $X_i (1 \leq i \leq n)$ appears one after another. We call this the *progressive confident condition.*

For meaningful rules, we have three additional criteria. First, there is a minimum support of the rule to ensure that the rule is general. Second, the probability of the first state X_1 leading to the end state C should be no less than some threshold min_conf1. This removes rules that begin with unrelated states. Third, the probability of the entire rule leading to C should be no less than a threshold min_conf2 to ensure that the rule is interesting.

We describe a depth-first mining algorithm to discover progressive confident rules and devise new pruning strategies for the large search space. The algorithm utilizes the concise set analysis in the mining process to further reduce the search space. Experimental results indicate the efficiency and scalability of the algorithm.

Progressive confident rules can also be utilized to predict an object's future state. This is a classification problem in essence. We incorporate the rules into three representative classifiers, C4.5, Support Vector Machine (SVM) and Bayes Classifier and in so doing, we show that their classification accuracies are greatly improved.

2. PROBLEM DEFINITION

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals called items. A state or an itemset $X \subseteq I$ is a set of items. The semantics of item and itemset is as follows. Every item is related with a characteristic. If a state (itemset) contains an item, it means the item's corresponding characteristic appears in the state. For example, if item a represents the symptom "sick", item b represents the symptom "fever", then state $\{a, b\}$ means patient is sick and has fever.

A pattern $P = X_1 \to X_2 \to \ldots \to X_n$ is an ordered set of states. The length of P is defined as the number of states in P. We use $|P|$ to represent the length of P. For example, if $P = \{a\} \to \{b, c\} \to \{d\}$, then $|P| = 3$.

An observation o is an itemset T with a time stamp (tid), i.e., $o = \langle tid, T\rangle$. If a state $X \subseteq T$, we say X is contained in the observation o. Sometimes we omit the time stamp tid and represent an observation by its itemset T only.

A sequence s is composed of a sequence id (sid) and an ordered list of observations, i.e. $s = \langle sid, o_1, o_2, \ldots, o_n\rangle$. Given a sequence $s = \langle sid, o_1, o_2, \ldots, o_n\rangle$ and a pattern $P = X_1 \to X_2 \to \ldots \to X_m$, we say that s matches P, or equivalently, P is contained in s, if there exist integers $j_1, j_2, \ldots, j_m$, such that $1 \le j_1 < j_2 < \ldots < j_m \le n$ and X_1 is contained in o_{j_1}, X_2 is contained in o_{j_2}, ..., X_m is contained in o_{j_m}. We represent this relationship by $P \sqsubseteq s$.

For example, pattern $P = \{a\} \to \{b, c\} \to \{d\}$ is contained in sequence $s_1 = \langle sid1, \{e\}, \{a, d\}, \{g\}, \{b, c, f\}, \{d\}\rangle$ because $\{a\} \subseteq \{a, d\}$, $\{b, c\} \subseteq \{b, c, f\}$, and $\{d\} \subseteq \{d\}$. Hence, $P \sqsubseteq s_1$. On the other hand, $P \not\sqsubseteq s_2 = \langle sid2, \{a, d\}, \{d\}, \{b, c, f\}\rangle$ because $\{b, c\}$ occurs before $\{d\}$ in P, which is not the case in s_2.

A database $\mathcal{D}$ is composed of a number of sequences whose end states are known. We use D_C to represent the set of sequences in D that end with state C, and $D_{\overline{C}}$ to represent the set of sequences in D that does not end with state C. Thus, $D = D_C \cup D_{\overline{C}}$. The sequences in $D_{\overline{C}}$ do not necessarily have the same end states, that is, there can be more than two end states in the database. We say that sequences with end state C are in class C.

The support of a pattern P in class C (represented by $sup(P, C)$) is defined as the number of sequences in D_C that match P. If $sup(P, C)$ is no less than a user specified support threshold ρ_s, we say P is a frequent pattern in class C.

We use $sup(P, \overline{C})$ to represent the number of sequences in $D_{\overline{C}}$ that match P. The confidence of a pattern P resulting in class C is

$$conf(P, C) = \frac{sup(P, C)}{sup(P, C) + sup(P, \overline{C})}$$

Given a pattern $P = X_1 \to X_2 \to \ldots \to X_n$, if $conf(X_1, C) \le conf(X_1 \to X_2, C) \le \ldots \le conf(P, C)$, we say P satisfies the progressive confident condition.

