Itemset Based Sequence Classification

Cheng Zhou[1,2], Boris Cule[1], and Bart Goethals[1]

[1] University of Antwerp, Belgium
[2] National University of Defense Technology, China
Cheng.Zhou@student.ua.ac.be

Abstract. Sequence classification is an important task in data mining. We address the problem of sequence classification using rules composed of interesting itemsets found in a dataset of labelled sequences and accompanying class labels. We measure the interestingness of an itemset in a given class of sequences by combining the cohesion and the support of the itemset. We use the discovered itemsets to generate confident classification rules, and present two different ways of building a classifier. The first classifier is based on the CBA (Classification based on associations) method, but we use a new ranking strategy for the generated rules, achieving better results. The second classifier ranks the rules by first measuring their value specific to the new data object. Experimental results show that our classifiers outperform existing comparable classifiers in terms of accuracy and stability, while maintaining a computational advantage over sequential pattern based classification.

1 Introduction

Many real world datasets, such as collections of texts, videos, speech signals, biological structures and web usage logs, are composed of sequential events or elements. Because of a wide range of applications, sequence classification has been an important problem in statistical machine learning and data mining.

The sequence classification task can be defined as assigning class labels to new sequences based on the knowledge gained in the training stage. There exist a number of studies integrating pattern mining techniques and classification, such as classification based on association rules [10], sequential pattern based sequence classifiers [8,13], and many others. These combined methods can output good results as well as provide users with information useful for understanding the characteristics of the dataset.

In this paper, we propose to utilise a novel itemset mining technique [4] in order to obtain an accurate sequence classifier. An itemset in a sequence should be evaluated based on how close to each other its items occur (cohesion) and how often the itemset itself occurs (support). We therefore propose a new method called sequence classification based on interesting itemsets (SCII), that greatly improves on the accuracy obtained by other classifiers based on itemsets, as they typically do not take cohesion into account. Moreover, we also achieve a reduction in complexity compared to classifiers based on sequential patterns, as

we generate and evaluate fewer patterns, and, yet, using cohesion, we still take the location of items within the sequences into account.

The main contribution of this paper consists of two SCII classifiers, both based on frequent cohesive itemsets. By using cohesion, we incorporate the sequential nature of the data into the method, while, by using itemsets, we avoid the complexity of mining sequential patterns. The two classifiers differ in how the rules are selected and ranked within the classifier, and we experimentally demonstrate that both give satisfactory results.

The rest of the paper is organised as follows. Section 2 gives a review of the related work. In Sections 3 and 4, we formally describe the sequence classification problem setting and present our two approaches for generating rules and building classifiers, respectively. We end the paper with an experimental evaluation in Section 5 and a summary of our conclusions in Section 6.

2 Related Work

The existing sequence classification techniques deploy a number of different approaches, ranging from decision trees, Naïve Bayes, Neural Networks, K-Nearest Neighbors (KNN), Hidden Markov Model (HMM) and, lately, Support Vector Machines (SVMs) [7].

In this section, we give an overview of pattern-based classification methods. Most such work can be divided into the domains of classification based on association rules and classification based on sequential patterns. The main idea behind the first approach is to discover association rules that always have a class label as their consequent. The next step is to use these patterns to build a classifier, and new data records are then classified in the appropriate classes. The idea of classification based on association rules (CBA) was first proposed by Liu et al. [10]. In another work, Li et al. [9] proposed CMAR, where they tackled the problem of overfitting inherent in CBA. In CMAR, multiple rules are employed instead of just a single rule. Additionally, the ranking of the rule set in CMAR is based on the weighted Chi-square of each rule replacing the confidence and support of each rule in CBA. Yin and Han [15] proposed CPAR which is much more time-efficient in both rule generation and prediction but its accuracy is as high as that of CBA and CMAR.

The concept of sequential pattern mining was first described by Agrawal and Srikant [2], and further sequential pattern mining methods, such as Generalized Sequential Patterns (GSP) [12], SPADE [16], PrefixSpan [11], and SPAM [3], have been developed since. A number of sequence classifiers have been based on these methods.

Lesh et al. [8] combined sequential pattern mining and a traditional Naïve Bayes classification method to classify sequence datasets. They introduced the FeatureMine algorithm which leveraged existing sequence mining techniques to efficiently select features from a sequence dataset. The experimental results showed that BayesFM (combination of Naïve Bayes and FeatureMine) is better than Naïve Bayes only. Although pruning is used in their algorithm, there

was still a great number of sequential patterns used as classification features. As a result, the algorithm could not effectively select discriminative features from a large feature space.

Tseng and Lee [14] proposed the Classify-By-Sequence (CBS) algorithm for classifying large sequence datasets. The main methodology of the CBS method is mining classifiable sequential patterns (CSPs) from the sequences and then assigning a score to the new data object for each class by using a scoring function, which is based on the length of the matched CSPs. They presented two approaches, CBS_ALL and CBS_CLASS. In CBS_ALL, a conventional sequential pattern mining algorithm is used on the whole dataset. In CBS_CLASS, the database is divided into a number of sub-databases according to the class label of each instance. Sequential pattern mining was then implemented on each sub-database. Experimental results showed that CBS_CLASS outperforms CBS_ALL. Later, they improved the CBS_CLASS algorithm by removing the CSPs found in all classes [13]. Furthermore, they proposed a number of alternative scoring functions and tested their performances. The results showed that the length of a CSP is the best attribute for classification scoring.

