Efficient Sequence Generator Mining and its Application in Classification

Chuancong Gao[1], Jianyong Wang[2], Yukai He[3] and Lizhu Zhou[4]
Tsinghua University, Beijing 100084, China
{gaocc07[1], heyk05[3]}@mails.tsinghua.edu.cn, {jianyong[2], dcszlz[4]}@tsinghua.edu.cn

ABSTRACT

Sequential pattern mining has drawn much attention and has been recently applied to classify text data, which can be viewed naturally as sequences of words. In this paper we introduce a new problem formulation of sequential pattern mining, namely, frequent sequence generator mining, which seems more meaningful from the classification point of view according to the Minimum Description Length principle, and devise a novel algorithm, *FEAT*, for mining all frequent sequence generators. Two effective search space pruning techniques and an efficient generator checking scheme are proposed to enhance the efficiency of the algorithm. We then propose a sequence generator based classification framework, including feature selection and model construction, and discuss some typical applications of subsequence pattern-based classification models. In order to further improve the efficiency of sequence generator based classification model construction, we devise another algorithm, *seqHarmony*, by introducing a new pruning strategy into *FEAT*, which can mine high quality instance-centric classification rules directly. We also present extensive performance study to evaluate the efficiency and scalability of the algorithms, the effectiveness of the pruning techniques, the classification quality of the sequence generator based classification model for detecting erroneous sentences and classifying consumer product reviews, in comparison with all sequential pattern based model and closed sequential pattern based model.

1 INTRODUCTION

As a well studied data mining task, sequential pattern mining has drawn much attention in recent years. This is partly because it has been shown very useful in several applications such as classifying sequence data [21], detecting erroneous sentences [23], identifying comparative sentences from web forum posting and product reviews [12], semi-structured data management [**?**], and Web log data analysis [6]. Since its introduction in [1], various sequential pattern mining algorithms have been developed, and most of these algorithms fall into two main categories: all sequential pattern mining and closed sequential pattern mining. Typical sequential pattern mining algorithms include *GSP* [22], *SPADE* [28], *PrefixSpan* [19], and *SPAM* [2], while *CloSpan* [27] and *BIDE* [24] are two well-known closed sequential pattern mining algorithms.

If we divide the set of all sequential patterns into a set of equivalence classes, where each equivalence class contains a set of sequential patterns (i.e., frequent subsequence patterns) which are supported by the same set of input sequences, the closed sequential patterns are those maximal ones in each equivalence class. It is evident that the set of closed sequential patterns is just a subset of all sequential patterns, and thus it is possible to identify some parts of search space which are unpromising to generate any closed patterns and can be pruned. Thus, closed sequential pattern mining can be potentially more efficient than all sequential pattern mining. In each equivalence class of sequential patterns, if we call the minimal ones sequence generators, similarly we get that the set of all sequence generators is a subset of all sequential patterns, and sequence generator mining can be potentially more efficient than all sequential pattern mining too. It is also evident that the average size (i.e., number of events) of sequence generators tends to be smaller than that of all sequential patterns (or closed sequential patterns).

As we introduced above, one of the important applications of sequential pattern mining is to classify sequence data. According to the Minimum Description Length (MDL) principle, generators are preferable in tasks like inductive inference and classification among the three types of sequential patterns (namely, all sequential patterns, closed sequential patterns, and frequent sequence generators) [14]. In this paper, we introduce an efficient frequent sequence generator mining algorithm *FEAT* (abbr. Frequent sEquence GenerATor Miner). In this algorithm we adopt the pattern-growth enumeration framework [19], and propose two novel pruning strategies, forward pruning and backward pruning, and an efficient generator checking scheme. To demonstrate the usefulness of sequence generator mining, we propose a sequence generator based classification framework, including data preprocessing, feature selection, and classification model construction. We apply the classification model to two typical categories of problems, erroneous sentence detection and consumer product review classification, and compare its classification accuracy with all sequential pattern based model and closed sequential pattern based model. Since the feature selection stage often takes a lot of time, inspired by the HARMONY algorithm, a rule-based classifier first proposed in [26] for transaction database, we further devise a new algorithm called *seqHarmony*, which mines at most η ($\eta \geq 1$) highest confidence covering rules for each input sequence instance directly. Our extensive performance study shows that *FEAT* is scalable and is more efficient than the state-of-the-art algorithms, sequence generators are very promising to be used in classifying sequence data, and *seqHarmony* is very efficient in mining sequence generator based classification rules.

The contributions of the paper are summarized as follows:

- We propose an efficient frequent sequence generator mining algorithm, *FEAT*. Several effective optimization techniques and new pruning strategies are proposed to enhance the algorithm efficiency.
- We introduce a sequence generator based classification framework, including data preprocessing, feature selection, and classification model construction.

- We devise an instance-centric classification rule mining algorithm, *seqHarmony*, to directly mine sequence generator based classification rules from sequence data.
- We present an extensive performance study for various classification models built from all sequential patterns, closed sequential patterns, and sequence generators, respectively. To our best knowledge, this is the first attempt to systematically compare classification models built from the three types of sequential patterns.

The rest of the paper is organized as follows. Section 2 describes the problem formulation. Section 3 introduces the related work. Section 4 presents the *FEAT* algorithm, with emphasis on the forward and backward pruning strategies, and the generator checking scheme. Section 5 introduces subsequence pattern based classification framework. Section 6 introduces the *seqHarmony* algorithm. The performance study of algorithms *FEAT* and *seqHarmony*, and the classification models is demonstrated and discussed in Section 7. Finally, Section 8 concludes the paper.

2 PROBLEM FORMULATION

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set of distinct items. A **sequence** S can be defined as a list of items (i.e., single-item events[1]) in I, where each item can occur multiple times, denoted by $S =< e_1, e_2, \cdots, e_m >$, or $S = e_1e_2 \ldots e_m$ for short. A sequence $S_a = a_1a_2 \ldots a_n$ is said to be **contained** in another sequence $S_b = b_1b_2 \ldots b_m$ if and only if there exist integers $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ such that $a_1 = b_{i_1}, a_2 = b_{i_2}, \cdots, a_n = b_{i_n}$. S_a is called a **subsequence** of S_b (or S_b is called a **super sequence** of S_a) if S_a is contained in S_b, which can be denoted by $S_a \sqsubseteq S_b$ (or $S_a \sqsubset S_b$ if $S_a \neq S_b$).

A **sequence database** *SDB* is a set of input sequences. The number of input sequences in *SDB* is called the **base size** of *SDB*, denoted by $|SDB|$. The **absolute support** of a sequence S in a sequence database *SDB* is the number of input sequences in *SDB* which contain S, denoted by sup_S^{SDB} or simply by sup_S if the context is clear; the **relative support** of S is the percentage of the input sequences in *SDB* which contain S, that is, $sup_S/|SDB|$. In the rest of the paper, both of them are used exchangeably if it is clear in the context. Given a user specified **minimum support** threshold *min_sup*, a subsequence S_a is said to be **frequent** in *SDB* if $sup_{S_a} \geq min\_sup$.

Among all non-empty frequent subsequences, those contained in the same set of input sequences in *SDB* form an **equivalence class**. In an equivalence class, the relation of "subsequence-of" forms a partial order. With respect to this partial order, the maximal subsequences are called **closed subsequence**s, while the minimal ones are called **sequence generator**s. In another word, a non-empty subsequence S_a is said to be closed if and only if $\nexists S_b$ such that $sup_{S_a} = sup_{S_b}$ and $S_b \sqsupset S_a$, while S_a is called a sequence generator if and only if $\nexists S_b$ such that $sup_{S_a} = sup_{S_b}$ and $S_b \sqsubset S_a$.