Finally, if a pattern P satisfies the following four conditions

$$sup(P, C) \ge \rho_s; \tag{1}$$

$$conf(X_1, C) \ge min\_conf1; \tag{2}$$

$$conf(X_1, C) \le conf(X_1 \to X_2, C) \le \ldots \le conf(X_1 \to X_2 \to \ldots \to X_n, C) \tag{3}$$

$$conf(X_1 \to X_2 \to \ldots \to X_n, C) \quad \ge min\_conf2; \tag{4}$$

where min_conf1, min_conf2 and ρ_s are three user specified parameters, we say P together with its confidence values $(conf(X_1), conf(X_1 \to \ldots \to X_i, C)$, where $1 < i \le n)$ is a progressive confident rule.

We observe that the set of all progressive confident rules contains redundant information.

Example 1. Suppose $P = \{a\} \to \{b, c\}$ and $Q = \{a\} \to \{b, c\} \to \{d\}$ are two progressive confident rules. If $conf(P, C) = 1$, by definition of confidence, $conf(Q, C)$ must also be 1, and the information captured in Q is redundant.

Example 2. Suppose $P = \{a\} \to \{b, c\}$ and $Q = \{a\} \to \{b, c\} \to \{d\}$ are two progressive confident rules. If $conf(P, C) < 1$, then all information in P is already contained in Q.

Given a pattern Q, we use $Q[i]$ to represent the i-th state (itemset) in Q, and $Q[i, j]$ ($j > i$) to represent the pattern $Q[i] \to Q[i+1] \to \ldots \to Q[j]$. For example, if $Q = \{a\} \to \{b, c\} \to \{d\}$, then $Q[1] = \{a\}$, $Q[2] = \{b, c\}$, $Q[3] = \{d\}$, and $Q[1, 2] = \{a\} \to \{b, c\}$.

A *base-pattern* of a pattern Q is obtained by removing the last few states (itemsets) from Q. For example, if $Q = \{a\} \to \{b, c\} \to \{d\}$, then $\{a\} \to \{b, c\}$ is Q's base-pattern, and $\{a\} \to \{b\}$ is not. If P is a base-pattern of Q, we also say Q is an *extender* of P.

Given a pattern Q, if $conf(Q, C) = 1$ and none of its base-pattern P satisfies the condition $conf(P, C) = 1$, we say Q is a *terminator pattern*.

Given a set of patterns V and a pattern $P \in V$, if P is not a base-pattern of any other pattern P' in V, we say P is a *maximal pattern* in V.

In Example 1, all the extenders of a terminator pattern can be ignored without loss of information. Further, if a pattern is neither a terminator pattern nor a maximal pattern (such as P in Example 2), then that pattern can be excluded from the concise set.

Based on the above discussion, we can obtain a concise set of rules from a set of progressive confident rules R in two steps: (1) delete all extenders of terminator patterns in R; (2) remove non-maximal patterns.

We define the problem of mining progressive confident rules as finding the concise set of all progressive confident rules in a given database $\mathcal{D}$ and a class C.

3. MINING CONCISE SET OF PCR

One common issue in data mining problems is that the search space is huge. Hence, many mining algorithms make use of the *Apriori* property to reduce the search space. The Apriori property states that if a pattern P satisfies the mining requirement, then all its subpatterns also meet the requirement. For example, the `Apriori` algorithm for finding large itemsets [1] and the `GSP` algorithm for mining sequential patterns [6] are based on this property. Unfortunately, the Apriori property does not hold in the progressive confident condition (i.e. $conf(P[1], C) \le conf(P[1, 2], C) \le \ldots \le conf(P, C)$).