Exarchos et al. [6] proposed a two-stage methodology for sequence classification based on sequential pattern mining and optimization. In the first stage, sequential pattern mining is used, and a sequence classification model is built based on the extracted sequential patterns. Then, weights are applied to both sequential patterns and classes. In the second stage, the weights are tuned with an optimization technique to achieve optimal classification accuracy. However, the optimization is very time consuming, and the accuracy of the algorithm is similar to FeatureMine.

Additionally, several sequence classification methods have been proposed for application in specific domains. Exarchos et al. [5] utilised sequential pattern mining for protein fold recognition, while Zhao et al. [17] used a sequence classification method for debt detection in the domain of social security.

The main bottleneck problem for sequential pattern based sequence classification being used in the real world is efficiency. Mining frequent sequential patterns in a dense dataset with a large average sequence length is time and memory consuming. None of the above sequence classification algorithms solve this problem well.

3 Problem Statement

In this paper, we consider multiple event sequences where an event e is a pair (i, t) consisting of an item $i \in I$ and a time stamp $t \in \mathbb{N}$, where I is the set of all possible items and $\mathbb{N}$ is the set of natural numbers. We assume that two events can never occur at the same time. For easier readibility, in our examples, we assume that the time stamps in a sequence are consecutive natural numbers. We therefore denote a *sequence* of events by $s = e_1, \cdots, e_l$, where l is the length of the sequence, and $1, \cdots, l$ are the time stamps.

Let L be a finite set of class labels. A sequence database *SDB* is a set of *data objects* (s, L_k), such that s is a sequence and $L_k \in L$ is a *class label* ($k = 1, 2, \cdots, m$, where m is the number of classes). The set of all sequences in *SDB* is denoted by S. We denote the set of sequences carrying class label L_k by S_k.

The patterns considered in this paper are itemsets, or sets of items coming from the set I. The support of an itemset is typically defined as the number of different sequences in which the itemset occurs, regardless of how many times the itemset occurs in any single sequence. To determine the interestingness of an itemset, however, it is not enough to know how many times the itemset occurs. We should also take into account how close the items making up the itemset occur to each other. To do this, we will define interesting itemsets in terms of both support and cohesion. Our goal is to first mine interesting itemsets in each class of sequences, and then use them to build a sequence classifier, i.e., a function from sequences S to class labels L.

We base our work on an earlier work on discovering interesting itemsets in a sequence database [4], and we begin by adapting some of the necessary definitions from that paper to our setting. The interestingness of an itemset depends on two factors: its support and its cohesion. Support measures in how many sequences the itemset appears, while cohesion measures how close the items making up the itemset are to each other on average.

For a given itemset X, we denote the set of sequences that contain all items of X as $N(X) = \{s \in S | \forall i \in X, \exists (i,t) \in s\}$. We denote the set of sequences that contain all items of X labelled by class label L_k as $N_k(X) = \{s \in S_k | \forall i \in X, \exists (i,t) \in s\}$. The *support* of X in a given class of sequences S_k can now be defined as $F_k(X) = \frac{|N_k(X)|}{|S_k|}$.

We begin by defining the length of the shortest interval containing an itemset X in a sequence $s \in N(X)$ as $W(X,s) = \min\{t_2 - t_1 + 1 | t_1 \leq t_2$ and $\forall i \in X, \exists (i,t) \in s,$ where $t_1 \leq t \leq t_2\}$. In order to calculate the cohesion of an itemset within class k, we now compute the average length of such shortest intervals in $N_k(X)$: $\overline{W_k}(X) = \frac{\sum_{s \in N_k(X)} W(X,s)}{|N_k(X)|}$. It is clear that $\overline{W_k}(X)$ is greater than or equal to the number of items in X, denoted as $|X|$. Furthermore, for a fully cohesive itemset, $\overline{W_k}(X) = |X|$. Therefore, we define cohesion of X in $N_k(X)$ as $C_k(X) = \frac{|X|}{\overline{W_k}(X)}$. Note that all itemsets containing just one item are fully cohesive, that is $C_k(X) = 1$ if $|X| = 1$. The *cohesion* of X in a single sequence s is defined as $C(X,s) = \frac{|X|}{W(X,s)}$.

In a given class of sequences S_k, we can now define the *interestingness* of an itemset X as $I_k(X) = F_k(X)C_k(X)$. Given an interestingness threshold *min_int*, an itemset X is considered interesting if $I_k(X) \geq min\_int$. If desired, minimum support and minimum cohesion can also be used as separate thresholds.

Once we have discovered all interesting itemsets in each class of sequences, the next step is to identify the classification rules we will use to build a classifier.

We define $r_{km} : \ p_m \Rightarrow L_k$ as a rule where p_m is an interesting itemset in S_k and L_k is a class label. p_m is the *antecedent* of the rule and L_k is the *consequent* of the rule. We further define the interestingness, support, cohesion

and size of r_{km} to be equal to the interestingness, support, cohesion and size of p_m, respectively. The *confidence* of a rule can now be defined as:

$$conf(p_m \Rightarrow L_k) = \frac{|N_k(p_m)|}{|N(p_m)|} \quad (1)$$

A rule $p_m \Rightarrow L_k$ is considered confident if its confidence exceeds a given threshold *min_conf*.

If all items in the antecedent of the rule can be found in the sequence of a given data object, we say that the rule *matches* the data object. We say that a rule *correctly classifies* or *covers* a data object in *SDB* if the rule matches the sequence part of the data object and the rule's consequent equals the class label part of the data object.

