Pattern Based Sequence Classification*

Cheng Zhou, Boris Cule and Bart Goethals

Abstract—Sequence classification is an important task in data mining. We address the problem of sequence classification using rules composed of interesting patterns found in a dataset of labelled sequences and accompanying class labels. We measure the interestingness of a pattern in a given class of sequences by combining the cohesion and the support of the pattern. We use the discovered patterns to generate confident classification rules, and present two different ways of building a classifier. The first classifier is based on an improved version of the existing method of classification based on association rules, while the second ranks the rules by first measuring their value specific to the new data object. Experimental results show that our rule based classifiers outperform existing comparable classifiers in terms of accuracy and stability. Additionally, we test a number of pattern feature based models that use different kinds of patterns as features to represent each sequence as a feature vector. We then apply a variety of machine learning algorithms for sequence classification, experimentally demonstrating that the patterns we discover represent the sequences well, and prove effective for the classification task.

Index Terms—sequence classification, interesting patterns, classification rules, feature vectors

1 Introduction

Sequential data is often encountered in a number of important settings, such as texts, videos, speech signals, biological structures and web usage logs, where a sequence is generally an ordered list of singletons. 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 (CBA) [2], sequential pattern based sequence classifier [3], the Classify-By-Sequence (CBS) algorithm [4], and so on. These combined methods can produce good results as well as provide users with information useful for understanding the characteristics of the dataset.

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 rules will not be discovered, and, with a low enough threshold, the rules that are discovered will be far too specific. In the other case, the class of a sequence is determined by items that occur in the sequence almost always in exactly the same order. At first glance, a sequence based classifier should outperform an itemset based classifier in this situation. However, itemset based classifiers will do better when the pattern sometimes occurs in an order different from the norm. This robustness means that itemset based classifiers can handle cases where small deviations in the subsequences that determine the class of the sequences occur. Moreover, due to a simpler candidate generation process, itemset based methods are much faster than those based on sequential patterns.

The above observations motivate the proposed research, **S**equence **C**lassification based on **I**nteresting **P**atterns (SCIP). First of all, we present algorithms to mine both types of interesting patterns — itemsets and subsequences. As a second step, we convert the discovered patterns into classification rules, and propose two methods to build classifiers to determine the class to which a new instance belongs. In the first method, we select the rules to be applied based on their confidence, while the second uses a novel approach by taking into account how cohesive the occurrence of the pattern that defines the rule is in the new instance. Finally, we step away from pattern based classification and evaluate the quality of our pattern miner by using our patterns as features in a variety of feature based classifiers.

When looking for interesting patterns in sequences, a pattern is typically evaluated based on how often it occurs (support). However, the proximity of the items that make up the pattern to each other (cohesion) is important, too [5]. If two classes are determined by exactly the same items, traditional pattern based classifiers may struggle. For example, if class A is determined by the occurrence of $\langle a, b\rangle$ with a shortest interval of 2 and class B by the occurrence of $\langle a, b\rangle$ with a shortest interval of 5, pattern $\langle a, b\rangle$ will not be enough to be able to tell the difference, and this will be solved by considering the cohesion information. Therefore, we use both cohesion and support to define interesting patterns in a sequence dataset. Finally, we utilise these interesting patterns to build classifiers. Experiments show the effectiveness of our classifiers and test which kind of

- *C. Zhou is with the Department of Mathematics and Computer Science, University of Antwerp, Belgium, and the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, China.*
  E-mail: cheng.zhou@uantwerpen.be
- *B. Cule is with the Department of Mathematics and Computer Science, University of Antwerp, Belgium, and the Computer & Decision Engineering Department, Université Libre de Bruxelles, Belgium.*
  E-mail: boris.cule@uantwerpen.be
- *B. Goethals is with the Department of Mathematics and Computer Science, University of Antwerp, Belgium.*
  E-mail: bart.goethals@uantwerpen.be

** A preliminary version appeared as "Itemset Based Sequence Classification", in the Proceedings of the 2013 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases [1].*

pattern works better in given circumstances. An additional advantage of our method is that the classifiers we build consist of easily understandable rules.

In this paper, we build on our previous work [1] by both improving and extending it. Firstly, we now present algorithms to mine two different types of patterns (only itemsets were handled in the previous work). Secondly, we replace the CBA based classifier used previously with a new classifier, based on the HARMONY scoring function [6], which achieves better performance. In addition, we now deploy a new top-k strategy for all the presented classifiers, instead of using only the highest ranked rule, as we did in our previous work. Moreover, we now propose using the discovered patterns as features in order to transform each sequence into a feature vector. We present a new feature vector representation approach by setting each feature value as the cohesion of the feature in a sequence. After this, we apply machine learning algorithms for sequence classification and find that this new feature vector representation approach outperforms the traditional presence-weighted feature vector representation approach.

The rest of the paper is organised as follows. Section 2 gives a review of the related work. In Section 3, we introduce the full description of our interestingness constraint for datasets consisting of multiple sequences. Section 4 presents our approach for generating classification rules and building the two types of classifiers. Experimental results are shown in Section 5 and we summarise our research in Section 6.

2 Related Work

The existing sequence classification techniques deploy a number of different machine learning methods, such as Naïve Bayes, k-nearest neighbours, decision trees, hidden Markov models, support vector machines, etc. [7]. Feature selection is a key step of classification by learning algorithms. Since different types of patterns can be informative, recent studies have proposed several effective classification methods based on pattern features, including itemset based approaches [8], [9] and subsequence based approaches [10], [11], [12], [13]. On top of these numerous learning methods, recent studies have proposed the use of pattern mining, such as association rules mining and sequential pattern mining, for building classification models.

2.1 Association Rule Based Classification

The main idea behind association classification is to discover class association rules (CARs) that always have a class label as their consequent. The next step is to use these rules (pattern $\Rightarrow$ class label) to build a classifier by selecting the most appropriate rules to classify new data records. Many association rule based classifiers have been proposed by adopting efficient association rule mining algorithms, e.g., Apriori [14] and FP-growth [15].

A well-known method of association classification, CBA proposed by Liu et al. [2] uses the Apriori-type association rule mining to generate CARs. An improved version of the method, CBA2 [16], solved the problem of unbalanced datasets by using multiple class minimum support values.

In another work, Li et al. [17] proposed CMAR by extending the FP-growth algorithm to mine large databases more efficiently than CBA. In CMAR, multiple rules are employed instead of just a single rule to avoid the overfitting inherent in CBA. On top of this, 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 used in CBA. Yin and Han [18] proposed CPAR which performs much faster in both rule generation and classification, but its accuracy is as high as that of CBA and CMAR. Wang and Karypis [6] proposed a new classifier, HARMONY, which directly mines the final set of classification rules. HARMONY uses a different scoring function which helps in improving the overall accuracy of the classifier. Chen et al. [19] proposed GARC, which prunes rules using information gain and certain strategies to filter out candidate itemsets in the generation process and shows similar accuracy performance to CBA. Recently, some more work has been done on developing alternative measures for finding informative patterns for association rule based classification [20], [21], [22], [23], [24], mostly achieving accuracy similar to HARMONY and CBA.

2.2 Sequential Pattern Based Classification

Since the concept of sequential pattern mining was first described by Agrawal and Srikant [25], other sequential pattern mining methods have been developed, such as Generalized Sequential Patterns (GSP) [26], SPADE [27], PrefixSpan [28], and SPAM [29]. PrefixSpan typically shows better performance than GSP and SPADE, but, when dealing with dense databases, the performance of PrefixSpan may be worse than that of SPADE. A number of sequence based classifiers have been based on these methods.

Lesh et al. [3] combined sequential pattern mining and a traditional Naïve Bayes classification method to classify sequences. 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 a pruning method is used in their algorithm, there was still a large 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 [4] 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. They proposed a number of alternative scoring functions and tested their performance. The results showed that the length of a CSP is the best attribute for classification scoring.

Exarchos et al. [30] proposed a two-stage methodology for sequence classification based on sequential pattern mining and optimisation. 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 optimisation technique to achieve optimal classification accuracy. However, the optimisation is very time consuming, and the accuracy of the algorithm is similar to that of FeatureMine.

Fradkin and Mörchen [31] proposed a direct sequential pattern mining approach, named BIDE-Discriminative, which uses class information for direct mining of predictive sequential patterns. They showed that their algorithm provides an efficient solution for sequence classification.

Additionally, several sequence classification methods have been proposed for application in specific domains. Exarchos et al. [32] utilized sequential pattern mining for protein fold recognition since it can contribute to the determination of the function of a protein whose structure is unknown. Zhao et al. [33] used a sequence classification method for debt detection in the domain of social security. They pointed out that the existing sequence classification methods based on sequential patterns consider only positive patterns. However, according to their experience in a large social security application, negative patterns are very useful in accurate debt detection. They therefore build a sequence classifier with both positive and negative sequential patterns. Experimental results show classifiers built with both positive and negative rules outperform traditional classifiers under most conditions.

All these approaches mine the frequent and confident (or discriminative) patterns for building a classifier. In this paper, we consider the cohesion of a pattern to identify interesting patterns and test whether the cohesion measure improves the classification performance.

3 Problem Statement

In this paper, we define an event e as 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 readability, and without any loss of generality, we assume that the time stamps in a sequence are consecutive natural numbers. We denote a *sequence* of such events by $s = \langle e_1, e_2, \ldots, e_l \rangle$ where l is the length of the sequence and thus s is an *l-sequence*. An event sequence $s' = \langle b_1, b_2, \ldots, b_m \rangle$ is said to be a *subsequence* of s if there exist integers $1 \leq i_1 < i_2 < \ldots < i_m \leq l$ such that $b_1 = e_{i_1}, b_2 = e_{i_2}, \cdots, b_m = e_{i_m}$, denoted as $s' \sqsubseteq s$ (if $s' \neq s$, written as $s' \sqsubset s$). A subsequence of s that starts at time stamp t_s and ends at time stamp t_e is denoted as $s_{(t_s,t_e)}$, e.g., $s_{(1,3)} = \langle e_1, e_2, e_3 \rangle$. A set $X = \{i_1, i_2, \ldots, i_n\} \subseteq I$ is called an *itemset*, or an *n-itemset* if it contains n items.