Class C

Sequence id	Observation	
	tid	Itemset
1	1	$\{a, e\}$
	2	$\{b, c\}$
	3	$\{d\}$
2	4	$\{a\}$
	5	$\{b, c\}$
	6	$\{d, f\}$
3	7	$\{a\}$
	8	$\{g\}$
	9	$\{e\}$

Classes not of C

Sequence id	Observation	
	tid	Itemset
4	11	$\{a\}$
	12	$\{b, c, d\}$
	13	$\{e\}$
5	14	$\{a\}$
	15	$\{b, d\}$
	16	$\{f\}$
6	17	$\{a\}$
	18	$\{d\}$
	19	$\{b\}$

Table 2: A sample database

Consider the database in Table 2. We set the user input parameters $\rho_s = 2$, $min\_conf1 = 0.5$ and $min\_conf2 = 0.9$. Pattern $P = \{a\} \rightarrow \{b, c\} \rightarrow \{d\}$ is a progressive confident rule because

- $sup(P, C) = 2 \geq \rho_s$;
- $conf(\{a\}, C) = \frac{3}{3+3} = 0.5 \geq min\_conf1$;
- $conf(\{a\} \rightarrow \{b, c\}, C) = \frac{2}{2+1} = 0.67 > conf(\{a\}, C)$; $conf(\{a\} \rightarrow \{b, c\} \rightarrow \{d\}, C) = \frac{2}{2+0} = 1 \geq conf(\{a\} \rightarrow \{b, c\}, C)$;
- $conf(\{a\} \rightarrow \{b, c\} \rightarrow \{d\}, C) = 1 \geq min\_conf2$.

However, P's subpattern $Q = \{a\} \rightarrow \{b\}$ does not satisfy the progressive confident condition, because $conf(\{a\} \rightarrow \{b\}, C) = \frac{2}{2+3} = 0.4 < conf(\{a\}, C) = 0.5$. Hence, the apriori property does not hold.

Another property that can be used to prune the search space is the prefix anti-monotonic property [5]. This property states that if a pattern P_1 meets the mining requirement and another pattern P_2 is obtained by removing some ending items from P_1, then P_2 also satisfies the mining requirement. In our example, Q is obtained by removing ending items c and d from P. Although P is a progressive confident rule, Q does not satisfy the progressive confident condition. Hence, the prefix anti-monotonic property is also not applicable here.

We put forward two theorems to help reduce the huge search space. The proofs of these theorems can be found in [8].

THEOREM 1. *Suppose a pattern P satisfies the conditions:*

$$\begin{cases} sup(P, C) \geq \rho_s; \\ conf(P[1], C) \geq min\_conf1; \\ conf(P[1], C) \leq conf(P[1,2], C) \leq \ldots \leq conf(P, C). \end{cases}$$

If Q is a base-pattern of P, then Q also satisfies the conditions:

$$\begin{cases} sup(Q, C) \geq \rho_s; \\ conf(Q[1], C) \geq min\_conf1; \\ conf(Q[1], C) \leq conf(Q[1,2], C) \leq \ldots \leq conf(Q, C). \end{cases}$$

Theorem 1 states that if a pattern Q does not satisfy either one of the support requirement, min_conf1 requirement or progressive confident condition, nor is any of Q's extenders. Thus, if we find such a pattern Q in the mining process, we do not need to consider Q's extenders as potential progressive confident rules. This will prune off a huge amount of search space. We denote the property described in Theorem 1 as the *base anti-monotonic property*.

The work in [5] developed a framework for mining sequential patterns with constraints that satisfy the prefix anti-monotonic property. Since our mining problem does not follow the prefix anti-monotonic property, we can use Theorem 1 to modify the framework in [5] by projecting on itemsets, instead of items. But this will be inefficient.

If a pattern P satisfies the progressive confident condition and the support requirement, we need to consider P's extender $P \rightarrow X$. At this point, $conf(P, C)$ is already known. To check whether $conf(P \rightarrow X, C) \geq conf(P, C)$, we need $conf(P \rightarrow X, C)$. From the definition of $conf(P \rightarrow X, C)$, we need to calculate both $sup(P \rightarrow X, C)$ and $sup(P \rightarrow X, \overline{C})$. The following theorem helps us to avoid computing $sup(P \rightarrow X, C)$ when $P \rightarrow X$ does not satisfies the progressive confident condition