In practice, most datasets used in the sequence classification task can be divided into two main cases. In the first case, the class of a sequence is determined by certain items that co-occur within it, though not always in the same order. In this case, a classifier based on sequential patterns will not work well, as the correct rule will not be discovered, and, with a low enough threshold, the rules that are discovered will be far too specific. For an itemset of size n, there are $n!$ orders in which this itemset could appear in a sequence, and therefore $n!$ rules that could be discovered (none of them very frequent). Our method, however, will find the correct rule. In the other case, the class of a sequence is determined by items that occur in the sequence always in exactly the same order. At first glance, a classifier based on sequential patterns should outperform our method in this situation. However, we, too, will discover the same itemset (and rule), only not in a sequential form. Due to a simpler candidate generation process, we will even do so quicker. Moreover, we will do better in the presence of noise, in cases when the itemset sometimes occurs in an order different from the norm. This robustness of our method means that we can handle cases where small deviations in the sequential patterns that determine the class of the sequences occur. For example, if a class is determined by occurrences of sequential pattern *abc*, but this pattern sometimes occurs in a different form, such as *acb* or *bac*, our method will not suffer, as we only discover itemset $\{a, b, c\}$. This means that, on top of the reduced complexity, our method often gives a higher accuracy than classifiers based on sequential patterns, as real-life data is often noisy and sequential classification rules sometimes prove to be too specific. On the other hand, in cases where two classes are determined by exactly the same items, but in different order, our classifier will struggle. For example, if class A is determined by the occurrence of *abc* and class B by the occurrence of *cba*, we will not be able to tell the difference. However, such cases are rarely encountered in practice.

4 Generating Rules and Building Classifiers

Our algorithm, SCII (Sequence Classification Based on Interesting Itemsets), consists of two stages, a rule generator (SCII_RG), which is based on the Apriori algorithm [1], and two different classifier builders, SCII_CBA and SCII_MATCH. This section discusses SCII_RG, SCII_CBA and SCII_MATCH.

4.1 Generating the Complete Set of Interesting Itemsets

The SCII_RG algorithm generates all interesting itemsets in two steps. Due to the fact that the cohesion and interestingness measures introduced in Section 3, are not anti-monotonic, we prune the search space based on frequency alone. In the first step, we use an Apriori-like algorithm to find the frequent itemsets. In the second step, we determine which of the frequent itemsets are actually interesting. An optional parameter, *max_size*, can be used to limit the output only to interesting itemsets with a size smaller than or equal to *max_size*.

Let *n-itemset* denote an itemset of size n. Let A_n denote the set of frequent *n-itemset*s. Let C_n be the set of candidate *n-itemset*s and T_n be the set of interesting *n-itemset*s. The algorithm for generating the complete set of interesting itemsets in a given class of sequences is shown in Algorithm 1.

Algorithm 1: GeneratingItemsets. An algorithm for generating all interesting itemsets in S_k.

input : S_k, minimum support threshold *min_sup*, minimum interestingness threshold *min_int*, max size constraint *max_size*
output : all interesting itemsets P_k

1 $C_1 = \{i | i \in I_k\}$, I_k is the set of all the items which occur in S_k;
2 $A_1 = \{f | f \in C_1, F_k(f) \geq min\_sup\}$;
3 $T_1 = \{f | f \in A_1, F_k(f) \geq min\_int\}$;
4 $n = 2$;
5 **while** $A_{n-1} \neq \emptyset$ and $n \leq max\_size$ **do**
6 $\quad T_n = \emptyset$;
7 $\quad C_n =$ candidateGen(A_{n-1});
8 $\quad A_n = \{f | f \in C_n, F_k(f) \geq min\_sup\}$;
9 $\quad T_n = \{f | f \in A_n, I_k(f) \geq min\_int\}$;
10 $\quad n++$;
11 $P_k = \bigcup_{i=1}^{n-1} T_i$;
12 **return** P_k;

Lines 1-2 count the supports of all the items to determine the frequent items. Lines 3 stores the interesting items in T_1 (note that the interestingness of a single item is equal to its support). Lines 4-12 discover all interesting itemsets of different sizes n ($n \geq max\_size \geq 2$). First, the already discovered frequent itemsets of size $n-1$ (A_{n-1}) are used to generate the candidate itemsets C_n using the candidateGen function (line 7). The candidateGen function is similar to the function Apriori-gen in the Apriori algorithm [1]. In line 8, we store the frequent itemsets from C_n into A_n. Line 9 stores the interesting itemsets (as defined in Section 3) from A_n into T_n. The final set of all interesting itemsets in S_k is stored in P_k and produced as output.

The time cost of generating candidates is equal to that of Apriori. We will now analyse the time needed to evaluate each candidate. We denote the time

needed for computing the interestingness of a frequent itemset f with $T_{I_k(f)}$. To get $I_k(f)$, we first need to find a minimal interval $W(f, s)$ of an itemset f in a sequence $s \in S_k$, whereby the crucial step is the computation of the candidate intervals $W(f, t_i)$ for the time stamps t_i at which an item of f occurs. In our implementation, we keep the set of candidate intervals associated with f in a list. To find the candidate interval around position t_i containing all items of f, we start by looking for the nearest occurrences of items of f both left and right of position t_i. We then start reading from the side on which the furthest element is closest to t_i and continue by removing one item at a time and adding the same item from the other side. This process can stop when the interval on the other side has grown sufficiently to make it impossible to improve on the minimal interval we have found so far. When we have found this minimal interval, we compare it to the smallest interval found so far in s, and we update this value if the new interval is smaller. This process can stop if we get a minimal interval which equals to $|f|$, and then $W(f, s) = |f|$. Otherwise, $W(f, s)$ equals to the smallest value in the list of candidate intervals.