The main goal of this paper is *to devise an efficient algorithm for mining all the frequent sequence generators and evaluate its utility in sequence data classification.*

Example 2.1. Table 1 shows an example of a sequence database. The database has a total of 3 distinct items and 4 input sequences. Suppose the absolute minimum support *min_sup*=2, we can find

Table 1: An example sequence database

Sequence ID	Sequence
1	C A A B C
2	A B C B
3	C A B C
4	A B B C A

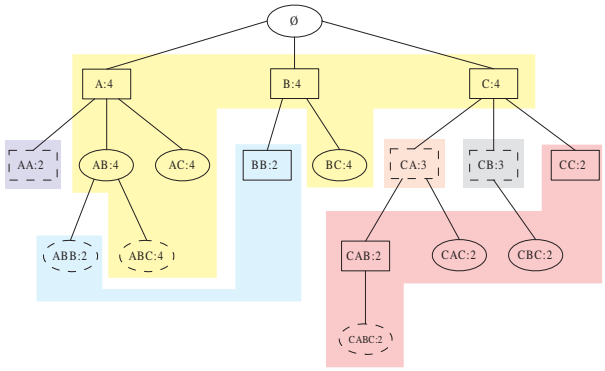

Figure 1: Frequent subsequences and equivalence classes in lexicographical order. (*min_sup* = 2)

17 frequent subsequences and organize them lexicographically in a sequence-tree structure as shown in Figure 1. Each node represents a frequent subsequence and the number after the colon in each node is the absolute support of the corresponding subsequence. Each colored area indicates an equivalence class. Among the 17 frequent subsequences, there are 6 closed subsequences as shown in the nodes with dotted borders (either eclipses or rectangles), and 9 generators as shown in the nodes within (either solid or dotted) rectangles. It can be seen that there can be more than one sequence generator in an equivalence class and a subsequence can be both a sequence generator and a closed subsequence[2]. □

3 RELATED WORK

The sequential pattern mining problem was first proposed in [1], and an improved algorithm, called Generalized Sequential Patterns [22], was later proposed. Since then, several efficient sequential pattern mining algorithms were developed for performance improvement, and typical examples include *SPADE* [28], *PrefixSpan* [19] and *SPAM* [2]. These algorithms mine the complete set of frequent subsequences, which may not be feasible (or necessary) for large datasets. Inspired by frequent closed itemset mining [17] [18] [29] [25], two representative closed sequential pattern mining algorithms, *CloSpan* [27] and *BIDE* [24], were proposed.

Sequential pattern mining has also shown many applications. Especially, frequent subsequences have been used as features for sequence data classification. [21] used frequent subsequence-based method to effectively predict outer membrane proteins, which outperforms the state-of-the-art methods in biological domain. [12] used frequent subsequences with multiple minimal supports for

[1] In this paper we only study the sequences of single-item events.

[2] Actually there can be multiple closed subsequences in an equivalence class either.

identifying comparative sentences. [23] devised some sequential pattern based classification models in order to detect erroneous sentences.

Usually a well-designed closed sequential pattern mining algorithm can remove some redundant patterns without loss of any information, and can be more efficient in many cases by pruning some unpromising parts of search space. Similar to closed sequential patterns, the set of frequent subsequence generators is more concise than the set of all sequential patterns. The generator pattern mining problem was first studied in the frequent item-set mining setting [5] [3]. [4] gave several important properties on itemset generators. While [14] proposed a algorithm called $Gr-Growth$ to help directly mine itemset generators. Their latter work in [15] shows a new algorithm called *DPMine* which could mine closed itemsets and itemset generators simultaneously together with their equivalence class representations.

One important application of frequent pattern mining is feature selection for building classification models. Currently there are several pieces of work which try to directly mine a set of itemset patterns for classification. [13] proposed a algorithm called DeEPS which could predict a class label by compactly summarizing the frequencies of discovered patterns based on a view to collectively maximizing the discriminating power o patterns, and the top-ranked patterns can help assess the importance of discovered patterns. [9] proposes another algorithm to mine top-K associative classification rules on gene data. The HARMONY algorithm [26] tries to directly mine k best rules for each transaction, and use them for building a rule-based classifier. [7] proves that information gain should be preferred to confidence in mining classification rules, and proposes an algorithm using information gain to select rules. While [8] devises an algorithm called DDPMine to directly mine rules using a sequential covering paradigm. There is no algorithm which directly mines a set of itemset generators for classification.

A two page poster paper presents a preliminary version of our sequence generator mining algorithm, *FEAT*, in [11]. In this paper, we provide more details to the *FEAT* algorithm and introduce a sequence generator based classification framework. The effectiveness of the subsequence pattern based classification model is also evaluated in comparison with the all sequential pattern based model and closed sequential pattern based model within two typical application domains, *erroneous sentence detection* and *consumer product review classification*. This is the first attempt to systematically compare the classification models with feature selection from three types of sequential patterns, namely, all sequential patterns, closed sequential patterns, and sequence generators. Furthermore, inspired by the instance-centric rule-based HARMONY classification algorithm [26], we devise a new algorithm called *seqHarmony*, which mines directly at most η highest confidence covering rules for each input sequence instance, and achieves better efficiency than the *FEAT* algorithm. In addition, in a parallel work of [16], a new algorithm called *GenMiner* was introduced during the same time frame for mining sequence generators and ranking them for further applications. Our performance study shows that algorithm *FEAT* is more efficient than *GenMiner*.

4 FEAT: AN EFFICIENT SEQUENCE GENERATOR MINING ALGORITHM

A naïve approach to mining the complete set of frequent sequence generators is to first apply a traditional sequential pattern mining algorithm to find the set of frequent subsequences, from which the set of frequent sequence generators can be further computed. However, it is inefficient as it cannot prune the unpromising parts of search space. In this section we propose two novel pruning methods, **Forward Prune** and **Backward Prune**, which can be integrated with the pattern-growth enumeration framework [19] to speed up the mining process. We also present an efficient generator checking scheme and the integrated sequence generator mining algorithm.

4.1 Sequence Enumeration Framework

FEAT algorithm adopts the pattern-growth enumeration framework to enumerate all frequent subsequences [19], which has been popularly used in many previous pattern discovery algorithms. It combines the divide-and-conquer and depth-first-search paradigm. Take Figure 1 as an example, the pattern-growth enumeration framework will first mine frequent subsequences with prefix A, then mine frequent subsequences with prefix B, finally mine frequent subsequences with prefix C. In mining frequent subsequences with prefix A, it will first mine frequent subsequences with prefix AA, then mine frequent subsequences with prefix AB, finally mine frequent subsequences with prefix AC. Thus, all the frequent subsequences in this running example will be discovered in the following order: A:4, AA:2, AB:4, ABB:2, ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2, CABC:2, CAC:2, CB:3, CBC:2, and CC:2.

Under the pattern-growth enumeration framework, a projected sequence database is constructed for a given prefix P, and all the locally frequent events can be discovered by scanning the projected database and are used to grow the corresponding prefix subsequence. Following we first give the definitions of **projected sequence** and **projected database**, which form the basis of the pattern-growth enumeration framework, then introduce the new definitions of **the i-th event missing subsequence** and **(i,j)-scope subsequence**, which will be used later.

Definition 4.1. (Projected Sequence) Given an input sequence $S = e_1e_2 \dots e_n$, a prefix subsequence $S_p = e'_1e'_2 \dots e'_i$, the projected sequence of S with respect to S_p is defined as the subsequence of S after the first appearance of S_p. For example, let $S = CAABC$ and $S_p = CA$, the projected sequence of S with respect to S_p is ABC.

Definition 4.2. (Projected Database) Given an input sequence database SDB, a prefix sequence S_p, the projected database SDB_{S_p} of SDB with respect to S_p is defined as the complete set of projected sequences in SDB with respect to S_p. For example, the projected database of prefix sequence AB in our example database as shown in Table 1 is $\{C, CB, C, BCA\}$.

Definition 4.3. (The i-th event missing subsequence) Given sequence $S = e_1e_2 \dots e_n$, we define the i-th event missing subsequence of S as the subsequence derived from S by removing its i-th event, denoted by $S^{(i)} = e_1e_2 \dots e_{i-1}e_{i+1} \dots e_n$. For example, let $S = ABBCA$, we have $S^{(3)} = ABCA$.

Definition 4.4. (The (i,j)-scope subsequence) Given a sequence $S = e_1e_2 \dots e_n$, we define the (i,j)-scope subsequence as $S_{(i,j)} = e_ie_{i+1} \dots e_j$. For example, let $S = ABBCA$, we have $S_{(2,4)} = BBC$.