THEOREM 2. *Given a pattern P and a state X, if $conf(P, C) < 1$ and $conf(P \rightarrow X, C) \geq conf(P, C)$, we have*

$$sup(P, C) \geq sup(P \rightarrow X, \overline{C}) \frac{conf(P, C)}{1 - conf(P, C)} and$$

$$sup(X, C) \geq sup(P \rightarrow X, \overline{C}) \frac{conf(P, C)}{1 - conf(P, C)}.$$

Theorem 2 provides a test on whether to extend a pattern P to $P \rightarrow X$. If $sup(P, C)$ or $sup(X, C)$ is not large enough (compared to $sup(P \rightarrow X, \overline{C}) \frac{conf(P,C)}{1-conf(P,C)}$), we have $conf(P \rightarrow X, C) < conf(P, C)$. This contradicts the progressive confident condition, and we do not need to extend P to $P \rightarrow X$, which saves the time for counting $sup(P \rightarrow X, C)$.

3.1 Algorithm

Based on Theorem 1 and Theorem 2, we design an efficient depth-first algorithm called FMP (Fast Mining of Progressive confident rules) to mine the concise set of progressive confident rules.

Algorithm FMP (see Figure 1) first finds all frequent itemsets in class C. Then for each frequent itemset that satisfies min_conf1 requirement, FMP processes its extenders in a depth-first order to search for other potential progressive confident rules. The core of FMP lies in the *extend* function (lines 7–28 of Figure 1). Given a pattern P, the function finds out all extenders of P that are in the concise set of progressive confident rules.

The *extend* function checks if $conf(P, C)$ is equal to 1. If so, P is a terminator pattern and we can ignore all P's extenders. P will be output. If $conf(P, C) \neq 1$, we check whether we need to append a state X to the end of P (lines 13–18). If X passes the test, we check if $P \rightarrow X$ meets the support requirement and progressive confident condition (lines 19–24). If yes, we re-call the *extend* function with a new value $P \rightarrow X$. Finally, if P cannot be extended anymore (i.e. P is a maximal pattern) and $conf(P, C) \geq min\_conf2$, we output P and the function terminates. The support of $P \rightarrow X$ is determined from the ID lists of P and X, not by scanning the database. The ID list of a pattern records the sequences and the time stamps where a pattern occurs.

The depth-first order and the careful control of the *extend* function guarantees that the rules obtained belongs to the concise set. The FMP algorithm is efficient for two reasons. First, it utilizes the two theorems to effectively reduce the search space. Second, it incorporates the concise set analysis into the mining process to avoid checking the extenders of terminator patterns.

4. EXPERIMENTAL EVALUATION

We compare Algorithm FMP with a naive method DMP (Direct Mining of Progressive confident rules). Algorithm DMP utilizes SPADE [7] to find all frequent patterns. Then it removes the frequent patterns that do not satisfy the other requirements of progressive confident rules. Finally, DMP computes the concise set of rules. Both FMP and DMP utilize ID lists to count the support of a pattern. We implemented both algorithms in C++ and carried out the experiments on a 750MHz Ultra Sparc III CPU with 1GB RAM, running SunOS 5.8.

```
Algorithm FMP ($D$, $\rho_s$, $min\_conf1$, $min\_conf2$)
  Find frequent itemsets in class $C$:
  $L_X = \{X | sup(X, C) \geq \rho_s\}$
  $\forall X \in L_X$
    calculate $conf(X, C)$
    if $conf(X, C) \geq min\_conf1$
      extend(X);
Function extend (pattern $P$)
  if $conf(P, C) = 1$
    output $P$;
    return;
  ext=false;
  $\forall X \in L_X$
    count $sup(P \to X, \overline{C})$
    stand = $sup(P \to X, \overline{C}) \frac{conf(P,C)}{1-conf(P,C)}$
    if $sup(P, C) < stand$
      continue with next itemset in $L_X$;
    if $sup(X, C) < stand$
      continue with next itemset in $L_X$;
    count $sup(P \to X, C)$
    if $sup(P \to X, C) < \rho_s$
      continue with next itemset in $L_X$;
    count $conf(P \to X, C)$
    if $conf(P \to X, C) < conf(P, C)$
      continue with next itemset in $L_X$;
    ext=true;
    extend($P \to X$);
  if ($ext = false \wedge conf(P) \geq min\_conf2$)
    output P;
```