Theoretically, in the worst case, the number of candidate intervals that need to be found can be equal to the length of sequence s, $|s|$. To find a candidate interval, we might need to read the whole sequence both to the left and to the right of the item. Therefore, the time to get a $W(f, t_i)$ is $O(|s|)$. So, $T_{I_k(f)}$ is $O(|s|^2)$. However, this worst case only materialises if we are computing $I_k(f)$ when f is composed of all items that appear in s, and even then only if item appearing at each end of s do not appear anywhere else.

4.2 Pruning the Rules

Once we have found all interesting itemsets in a given class, all confident rules can be found in a trivial step. However, the number of interesting itemsets is typically very large, which leads to a large amount of rules. Reducing the number of rules is crucial to eliminate noise which could affect the accuracy of the classifier, and to improve the runtime of the algorithm.

We therefore try to find a subset of rules of high quality to build an efficient and effective classifier. To do so, we use the idea introduced in CMAR [9], and prune unnecessary rules by the database coverage method.

Before using the database coverage method, we must first define a total order on the generated rules R including all the rules from every class. This is used in selecting the rules for our classifier.

Definition 1. *Given two rules in R, r_i and r_j, $r_i \succ r_j$ (also called r_i* precedes *r_j or r_i has a higher precedence than r_j) if:*

1. the confidence of r_i is greater than that of r_j, or

2. their confidences are the same, but the interestingness of r_i is greater than that of r_j, or

3. both the confidences and interestingnesses of r_i and r_j are the same, but the size of r_i is greater than that of r_j

4. all of the three parameters are the same, r_i is generated earlier than r_j.

We apply the database coverage method to get the most significant subset of rules. The main idea of the method is that if a rule matches a data object that has already been matched by a high enough number of higher ranked rules (this number is defined by a user chosen parameter δ, or the coverage threshold), this rule would contribute nothing to the classifier (with respect to this data object). The algorithm for getting this subset is described in Algorithm 2. The algorithm has 2 main steps. First, we sort the set of confident rules R according to definition 1 (line 1). This makes it faster to get good rules for classifying. Then, in lines 2-13, we prune the rules using the database coverage method. For each rule r in sorted R, we go through the dataset D to find all the data objects correctly classified by r and increase the cover counts of those data objects (lines 3-7). We mark r if it correctly classifies a data object (line 8). If the cover count of a data object passes the coverage threshold, its id will be stored into $temp$ (line 9). Finally, if r is marked, we store it into PR and remove those data objects whose ids are in $temp$ (lines 10-13). Line 14 returns the new set of rules PR. In the worst case, to check whether a data object is correctly classified by r, we might need to read the whole sequence part s of the data object, resulting in a time complexity of $O(|s|)$.

Algorithm 2: PruningRules. An algorithm for finding the most significant subset among the generated rules.

```
   input  : training dataset D, a set of confident rules R, coverage threshold δ
   output : a new set of rules PR
 1 sort R according to Definition 1;
 2 foreach data object d in D do d.cover_count = 0;
 3 foreach rule r in sorted R do
 4     temp = ∅;
 5     foreach data object d in D do
 6         if rule r correctly classifies data object d then
 7             d.cover_count++;
 8             mark r;
 9             if d.cover_count >= δ then store d.id in temp;
10     if r is marked then
11         select r and store it into PR;
12         foreach data object d in D do
13             if d.id ∈ temp then delete d from D;
14 return PR;
```

4.3 Building the Classifiers

Based on the generated rules, we now propose two different ways to build a classifier, SCII_CBA and SCII_MATCH.

SCII_CBA. We build a classifier using the rules we discovered after pruning based on the CBA-CB algorithm (the classifier builder part of CBA [10]). In other words, we use rules generated in section 4.2 instead of using all the rules before pruning as the input for CBA-CB. We can also skip the step of sorting rules in CBA-CB because we have already sorted them in the pruning phase. However, the total order for generated rules defined in CBA-CB is different from that given in Definition 1. We use interestingness instead of support and, if the confidence and the interestingness of two rules are equal, we consider the larger rule to be more valuable than the smaller rule.

After building the classifier using the CBA-CB method, the classifier is of the following format: $< r_1, r_2, \ldots, r_n, default\_class >$, where $r_i \in R$, $r_a \succ r_b$ if $a < b$, and $default\_class$ is the default class produced by CBA-CB. When classifying a new data object, the first rule that matches the data object will classify it. This means that the new data object will be classified into a class which the consequent of this rule stands for. If there is no rule that matches the data object, it is classified into the default class.

SCII_MATCH. Rather than ranking the rules using their confidence, interestingness and size, we now propose incorporating the cohesion of the antecedent of a rule in the new data object into the measure of the appropriateness of the rule for classifying the object. Obviously, we cannot entirely ignore the confidence of the rules. Therefore, we will first find all rules that match the new object, and then compute the product of the rule's confidence and the antecedent's cohesion in the new data object. We then use this new measure to rank the rules, and classify the object using the highest ranked rule.