Let L be a finite set of class labels and $|L|$ the number of classes. A sequence database *SDB* is a set of *data objects* and a data object d is denoted by (s, L_k), where s is a sequence and $L_k \in L$ is a *class label* ($k = 1, 2, \ldots, |L|$). 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, such that $S_k \subseteq S$.

Example 1. *Given a training dataset of two classes as shown in Table 1. Then,* $S = \{s_1, s_2, \ldots, s_8\}$ *and* $S_1 = \{s_1, s_2, s_3, s_4\}$.

The patterns considered in this paper could be both itemsets or subsequences (sequential patterns), which is why we provide formal definitions applying to both pattern types below. The support count of a pattern is defined as the number of different sequences in which the pattern occurs; regardless of how many times the pattern occurs in any single sequence. In other words, when looking for the support count of a pattern alone, we can stop looking at a sequence as soon as we have encountered the first occurrence of the pattern.

TABLE 1
An Example of a Training Dataset

ID	Sequence	Class Label
1	$c\ c\ x\ y\ a\ b\ d$	L_1
2	$a\ b\ e\ e\ x\ x\ e\ c\ f\ d$	L_1
3	$c\ g\ h\ a\ b\ d\ d$	L_1
4	$d\ d\ e\ c\ f\ b\ a$	L_1
5	$a\ d\ z\ z\ c\ d\ b$	L_2
6	$b\ x\ y\ d\ d\ c\ d\ d\ d\ x\ a$	L_2
7	$b\ d\ c\ c\ c\ c\ a\ y$	L_2
8	$a\ x\ x\ c\ d\ b$	L_2

To determine the interestingness of a pattern, however, it is not enough to know how many times the items making up the pattern occur. We would also like to know how close they appear to each other. To do this, we will define interesting patterns in terms of both support and cohesion.

Our goal is to first mine interesting patterns in each class of sequences, and then use them to build a sequence classifier, i.e., a function from the set of sequences S to the set of class labels L.

3.1 Definition of an Interesting Pattern

The interestingness of a pattern depends on two factors: its support and its cohesion. Support measures in how many sequences the pattern appears, while cohesion measures how close the items making up the pattern are to each other on average, using the lengths of the shortest intervals containing the pattern in different sequences. As the patterns considered in this paper could be itemsets or subsequences, we give the definition of an interesting pattern based on itemsets and subsequences respectively.

3.1.1 Support.

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 $N_k(X) = \{s \in S_k | \forall i \in X, \exists (i, t) \in s\}$.

For a given subsequence s', we denote the set of sequences that contain s' as $N(s') = \{s \in S | s' \sqsubseteq s\}$. We denote the set of sequences that contain s' labelled by class label L_k as $N_k(s') = \{s \in S_k | s' \sqsubseteq s\}$.

The support of a pattern P in a given class of sequences S_k can now be defined as $F_k(P) = \frac{|N_k(P)|}{|S_k|}$, where P could be X or s'.

Example 2. *Consider the dataset shown in Table 1. Given* $X = \{a, b\}$*, we see that* $F_1(X) = \frac{4}{4} = 1$*. If* $s' = \langle a, b \rangle$*, then* $F_1(s') = \frac{3}{4} = 0.75$.

3.1.2 Cohesion.

We begin with defining the length of the shortest interval containing an itemset X in a sequence $s \in N(X)$ as $W(X, s) = \min\{t_2 - t_1 + 1 | t_1 \leq t_2 \text{ and } \forall i \in X, \exists (i, t) \in s, \text{ where } t_1 \leq t \leq t_2\}$.

We define the length of the shortest interval of a subsequence s' in a sequence $s \in N(s')$ as $W(s', s) = \min\{t_e - t_s + 1 | t_s \leq t_e \text{ and } s' \sqsubseteq s_{(t_s,t_e)}\}$.

In order to compute the cohesion of a pattern P within class k, we now compute the average length of such shortest intervals in $N_k(P)$: $\overline{W_k}(P) = \frac{\sum_{s \in N_k(P)} W(P,s)}{|N_k(P)|}$, where P could be X or s'.

It is clear that $\overline{W_k}(P)$ is greater than or equal to the number of items in P, denoted as $|P|$. Furthermore, for a fully cohesive pattern, $\overline{W_k}(P) = |P|$. Therefore, we define cohesion of P in $N_k(P)$ as $C_k(P) = \frac{|P|}{\overline{W_k}(P)}$.

Note that all patterns containing just one item are fully cohesive, that is, $C_k(P) = 1$ if $|P| = 1$.

The cohesion of P in a single sequence s is defined as $C(P, s) = \frac{|P|}{W(P,s)}$.

Example 3. *Consider again the dataset given in Table 1, assume* $X = \{b, d\}$*, then* $C_2(X) = \frac{2}{(2+4+2+2)/4} = 0.8$ *and* $C(X, s_3) = \frac{2}{2} = 1$.

3.1.3 Interestingness.

In a given class of sequences S_k, we can now define the interestingness of a pattern P as $I_k(P) = F_k(P) \times C_k(P)$, where P could be X or s'.

Given a minimum support threshold *min_sup* and a minimum interestingness threshold *min_int*, a pattern P is considered interesting in a set of sequences labelled by class label L_k, if $F_k(P) \geq$ *min_sup* and $I_k(P) \geq$ *min_int*.

3.2 Classification Rules

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

We define $r : P \Rightarrow L_k$ as a classification rule where P is an interesting pattern in S_k and L_k is a class label. P 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 to be equal to the interestingness, support, cohesion and size of P, respectively.

The confidence of a rule can now be defined as $conf(P \Rightarrow L_k) = \frac{|N_k(P)|}{|N(P)|}$, where P could be an itemset X (r is an *itemset rule*) or a subsequence s' (r is a *sequence rule*). A rule $P \Rightarrow L_k$ is considered confident if its confidence exceeds a given threshold *min_conf*.

An itemset X is said to *match* a sequence s if s contains all items in X, while a subsequence s' is said to *match* s if $s' \sqsubseteq s$. Therefore, if the antecedent of the rule *matches* 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.

4 Rule Based Classifiers

Our algorithm, SCIP (sequence classification based on interesting patterns), consists of two stages, rule generation (called SCIP-RG, with one variant using interesting itemsets (SCII-RG) and another using interesting subsequences (SCIS-RG)), and classifier building (called SCIP-CB).

4.1 Generating Interesting Itemsets

The rule generator for interesting itemsets (SCII-RG) 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 support 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*. The algorithm for generating the complete set of interesting itemsets in a given class of sequences is shown in Algorithm 1, with A_n denoting the set of frequent *n-itemset*s, C_n the set of candidate *n-itemset*s and T_n the set of interesting *n-itemset*s.

Algorithm 1: Generating interesting itemsets

Input : S_k, *min_sup*, *min_int*, *max_size*
Output: all interesting itemsets $\mathcal{X}_k$

```
1  A_1 ← {frequent items in S_k};
2  T_1 ← {X | X ∈ A_1, F_k(X) ≥ min_int};
3  n ← 2;
4  while A_{n-1} ≠ ∅ and n ≤ max_size do
5      C_n ← candidateGen(A_{n-1});
6      A_n ← {X | X ∈ C_n, F_k(X) ≥ min_sup};
7      T_n ← {X | X ∈ A_n, I_k(X) ≥ min_int};
8      n++;
9  X_k ← ⋃ T_i;
10 return X_k;
```

Line 1 counts the supports of all the items to determine the frequent items. Line 2 stores the interesting items in T_1 (note that the interestingness of a single item is equal to its support). Lines 3-10 discover all interesting itemsets of different sizes n (*max_size* $\geq n \geq$ 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 5). The candidateGen function is similar to the function Apriori-gen in the Apriori algorithm [14]. In line 6, we store the frequent itemsets from C_n into A_n. Line 7 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 $\mathcal{X}_k$ 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. To get $I_k(X)$, we first need to find a shortest interval $W(X, s)$ of an itemset X in each sequence $s \in N_k(X)$, as shown in Algorithm 2. The crucial step is the computation of the candidate intervals $W(X, t_j)$ for the time stamps t_j at which an item of X occurs. Clearly, to find the shortest interval in the entire sequence, it is enough to select a single item in X and to identify the shortest intervals for each time stamp at which this item occurs. To make the loop process faster, we pick the item in X that has the lowest frequency in s. We keep the set of candidate intervals associated with X in a list CW (line 1). To find the candidate interval around position t_j containing all items of X, we start by looking for the nearest occurrences of items of X both left and right of position t_j (lines 4-8). Note that if an

item a_k does not occur at all before the current time stamp, we set its left time stamp l_{jk} to $-\infty$, and if it does not occur after t_j, we set its right time stamp r_{jk} to ∞. In lines 9-10, we get an ordered list L_j of ascending left time stamps and a list R_j of right time stamps following the same order of items as L_j. In line 11, function getInterval computes the shortest interval for time stamp t_j. If we get a shortest interval which equals to $|X|$, we return $W(X, s) = |X|$, and we need to look no further, since no later interval can be shorter (line 12). Otherwise, $W(X, s)$ equals to the smallest value in CW (line 14).