4.2 Pruning Strategies

As we described above, the set of frequent sequence generators is a subset of all sequential patterns, it is possible that some parts of the search space may not contain any sequence generators, thus can be pruned. In the following we first introduce Theorems 1 and 2 which form the basis of the pruning methods, then describe the two new pruning techniques which can be used to enhance the efficiency of *FEAT* algorithm.

Theorem 4.5. *(SDB Equivalence) Given two sequences S_{p_1} and S_{p_2}, if $S_{p1} \sqsubset S_{p_2}$ (i.e., S_{p_1} is a proper subsequence of S_{p_2}) and $SDB_{S_{p_1}} = SDB_{S_{p_2}}$, then any extension to S_{p_2} cannot be a generator.*[3]

Proof. Assume there exists any subsequence S, which can be used to grow S_{p_2} to get a subsequence S'=$S_{p_2} \diamond s$[4], we can always use S to grow S_{p_1} and get another subsequence $S'' = S_{p_1} \diamond S$. Since $S_{p_1} \sqsubset S_{p_2}$ and $SDB_{S_{p_1}} = SDB_{S_{p_2}}$ hold, we can get $sup_{S'} = sup_{S''}$ and $S'' \sqsubset S'$, thus S' cannot be a generator. □ □

Theorem 4.6. *Given subsequence $S_p = e_1e_2 \dots e_n$ and an item e', if $SDB_{S_p} = SDB_{S_p^{(i)}} (i = 1, 2, \dots, n)$, then we have $SDB_{S_p \diamond e'} = SDB_{S_p^{(i)} \diamond e'}$.*

Proof. It can be directly proved based on the definition of a projected database. □ □

Lemma 4.7. *(Forward Prune) Given subsequence $S_p = e_1e_2 \dots e_n$ and an item e', let $S_p^* = S_p \diamond e'$. If $sup_{S_p} = sup_{S_p^*}$ holds and for any local frequent item u of S_p^* we always have $SDB_{S_p \diamond u} = SDB_{S_p^* \diamond u}$, then S_p^* can be safely pruned.*

Proof. Because $sup_{S_p} = sup_{S_p^*}$, S_p^* cannot be a generator. Furthermore, since for any local frequent item u of S_p^* we always have $SDB_{S_p \diamond u} = SDB_{S_p^* \diamond u}$, $S_p^* \diamond u$ and any extension to $S_p^* \diamond u$ cannot be generators either according to Theorem 4.5. Thus we can safely prune S_p^*. □ □

Lemma 4.8. *(Backward Prune) Given $S_p = e_1e_2 \dots e_n$, if there exists an index i ($i = 1, 2, \cdots, n-1$) and a corresponding index j ($j = i+1, i+2, \cdots, n$) such that $SDB_{(S_p)_{(1,j)}} = SDB_{((S_p)_{(1,j)})^{(i)}}$, then S_p can be safely pruned.*

Proof. Because $SDB_{(S_p)_{(1,j)}} = SDB_{((S_p)_{(1,j)})^{(i)}}$ holds, according to Theorem 4.6 we can infer that $SDB_{S_p} = SDB_{(S_p)^{(i)}}$. Furthermore, as $(S_p)^{(i)} \sqsubset S_p$ also holds, according to Theorem 4.5 we can safely prune S_p. □ □

[3]Note that a similar checking has been adopted in a closed sequential pattern mining algorithm, *CloSpan* [27]. Here we adapted the technique to the setting of sequence generator mining.

[4]Here we use $A \diamond B$ to denote the concatenation of two subsequences of A and B.

4.3 Generator Checking Scheme

The preceding pruning techniques can be used to prune the unpromising parts of search space, but they cannot assure each mined frequent subsequence $S = e_1e_2 \dots e_n$ is a generator. We devise a generator checking scheme as shown in Theorem 4.9 in order to perform this task, and it can be done efficiently during pruning process by checking whether there exists such an index i ($i = 1, 2, \cdots, n$) that $|SDB_S| = \left|SDB_{S^{(i)}}\right|$, as $sup_S = |SDB_S|$ holds.

Theorem 4.9. *A sequence $S = e_1e_2 \dots e_n$ is a generator if and only if $\nexists i$ such that $1 \le i \le n$ and $sup_S = sup_{S^{(i)}}$.*

Proof. It can be easily derived from the definition of a generator and the well-known downward closure property of a sequence.[5] □ □

4.4 Algorithm

By integrating the preceding pruning methods and generator checking scheme with a traditional pattern growth framework [19], we can easily derive the FEAT algorithm as shown in Algorithm 1. Given a prefix sequence S_P, FEAT first finds all its locally frequent items, uses each locally frequent item to grow S_P, and builds the projected database for the new prefix (lines 2,3,4). It adopts both the *forward* and *backward* pruning techniques to prune the unpromising parts of search space (lines 8,12), and uses the generator checking scheme to judge whether the new prefix is a generator (lines 7,9,12,14). Finally, if the new prefix cannot be pruned , FEAT recursively calls itself with the new prefix as its input (lines 17,18).

Algorithm 1: *FEAT*

Input : Prefix sequence S_P, S_P's projected database SDB_{S_p}, minimum support *min_sup*, and result set *FGS*

1 **foreach** *i in localFrequentItems*(SDB_{S_p}, *min_sup*) **do**
2 $S_p^i \leftarrow S_p \diamond i$;
3 $SDB_{S_p^i} \leftarrow projectedDatabase(SDB_{S_p}, S_p^i)$;
4 $canPrune \leftarrow false$;
5 $isGenerator \leftarrow true$;
6 **if** $|SDB_{S_p}| = |SDB_{S_p^i}|$ **then**
7 $canPrune \leftarrow ForwardPrune(S_p, SDB_{S_p}, S_p^i, SDB_{S_p^i})$;
8 $isGenerator \leftarrow false$;
9 **if** *not canPrune* **then**
10 $BackwardPrune(S_p^i, SDB_{S_p^i}, canPrune, isGenerator)$;
11 **if** *isGenerator* **then**
12 $FGS \leftarrow FGS \cup \{S_P^i\}$;
13 **if** *not canPrune* **then**
14 $FEAT(S_P^i, SDB_{S_p^i}, min\_sup, FGS)$;

5 SEQUENCE GENERATOR-BASED CLASSIFICATION

One important application of frequent subsequence patterns is to use them as the features to build accurate classification models in

[5]The downward closure property of a sequence says that given a subsequence p, the support of any subsequence of p is no smaller than that of p.

Algorithm 2: *forwardPrune*

Input : previous prefix sequence S_P, previous projected sequence database SDB_{S_p}, prefix sequence S_P^*, projected sequence database $SDB_{S_p^*}$, minimum support sup_{min}

Output : whether S_P^* can be pruned

```
foreach i in localFrequentItems(SDB_{S_p^*}, sup_min) do
    if projectedDatabase(SDB_{S_p}, S_P ◇ i) ≠
     projectedDatabase(SDB_{S_p^*}, S_P^* ◇ i) then
        return false;
return true;
```

Algorithm 3: *backwardPrune*

Input : prefix sequence S_P, projected sequence database SDB_{S_p}, whether S_P can be pruned, whether S_P is a generator

```
for i ← |S_P| − 1 to 1 do
    for j ← i + 1 to |S_P| do
        S_P^* ← ((S_P)_(1,j))^(i);
        SDB_{S_p^*} ← projectedDatabase(SDB_{S_p^*}, S_P^*);
        S_P^** ← (S_P)_(1,j);
        if |SDB_{S_p^*}| = |SDB_{S_p^**}| then
            isGenerator ← false;
            if SDB_{S_p^*} = SDB_{S_p^**} then
                canPrune ← true;
                return;
```

order to classify sequence data such as protein sequence data and text data. Previous studies have shown that they are very successful for classifying biological data [10] [21], detecting erroneous sentences [23], and identifying comparative sentences from Web forum posting and product reviews [12]. However, to our best knowledge, all these approaches select features from the set of all sequential patterns (i.e., frequent subsequences), but have not ever tried sequence generators and closed sequential patterns. As we introduced in Section 1, both the set of frequent sequence generators and the set of closed sequential patterns are subsets of all sequential patterns, sequence generator mining (or closed sequential pattern mining) can be potentially more efficient than traditional sequential pattern mining. In addition, as the average pattern size of frequent sequence generators tends to be smaller than that of all sequential patterns (or closed sequential patterns), we expect the classification model built from sequence generators is simpler. Another task of this paper is to experimentally compare the classification models built from sequential patterns, closed sequential patterns, and sequence generators, respectively.