Considering there may not exist a rule matching the given data object, we must also add a default rule, of the form $null \Rightarrow L_d$, to the classifier. If there is no rule that matches the given data object, the default rule will be used to classify the data object. To find the default rule, we first delete the data objects matched by the rules in PR. Then we count how many times each class label appears in the remainder of the dataset. Finally we set the label that appears the most times as the default class label L_d. If multiple class labels appear the most times, we choose the first one as the default class label. So the default rule $default\_r$ is $null \Rightarrow L_d$. In the worst case, to check whether a data object is matched by rule r in PR, we might need to read the whole sequence part s of the data object. Since we need to do this for all data objects and all rules, the time complexity of finding the default rule is $O(|PR| \sum_{s \in D} |s|)$.

The classifier is thus composed of PR and the default rule $default\_r$. We now show how we select a rule for classifying the sequence in a new data object. The algorithm for finding the rule used to classify a new data object is shown in Algorithm 3.

First, we find all the rules that match the given data object d and store them into MR (lines 1). Then, we handle three different cases:

1. (lines 2-7): If the size of MR is greater than 1, we go through MR to compute the cohesion of each rule in MR with respect to the given data object.

Algorithm 3: ClassifyingRule. An algorithm for finding the rule used to classify a new sequence.

input : PR and $default\_r$, a new unclassified data object $d = (s, L_?)$
output : the classifying rule r_c

```
1  MR = {r ∈ PR | r matches d};
2  if MR.size > 1 then
3      foreach rule r : p ⇒ L_k in MR do
4          if r.length > 1 then r.measure = r.confidence * C(p, d.s);
5          else r.measure = r.confidence;
6      sort rules in MR in descending order by r.measure;
7      return the first rule in sorted MR;
8  else
9      if MR.size == 1 then return the only rule in MR;
10     else return default_r;
```

Let us go back to the cohesion defined in section 3. We use the antecedent of a rule to take the place of itemset X to compute the cohesion of a rule. We then compute the value of every rule in MR (the product of the rule's confidence and the antecedent's cohesion), and sort the rules according to their value (the higher the value, the higher the precedence). We then utilize the first rule in the sorted MR to classify the given data object.

2. (line 9): If the size of MR is 1, then we classify the sequence using the only rule in MR.

3. (line 10): If there is no rule in MR, then we use the default rule to classify the given data object.

The only time-consuming part of Algorithm 3 is the computation of $C(p, d.s)$. The time complexity of this computation has already been analysed at the end of Section 4.1.

4.4 Example

To illustrate how our methods work, we will use a toy example. Consider the training dataset consisting of the data objects (sequences and class labels) given in Table 1. We can see that itemset *abcd* exists in all sequences regardless of class. It is therefore hard to distinguish the sequences from different classes using the traditional frequent itemset methods. We now explain how our approach works.

Using the definitions given in Section 3 and Algorithm 1, assume *min_sup* = *min_int* = $\frac{2}{3}$, *max_size* = 4 and make the sequences of class 1 as input S_1 in Algorithm 1. First, we discover frequent itemsets in S_1, which turn out to be itemset *abcd* and all its subsets. Then we generate interesting itemsets from frequent itemsets, and find itemsets *ab*, *a*, *b*, *c* and *d*, whose interestingness is equal to 1. Meanwhile, in the sequences of class 2, S_2, itemsets *bcd*, *cd* *a*, *b*, *c* and *d* are interesting. If we now set *min_conf* = 0.5, we get the confident rules sorted using Definition 1, as shown in Table 2.

Table 1. An example of a sequence dataset

ID	Sequence	Class Label	ID	Sequence	Class Label
1	$c\ c\ x\ y\ a\ b\ d$	class1	5	$a\ d\ z\ z\ c\ d\ b$	class2
2	$a\ b\ e\ e\ x\ x\ e\ c\ f\ d$	class1	6	$b\ x\ y\ d\ d\ c\ d\ d\ d\ x\ a$	class2
3	$c\ g\ h\ a\ b\ d\ d$	class1	7	$b\ d\ c\ c\ c\ c\ a\ y$	class2
4	$d\ d\ e\ c\ f\ b\ a$	class1	8	$a\ x\ x\ c\ d\ b$	class2

Table 2. Sorted rules from the example

Rule	Cohesion	Confidence	Rule	Cohesion	Confidence
$a\ b \Rightarrow Class1$	1.0	0.5	$c \Rightarrow Class2$	1.0	0.5
$c\ d \Rightarrow Class2$	1.0	0.5	$a \Rightarrow Class2$	1.0	0.5
$c \Rightarrow Class1$	1.0	0.5	$b \Rightarrow Class2$	1.0	0.5
$a \Rightarrow Class1$	1.0	0.5	$d \Rightarrow Class2$	1.0	0.5
$b \Rightarrow Class1$	1.0	0.5	$c\ b\ d \Rightarrow Class2$	0.8	0.5
$d \Rightarrow Class1$	1.0	0.5	$b\ d \Rightarrow Class2$	0.8	0.5

Given a new input sequence $s_9 = a\ x\ b\ y\ c\ d\ z$, we can see that it is not easy to choose the correct classification rule, as all rules match the input sequence, and only the last two score lower than the rest. The first two rules are ranked higher due to the size of the antecedent, but the CBA, CMAR and SCII_CBA methods would have no means to distinguish between the two rules, and would classify s_9 into class 1, simply because rule $a\ b \Rightarrow Class1$ was generated before rule $c\ d \Rightarrow Class2$. Using the SCII_MATCH method, however, we would re-rank the rules taking the cohesion of the antecedent in s_9 into account. In the end, rule $c\ d \Rightarrow Class2$ is chosen, as $C(cd, s_9) = 1$, while $C(ab, s_9) = \frac{2}{3}$. The cohesion of all antecedents of size 1 in s_9 would also be equal to 1, but rule $c\ d \Rightarrow Class2$ would rank higher due to its size. We see that the SCII_MATCH method classifies the new sequence correctly, while other methods fail to do so.