Algorithm 2: Finding the shortest interval of an itemset

Input : sequence s, itemset $X = \{a_1, a_2, \ldots, a_{|X|}\}$ (where a_1 has the lowest frequency in s)
Output: the length of the shortest interval $W(X, s)$

1 $CW \leftarrow \emptyset$;
2 $G_i \leftarrow$ time stamps of a_i, $i = \{1, \ldots, |X|\}$;
3 **foreach** *time stamp* t_j *in* G_1 **do**
4 **for** $k = 2$ *to* $|X|$ **do**
5 **if** $\forall\, t \in G_k,\ t > t_j$ **then** $l_{jk} \leftarrow -\infty$;
6 **else** $l_{jk} \leftarrow \max\{t \in G_k | t < t_j\}$;
7 **if** $\forall\, t \in G_k,\ t < t_j$ **then** $r_{jk} \leftarrow \infty$;
8 **else** $r_{jk} \leftarrow \min\{t \in G_k | t > t_j\}$;
9 $L_j \leftarrow [l_{jk} |\ k = \{2, \ldots, |X|\}]$;
10 $R_j \leftarrow [r_{jk} |\ k = \{2, \ldots, |X|\}]$;
11 $w \leftarrow$ getInterval(L_j, R_j, t_j);
12 **if** $w == |X|$ **then return** w;
13 $CW \leftarrow CW \bigcup \{w\}$
14 **return** $\min\{w \in CW\}$;

Algorithm 3 shows the pseudocode of the getInterval function, which computes the size of the shortest occurrence of X around time stamp t_j. Throughout the procedure, $[l_j, r_j]$ represents the current candidate interval, and we evaluate all possible intervals with $l_j \in L_j$ and $r_j \in R_j$. We start off with l_j set to the smallest time stamp in L_j and r_j set to t_j, which represents the smallest occurrence of X such that all items in X occur at time stamps smaller than or equal to t_j. The process continues by moving to the next starting point l_j (lines 5-6) and gets a new ending point r_j corresponding to the item of the last starting point (lines 7-8). If the new interval is shorter than the shortest interval found so far, we update the shortest interval length (line 10). Once again, if we have found an interval of length $|X|$ or the interval on the other side has grown sufficiently to make it impossible to get a smaller shortest interval, we stop the procedure and return w (line 11). Otherwise, the procedure stops when we have processed all the time stamps in L_j.

Example 4. *Assume $s = \langle a, b, c, d, a, c\rangle$ and we want to find the shortest interval of $X = \{a, b, c, d\}$. First, we pick the the least frequent item b and thus $t_j = 2$. We obtain $L_j = [-\infty, -\infty, 1]$ and $R_j = [3, 4, 5]$ for items c, d and a. We then get the first candidate interval of length ∞ based on lines 1-3 of Algorithm 3. Executing the for loop of Algorithm 3, we then get a new candidate interval of length ∞ with $l_j = -\infty$, $r_j = 3$ when $k = 2$ and finally get another candidate interval of length 4 with $l_j = 1$, $r_j = 4$ when $k = 3$. Therefore, we stop the procedure and return $W(X, s) = 4$.*

Algorithm 3: getInterval(L_j, R_j, t_j)

Input : L_j, R_j, t_j
Output: a candidate interval w

1 $l_j \leftarrow L_j[1]$;
2 $r_j \leftarrow t_j$;
3 $w \leftarrow r_j - l_j + 1$;
4 **for** $k = 2$ *to* $|X|$ **do**
5 **if** $k < |X|$ **then** $l_j \leftarrow L_j[k]$;
6 **else** $l_j \leftarrow t_j$;
7 $r'_j \leftarrow R_j[k-1]$;
8 **if** $r'_j > r_j$ **then** $r_j \leftarrow r'_j$;
9 $cw \leftarrow r_j - l_j + 1$;
10 **if** $cw < w$ **then** $w \leftarrow cw$;
11 **if** $w == |X|$ or $w \leq r_j - t_j + 1$ **then return** w;
12 **return** w;

Assume that s contains k_i occurrences of the ith item of X. Since the sequence is ordered, we obtain the sorted list of occurrences for each item when loading the dataset. Therefore, in order to find all the shortest intervals around each occurrence of a particular item in X, in the worst case we might need to read the complete occurrence lists of all items in X. Therefore, the time complexity of this process is $O(\sum_{i=1}^{|X|} k_i)$. Theoretically, in the worst case, the time complexity to get $W(X, s)$ is $O(|s|)$. However, this worst case can only materialise if X is composed of all items that appear in s. When evaluating an itemset of a limited size, we essentially need to do no more than go through the occurrence lists of the items that make up the itemset. Finally, in order to compute $I_k(X)$, we need to repeat this process in each sequence that contains X. Experiments confirm that this method indeed works efficiently.

4.2 Generating Interesting Subsequences

We base our method for generating interesting subsequences on the well-known SPADE algorithm [27], which is capable of efficiently finding all frequent subsequences. To determine the support of any l-sequence, SPADE looks at intersections of the id-lists of any two of its subsequences of length $(l - 1)$ since such an id-list keeps a list of the sequences in which a pattern occurs, as well as where in those sequences the items making up the pattern occur. Algorithms 4 and 5 show the process of generating all interesting subsequences. In Algorithm 4, lines 1-2 store the interesting 1-sequences in T_1. Line 4 calls Algorithm 5 to get interesting n-sequences ($2 \leq n \leq$ *max_size*). Finally, we get the complete set of interesting subsequences $\mathcal{Y}_k$ (line 5).

Algorithm 4: Generating interesting subsequences

Input : S_k, *min_sup*, *min_int*, *max_size*
Output: all interesting subsequences $\mathcal{Y}_k$

1 $A_1 \leftarrow$ {frequent items or 1-sequences in S_k};
2 $T_1 \leftarrow \{s' | s' \in A_1, F_k(s') \geq$ *min_int*$\}$;
3 $\mathcal{Y}_k \leftarrow T_1$;
4 Enumerate-Sequence(A_1);
5 **return** $\mathcal{Y}_k$;

In Algorithm 5, given any two l-sequences α_i and α_j that share the same $(l-1)$ length prefix, we generate a candidate sequence s' of length $(l+1)$ by adding the last item in α_j to α_i (line 4). In order to determine the cohesion of a subsequence s', needed to compute $I_k(s')$ in line 7, we had to make a modification to the original SPADE algorithm. We added a function that finds the shortest intervals of a frequent subsequence s' in an input sequence s by tracking the occurrences in s of all the items contained in s'.

Algorithm 5: Enumerate-Sequence(Q)

Input : a set Q of sequences of size l sharing the first $l-1$ elements

```
1  foreach α_i in Q do
2      A_i ← ∅, T_i ← ∅;
3      foreach α_j in Q do
4          s' ← α_i + α_j[l];
5          if F_k(s') ≥ min_sup then
6              A_i ← A_i ∪ {s'};
7              if I_k(s') ≥ min_int then
8                  T_i ← T_i ∪ {s'};
9      Y_k ← Y_k ⋃ T_i;
10     if l < max_size then Enumerate-Sequence(A_i);
```

Similar to the case of itemsets, the time complexity for getting the shortest interval of s' in s is $O(\sum_{i=1}^{|s'|} k_i)$. Again, in the theoretical worst case, this could be equal to $O(|s|)$, but is in reality much smaller. This computation, naturally, has to be done in each sequence $s \in S_k$ that contains s'.

4.3 Pruning the Rules

Once we have found all interesting patterns in a given class, all confident classification rules can be found in a trivial step — for each pattern P that is interesting in class L_k, we generate rule $P \Rightarrow L_k$. However, the number of patterns is typically very large, which leads to a large number 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 [17], and prune unnecessary rules using the database coverage method.

Before using the database coverage method, we must first define a total order on the set of all generated rules R. 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. the confidence of r_i and r_j is the same, but the interestingness of r_i is greater than that of r_j, or

3. both the confidence and the interestingness 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, but r_i is generated earlier than r_j.

We apply the database coverage method to get the most important 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 6.