In the following, we first present some methods for feature selection from a given set of subsequence patterns (no matter they are sequential patterns, closed sequential patterns, or sequence generators), then give two typical applications of the subsequence-based classification models, and summarize the subsequence pattern based classification framework.

5.1 Subsequence-based Feature Selection

One naïve method for feature selection is to use the complete set of subsequence patterns as the feature set. However, there are some problems with this method. First, the number of subsequence patterns can be huge, which leads to extremely high dimensionality for the transformed data. Although the set of frequent sequence generators (or closed sequential patterns) is more concise than the set of all sequential patterns, the number of generator patterns is still very large. According to our experience, it is not uncommon to mine millions of generator patterns for a dense sequence dataset. Second, the set of subsequence patterns contains redundant patterns. As we have shown in our running example, each equivalence class may contain more than one sequence generators, which have the same power from the classification point of view. In addition, not all the mined subsequence patterns have high quality for building classification models. Thus, an effective feature selection method is crucial and necessary. Below we introduce the feature selection methods adopted in this paper.

Given a labelled training sequence dataset, which contains a total of n input sequences, $\{S_1, S_2, \ldots, S_n\}$, and has k distinct class labels, $\{l_1, l_2, \ldots, l_k\}$, we denote the set of subsequence patterns (e.g., frequent sequence generators) discovered by a certain sequential pattern mining algorithm (e.g., FEAT) with a minimum support threshold *min_sup* by $\{p_1, p_2, \ldots, p_m\}$ (Here m is the total number of subsequence patterns). For each pattern p_i, we can compute k classification rules corresponding to the k class labels respectively. The classification rule R_{ij} built from pattern p_i ($1 \leq i \leq m$) and class label l_j ($1 \leq j \leq k$) has the following form:

$$p_i \rightarrow l_j : sup_{p_i}^{l_j}, conf_{p_i}^{l_j}$$

Where p_i is the rule body, l_j is the rule head, $sup_{p_i}^{l_j}$ is the rule support which equals the number of training input sequences which contain p_i and are labelled with l_j, $conf_{p_i}^{l_j}$ is the rule confidence which is equal to $\frac{sup_{p_i}^{l_j}}{sup_{p_i}}$. If the rule body p_i of classification rule R_{ij} is contained in an input sequence S and the rule head is the same as the class label of S, rule R_{ij} is called a covering rule of input sequence S.

After introducing the above concepts, we now discuss the feature selection methods. The first one is called the **confidence based approach**. Given a user specified minimum confidence threshold *min_conf*, we compute all the classification rules with a confidence no smaller than *min_conf*. The set of rule bodies (i.e., subsequence patterns) of these selected classification rules form the feature set.

The preceding confidence-based approach is simple but has a problem. It accepts as input a user specified minimum confidence threshold *min_conf*. If *min_conf* is too high, some high quality rules can be missed, while if *min_conf* is too low, the feature set may contain some low quality rules. In addition, it does not consider the coverage of training dataset by the selected rules. Therefore, we propose to use the **Instance-Centric approach**, which is inspired by our HARMONY algorithm [26]. In this approach, we compute for each training sequence its η covering rules with top η highest confidences, where η is a small number with a default value of 1 (and usually no greater than 3), the complete set of rule bodies of covering rules selected in this way form the final feature set.

There are potentially other factors which could be used to improve both the **confidence-based** and **Instance-Centric** approaches. A typical example is that we may use the concept of equivalence class to remove redundancy of the selected features, based on the observation that the selected features of the same equivalence class have the same support and confidence, and cover the same set of training sequences. Thus, we may consider only retaining one feature for each equivalence class.

5.2 Typical Applications of Subsequence-based Classification Model

After the feature selection phase is finished, the classification model can be very easily constructed. We use the feature set to convert the training sequence data and build some traditional classifiers such as *Naïve Bayes* and *Support Vector Machine (SVM)*. We evaluate the sequence generator based classification models within two typical application domains. The first one relates to *erroneous sentence detection*. Given a set of native or non-native sentences, we want to detect those non-native sentences automatically using the classification model. The second application is about *consumer product review classification*. There are abundant product reviews on some commercial Web site (e.g., Amazon.com), which are labelled as (or can be transformed to be) either positive or negative. The training datasets used in these two domains all contain a set of sentences of words. Following we introduce how to preprocess data in order to get better classification models.

Data Preprocessing. Firstly, all words in the training dataset are converted to their lowercase form, and the transformed words as well as the punctuations are all viewed as events.

Secondly, there are several options which can be used to further preprocess the data:

- *Choice 1:* Use the *POS (Part Of Speech)* tagging technique [20] to convert all words except those in the list of *function-words* and *time-words*[6] to their *POS* tags.
- *Choice 2:* The words in the *sentimental-word-list*[7] are kept, while all other words are removed. If a sentence becomes empty, a special word *NIL* is added to it.
- *Choice 3:* On the basis of Choice 2, all *degree tag* of the kept words are used for further processing.
- *Choice 4:* Do not perform any further preprocessing.

For example, under the 'Choice 1' preprocessing method, a sentence of *"There are formula and name indexes covering both parts."* may be converted into *"there are NN and NN NNS VBG both NNS."*, where *NN*, *NNS*, and *VBG* are all *POS* tags.

5.3 The Subsequence Pattern based Classification Framework

To give an overview of our approach, here we summarize the entire subsequence pattern based classification framework in Figure 2. It accepts any sequence data (e.g., a text database) as input. After the two-step preprocessing (Note that Pp1, Pp2, Pp3, and Pp4 in Figure 2 represent the four preprocessing options in the second step), it applies any sequential pattern mining algorithm (e.g., *FEAT*, *BIDE*, or *PrefixSpan*) on the training data divided from the whole input data, to get a set of subsequence patterns, which forms the candidate feature set. Then it applies either confidence based or instance centric based approach to do feature selection. The final feature set can be used to convert the dataset and build a classification model (e.g., *SVM* or *Naïve Bayes*). Finally, evaluation results are obtained by applying the classification model on testing data which composes the input data together with training data. Note that we actually adopt 10-fold cross validation to evaluate the model, and Figure 2 only depicts the whole process on one fold of validation.

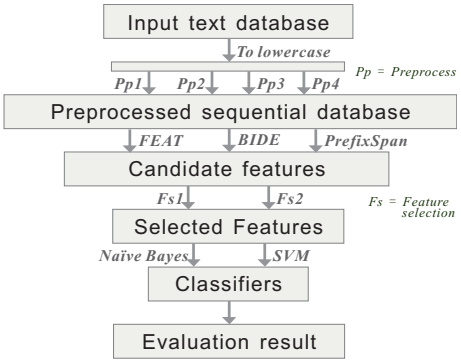

Figure 2: The whole process of the classification

6 SEQHARMONY: AN INSTANCE-CENTRIC RULE MINING ALGORITHM

In the preceding section, we discussed how to build a classification model from the complete set of sequence generators returned by algorithm *FEAT*. One problem with this classification framework is that algorithm FEAT usually generates a huge result set. Both mining the complete set of frequent sequence generators and selecting the features for model construction from this large result set could be very time consuming. In this section we introduce a new algorithm, *seqHarmony*, which is derived from *FEAT*, and mines a set of high quality classification rules directly from the sequence data. *seqHarmony* incorporates some basic ideas from algorithm *HARMONY* [26], and tries to find a set of highest confidence covering classification rules for each instance (i.e., an input sequence). In the following we first introduce the concept of the highest confidence covering rules and show that among the highest confidence covering rules w.r.t. a given input sequence instance, at least one of them must have a rule body which is a sequence generator, then explore how to derive a search space pruning method which can be incorporated into FEAT to accelerate the mining process of η highest confidence covering classification rules for each sequence instance, where $\eta \geq 1$ is a user-specified parameter.