5 Experiments

We compared our classifiers SCII_CBA and SCII_MATCH with five classifiers: CBA, CMAR, BayesFM [8] and CBS [13]. The CBS paper proposes a number of different scoring functions [13], and we chose the length policy as it gave the best results. For better comparison, we also added a *max_size* constraint into the pattern mining stage of CBS. Our methods, BayesFM and CBS are implemented in Java of Eclipse IDE, while CBA and CMAR are implemented in LUCS-KDD Software Library[3]. We use SPADE [16] to mine subsequential patterns for BayesFM and CBS and we transform the sequence dataset into a transaction dataset for CBA and CMAR. All experiments are performed on a

[3] `http://cgi.csc.liv.ac.uk/~frans/KDD/Software/`

laptop computer with Intel i7 (2 CPUs 2.7GHz), 4GB memory and Windows 7 Professional.

In order to evaluate the proposed methods, we used four real-life datasets. Three of these datasets were formed by making a selection from the Reuters-21578 dataset[4], consisting of news stories, assembled and indexed with categories by Reuters Ltd personnel. We consider the words appearing in the texts as items and treat each paragraph as a sequence. We formed the *Reuters1* dataset using the two biggest classes in the Reuters-21578 dataset, "acq" (1596 paragraphs) and "earn" (2840 paragraphs). *Reuters2* consists of the four biggest classes in Reuters-21578, "acq", "earn", "crude" (253 paragraphs) and "trade" (251 paragraphs), and is therefore an imbalanced dataset. *Reuters3* is a balanced dataset obtained from *Reuters2* by keeping only the first 253 paragraphs in the top two classes. *Reuters1* consists of 4436 sequences composed of 11947 different items, *Reuters2* of 4940 sequences containing 13532 distinct items, and *Reuters3* of 1010 sequences composed of 6380 different items.

Our fourth dataset is a protein dataset obtained from PhosphoELM[5]. The data consists of different combinations of amino acids for each kind of protein. We chose two of the biggest protein groups (PKA with 362 combinations and SRC with 304 combinations) to form the *Protein* dataset. We treat each combination of amino acids as a sequence and consider each amino acid as an item. Each sequence is labelled by the protein group it belongs to. This dataset consists of 666 sequences containing 20 different items. All the reported accuracies in all of experiments were obtained using 10-fold cross-validation.

5.1 Analysis of the Predictive Accuracy

Table 3 reports the accuracy results of all six classifiers. In the experiments, we set *min_conf* to 0.6 and *min_sup* to 0.1 for all of the classifiers, while varying the *max_size* threshold. Additionally, we set *min_int* to 0.05 for the SCII classifiers. For the SCII methods and CMAR, the database coverage threshold was set to 3. The best result for each dataset is highlighted in bold. As shown in Table 3, the SCII algorithms generally outperform other classifiers.

To further explore the performance of the six classifiers, we conducted an analysis of the predictive accuracy under different support, confidence, and interestingness thresholds, respectively. We first experimented on *Reuters1* and *Protein*, with *min_conf* fixed at 0.6, *max_size* set to 3 and *min_int* for SCII classifiers set to 0.05. We can see in Fig. 1 that the SCII_C classifier is not sensitive to the minimum support thresholds as minimum interestingness threshold is the main parameter deciding the output rules. As the number of output rules drops, SCII_M begins to suffer, as it picks just the highest ranked rules to classify a new object. SCII_C compensates by using a combination of rules, and the accuracy therefore does not suddenly drop once some rules drop out of the classifier.

[4] `http://web.ist.utl.pt/~acardoso/datasets/r8-train-stemmed.txt`
[5] `http://phospho.elm.eu.org/`

Table 3. Comparison of Predictive Accuracy (%)

Dataset	*max_size*	SCII_C	SCII_M	CBA	CMAR	BayesFM	CBS
Reuters1	2	**92.74**	92.54	67.05	65.76	92.38	89.27
	3	92.74	92.54	66.63	65.65	**92.88**	88.12
	4	92.74	92.54	66.63	65.54	**92.79**	88.25
	5	92.74	92.54	66.63	65.54	**92.76**	88.82
	∞	**92.74**	92.54	66.63	65.54	92.72	88.88
Protein	2	86.63	**87.56**	81.34	81.91	52.66	54.92
	3	87.69	**88.59**	86.55	80.88	72.97	74.94
	4	**91.01**	90.71	87.93	78.64	85.14	86.81
	5	90.71	**91.73**	89.14	78.94	85.14	86.96
	∞	90.56	**91.86**	89.14	78.94	85.14	86.96
Reuters2	2	**90.40**	90.16	57.69	57.45	83.68	78.87
	3	**90.51**	90.22	57.65	57.17	83.22	75.99
	4	**90.67**	90.28	57.65	57.09	82.94	74.31
	5	**90.75**	90.26	57.65	57.09	82.87	72.96
	∞	**90.61**	90.45	57.65	57.09	82.82	72.11
Reuters3	2	92.48	**92.97**	78.61	62.28	78.71	87.82
	3	92.28	**92.87**	78.71	62.38	74.16	88.22
	4	**92.87**	92.67	78.71	62.38	72.57	87.92
	5	92.77	**92.97**	78.71	62.38	72.18	87.52
	∞	**93.07**	92.77	78.71	62.38	71.98	86.73