Algorithm 6: Finding the most important 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
3      d.cover_count ← 0;
4  foreach rule r in sorted R do
5      foreach data object d in D do
6          if rule r correctly classifies data object d then
7              store r into PR;
8              d.cover_count++;
9              if d.cover_count ≥ δ then
10                 delete d from D;
11 return PR;
```

The algorithm has two 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 2-10, 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 4-8). We store the rule r into *PR* if it correctly classifies a data object (line 7). If the cover count of a data object passes the coverage threshold, we remove the data object (line 10). Line 11 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 $O(|s|)$.

Example 5. *To illustrate how the algorithms work, consider the training dataset given in Table 1 again. Assume min_sup = min_int = 0.6, max_size = 3 and min_conf = 0.5. After finding frequent patterns in S_1 and S_2 as described in Sections 4.1 and 4.2, we get the confident rules sorted using Definition 1 for itemset rules and sequence rules respectively, as shown in Table 2. Assuming we use a database coverage threshold $\delta = 1$, only the rules shown in bold would survive the pruning stage.*

4.4 Building the Classifiers

This subsection presents the SCIP-CB algorithm for building a classifier using the interesting patterns discovered by SCIP-RG. CBA [2] has successfully shown the competitive accuracy performance of an associative classifier. However, HARMONY [6] uses a different scoring function to improve the overall accuracy of the classifier. When classifying a new data object, HARMONY computes the score of a class label L_k as the sum of the top λ highest confidences of the rules carrying class label L_k and matching the data object. Since HARMONY outperforms CBA (as shown later in Table 4

TABLE 2
Sorted Itemset and Sequence Rules

Itemset Rule	$C_k(r)$	$conf(r)$	Sequence Rule	$C_k(r)$	$conf(r)$
$\boldsymbol{ab \Rightarrow L_1}$	1.0	0.5	$\boldsymbol{\langle a,b\rangle \Rightarrow L_1}$	1.0	0.6
$\boldsymbol{cd \Rightarrow L_2}$	1.0	0.5	$\boldsymbol{\langle c,d\rangle \Rightarrow L_2}$	1.0	0.5
$c \Rightarrow L_1$	1.0	0.5	$\boldsymbol{\langle c\rangle \Rightarrow L_1}$	1.0	0.5
$a \Rightarrow L_1$	1.0	0.5	$\langle a\rangle \Rightarrow L_1$	1.0	0.5
$b \Rightarrow L_1$	1.0	0.5	$\langle b\rangle \Rightarrow L_1$	1.0	0.5
$d \Rightarrow L_1$	1.0	0.5	$\langle d\rangle \Rightarrow L_1$	1.0	0.5
$c \Rightarrow L_2$	1.0	0.5	$\boldsymbol{\langle c\rangle \Rightarrow L_2}$	1.0	0.5
$a \Rightarrow L_2$	1.0	0.5	$\langle a\rangle \Rightarrow L_2$	1.0	0.5
$b \Rightarrow L_2$	1.0	0.5	$\langle b\rangle \Rightarrow L_2$	1.0	0.5
$d \Rightarrow L_2$	1.0	0.5	$\langle d\rangle \Rightarrow L_2$	1.0	0.5
$cbd \Rightarrow L_2$	0.8	0.5			
$bd \Rightarrow L_2$	0.8	0.5			

in Section 5), we first propose a method named SCIP_HAR (**HAR**MONY based classifier), whereby we also, like CBA2, allow different support thresholds for different classes.

Rather than ranking the rules using their confidence, we further propose incorporating the cohesion of the antecedent of a rule in the new data object into the measure of the appropriateness of a rule for classifying the object. 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 top λ ranked rules. We call this method SCIP_MA (**Ma**tching cohesive rules based classifier).

HARMONY does not account for the possibility that there may not exist a rule matching the given data object which decreases its accuracy performance. We fix this by adding a default rule, $null \Rightarrow L_d$, 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.

The algorithm for choosing the default rule for both SCIP_HAR and SCIP_MA is given in Algorithm 7. Lines 1-3 delete the data objects matched by rules in *PR*. Lines 4-6 count how many times each class label appears in the remainder of the dataset. Line 7 sets the label that appears the most time 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. Finally, line 8 generates the default rule. Since we need to do this for all data objects and all rules, the time complexity of finding the default rule is $O(|PR| \sum_{s \in D} |s|)$.

Algorithm 7: Getting the default rule

Input : pruned rules *PR*, training dataset D
Output: the default rule *default_r*

```
1 foreach rule r in PR do
2     foreach data object d in D do
3         if r matches d then delete d from D;
4 counter ← a new array of size |L|;
5 foreach data object (s, L_k) in D do
6     counter[k]++;
7 L_d ← the class label L_k with largest counter[k];
8 return default_r : null ⇒ L_d;
```

The classifiers are thus composed of *PR* and the default rule *default_r*. We now show how we select the top λ rules and classify the sequence in a new data object. The algorithm for finding the rules and classifying a new sequence is shown in Algorithm 8.

Algorithm 8: Classifying a new sequence

Input : *PR*, *default_r*, λ, a new unclassified data object $d = (s, L_?)$
Output: a class label