6.1 Highest Confidence Covering Rule

A classification rule discovered from the sequence data has the following form: "$Y \rightarrow c : sup_Y^c, conf_Y^c$", where Y is a subsequence pattern, c is a class label, sup_Y^c is the support of Y associated with a class label c (i.e., the number of input sequences with a class label c which contain Y), and $conf_Y^c$ is defined as $\frac{sup_Y^c}{sup_Y}$. Given a

[6]The list of function-words and time-words is composed of 501 words. They are believed helpful for classification and have been used in a similar way in [23].
[7]This *sentimental-word-list* contains some words related to sentiment, grade, etc.

minimum support threshold, *min_sup*, the subsequence Y is frequent if $sup_Y \geq min\_sup$. A frequent subsequence Y supported by any training sequence instance $< S_i, c >$ ($1 \leq i \leq |SDB|$, and c is a class label) is called a frequent covering subsequence of instance S_i, and "$Y \rightarrow c : sup_Y^c, conf_Y^c$" is called a frequent covering rule of instance S_i. Among the frequent covering rules of an instance S_i, those with the highest confidence are called the Highest Confidence Covering Rules (HCCR) with regard to instance S_i. We denote a highest confidence covering rule with regard to instance S_i by $HCCR_{S_i}$, and use $HCCR_{S_i}^{sup}$ and $HCCR_{S_i}^{conf}$ to denote its support and confidence.

Theorem 6.1. *Given an input sequence S_i, among the highest confidence covering rules regarding S_i (i.e., $HCCR_{S_i}$'s), there must exist at least one highest confidence covering rule such that its rule body is a generator.*

Proof. Given any a highest confidence covering rule of input sequence S_i, $HCCR_{S_i}$, which is of the form of "$Y \rightarrow c : sup_Y^c, conf_Y^c$", there exists an equivalence class of subsequences where subsequence Y belongs to. In this equivalence class, there is at least one subsequence which is a generator, and we denote it by Y'. As both Y' and Y belong to the same equivalence class, Y' must be a subsequence of S_i, and $sup_Y = sup_{Y'}$ and $sup_Y^c = sup_{Y'}^c$ hold, we further have $conf_Y^c = conf_{Y'}^c$. Thus, the classification rule of "$Y' \rightarrow c : sup_{Y'}^c, conf_{Y'}^c$" must be a highest confidence covering rule of input sequence S_i. □ □

From Theorem 6.1 we know that for each input sequence, we can compute at least one highest confidence covering rule for it whose rule body is a sequence generator, while there may also exist some highest confidence covering rules whose rule bodies are not sequence generators. According to the Minimum Description Length principle, the generator based highest confidence covering rules are more preferable from the classification point of view. Following we will explore how to devise a search space pruning method in order to mine the set of generator based highest confidence covering rules directly under the framework of algorithm *FEAT*.

6.2 Unpromising Projected Database Pruning

In algorithm *HARMONY* [26], several pruning techniques are proposed and basically they can be used in mining the sequence generator based highest confidence covering rules. In *seqHarmony*, we extend the **unpromising projected database pruning method** from the itemset setting to the sequence setting. Here we introduce the basic idea of the pruning method, for more details we refer the readers to our *HARMONY* algorithm [26].

Given the user specified support threshold, *min_sup*, and the current prefix sequence Y, let Y^* be a subsequence extended from Y (Y^* can be equal to Y), for any class label c, the confidence of rule "$Y^* \rightarrow c$", $conf_{Y^*}^c$, must satisfy the following equation:

$$conf_{Y^*}^c = \frac{sup_{Y^*}^c}{sup_{Y^*}} \leq \frac{sup_{Y^*}^c}{min\_sup} \leq \frac{sup_Y^c}{min\_sup}$$

In addition, as $conf_{Y^*}^c \leq 1$ holds, we further have $conf_{Y^*}^c \leq \min\left\{1, \frac{sup_Y^c}{min\_sup}\right\}$. Then we formally introduce the unpromising projected database pruning technique in the following lemma.

Lemma 6.2. *(Unpromising projected database pruning) For any input sequence containing subsequence Y, S_i, (i.e., $<S_i, c> \in SDB_Y$, $1 \leq i \leq |SDB_Y|$, and c is class label), if the following equation always holds, the projected database SDB_Y can be safely pruned.*

$$HCCR_{S_i}^{conf} \geq \min\left\{1, \frac{sup_Y^c}{min\_sup}\right\}$$

Proof. Following the above analysis, we can derive that for any subsequence X (X can be $\emptyset$) and $\forall i (1 \leq i \leq |SDB_Y|)$, the following holds:

$$conf_{Y \diamond X}^c \leq HCCR_{S_i}^{conf}$$

This means that any rule mined by growing prefix Y will have a confidence no greater than the currently maintained highest confidence rules of any projected instance in SDB_Y, thus the whole projected database SDB_Y can be safely pruned. □ □

6.3 Algorithm

By incorporating the unpromising projected database pruning technique with algorithm *FEAT*, we get the *seqHarmony* algorithm, which can mine the set of the sequence generator based highest confidence covering classification rules. The algorithm is summarized in Algorithm 4. We see that algorithm *seqHarmony* is very similar to *FEAT*, the main difference is that *seqHarmony* needs to maintain the highest confidence covering rules for each input sequence (line 18) and applies the unpromising projected database pruning method (line 7 and line 10). The final rule set is composed with the union of all coving rules on each instance.

Algorithm 4: *seqHarmony*

Input : Prefix sequence S_P, S_P's projected database SDB_{S_p}, and minimum support *min_sup*

1 **foreach** *i in localFrequentItems*(SDB_{S_p}, min_sup) **do**
2 $S_p^i \leftarrow S_p \diamond i$;
3 $SDB_{S_p^i} \leftarrow projectedDatabase(SDB_{S_p}, S_p^i)$;
4 $canPrune \leftarrow false$;
5 $isGenerator \leftarrow true$;
6 $canPrune \leftarrow UnpromisingDBPrune(S_p, SDB_{S_p})$;
7 **if** $\left|SDB_{S_p}\right| = \left|SDB_{S_p^i}\right|$ **then**
8 $isGenerator \leftarrow false$;
9 **if** *not canPrune* **then**
10 $canPrune \leftarrow forwardPrune(S_p, SDB_{S_p}, S_p^i, SDB_{S_p^i})$;
11 **if** *not canPrune* **then**
12 $backwardPrune(S_p^i, SDB_{S_p^i}, \&canPrune, \&isGenerator)$;
13 **if** *isGenerator* **then**
14 Update *HCCR* for each instance;
15 **if** *not canPrune* **then**
16 $seqHarmony(S_P^i, SDB_{S_p^i}, min\_sup)$;

Table 2: Data Characteristics for *Gazelle*, *ProgramTrace*, and *TCAS* datasets.

Dataset	# seq.	# items	avg. len.	max. len.
Gazelle	29,369	1,423	3	651
ProgramTrace	10	105	488	989
TCAS	1,578	287	61	97

Table 3: Data Characteristics for ESL and product review datasets.

Dataset	# seq.	# $\mathbb{P}$	# $\mathbb{N}$	ave. len.
ESL-C	5765	2879	2886	18.4
ESL-J	31369	15800	15569	13.1
Office07Review	320	240	80	94
VistaReview	597	254	343	234

7 EXPERIMENTAL STUDY

In the following we first present the experimental results to evaluate the performance of algorithm *FEAT*, including the efficiency test in comparison with various sequential pattern mining algorithms (e.g., sequential pattern mining algorithm *PrefixSpan* [19], Closed sequential pattern mining algorithm *BIDE* [24], and frequent sequence generator mining algorithm *GenMiner* [16]), peak memory usage test, the effectiveness of the pruning techniques, scalability test, and efficiency comparison between *FEAT* and *seqHarmony*. Then we evaluate the classification quality of the sequence generator based classification models in comparison with the all sequential pattern based and closed sequential pattern based models.