Figure 1: Algorithm FMP

Parameter	Description	Value
$\|D\|$	Number of sequences in database $\mathcal{D}$	100,000
$\|D_C\|$	Number of sequences in class C	50,000
$\|D_{\overline{C}}\|$	Number of sequences not in class C	50,000
O	Average number of observations per sequence	8
T	Average number of items per observation	2.5
S	Average number of itemsets in maximal potentially frequent patterns	6
I	Average size of itemsets in maximal potentially frequent patterns	1.25
N_S	Number of maximal potentially frequent patterns	500
N_I	Number of maximal potentially frequent itemsets	2,500
N	Number of items	1,000

Table 3: Parameters and default values of data generator.

The test dataset is generated as follows. We first create a table of potentially frequent itemsets. The size of each itemset is computed using a Poisson distributed random number with mean=I. Except for the first itemset, every other itemset shares some common items with its immediate preceding one. The number of items shared is determined by an exponentially distributed random number. We will generate N_I itemsets from N different items. Each generated itemset is assigned a confidence value c which determines the probability an itemset will appear in class C. c is given by:

$$c = \begin{cases} r & \text{if } 0 \leq r \leq 1; \\ 0 & \text{if } r < 0; \\ 1 & \text{if } r > 1, \end{cases}$$

where r is a normal-distributed random number with mean=0.5.

Next, we generate two pattern tables for datasets D_C (corresponds to class C) and $D_{\overline{C}}$ (corresponds to classes other than C) respectively. We call the tables T_C and $T_{\overline{C}}$. The length of a pattern p is determined by a Poisson distributed random number with mean=S. The content of p is a series of itemsets selected randomly from the itemset table. If the confidence of a selected itemset is c, we append it to a pattern belonging to T_C with a probability of c, and append it to a pattern in $T_{\overline{C}}$ with the probability of $1 - c$. When a pattern reaches its length, we insert it into the corresponding pattern table. We will generate N_s patterns for both T_C and $T_{\overline{C}}$. We assign a weight and a corruption level to each generated pattern p. The weight indicates the probability that p is chosen to generate a sequence, and the corruption level controls how many items is dropped from p before it is inserted into a sequence.

Finally, we generate sequences for our test dataset. Sequences in D_C is computed from pattern table T_C, and sequences in $D_{\overline{C}}$ is generated from $T_{\overline{C}}$. A sequence length is determined by a Poisson distributed variable whose mean is O. The average number of items in a sequence's itemsets is determined by a Poisson distributed variable with mean=T. We randomly choose a pattern from the corresponding pattern table to fill it. If the pattern is too large for the sequence, we discard it in half the cases, and push it into the sequence in the other half cases. We will generate $|D_C|$ and $|D_{\overline{C}}|$ sequences from Tables T_C and $T_{\overline{C}}$ respectively.

Table 3 summarizes the parameters of the data generator and their default values.

4.1 Effect of Varying Support Threshold

We first test the performance of the two algorithms under different support thresholds. Figure 2 shows that FMP is much faster than DMP (the runtime is given in log scale). When the support threshold is low ($\rho_s \leq 0.0019$), FMP is one order faster than DMP. This is because a low ρ_s results in many frequent patterns and DMP has to mine all of them; while FMP only needs to deal with patterns that satisfy both the support and confidence requirements. When the support threshold increases, DMP requires less effort to mine the fewer frequent patterns, thus narrowing the gap between the two algorithms. However, FMP is still about 2 times faster than DMP.

4.2 Effect of Varying Confidence Thresholds

Next, we examine the influence of the parameters min_conf1 and min_conf2 on the two algorithms. Figure 3 shows the runtime for DMP and FMP in log scale when min_conf1 changes from 0.4 to 0.65. When min_conf1 increases, the running times of both algorithms decrease. For DMP, a higher min_conf1 means more patterns could be pruned before the concise set computation, which decreases the running time. When min_conf1 increases from 0.4 to 0.65, DMP saves $(566.36 - 481.23)/566.36 = 15\%$ execution time. Algorithm FMP incorporates the min_conf1 test into the mining process, i.e., it does not extend an itemset if its confidence is less than min_conf1. Hence, the effect of min_conf1 on FMP is greater than that on DMP. When min_conf1 changes from 0.4 to 0.65, FMP reduces $(42.63 - 29.40)/42.63 = 31\%$ running time.