```
1  MR ← ∅;
2  foreach rule r in PR do
3      if r matches d then store r into MR;
4  if MR.size > 0 then
5      foreach rule r : P ⇒ L_k in MR do
6          if use SCIP_HAR then
7              r.value ← r.confidence;
8          if use SCIP_MA then
9              r.value ← r.confidence × C(P, d.s);
10     sort rules in MR by descending r.value;
11     CR ← {the top λ rules in sorted MR};
12     score ← a new array of size |L|;
13     foreach rule r' : P ⇒ L_k in CR do
14         score[k] ← score[k] + r'.value;
15     return the class label L_k with largest score[k];
16 else return the class label of default_r;
```

First, we find all the rules that match the given data object d and store them into *MR* (lines 1-3). Then, we handle two different cases:

1. (lines 4-15): If the size of *MR* is greater than 0, we compute the $r.value$ of every rule in *MR* and sort the rules according to $r.value$ (the higher the value, the higher the precedence). Lines 6-9 compute the score of a rule for SCIP_HAR and SCIP_MA, respectively. We finally utilise the top λ rules in the sorted *MR*, denoted by *CR*, to classify the given data object. If the number of rules in *MR* is smaller than λ, we simply use all the rules in *MR* as *CR*. Lines 12-14 compute the sum of $r.value$s of each rule in *CR* according to their class labels and line 15 returns the class label which has the largest sum.

2. (line 16): If *MR* is empty, we return the class label of the default rule.

The only time-consuming part of Algorithm 8 is the computation of $C(P, d.s)$. The time complexity of this computation has already been analysed at the end of Sections 4.1 and 4.2.

Since we investigate building classifiers by both itemset rules and sequence rules, the complete set of SCIP classifiers contains four classifiers — two itemset based classifiers (SCII_HAR and SCII_MA) and two sequence based classifiers (SCIS_HAR and SCIS_MA).

Example 6. *Returning to our running example, assume we are given a new data object (s_9, L_2), with $s_9 = \langle a, x, b, y, c, d, z\rangle$. If we try using the rules discovered in Example 5, we can see that it is not easy to choose the correct classification rule, as all the remaining rules match s_9. We now illustrate how our classifiers classify the new data object. For simplicity, we set $\lambda = 1$.*

1. Classify by SCII_HAR and SCII_MA. As can be seen in Table 2, the CBA, CMAR and SCII_HAR methods would have no means to distinguish between the top two itemset rules, and would classify s_9 into class 1, simply because rule $ab \Rightarrow L_1$ was generated before rule $cd \Rightarrow L_2$. Using SCII_MA, however, we would re-rank the rules taking the cohesion of the antecedent in s_9 into account. In the end, rule $cd \Rightarrow L_2$ is chosen, as $C(cd, s_9) = 1$, while $C(ab, s_9) = \frac{2}{3}$. We see that SCII_MA classifies the new sequence correctly, while other methods fail to do so.

2. Classify by SCIS_HAR and SCIS_MA. It is obvious that s_9 would be classified into class 1 by SCIS_HAR since $r_1 : \langle a, b\rangle \Rightarrow L_1$ is the rule with the highest confidence. Meanwhile, using SCIS_MA, rule $r_2 : \langle c, d\rangle \Rightarrow L_2$ is chosen. That is because $C(\langle c, d\rangle, s_9) = 1$, while $C(\langle a, b\rangle, s_9) = \frac{2}{3}$, and thus $r_2.value = 1 \times 0.5 = 0.5 > r_1.value = \frac{2}{3} \times 0.6 = 0.4$. We see that SCIS_MA classifies the new sequence correctly.

5 Experiments

In this section, we present the results of our experiments. The performed experiments can be divided into two types. In the first set of experiments, we compared our SCIP classifiers with other comparable rule based classifiers in terms of accuracy and scalability. In the second set of experiments, we evaluated the usefulness of our pattern mining method by utilising the discovered patterns to represent the input sequences as feature vectors and then applying learning algorithms for sequence classification. All experiments were performed on a PC with Intel Xeon CPU at 2.90GHz, Ubuntu 12.04.4 and the maximum heap size was set to 4G (using -Xmx4G). All the reported accuracies in all experiments were obtained using 10-fold cross-validation.

5.1 Datasets

In order to evaluate the proposed methods, we used six real-life datasets, as summarised in Table 3. The first dataset was formed by making a selection from the Reuters-21578 dataset[1], consisting of news stories, assembled and indexed with categories by Reuters Ltd personnel. We formed the *Reuters* dataset using the four biggest classes in the Reuters-21578 dataset, "acq" (1596 documents), "earn" (2840 documents), "crude" (253 documents) and "trade" (251 documents). Then we balanced the dataset by keeping only the first 253 paragraphs in the top two classes.

TABLE 3
Summary of the Datasets

Dataset	# Classes	$\|D\|$	# Items	Average Length
Reuters	4	1010	6380	93.84
News	5	4976	27884	139.96
WebKB	3	3695	7737	128.40
Protein	2	538	20	14.86
Unix	4	5472	1711	32.34
Robot	2	4302	95	24.00

The second dataset *News* was formed by selecting the five biggest groups of documents from the 20Newsgroups[1] dataset. The five groups are rec.sport.hockey (999 documents), rec.motorcycles (996 documents), soc.religion.christian (996 documents), rec.sport.baseball (994 documents) and sci.crypt (991 documents).

The third dataset *WebKB* consists of the content of webpages from the WebKB collection[2] collected from computer science departments of various universities in January 1997 by the World Wide Knowledge Base project of the CMU text learning group. In our experiments, we used the three largest classes (*student* with 1641 documents, *faculty* with 1124 documents and *course* with 930 documents).

In the three datasets described above, we kept only letters, while other symbols and punctuation were converted into spaces. We then applied stemming and stop word removal, after which we considered the stemmed words appearing in the texts as items and treated each separate document as a sequence.

The fourth dataset is a protein dataset obtained from PhosphoELM[3]. The data consists of different combinations of amino acids for each kind of protein. We chose two of the biggest Kinase groups (PKA_group with 381 combinations and SRC with 157 combinations) to form the *Protein* dataset. Each combination of amino acids is a segment from the position of the S/T/Y phosphorylation site. 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.

The fifth dataset deals with the problem of differentiating the users by their UNIX shell command data. The *Unix* dataset consists of the UNIX command sessions of four UNIX users (user 6, 8, 4 and 5) from a UNIX user dataset[4]. The dataset is collected from the Purdue MILLENNIUM machine learning lab [34] over varying periods of time. We treat each command session as a sequence and consider each command token as an item. Each sequence is labelled by the user it belongs to.

Our last dataset *Robot* was formed by selecting the largest two classes (Move-Forward with 2205 samples and Sharp-Right-Turn with 2097 samples) of the data[5] collected as the robot navigates through the room following the wall in a clockwise direction. Each sequence consists of ultrasound readings collected by the sensors carried by the robot. Since these readings are real numbers, we discretised them into items with a step of 0.05 to meet the needs of frequent pattern mining, e.g., values 0.02 and 5.00 are discretised to 1 and 100, respectively.

5.2 Comparison of Rule Based Classifiers and HMM

We compared our four classifiers, SCII_HAR, SCII_MA, SCIS_HAR and SCIS_MA, with six benchmark classifiers: CBA, CMAR, HARMONY, BayesFM, CBS and HMM. The CBS paper proposed a number of different scoring functions used for selecting the classification rules, and we chose the length policy as it gave the best results. For fairer comparison, we also added a *max_size* constraint into the pattern mining stage of CBS. We implemented our classifiers, BayesFM and CBS in Java[6], while we downloaded

1. http://web.ist.utl.pt/acardoso/datasets/

2. http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/

3. http://phospho.elm.eu.org/

4. https://archive.ics.uci.edu/ml/datasets/UNIX+User+Data

5. https://archive.ics.uci.edu/ml/datasets/Wall-Following+Robot+Navigation+Data

6. http://adrem.ua.ac.be/sites/adrem.ua.ac.be/files/SCIP.zip

TABLE 4
Comparison of Predictive Accuracy (%)

Dataset	*max_size*	SCII_HAR	SCII_MA	SCIS_HAR	SCIS_MA	CBA	CMAR	HARMONY	BayesFM	CBS	HMM
	2	94.75	95.14	**96.24**	96.04	92.48	91.29	94.55	84.46	89.90	
	3	95.05	95.25	**95.74**	95.64	92.87	92.28	92.87	82.77	90.50	
Reuters	4	94.85	94.85	**95.64**	95.24	93.07	92.28	93.07	81.78	89.90	
(top 11 rules)	5	94.85	94.85	**95.64**	95.24	93.07	92.38	92.97	81.78	90.89	
	∞	94.85	94.85	**95.84**	95.74	93.07	92.38	92.97	81.68	N/A	95.05
	2	93.93	**94.05**	93.72	93.69	81.31	75.24	93.76	76.43	72.14	
	3	93.85	**94.01**	93.65	93.73	81.21	74.22	93.67	67.79	61.70	
News	4	93.85	**94.01**	93.65	93.73	81.19	73.98	93.30	61.13	52.55	
(top 1 rule)	5	93.87	**94.01**	93.71	93.73	81.17	73.90	93.24	56.17	66.71	
	∞	93.89	**94.01**	93.71	93.71	81.17	73.90	93.20	N/A	N/A	84.00
	2	90.12	**90.31**	89.77	89.06	86.00	73.69	89.84	65.88	67.55	
	3	89.22	**90.68**	89.11	90.00	86.30	76.18	88.41	61.98	64.12	
WebKB	4	89.12	**90.52**	89.15	89.98	86.22	77.00	87.92	60.14	61.98	
(top 11 rules)	5	89.20	**90.60**	89.06	90.09	86.11	77.37	87.51	59.68	60.79	
	∞	89.20	**90.66**	89.06	90.09	86.11	77.54	87.46	59.33	70.51	86.31