7.1 Test Environment and Datasets

We conducted the performance study on a computer with Intel Core Duo 2 E6550 CPU and 2GB memory installed. To test the performance of *FEAT*, we used three real datasets. The first dataset, *Gazelle*, is a sparse web click-stream dataset, which contains 29,369 sequences, 87,546 events (i.e., page views), and 1,423 distinct items (i.e., web pages); the second dataset, *ProgramTrace*, is an extremely dense program trace dataset, which contains 105 distinct items and 10 long sequences with a maximal sequence length of 989; the third dataset, *TCAS*, is also a program trace dataset, contains 1,578 sequences and 287 unique items with a maximal sequence length of 97. Table 2 summarizes the characteristics of datasets *Gazelle*, *ProgramTrace*, and *TCAS*, including the number of sequences, number of distinct items, the average sequence length, and the maximum sequence length, respectively.

In order to evaluate the effectiveness of the sequence generator based classification models, we used several real datasets from two application domains, *erroneous sentence detection* and *product review classification*. The datasets for *erroneous sentence detection* are from the writers of English as a Second Language (ESL), all sentences in it are labelled as either "Native"($\mathbb{P}$) or "Non-native"($\mathbb{N}$). There are two ESL datasets, one is from Chinese students and the other is from Japanese students. We denote them by *ESL-C* and *ESL-J*, respectively. A similar set of datasets were previous used in [23]. For the application domain of product review classification, we used two datasets, *Office07Review* and *VistaReview*, which were collected from Amazon.com. In these two product review datasets all the sentences within the quotation marks were removed. The *VistaReview* dataset contains reviews for four different Vista versions. The two ESL datasets and the two product review datasets were preprocessed using the *Choice 1* and *Choice 2* options respectively(See Section 5.2). The characteristics of these datasets are illustrated in Table 3. We also used the *VistaReview* dataset to evaluate the effectiveness of *Unpromising projected database Pruning* technique (namely, efficiency comparison between *FEAT* and *seqHarmony*).

Table 4: Peak Memory Usage (in KB) Comparison

Dataset	*min_sup*	*BIDE*	*GenMiner*	*FEAT*
Gazelle	0.014%	6,584	259,472	146,860
ProgramTrace	80%	892	4,712	2,156
TCAS	5%	249,772	250,160	12,556

7.2 Performance Study of FEAT Algorithm

7.2.1 Efficiency test. We first evaluated the runtime efficiency of *FEAT* in comparison with various sequential pattern mining algorithms, including all sequential pattern mining algorithm *PrefixSpan*, closed sequential pattern mining algorithm *BIDE*, and frequent sequence generator mining algorithm *GenMiner*. Figure 3 demonstrates the comparison results for sparse dataset *Gazelle*. We see that with a high support all four algorithms have similar performance for *Gazelle* dataset, however, once we lower the support threshold further, the sequential pattern mining algorithm *PrefixSpan* runs significantly slower than the closed sequential pattern mining algorithm *BIDE* and the two sequence generator mining algorithms, *FEAT* and *GenMiner*. *BIDE* and *FEAT* show similar performance for this dataset at various support levels. Figure 3 also shows that at a low support, *FEAT* can be orders of magnitude faster than *GenMiner* for sparse dataset *Gazelle*.

ProgramTrace is an extremely dense datasets which contains a few very long sequences. Figure 4 shows the comparison results for this dataset. We see that *PrefixSpan* is always much slower than the other three algorithms, which indicates that for dense datasets mining closed sequential patterns or sequence generators can be much more efficient than all sequential pattern mining. Comparing the two sequence generator mining algorithms, we can see than *FEAT* is always faster than *GenMiner* for this dense dataset.

TCAS dataset is also a dense dataset with a moderate average sequence length. Figure 5 shows the comparison results for *BIDE*, *GenMiner*, and *FEAT*. As *PrefixSpan* is also much slower than the other algorithm, we do not show its curve in the figure. We see that *FEAT* is also faster than *GenMiner*. For example, with a minimum support of 5%, *FEAT* takes about 129 seconds to finish, while *GenMiner* costs about 1415 seconds. Comparing with closed pattern mining algorithm *BIDE*, FEAT is faster at a high support while it is slower at a low support.

We also measured the peak memory usage of the three algorithms *BIDE*, *GenMiner*, and *FEAT*, at the lowest measurable *min_sup* on the three datasets. The results can be found in table 4. We see that *FEAT* always consumes less memory than *GenMiner* for all the three datasets. While *BIDE* uses the least memory for the *Gazelle*

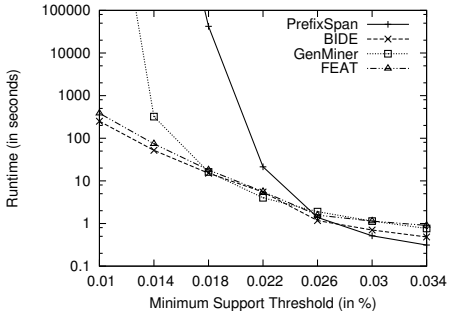

Figure 3: Efficiency Comparison (*Gazelle*)

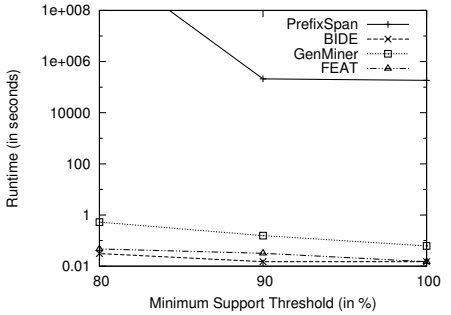

Figure 4: Efficiency Comparison (*ProgramTrace*)

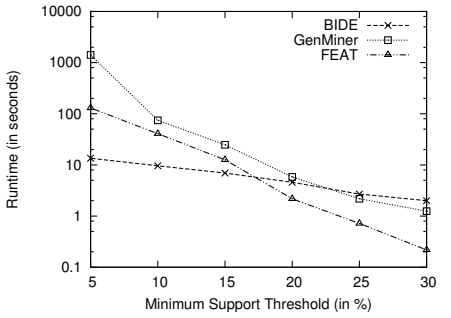

Figure 5: Efficiency Comparison (*TCAS*)

and *ProgramTrace* datasets, *FEAT* consumes the least memory for the *TCAS* dataset. Thus, our performance study also demonstrates that *FEAT* is also a space-efficient algorithm.

7.2.2 Effectiveness of the pruning techniques. Although the above experimental results demonstrate that *FEAT* is significantly faster than *PrefixSpan*, which already validates the effectiveness of the pruning techniques, here we present some experimental results to further isolate the effectiveness of each pruning technique. Figure 6 shows the results for *Gazelle* dataset. We see that both the *forward prune* and *backward prune* methods are very effective in improving the algorithm efficiency. However, the *backward prune* method is much more effective than the *forward prune* method and if we combine the two methods we can get almost the same performance as the one which only applies *backward prune* technique. We also

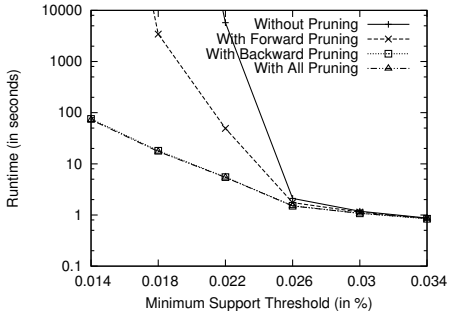

Figure 6: Effectiveness of the pruning techniques (*Gazelle*)

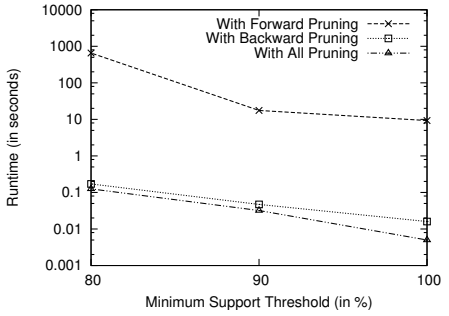

Figure 7: Effectiveness of the pruning techniques (*ProgramTrace*)

used the *ProgramTrace* and *TCAS* datasets to evaluate the effectiveness of the pruning methods. Figures 7 and 8 also show that the *forward prune* method performs much worse than the *backward prune* method on dense datasets like *ProgramTrace* and *TCAS*.