Figure 4 shows the runtime for DMP and FMP when min_conf2 changes from 0.75 to 1. Since FMP does not use min_conf2 for pruning, min_conf2 has nearly no effect on its performance. For DMP, the savings obtained from a smaller min_conf2 value is also negligible.

4.3 Effect of Parameters O, T, S and I

We examine the influence of generator parameters O, T, S and I on the performance of the algorithms. We first fix the values of T, S, I, and vary O (average number of observations per sequences) from 6 to 12. Figure 5 shows that as O increases, the running time of both algorithms increases as well. A larger O implies more frequent patterns and more progressive confident rules, and hence additional execution time. The increase in running time

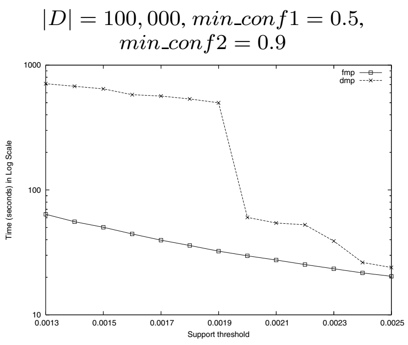

Figure 2: Varying ρ_s

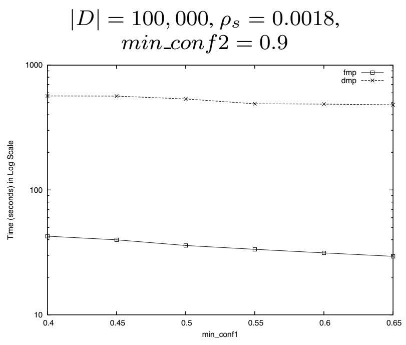

Figure 3: Varying min_conf1

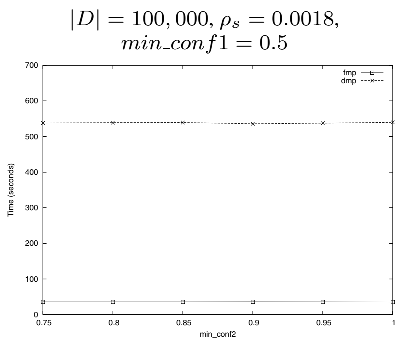

Figure 4: Varying min_conf2

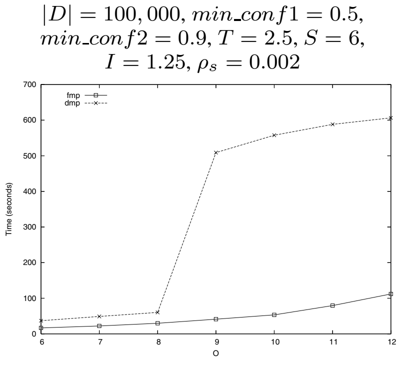

Figure 5: Varying O

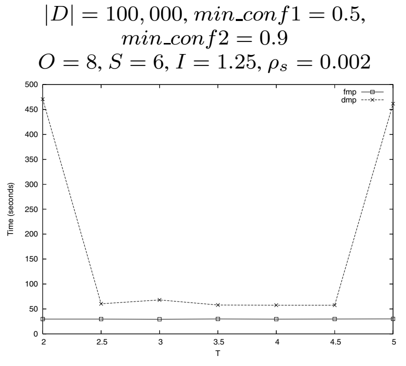

Figure 6: Varying T

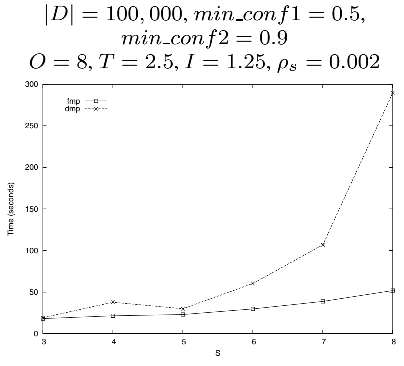

Figure 7: Varying S

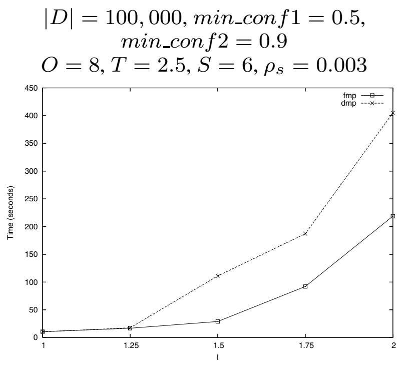

Figure 8: Varying I

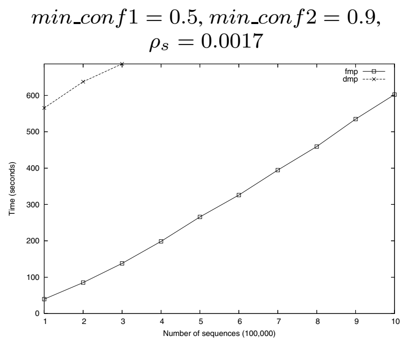

Figure 9: Varying No. of sequences

data	class C	class $\overline{C}$	**total**
training cases	25,000	25,000	50,000
test cases	25,000	25,000	50,000
classifier accuracy	class C	class $\overline{C}$	**avg.**
C4.5	74.08%	79.44%	76.76%
C4.5_PCR	77.70%	99.62%	88.66%
SVM	75.99%	77.49%	76.74%
SVM_PCR	99.63%	77.69%	88.66%
BC	88.06%	62.23%	75.15%
BC_PCR	99.12%	74.60%	86.86%

Table 4: Classification result on synthetic data

data	FMAC-N	FMAC-Y	**total**
training cases	100	100	200
test cases	10634	11	10645
classifier accuracy	FMAC-N	FMAC-Y	**g_mean**
C4.5	57.22%	9.09%	22.81%
C4.5_PCR	76.47%	27.27%	45.67%
SVM	67.84%	63.64%	65.70%
SVM_PCR	64.25%	90.91%	76.42%
BC	89.21%	18.18%	40.27%
BC_PCR	60.06%	90.91%	73.89%

Table 5: Classification result on diabetic retinal data