	2	90.17	90.90	89.95	**91.61**	85.86	86.25	89.79	84.78	73.32	
	3	91.60	92.34	93.66	**94.98**	83.78	88.31	90.68	93.31	88.61	
Protein	4	90.12	91.23	94.24	**94.82**	83.92	89.22	89.01	91.28	94.60	
(top 11 rules)	5	89.94	91.07	94.06	**94.82**	83.45	89.22	87.90	91.28	94.60	
	∞	89.96	90.34	94.06	**94.82**	83.64	89.22	88.31	91.28	94.60	91.82
	2	88.15	88.05	88.12	87.88	78.75	76.42	**88.72**	83.75	85.60	
	3	**88.23**	88.08	88.19	87.94	78.75	76.13	87.82	82.58	83.75	
Unix	4	**88.21**	88.16	88.12	87.81	78.75	76.13	87.80	81.93	81.58	
(top 1 rule)	5	**88.21**	88.16	88.12	87.81	78.75	76.13	87.79	81.03	79.62	
	∞	88.21	88.16	88.12	87.99	78.75	76.13	87.79	81.10	N/A	**89.09**
	2	79.43	79.82	80.21	79.17	76.24	71.22	**82.31**	76.85	79.10	
	3	80.43	81.52	82.12	82.31	75.69	71.71	**82.33**	76.48	78.22	
Robot	4	80.82	81.73	82.72	**83.26**	75.64	71.69	82.98	76.57	78.71	
(top 11 rules)	5	80.94	81.89	83.61	**83.73**	75.64	71.69	83.05	76.89	78.29	
	∞	80.94	81.87	83.52	**83.89**	75.64	71.69	83.05	77.31	78.92	83.24
avg rank (∞)		4.08	3.42	2.92	**2.42**	7.83	8.50	6.00	8.08	7.92	3.83

implementations of CBA and CMAR from the LUCS-KDD Software Library[7]. The implementation of HMM was downloaded from a package[8] for the machine learning environment WEKA [35]. The database coverage threshold was set to 1 for our classifiers as a default value.

5.2.1 Comparison of Predictive Accuracy

Table 4 reports the accuracy results of all ten classifiers using various *max_size* thresholds. In the experiments, we set *min_conf* to 0.5 and *min_sup* to 0.05 for all of the classifiers except HMM (since HMM cannot limit the pattern size, we only report its accuracy at *max_size* $= \infty$). Additionally, we set *min_int* to 0.02 for the SCIP classifiers. The best results are highlighted in bold and the N/As mean that the algorithms ran out of memory. As shown in Table 4, the SCIP classifiers generally outperform other classifiers. Sicne there are only 20 different items (amino acids) in the *Protein* dataset, most of the combinations of items are frequent in different classes. Therefore, SCIS classifiers differentiate different classes better than SCII classifiers as they keep the most information about the order of items in a sequence. Furthermore, we applied the Friedman test and the average ranks of the various classifiers at *max_size* $= \infty$ are reported at the bottom of Table 4, confirming the above observations. Overall, SCIS_MA produced the best results. Finally, we performed the pair-wise sign test on all possible pairs of classifiers, to test if the differences mentioned above were statistically significant. The results confirmed that SCIS_MA statistically significantly outperformed CBA, CMAR, HARMONY, BayesFM and CBS (at the level of 0.05). SCIS_HAR significantly outperformed CBA, CMAR, HARMONY and BayesFM, while the two SCII classifiers significantly outperformed CBA and CMAR. While none of our methods proved statistically significantly better than HMM, we can still see in Table 4 that our SCIS methods regularly outperformed HMM in most cases.

Additionally, we experimented with various values of the *max_size* parameter to check its impact on accuracy and runtime. We found that a larger *max_size* requires much more runtime but typically achieves similar accuracy. For example, the runtime of the SCIS classifiers on the *Reuters* dataset is 78.54s with *max_size* $= 3$ and 827.58s with *max_size* $= 5$ while the accuracy is almost the same. Therefore, in all remaining experiments, we kept *max_size* $= 3$ since this setting proved sufficient for the classifiers to achieve satisfactory results.

The default value of λ (the number of rules used for classifying) in HARMONY is ∞. However, we ran our SCIP classifiers with various values of λ, and found that the classifiers do not always achieve the best accuracy with $\lambda = \infty$, as was also reported in the work of HARMONY. Moreover, once λ is large enough, the accuracy does not change much with further increases. Therefore, we only report the results for the value of λ that, on average, produced the highest accuracy (the used λ is reported under the name of each dataset in Table 4). In all remaining experiments, we kept λ fixed at the reported value for each dataset.

7. http://cgi.csc.liv.ac.uk/~frans/KDD/Software/
8. http://doc.gold.ac.uk/~mas02mg/software/hmmweka/#about

For HMM, all items that appear in the dataset become primary candidates for input features. The computation complexity is sensitive to the size and the quality of the features and is somewhat related to the performance of classification [36]. Therefore, we removed items occurring in less than $K\%$ ($K = \{1, 2, 3, \ldots, 20\}$) of the dataset and only the peak value of the accuracy was reported.

5.2.2 Impact of Different Thresholds

To further explore the performance of the classifiers presented above, we conducted an analysis of the predictive accuracy under different support, confidence, and interestingness thresholds, respectively (HMM is omitted , since it uses no such thresholds). We first experimented on different support thresholds, where *min_conf* is fixed at 0.5 and *min_int* for SCIP classifiers is set to 0.05, as shown in Fig. 1. Overall, the SCIP classifiers generally outperform other classifiers and the performance of all classifiers decreases with increasing support thresholds. Table 5 shows the average number of generated rules for each cross-validation with respect to different support thresholds on *Reuters* (CBA and CMAR have the same generated rules). We can see that the performance of SCIP classifiers and CBS are not that sensitive to the minimum support thresholds since, unlike the other classifiers, they mine frequent patterns from each class separately, resulting in more generated rules even with a large support threshold. For example, on *Reuters*, the accuracies of CBA and CMAR drop fast since they fail to find enough rules to build the classifier when $min\_sup > 0.12$.

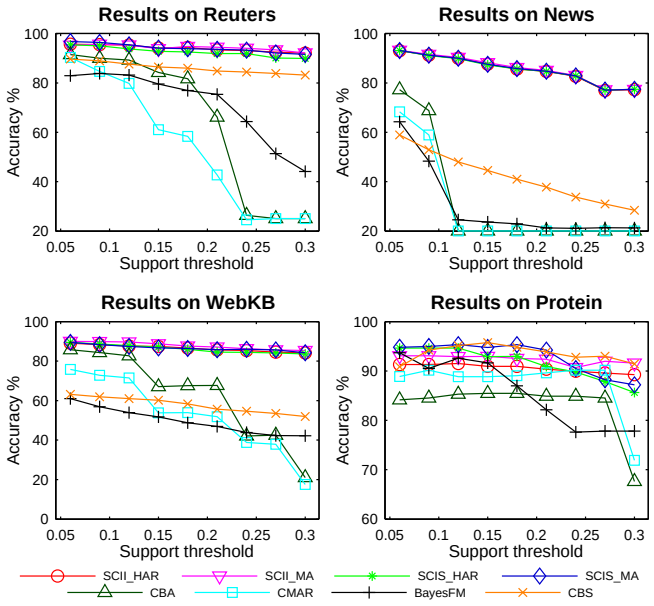

Fig. 1. The impact of the support threshold on accuracy.

Fig. 2 compares the predictive accuracy of the classifiers at various confidence thresholds (CMAR and BayesFM are omitted since they do not use a confidence threshold). Here, *min_sup* is fixed at 0.1 and *min_int* for the SCIP classifiers is set to 0.06. From Fig. 2, we can see that the SCIP classifiers are not sensitive to the minimum confidence thresholds at all until $min\_conf > 80\%$. On top of that, they always outperform CBA and CMAR. The number of generated

TABLE 5
Average Number of Rules at Varying Support Thresholds in *Reuters*

min_sup	SCII	SCIS	CBA/CMAR	BayesFM	CBS
0.06	826.2	951.6	1087.0	5324.4	92846.9
0.12	563.9	666.4	43.5	731.4	14983.9
0.18	406.3	470.4	9.2	216.3	4726.6
0.24	304.9	337.7	0.1	85.3	2007.3
0.30	211.4	212.5	0.0	39.8	941.9

rules on the *Reuters* dataset is reported in Table 6. The results for CBA and CMAR show that their performance is strongly related to the number of produced rules, as they fail to build a successful classifier if the number of rules is limited (as is the case at high confidence thresholds). Fig. 2 shows that this is particularly the case with CMAR although it gets the same rules as CBA before building the classifier.

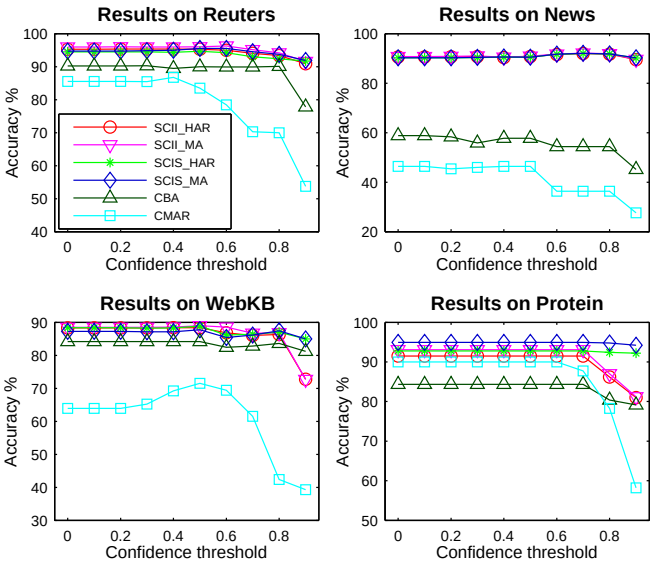

Fig. 2. The impact of the confidence threshold on accuracy.

TABLE 6
Average Number of Rules at Varying Confidence Thresholds in *Reuters*

min_conf	SCII	SCIS	CBA/CMAR
0.1	806.3	894.6	245.4
0.3	701.6	787.6	213.8
0.5	521.9	605.3	138.2
0.7	388.6	474.8	107.1
0.9	268.1	362.2	45.6

Fig. 3 shows the accuracy of the SCIP classifiers with different minimum interestingness thresholds, varying from 0.01 to 0.19 with a step size of 0.02. Here, *min_sup* is fixed at 0.1 and *min_conf* at 0.5. We can see that the accuracies of the SCIP classifiers first increase with an increasing *min_int*, and then start to decrease again with $min\_int > 0.1$, as shown in Table 7. We conclude that setting the interestingness threshold too low can lead to useless rules finding their way into the classifier, while setting it too high will result in not having enough rules to correctly classify new data objects.

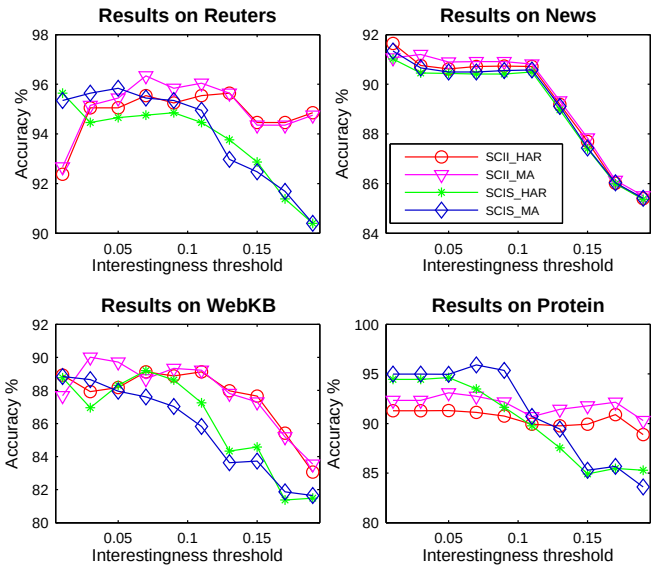

Fig. 3. The impact of the interestingness threshold on accuracy.

TABLE 7
Average Number of Rules at Varying Interestingness Thresholds

min_int	Dataset	SCII	SCIS	Dataset	SCII	SCIS
0.03		1247.3	1519.0		417.3	370.2
0.07		442.5	496.2		212.5	203.3
0.11	*Reuters*	259.8	281.2	*News*	163.5	161.7
0.15		142.0	148.4		86.5	86.3
0.19		95.7	100.9		58.7	58.7

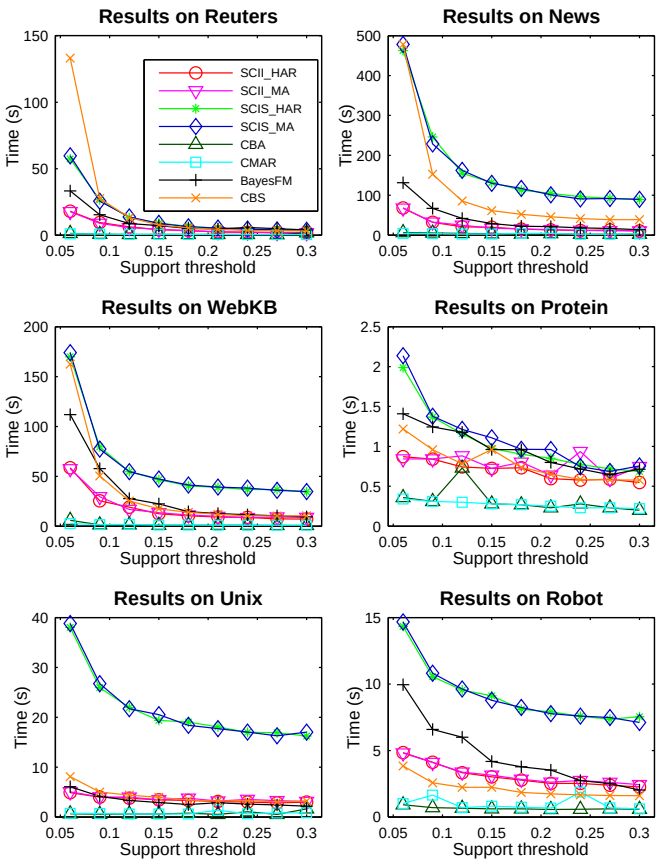

Fig. 4. The impact of the support threshold on runtime.

TABLE 8
Average Number of Patterns at Varying Support Thresholds in *Reuters*

min_sup	SCII	SCIS/CBS	CBA/CMAR	BayesFM
0.06	12385.0	93774.9	9596.1	5324.4
0.12	3081.9	15254.3	884.4	731.4
0.18	1242.4	4843.4	219.4	216.3
0.24	622.9	2050.1	84.3	85.3
0.30	361.1	969.9	39.1	39.8

As expected, for most datasets, the number of rules tends to decrease as the minimum support, minimum confidence or minimum interestingness threshold increases. When the thresholds increase, the number of rules that can be used decreases, and, thus, the classifiers become more concise. If a threshold increases too much, some useful rules will be eliminated and the performance of classifiers decreases. As shown in Figs. 1, 2 and 3, the SCIP classifiers are relatively stable at different thresholds. Therefore, the user just needs to set the thresholds to generate enough rules for classification, e.g., about 300 generated rules in the datasets we used. Note that the minimum interestingness threshold is normally set smaller than the minimum support threshold since the interestingness of a pattern is always smaller than or equal to the support of a pattern as defined in Section 3.1.