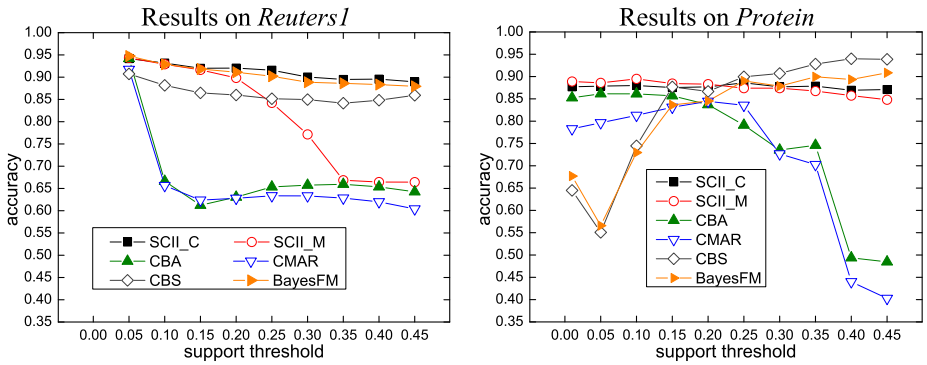

Fig. 1. The impact of varying the support threshold on various classifiers

We then compared the predictive accuracy of the classifiers using different minimum confidence thresholds on the *Protein* dataset. We compare just four classifiers of the classifiers, as BayesFM and CBS do not use a confidence threshold. Here, *min_sup* is fixed at 0.1, *max_size* is set to 3 and *min_int* for the SCII classifiers is set to 0.05. From Fig. 2, we can see that the SCII classifiers are not sensitive to the minimum confidence threshold at all. When the confidence threshold is not lower than 0.8, the accuracies of CBA and CMAR decline

sharply. It shows the performance of CBA and CMAR is strongly related to the number of produced rules.

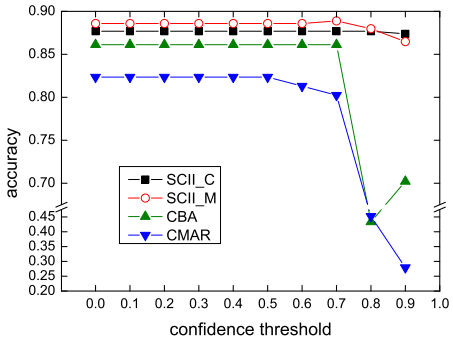

Fig. 2. The impact of varying the confidence threshold on various classifiers

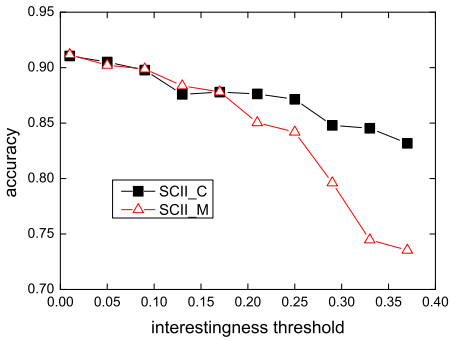

Fig. 3. The impact of varying the interestingness threshold on the SCII classifiers

Fig. 3 shows the accuracy of the SCII classifiers on the *Reuters2* dataset with different minimum interestingness thresholds. Here, *min_sup* is fixed at 0.1, *min_conf* at 0.6, and *max_size* is set to 3. We can see that the accuracies of both SCII_C and SCII_M decrease when the minimum interestingness threshold increases. When the minimum interestingness threshold is greater than 0.2, fewer rules are discovered, and the accuracy of SCII_M, once again, declines faster than that of SCII_C. We can conclude that the selection of good classification rules is already done using the support and confidence threshold, and there is no need to prune further with the interestingness threshold. The interestingness of an itemset, however, remains a valuable measure when ranking the selected classification rules.

5.2 Analysis of the Scalability for Different Methods

Fig. 4 shows the performance of the six classifiers on *Reuters1* and *Protein* for a varying number of sequences (*#sequences*). We start off by using just 10% of the dataset, adding another 10% in each subsequent experiment. In *Reuters1* the number of items (*#items*) increases when *#sequences* increases, while *#items* is a fixed number in *Protein*, as there are always exactly 20 amino acids. In this experiment we set $min\_int = 0.01$ for SCII, $max\_size = 3$ and $min\_conf = 0.6$ for all methods.