of FMP is much smaller than that of DMP. The reason is that FMP could efficiently avoid processing many new frequent patterns that do not contribute to new progressive confident rules in the concise set; while DMP has to process all the new frequent patterns.

Next, we fix the values of O, S, I, and vary T (the average number of items in the itemset). Figure 6 shows the performance of DMP and FMP when T changes from 2 to 5. The runtime of DMP varies greatly under different T values. This is because T is used to control the size limit of a sequence, and different T values result in different set of frequent sequences. On the other hand, T does not influence the pattern table, which contains potential progressive confident rules. By mining progressive confident rules directly, the performance of FMP is hardly influenced by T.

We also study the effect of varying S (average length of potentially maximal frequent patterns) from 3 to 8. The result in Figure 7 shows that the execution times of both algorithms increase since a larger S implies more frequent patterns and more progressive confident rules. However, the runtime of FMP increases slowly, while that of DMP increases dramatically.

We examine the performance of DMP and FMP when the average number of items in an itemset (I) changes. With a larger I, the frequencies of items increase in the database, which results in more frequent patterns and more progressive confident rules. Hence, the execution times of both DMP and FMP increase as I becomes larger (see Figure 8). Again, the rate of increase for FMP is lower than that for DMP, because FMP incorporates other condition checks in the mining process to prune the search space.

4.4 Scalability

Finally, we test the scalability of DMP and FMP. Figure 9 shows that FMP scales linearly with the number of sequences. Algorithm DMP scales linearly at first, and runs out of memory when the number of sequences reaches 400,000. Increasing the number of sequences leads to longer ID lists. DMP mines all frequent patterns before filtering and thus, requires a lot of memory to store the long ID lists. In contrast, FMP combines the mining and filtering processes and is able to prune a lot of patterns and their ID lists directly.