5.2.3 Scalability of Different Methods

Fig. 4 shows the runtimes of the algorithms with various support thresholds where *min_conf* is fixed at 0.5 and *min_int* for SCIP classifiers is set to 0.05. The results show that, as the *min_sup* value increases, the runtimes of all methods decrease. This is because a larger *min_sup* value will reduce the number of candidate patterns that need to be processed, as shown in Table 8. We find that sequence based classifiers are more time consuming though BayesFM is faster since it mines patterns from the whole dataset while others mine patterns from each class. As expected, CMAR and CBA, which take no sequential information about the itemsets into account, are the fastest, but as was already seen in Table 4, their accuracy was unsatisfactory.

Fig. 5 shows the runtimes of classifiers for a varying number of data objects. We start off by using just 10% of the dataset, adding another 10% in each subsequent experiment. In this experiment we set $min\_int = 0.06$ for the SCIP classifiers, $min\_sup = 0.1$ and $min\_conf = 0.5$ for all classifiers. For the eight classifiers, the runtime grows similarly, with sequence based classifiers (i.e., SCIS, CBS, and BayesFM) the slowest and the least stable, and the classifiers that take no sequential information into account the fastest. On the *Reuters* and *Protein* datasets, there is actually a drop in runtime from 10% of the dataset to 20% of the dataset. This occurs because the algorithm generates many more patterns when using only 10% of the dataset, as can be seen in Table 9. We find that the runtimes increase with larger dataset size, but the number of frequent patterns do not increase at the same time. That is because the shortest interval computation of a pattern has to be done in each

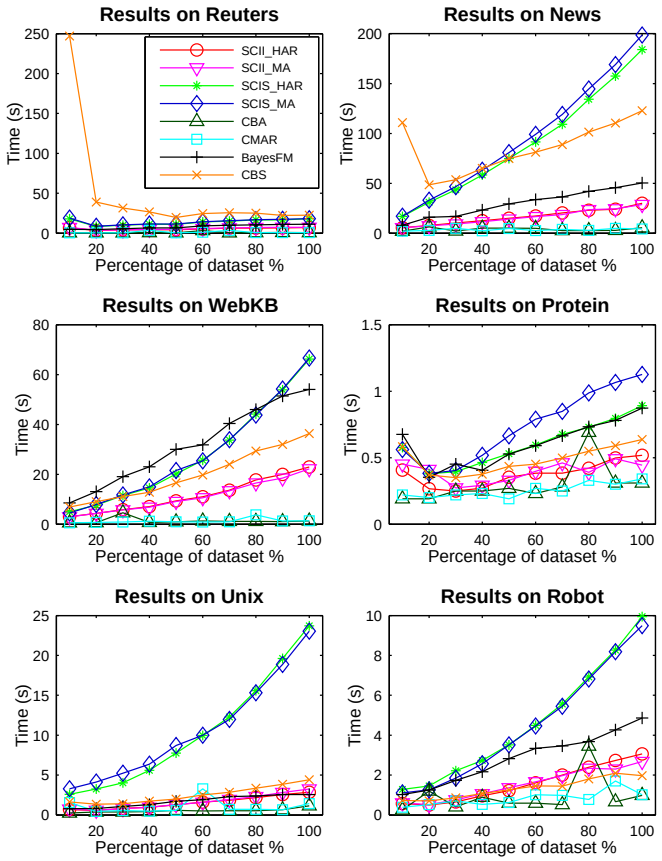

Fig. 5. The impact of the dataset size on runtime.

TABLE 9
Average Number of Patterns at Varying Dataset Sizes (*Reuters*)

Size	SCII	SCIS/CBS	CBA/CMAR	BayesFM
10%	13369.4	182889.4	2450.1	1888.8
30%	6530.1	45753.7	2454.6	1746.2
50%	5080.7	31598.0	1912.9	1347.7
70%	5234.6	31439.6	2071.5	1431.5
90%	4726.6	26641.9	1766.9	1261.7

sequence containing the pattern, and there will be more such sequences when the number of data objects increases.

5.3 Comparison of Learning Based Classifiers

To demonstrate that our method is effective in finding informative patterns to represent sequences, four representative classifiers, Naïve Bayes (NB), k nearest neighbours (KNN), decision trees (C4.5), and support vector machines (SVM) are used in our experiments.

Firstly, the mined patterns from a dataset are collected to construct a feature set. Secondly, each sequence s in the dataset is represented using the constructed feature set by a feature vector $(n_1(s), \ldots, n_m(s))$. We evaluate two policies to set the value of $n_i(s)$.

1. Method_P: presence-weighted feature vector. $n_i(s) = 1$ if and only if feature f_i occurs in s, otherwise $n_i(s) = 0$.

2. Method_C: cohesion-weighted feature vector. In order to investigate whether cohesion information could result in a higher performance of learning algorithms, we set $n_i(s)$ to the cohesion of feature f_i in s ($C(f_i, s)$) if and only if feature f_i occurs in s, otherwise $n_i(s) = 0$.

Note that some existing methods have already used frequent patterns (itemsets and subsequences) to represent a data object for classification [8], [9], [37]. Here, we investigate how the interesting patterns we discover compare to the traditional patterns such as frequent itemsets or sequential patterns, and, in addition, we evaluate the novel approach of using the cohesion of a pattern as a feature value to boost classifier performance. We implement the following feature vector representation approaches to convert each sequence to a feature vector based on different kinds of patterns mined from the dataset.

- Interesting pattern based methods. SCIP-RG provides a scalable approach for finding interesting patterns. Note that the pattern mentioned here could be an itemset or a sequence, so we get four SCIP feature vector representation approaches here (SCII_P and SCIS_P using Method_P, SCII_C and SCIS_C using Method_C).
- Frequent itemset based methods. SET_P: mine all frequent itemsets and construct a feature set. Then represent each sequence as a feature vector as described in Method_P. SET_C: set $n_i(s)$ to the cohesion of a feature f_i in s as described in Method_C.
- Frequent subsequence based methods. SEQ_P: mine all frequent subsequences and get a feature set of frequent subsequences. Then convert each sequence to a feature vector as described in Method_P. SEQ_C: set $n_i(s)$ to the cohesion of a feature as described in Method_C.
- TFIDF: short for term frequency-inverse document frequency. The TFIDF value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus. Then, each sequence (document) is represented as a TFIDF-weighted word vector [38]. Due to the large number of items (words), we use only those items which occur in at least 5% of the whole sequences.

In all experiments, the default parameter settings are as follows. Minimum support threshold *min_sup* is set to 0.05 for all approaches. For the SCIP based approaches *min_int* is set to 0.02. Some of the learning classifiers also use a number of additional parameters, which we set to the default values used in WEKA [35] (implementations of KNN, C4.5 and SVM in WEKA are IBk, J48 and SMO, respectively), for example, $k = 1$ for KNN.

Table 10 reports the performance of the proposed methods by using a variety of learning algorithms. Column AVG reports the average accuracy of the SCIP rule based classifiers from Table 4 for the highest-scoring *max_size* for a given dataset. The results show that the proposed approaches based on pattern features achieve a higher level of accuracy than the TFIDF approach, while SVM generally outperforms other classifiers. We used the SVM results to obtain the average ranks reported at the bottom of the table, which show that the classifiers usually perform better when using the cohesion of a feature as the feature value. Furthermore, we performed the pair-wise sign test to compare SCII_P

TABLE 10
Predictive Accuracy of Learning Algorithms(%)

Dataset	Classifier	SCII_P	SCII_C	SCIS_P	SCIS_C	SET_P	SET_C	SEQ_P (random)	SEQ_C (random)	TFIDF	AVG
	NB	**88.12**	87.43	87.52	87.43	78.42	82.67	80.00 (76.34)	76.63 (76.42)	84.75	
	KNN	90.99	92.67	90.99	**93.47**	90.00	88.51	89.60 (75.84)	89.60 (67.43)	86.63	
Reuters	C4.5	89.21	89.90	90.40	89.90	**90.79**	90.50	89.21 (82.18)	90.20 (80.99)	90.30	95.54
	SVM	96.34	**96.63**	96.14	96.24	95.54	**96.63**	95.74 (81.49)	96.34 (80.69)	94.55	
	NB	72.75	65.17	65.13	58.12	56.95	**73.27**	48.77 (46.02)	62.68 (54.80)	62.12	
	KNN	53.68	72.97	56.75	**73.63**	61.98	70.42	59.71 (52.23)	69.51 (46.50)	59.26	
News	C4.5	86.39	86.43	86.11	**86.56**	85.63	85.43	85.03 (63.26)	85.83 (62.58)	86.33	93.85
	SVM	85.85	**88.08**	85.45	87.78	84.18	88.06	83.72 (64.27)	87.74 (63.50)	80.89	
	NB	**77.86**	76.13	77.65	75.40	73.21	73.91	67.69 (65.66)	62.87 (68.90)	67.14	
	KNN	80.41	**82.71**	81.27	82.54	78.89	77.00	80.24 (75.40)	77.92 (65.82)	78.38	
WebKB	C4.5	87.01	87.60	**87.85**	86.85	85.87	87.33	87.42 (79.65)	87.47 (79.57)	86.96	89.82
	SVM	90.85	91.39	90.99	91.64	90.91	92.50	91.75 (84.11)	**92.69** (82.11)	87.90	
	NB	87.17	87.55	93.87	93.49	84.94	91.26	**94.05** (94.05)	93.31 (93.12)	84.75	
	KNN	87.92	89.96	91.82	92.38	85.69	89.22	92.01 (92.01)	**92.57** (92.01)	86.63	
Protein	C4.5	89.96	93.12	93.49	92.19	90.33	**93.87**	93.31 (93.12)	92.19 (92.19)	90.30	93.15
	SVM	89.03	92.19	95.91	95.91	89.96	92.75	96.10 (95.72)	**96.84** (95.91)	94.55	
	NB	77.92	**79.82**	65.44	73.74	74.10	63.78	61.88 (60.38)	73.45 (64.49)	74.71	
	KNN	91.75	**93.79**	83.52	84.45	84.76	85.36	83.33 (78.82)	83.94 (79.42)	83.99	
Unix	C4.5	86.84	**88.66**	86.60	86.68	86.35	86.33	86.13 (80.68)	86.79 (81.41)	86.60	88.12
	SVM	86.33	**88.96**	87.74	87.96	87.12	87.12	87.23 (81.78)	87.65 (82.40)	79.95	
	NB	78.43	81.80	81.54	83.59	77.55	79.85	81.33 (81.26)	**85.26** (84.26)	78.27	
	KNN	92.03	94.03	**94.98**	94.79	92.49	93.79	94.61 (93.93)	94.86 (94.68)	92.68	
Robot	C4.5	87.59	89.01	88.96	**91.45**	87.63	89.35	89.12 (89.12)	91.35 (90.49)	87.98	82.56
	SVM	87.42	90.73	90.96	91.70	86.96	89.63	91.72 (90.42)	**94.12** (91.42)	83.66	
avg rank (SVM)		6.75	3.42	4.75	3.42	7.42	4.00	4.50	**2.42**	8.33	

with SCII_C, SCIS_P with SCIS_C, SET_P with SET_C, and SEQ_P with SEQ_C, and we found that Method_C performed statistically significantly better than Method_P in all four cases (at the level of 0.05).

Finally, we note that the SCIP feature vector representation approaches achieve comparable or better performance than other approaches with much fewer features, e.g., there are more than 8000 features for the SEQ methods while only 483 features for the SCIS methods on the *Reuters* dataset. In a further experiment with the SEQ methods, we randomly picked a number of features equal to that used by the SCIS variant, which showed that the accuracy of the SEQ methods (reported in brackets) always either dropped or stayed the same when using fewer features.