7.2.3 FEAT vs. seqHarmony. In Section 6, we extended algorithm *FEAT* to mine high confidence classification rules directly and devised the *seqHarmony* algorithm. The main enhancement of *seqHarmony* to *FEAT* is that it introduces the unpromising projected database pruning technique to algorithm *FEAT*. Figure 9 compares the two algorithms for dataset *VistaReview*. We can see that *seqHarmony* is much faster than *FEAT* at a low minimum support, which indicates the *Unpromising projected database Pruning* method is very effective in improving the algorithm efficiency. We note here that as *seqHarmony* directly mines a set of high quality sequence generator based classification rules, which means that we no longer need to do feature selection for classification model construction. Since the result set of *FEAT* may contain a huge set of sequence generators, from which selecting features for classification can be very costly.

7.2.4 Scalability test. We also tested the scalability of the *FEAT* algorithm. We replicated the *Gazelle* dataset 1, 2, 4, and 8 times respectively and ran *FEAT* algorithm with the minimum support set at 0.026%, 0.03%, and 0.034% respectively. The performance results are shown in Figure 10, where each curve corresponds to a minimum support. We see that the runtime increases linearly

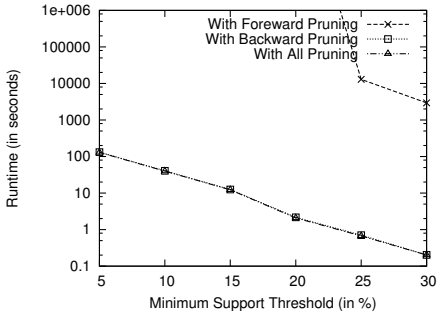

Figure 8: Effectiveness of the pruning techniques (*TCAS*)

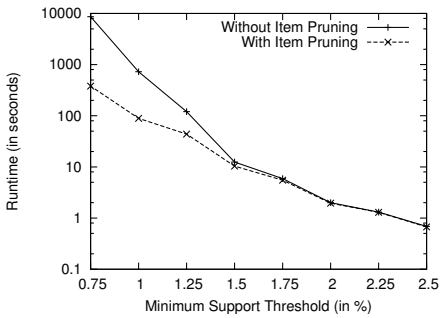

Figure 9: Effectiveness of the unpromising projected database pruning technique (*VistaReview*)

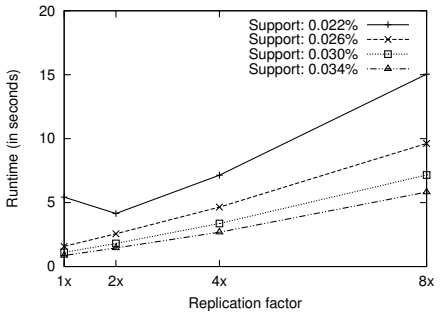

Figure 10: Scalability test (*Gazelle*)

with the increase of the base size. This shows that *FEAT* has good scalability.

7.3 Performance Study of the Sequence Generator based Classification Model

Following we will explore two typical applications of the sequence generator based classification models, *erroneous sentence detection* and *consumer product review classification*. Specifically, we will evaluate the subsequence based feature selection methods and various classification models which are based on frequent sequence generators, closed sequential patterns, and all sequential patterns, respectively. In the meantime we also present some examples of salient sequence generator patterns.

7.3.1 Confidence-based vs. Instance-centric feature selection. We first evaluated the effectiveness of the two feature selection methods, namely, **confidence-based approach** and **instance-centric approach** using the ESL and product review datasets. Some previous algorithms use some variants of the confidence-based approach to select a set of frequent subsequences as features [21] [23]. Our experiments show that the confidence-based approach is not as good as our newly proposed instance centric approach. We first built *SVM* and *Naïve Bayes* models based on sequence generators using different feature selection methods. In the experiments, we set the minimum support threshold at 2.5% and minimum confidence at 50%. We varied the parameter η from 1 to 3 for the instance centric approach. Table 5 compares the accuracy of different models based on different feature selection methods for dataset *VistaReview*. We see the instance-centric approach achieves much higher accuracy than the confidence-based approach. In the meantime, we also see that *SVM* is much better than *Naïve Bayes* classifiers for this dataset. We tried higher confidence (e.g., 75%) for the confidence-based approach, the experimental results show that it does not improve the classification too much. For example, at min_conf=75%, the confidence-based *SVM* and *Naïve Bayes* classifiers can only achieve an accuracy of 64.48% and 52.00% respectively for the same *VistaReview* dataset. We also used the closed sequential patterns to build classification models and compared the effectiveness of the confidence-based approach with that of instance centric approach. Table 6 shows the comparison results for dataset *VistaReview*. We see that the instance-centric approach is also much better than the confidence-based approach. In the following experiments, we used the instance-centric approach to do feature selection by default.

Salient Sequence Generator Rules. Although not all the features are meaningful from the classification point of view, many of them seem quite interesting. Following we give a few examples.

Example 7.1. From the ESL-C dataset we can find the following rule: "$\langle this, NNS\rangle \longrightarrow$ Non-native", where NNS represents a plural noun. A typical sentence which supports this rule is "In all this skills are make us dazzled". □

Example 7.2. From the Office07Review dataset, we can find the following feature: "$\langle new, much, more\rangle \longrightarrow$ Positive", which is supported by sentences like "I love the new edition. It is much more ...". □

Example 7.3. From the *Office07Review* dataset, we can find another interesting feature: "$\langle trying, how\rangle \longrightarrow$ Negative", which is supported by sentences like "I'm still trying to figure out how to type in Chinese ...". □

7.3.2 Comparison of various subsequence pattern based classification models. Previous studies on sequence data classification adopt a set of sequential patterns to build classifiers [10] [21] [12] [23], and we have not seen any sequence data classifier which is built on sequence generators or closed sequential patterns. It will be very interesting to compare the classification models built from the three different subsequence patterns. We first ran *FEAT*, *BIDE*, and *PrefixSpan* to mine the complete set of frequent sequence generators, the complete set of closed sequential patterns, and the complete set of sequential patterns for *ESL-C* and *ESL-J* datasets

Table 5: Feature selection method comparison (*VistaReview* dataset, sequence generator based classification model, min_sup = 2.5%, *min_conf*=50%).

Feature selection method	*SVM*	*Naïve Bayes*
Confidence-based	65.5%	45.77%
Instance-centric ($\eta = 1$)	72.56%	65.98%
Instance-centric ($\eta = 2$)	72.21%	62.08%
Instance-centric ($\eta = 3$)	72.57%	60.24%

Table 6: Feature selection method comparison(*VistaReview* dataset, closed sequential pattern based classification model, min_sup = 2.5%, *min_conf*=50%).

Feature selection method	*SVM*	*Naïve Bayes*
Confidence-based	65.5%	45.77%
Instance-centric ($\eta = 1$)	72.73%	66.66%
Instance-centric ($\eta = 2$)	72.22%	62.42%
Instance-centric ($\eta = 3$)	72.38%	62.25%

with a minimum support set at 1% and 2.5% respectively, then used the instance-centric approach to do feature selection (the η parameter was set at 2). The final feature set was then used to build *SVM* and *Naïve Bayes* classifiers. Tables 7 and 8 show the comparison results in terms of classification accuracy. Similarly, we conducted experiments for *Office07Review* dataset with parameter η set at 1. The comparison results are shown in Table 9. The last row in Tables 7, 8 and 9 illustrates the average accuracy. We see that the models built from the three types of subsequence patterns have almost the same accuracy with a small variance. As shown in Figures 3 and 4, sequence generator mining can be orders of magnitude faster than traditional sequential pattern mining when the minimum support is low, which demonstrates that the sequence generator based classification model has an edge over the sequential pattern based model in terms of efficiency of model construction. While according to the MDL principle, the sequence generator based classification model is preferable to both the sequential pattern based and closed sequential pattern based models. Figure 11 compares the number of patterns mined from dataset *ESL-C* by *FEAT*, *BIDE* and *PrefixSpan* respectively, while Figure 12 compares the average pattern size (i.e., number of events) of sequence generators, closed sequential patterns, and all sequential patterns. We see the number of sequence generators is almost the same as that of closed sequential patterns, and the average size of the sequence generators is much smaller than that of closed sequential patterns. Thus, sequence generator based classifier has an advantage over the closed sequential pattern based model in terms of model complexity.