5. APPLYING PCR TO CLASSIFICATION

One application of progressive confident rules is to predict a future state of an object based on its past state sequence. In this section, we describe how progressive confident rules can be utilized in three representative classifiers, C4.5, Support Vector Machine (SVM) and Bayes Classifier (BC).

We map each progressive confident rule r in the concise set to a new feature f_r. If the state sequence of an object matches r, the value of f_r is set to 1. Otherwise, it is set to 0. Given n progressive confident rules, the state sequence of an object is transformed to n $\{$feature, value$\}$ pairs. Three classifiers, C4.5, SVM and BC are built from the converted feature set. We use C4.5_PCR (C4.5 with Progressive Confident Rules), SVM_PCR and BC_PCR to indicate classifiers incorporated with PCRs, while C4.5, SVM, and BC to indicate classifiers running on the last states of objects.

The first set of experiments is carried out on our synthetic dataset with two classes C and $\overline{C}$. Each class has 50,000 cases. We select 50% cases from each class for training, and use the remaining cases for testing. The mining algorithm generated 1,154 progressive confident rules from two classes. Table 4 shows the various classification accuracies in two classes and in the whole test data. We see that classifiers incorporated with PCRs, i.e. C4.5_PCR, SVM_PCR and BC_PCR, outperform their counterparts C4.5, SVM, BC. On average, they could achieve about 12% more accuracy.

We also carried out experiments on a real-life dataset which captures the retinal examination data of 10,845 diabetic patients. Each patient has around 2-6 examination records at different times. There are two classes in this dataset: FMAC-Y and FMAC-N. This real-life dataset is very biased. The size of class FMAC-N is about 100 times that of class FMAC-Y. For such biased data, *geometric mean (g_mean)* is typically used to measure the total classification accuracy for all classes. $g\_mean = \sqrt{TP \times TN}$, where TP is recall or true positive rate, and TN is true negative rate. We set the training size of two classes to be 90% that of class FMAC-Y. There are 1,268 progressive confident rules in the training data. Table 5 shows the classification accuracy of the classifiers in two classes and the g_mean values.

6. RELATED WORK

Our work is related to the sequential pattern mining problem [2]. Efficient algorithms for this problem include GSP [6], SPADE [7] and PrefixSpan [4]. GSP utilizes the apriori property to prune search space [6]. SPADE [7] works on a vertical representation of the database where every database item is associated with an id-list and the supports of sequences can be computed from the id-lists directly. PrefixSpan [4] mines frequent sequences by projecting databases on individual items, thus avoiding the time-consuming subset testing and candidate generation. Our work mines frequent sequences with extra constraints.

The work in [3] uses regular expressions as a tool for users to specify the kind of frequent sequences the system should return and propose a family of algorithms to mine frequent sequences with regular expression constraints. Jian et al. [5] examine sequential patterns with constraints that obey the prefix anti-monotonic property and design the prefix-growth algorithm which also utilizes a database projection method. Our work is different from theirs in that our constraints neither follow regular expressions nor obey the prefix anti-monotonic property.

7. CONCLUSION

In this paper, we have described a new kind of pattern called progressive confident rules. The rules capture the state change of objects that leads to a certain end state with increasing confidence. We formalized the concept of progressive confident rules and put forward new pruning strategies to reduce the huge search space. We designed a depth-first mining algorithm FMP and demonstrated its efficiency. We have also shown how progressive confident rules can be used to predict a future state of an object given its past state sequence. Experiments on both synthetic and real datasets show that classifiers incorporated with progressive confident rules could improve the classification accuracy greatly.

8. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD*, 1993.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *IEEE ICDE*, 1995.

[3] M.N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *VLDB*, 1999.

[4] J. Pei, J. Han, and B. Mortazavi-Asl et. al. Prefixspan: Mining sequential patterns by prefix-projected growth. In *IEEE ICDE*, 2001.

[5] J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *ACM CIKM*, 2002.

[6] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, 1996.

[7] M.J. Zaki. Efficient enumeration of frequent sequences. In *ACM CIKM*, 1998.

[8] M. Zhang, W. Hsu, and M.L. Lee. Mining progressive confident rules. Technical Report TRA6/06, Dept. of Computer Science, National University of Singapore, June 2006.