Compared with the accuracies reported in Table 4, we find that the best accuracy achieved by learning based classifiers for each dataset is generally higher than that of the rule based classifiers, except on the *News* dataset. However, our SCIP rule based classifiers mostly produce better results than NB, KNN and C4.5, and have comparable accuracy to SVM, except on the *Robot* dataset.

6 Conclusions

In this paper, we introduce a sequence classification method based on interesting patterns named SCIP. We present four concrete classifiers, using various pattern types and classification strategies. Through experimental evaluation, we show that the SCIP rule based classifiers in most cases provide higher classification accuracy compared to existing methods. The experimental results show that SCIP is not overly sensitive to the setting of a minimum support threshold or a minimum confidence threshold. In addition, the SCIP method proved to be scalable, with runtimes dependent on the minimum support threshold and the number of data objects. What is more, by using the discovered patterns as input for a number of learning based classification algorithms, we demonstrate that our pattern mining method is effective in finding informative patterns to represent the sequences, leading to classification accuracy that is in most cases higher than the baselines. Therefore, we can conclude that SCIP is not only an effective and stable method for classifying sequence data, but also that its first, pattern mining, step provides a valuable tool for discovering representative patterns.

In future work, we intend to explore a more general setting where several events may sometimes occur at the same time stamp. Additionally, we will attempt to reduce the number of user-chosen parameters that are currently needed by the classifiers.

Acknowledgments

Cheng Zhou is financially supported by the China Scholarship Council (CSC). Boris Cule is partially supported by the SPICES project funded by Innoviris.

References

[1] C. Zhou, B. Cule, and B. Goethals, "Itemset based sequence classification," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 353–368.

[2] B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rule mining," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 80–86.

[3] N. Lesh, M. J. Zaki, and M. Ogihara, "Scalable feature mining for sequential data," *IEEE Intell. Syst.*, vol. 15, no. 2, pp. 48–56, 2000.

[4] V. S. Tseng and C.-H. Lee, "Effective temporal data classification by integrating sequential pattern mining and probabilistic induction," *Expert Systems with Applications*, vol. 36, no. 5, pp. 9524–9532, 2009.

[5] B. Cule, B. Goethals, and C. Robardet, "A new constraint for mining sets in sequences," in *Proceedings of the SIAM International Conference on Data Mining*, 2009, pp. 317–328.

[6] J. Wang and G. Karypis, "Harmony: Efficiently mining the best rules for classification," in *Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2005, pp. 205–216.
[7] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *ACM SIGKDD Explorations Newsletter*, vol. 12, no. 1, pp. 40–48, 2010.
[8] T. Quack, V. Ferrari, B. Leibe, and L. Van Gool, "Efficient mining of frequent and distinctive feature configurations," in *Proceedings of the 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.
[9] H. Cheng, X. Yan, J. Han, and C.-W. Hsu, "Discriminative frequent pattern analysis for effective classification," in *Proceedings of the 23rd International Conference on Data Engineering*. IEEE, 2007, pp. 716–725.
[10] S. Nowozin, G. Bakir, and K. Tsuda, "Discriminative subsequence mining for action classification," in *Proceedings of the 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.
[11] M. J. Zaki, C. D. Carothers, and B. K. Szymanski, "Vogue: A variable order hidden markov model with duration based on frequent sequence mining," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 1, p. 5, 2010.
[12] H. T. Lam, F. Moerchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Statistical Analysis and Data Mining*, vol. 7, no. 1, pp. 34–52, 2014.
[13] G. Dafé, A. Veloso, M. Zaki, and W. Meira Jr, "Learning sequential classifiers from long and noisy discrete-event sequences efficiently," *Data Mining and Knowledge Discovery*, pp. 1–24, 2014.
[14] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers, 1994, pp. 487–499.
[15] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM SIGMOD Record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.
[16] B. Liu, Y. Ma, and C.-K. Wong, "Classification using association rules: weaknesses and enhancements," in *Data mining for scientific and engineering applications*. Springer, 2001, pp. 591–605.
[17] W. Li, J. Han, and J. Pei, "Cmar: Accurate and efficient classification based on multiple class-association rules," in *Proceedings of the 2001 IEEE International Conference on Data Mining*. IEEE Computer Society, 2001, pp. 369–376.
[18] X. Yin and J. Han, "Cpar: Classification based on predictive association rules," in *Proceedings of the SIAM International Conference on Data Mining*, 2003, pp. 331–335.
[19] G. Chen, H. Liu, L. Yu, Q. Wei, and X. Zhang, "A new approach to classification based on association rule mining," *Decision Support Systems*, vol. 42, no. 2, pp. 674–689, 2006.
[20] Y.-L. Chen and L. T.-H. Hung, "Using decision trees to summarize associative classification rules," *Expert Systems with Applications*, vol. 36, no. 2, pp. 2338–2351, 2009.
[21] S. Zhao, E. C. Tsang, D. Chen, and X. Wang, "Building a rule-based classifierła fuzzy-rough set approach," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 5, pp. 624–638, 2010.
[22] X. Zhang, G. Chen, and Q. Wei, "Building a highly-compact and accurate associative classifier," *Applied Intelligence*, vol. 34, no. 1, pp. 74–86, 2011.
[23] L. T. Nguyen, B. Vo, T.-P. Hong, and H. C. Thanh, "Classification based on association rules: A lattice-based approach," *Expert Systems with Applications*, vol. 39, no. 13, pp. 11 357–11 366, 2012.
[24] H. Deng, G. Runger, E. Tuv, and W. Bannister, "Cbc: An associative classifier with a small number of rules," *Decision Support Systems*, vol. 59, pp. 163–170, 2014.
[25] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the 11th International Conference on Data Engineering*, 1995, pp. 3–14.
[26] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proceedings of the 5th International Conference on Extending Database Technology*. Springer-Verlag, 1996, pp. 3–17.
[27] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, no. 1-2, pp. 31–60, 2001.
[28] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 11, pp. 1424–1440, 2004.
[29] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002, pp. 429–435.
[30] T. P. Exarchos, M. G. Tsipouras, C. Papaloukas, and D. I. Fotiadis, "A two-stage methodology for sequence classification based on sequential pattern mining and optimization," *Data & Knowledge Engineering*, vol. 66, no. 3, pp. 467–487, 2008.
[31] D. Fradkin and F. Mörchen, "Mining sequential patterns for classification," *Knowledge and Information Systems*, pp. 1–19, 2015.
[32] T. P. Exarchos, C. Papaloukas, C. Lampros, and D. I. Fotiadis, "Mining sequential patterns for protein fold recognition," *Journal of Biomedical Informatics*, vol. 41, no. 1, pp. 165–179, 2008.
[33] Y. Zhao, H. Zhang, S. Wu, J. Pei, L. Cao, C. Zhang, and H. Bohlscheid, "Debt detection in social security by sequence classification using both positive and negative patterns," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*. Springer-Verlag, 2009, pp. 648–663.
[34] T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 295–331, 1999.
[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
[36] Y. Yang, "An evaluation of statistical approaches to text categorization," *Information retrieval*, vol. 1, no. 1-2, pp. 69–90, 1999.
[37] S. Matsumoto, H. Takamura, and M. Okumura, "Sentiment classification using word sub-sequences and dependency sub-trees," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2005, pp. 301–311.
[38] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *The Journal of Machine Learning Research*, vol. 2, pp. 45–66, 2002.

Cheng Zhou is a post-doctoral researcher at the University of Antwerp in Belgium, where he received his Ph.D. in Computer Science in 2015. Prior to that, he received a Master degree in Management of Information Systems from the National University of Defense Technology, China, in 2011. His research interests include data mining and its applications.

Boris Cule is a post-doctoral researcher currently affiliated with the University of Antwerp and Université Libre de Bruxelles in Belgium. He obtained his Ph.D. in Computer Science in 2012, following a Master in Mathematics degree obtained in 2007. His very first paper introduced the cohesion measure for evaluating the quality of itemsets, which has since been successfully applied to various problem settings in a wide range of domains.

Bart Goethals is a professor at the Department of Mathematics and Computer Science of the University of Antwerp in Belgium. His primary research interests are the study of data mining techniques to efficiently find interesting patterns and properties in large databases. He received the IEEE ICDM 2001 Best Paper Award and the PKDD 2002 Best Paper Award for his theoretical studies on frequent itemset mining. He is associate editor of the Data Mining and Knowledge Discovery journal and the IEEE TKDE journal, and served as program chair of ECML PKDD 2008 and SIAM DM 2010, as well as general chair of the IEEE ICDM 2012 conference.