The first two plots in Fig. 4 show the effect of an increasing dataset size on the run-times of all six algorithms. We began with a small subset of *Reuters1*, adding further sequences until we reached the full dataset. We plot the runtimes compared to the number of sequences, and the number of different items encountered in the sequences. For all six algorithms, the run-times grew similarly,

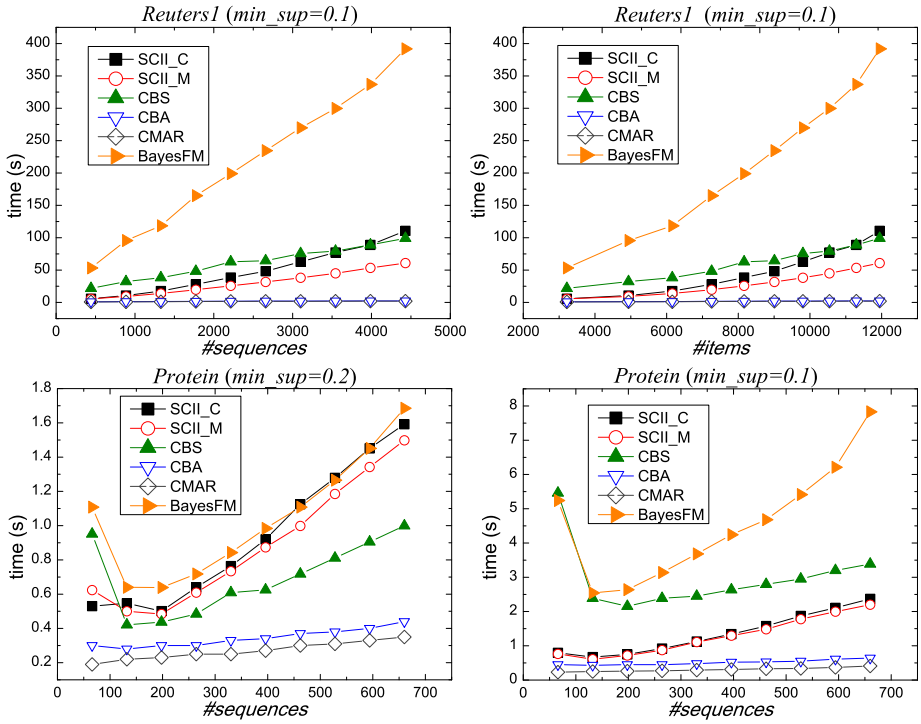

Fig. 4. Scalability analysis

with the classifiers based on sequential patterns the slowest, and the classifiers that took no sequential information into account the fastest.

The last two plots show the run-times of the algorithms on the *Protein* dataset. Here, too, we kept increasing the dataset size, but the number of items was always equal to 20. We performed the experiments with two different support thresholds, and it can be noted that classifiers based on sequential patterns are much more sensitive to the lowering of the threshold than our two classifiers. Once again, as expected, CMAR and CBA were fastest, but as was already seen in Table 3, their accuracy was unsatisfactory.

6 Conclusions

In this paper, we introduce a sequence classification method based on interesting itemsets named SCII with two variations. Through experimental evaluation, we confirm that the SCII methods provide higher classification accuracy compared to existing methods. The experimental results show that SCII is not sensitive to the setting of a minimum support threshold or a minimum confidence threshold. In addition, the SCII method is scalable as the runtime is proportional to the dataset size and the number of items in the dataset. Therefore, we can conclude

that SCII is an effective and stable method for classifying sequence data. What is more, the output rules of SCII are easily readable and understandably represent the features of datasets.

Acknowledgement. Cheng Zhou is financially supported by the China Scholarship Council (CSC).

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB'94. pp. 487–499. Morgan Kaufmann Publishers (1994)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE'95. pp. 3–14 (1995)
3. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: KDD'02. pp. 429–435. ACM (2002)
4. Cule, B., Goethals, B., Robardet, C.: A new constraint for mining sets in sequences. In: SDM'09. pp. 317–328 (2009)
5. Exarchos, T.P., Papaloukas, C., Lampros, C., Fotiadis, D.I.: Mining sequential patterns for protein fold recognition. Journal of Biomedical Informatics 41(1), 165–179 (2008)
6. Exarchos, T.P., Tsipouras, M.G., Papaloukas, C., Fotiadis, D.I.: A two-stage methodology for sequence classification based on sequential pattern mining and optimization. Data & Knowledge Engineering 66(3), 467–487 (2008)
7. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques, third edition. Morgan kaufmann (2011)
8. Lesh, N., Zaki, M.J., Ogihara, M.: Scalable feature mining for sequential data. IEEE Intelligent Systems 15(2), 48–56 (2000)
9. Li, W., Han, J., Pei, J.: Cmar: Accurate and efficient classification based on multiple class-association rules. In: ICDM'01. pp. 369–376. IEEE Computer Society (2001)
10. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: KDD'98. pp. 80–86 (1998)
11. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge and Data Engineering 16(11), 1424–1440 (2004)
12. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT'96. pp. 3–17. Springer-Verlag (1996)
13. Tseng, V.S., Lee, C.H.: Effective temporal data classification by integrating sequential pattern mining and probabilistic induction. Expert Systems with Applications 36(5), 9524–9532 (2009)
14. Tseng, V.S.M., Lee, C.h.: Cbs: A new classification method by using sequential patterns. In: SDM'05. pp. 596–600 (2005)
15. Yin, X., Han, J.: Cpar: Classification based on predictive association rules. In: SDM'03. pp. 331–335 (2003)
16. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Machine Learning 42(1-2), 31–60 (2001)
17. Zhao, Y., Zhang, H., Wu, S., Pei, J., Cao, L., Zhang, C., Bohlscheid, H.: Debt detection in social security by sequence classification using both positive and negative patterns. In: Machine Learning and Knowledge Discovery in Databases, pp. 648–663. Springer (2009)