8 CONCLUSIONS

In this paper we study the problem of mining sequence generators and explore its applications in sequence data classification. We present an efficient generator checking scheme and two novel pruning methods, *backward prune* and *forward prune*, which can prune the unpromising parts of search space efficiently, and devise a frequent sequence generator mining algorithm, *FEAT*. We also present a subsequence pattern based classification framework, which includes data preprocessing, feature selection, and model construction. An extensive performance study shows that the optimization techniques are effective in improving the algorithm efficiency, *FEAT* algorithm is more efficient than the state-of-the-art sequence generator mining algorithm, *GenMiner*, and the sequence generator based classification model is very effective in detecting erroneous sentences and classifying Web product reviews. In order to avoid the time consuming feature selection process, we devise

Table 7: Classification accuracy comparison for ESL datasets (*SVM*, instance-centric, η = 2, *min_conf*=50%).

Dataset	min_sup	Generator	Closed	all sequential
ESL-C	2.5%	80.05%	80.02%	79.93%
ESL-C	1%	81.04%	81.02%	80.79%
ESL-J	2.5%	80.66%	80.66%	80.68%
ESL-J	1%	81.03%	81.07%	81.01%
-	-	80.70%	80.69%	80.60%

Table 8: Classification accuracy comparison for ESL datasets (*Naïve Bayes*, instance-centric, η = 2, *min_conf*=50%).

Dataset	min_sup	Generator	Closed	all sequential
ESL-C	2.5%	72.91%	72.89%	73.89%
ESL-C	1%	77.69%	77.57%	77.86%
ESL-J	2.5%	52.56%	52.55%	52.61%
ESL-J	1%	53.76%	53.82%	53.85%
-	-	64.23%	64.21%	64.55%

Table 9: Classification accuracy comparison for *Office07Review* dataset (instance-centric, η = 1, *min_conf*=50%).

Model	min_sup	Generator	Closed	all sequential
SVM	2%	76.87%	77.5%	77.5%
SVM	1.5%	77.81%	77.5%	76.56%
Naïve Bayes	2%	80.31%	79.68%	79.69%
Naïve Bayes	1.5%	79.37%	80.31%	78.12%
-	-	78.59%	78.74%	77.97%

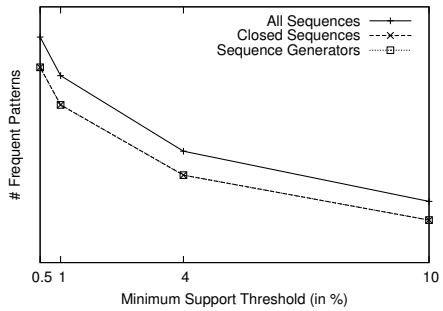

Figure 11: Comparison of # patterns (*ESL_C*)

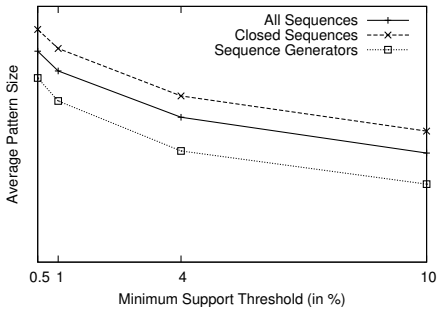

Figure 12: Average pattern size comparison (*ESL_C*)

a new algorithm *seqHarmony*, which can mine classification rules directly from the input sequence database, and evaluate its effectiveness. In future we will further explore the applications of sequence generator mining in Web page classification and click stream data analysis.

ACKNOWLEDGEMENTS

We thank David Lo, Siau-Cheng Khoo, and Jinyan Li for sending us the TCAS dataset, and their binary code of the GenMiner algorithm. We also appreciate their post conference clarification and correction through our personal communication in terms of the performance comparison between algorithms BIDE and CloSpan, which can be found from the link of *http://www.comp.nus.edu.sg/~dlo/papers/sdm08.pdf*.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society.

[2] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 429–435, Edmonton, Alberta, Canada, 2002. ACM.

[3] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Proceedings of the First International Conference on Computational Logic*, pages 972–986, London, UK, 2000. Springer.

[4] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, and Lotfi Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, 2000.

[5] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queris by means of free-sets. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 75–85, Lyon, France, 2000. Springer.

[6] Jinlin Chen and Terry Cook. Mining contiguous sequential patterns from web logs. In *Proceedings of the 16th International Conference on World Wide Web*, pages 1177–1178, Banff, Alberta, Canada, 2007. ACM.

[7] Hong Cheng, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 716–725, Istanbul, Turkey, 2007. IEEE.

[8] Hong Cheng, Xifeng Yan, Jiawei Han, and Philip S. Yu. Direct discriminative pattern mining for effective classification. In *Proceedings of the 24th International Conference on Data Engineering*, pages 169–178, Cancún, México, 2008. IEEE.

[9] Gao Cong, Kian-Lee Tan, Anthony K. H. Tung, and Xin Xu. Mining top-k covering rule groups for gene expression data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 670–681, Baltimore, Maryland, USA, 2005. ACM.

[10] Mukund Deshpande and George Karypis. Evaluation of techniques for classifying biological sequences. In *Proceedings of the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 417–431, Taipei, Taiwan, 2002. Springer.

[11] Chuancong Gao, Jianyong Wang, Yukai He, and Lizhu Zhou. Efficient mining of frequent sequence generators. In *Proceedings of the 17th International Conference on World Wide Web*, pages 1051–1052, Beijing, China, 2008. ACM.

[12] Nitin Jindal and Bing Liu. Identifying comparative sentences in text documents. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 244–251, Seattle, Washington, USA, 2006. ACM.

[13] Jinyan Li, Guozhu Dong, Kotagiri Ramamohanarao, and Limsoon Wong. Deeps: A new instance-based lazy discovery and classification system. *Machine Learning*, 54(2):99–124, 2004.

[14] Jinyan Li, Haiquan Li, Limsoon Wong, Jian Pei, and Guozhu Dong. Minimum description length principle: Generators are preferable to closed patterns. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, Massachusetts, USA, 2006. AAAI Press.

[15] Jinyan Li, Guimei Liu, and Limsoon Wong. Mining statistically important equivalence classes and delta-discriminative emerging patterns. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 430–439, San Jose, California, USA, 2007.

[16] David Lo, Siau-Cheng Khoo, and Jinyan Li. Mining and ranking generators of sequential patterns. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 553–564, Atlanta, Georgia, USA, 2008. SIAM.

[17] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory*, pages 398–416, Jerusalem, Israel, 1999. Springer.

[18] Jian Pei, Jiawei Han, and Runying Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[19] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, Heidelberg, Germany, 2001. IEEE Computer Society.

[20] Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. *CoRR*, cmp-lg/9704002, 1997.

[21] Rong She, Fei Chen, Ke Wang, Martin Ester, Jennifer L. Gardy, and Fiona S. L. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 436–445, Washington, DC, USA, 2003. ACM.

[22] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17, Avignon, France, 1996. Springer.

[23] Guihua Sun, Xiaohua Liu, Gao Cong, Ming Zhou, Zhongyang Xiong, John Lee, and Chin-Yew Lin. Detecting erroneous sentences using automatically mined sequential patterns. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic, 2007. The Association for Computer Linguistics.

[24] Jianyong Wang, Jiawei Han, and Chun Li. Frequent closed sequence mining without candidate maintenance. *IEEE Trans. Knowl. Data Eng.*, 19(8):1042–1056, 2007.

[25] Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 236–245, Washington, DC, USA, 2003. ACM.

[26] Jianyong Wang and George Karypis. On mining instance-centric classification rules. *IEEE Trans. Knowl. Data Eng.*, 18(11):1497–1511, 2006.

[27] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large databases. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, San Francisco, CA, USA, 2003. SIAM.

[28] Mohammed Javeed Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[29] Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM International Conference on Data Mining*, Arlington, VA, USA, 2002. SIAM.