Margin-Closed Frequent Sequential Pattern Mining

Dmitriy Fradkin  
Siemens Corporate Research  
755 College Road East  
Princeton, NJ 08540  
dmitriy.fradkin@siemens.com

Fabian Moerchen  
Siemens Corporate Research  
755 College Road East  
Princeton, NJ 08540  
fabian.moerchen@siemens.com

ABSTRACT

We present a new approach to mining sequential patterns that significantly reduces the number of patterns reported, favoring longer patterns and suppressing shorter patterns with similar frequencies. This is achieved by mining only margin-closed patterns whose support differs by more than some margin from any extension. Our approach extends the efficient BIDE algorithm to enforce the margin constraint. The set of margin-closed patterns can be significantly smaller than a set of just closed patterns while retaining the most important information about the dataset. This is shown by an extensive empirical evaluation on six real life databases.

1. INTRODUCTION

Temporal data mining exploits temporal information in data sources in the context of data mining tasks such as clustering or classification. Many scientific and business data sources are dynamic and thus promising candidates for application of temporal mining methods. For an overview of methods to mine time series, sequence, and streaming data see [17, 14].

One particular type of temporal data are sequences of (sets of) discrete items associated with time stamps, for example histories of transactions of customers in an online shop or log messages emitted by machines or telecommunication equipment during operation. A common task is to mine for local regularities in this data by looking for sequential patterns [2] that represent a sequence of itemsets possibly with gaps in the observation sequences.

It is well know that frequent itemset mining suffers from a combinatorial explosion of results when lowering the minimum support threshold. When mining sequential patterns, i.e., sequences of itemsets on sequential databases, this effect typically becomes even stronger. A lossless way of reducing the number of reported patterns that favors longer, thus more interpretable patterns, is mining of closed patterns. Only patterns that cannot be extended by additional items without lowering their support are reported. A straightforward extension of closed itemset mining are margin-closed itemsets, also known as δ-tolerance itemsets [12]. Margin-closed patterns are used, for example, to find patterns with gaps in the observation sequences.

Margin-closed itemsets have been previously proposed by the authors for exploratory knowledge discovery tasks in the context of temporal data mining [25, 27] and independently as δ-tolerance itemsets for frequency estimation in [12]. Margin-closed patterns are a specialization of closed itemsets with a constraint to limit the redundancy among reported patterns. An itemset is closed if no superset with the same frequency exists. An itemset is margin-closed if no superset with almost the same frequency exists, where ‘almost’ is defined by a threshold δ on the relative (or absolute) difference of the frequencies. The threshold ensures a frequency margin among the reported patterns. An efficient algorithm for mining margin-closed itemsets, extending the well-known DCL_Closed algorithm [21], has been proposed in [24].

A related line of work is motivated by the fact that transaction data is often noisy. The strict definition of support, requiring all

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  

UP’10, July 25th, 2010, Washington, DC, USA.
Copyright 2010 ACM 978-1-4503-0216-6/10/07 ...$10.00.
items of an itemset to be present in a transaction, is relaxed, see [16] and references therein. These approaches can reveal important structures in noisy data that might otherwise get lost in a huge amount of fragmented patterns. One needs to be aware though that such approaches report approximate support values and possibly list itemsets that are not observed as such in the collection at all [1] or with much smaller support.

2.2 Sequential mining

An overview of algorithms for sequential pattern mining is given in [44]. Our approach extends the BIDE algorithm [39] that uses a smart search space pruning and does not require the patterns found to remain in memory until the algorithm terminates.

Motivated by approaches that have worked on itemsets, research on reduction of the output of sequential pattern mining algorithms includes compression of the mining result in a post-processing step [10, 41], a condensed representation to evaluate sequential association rules [35], and approximate patterns [45] under the Hamming distance.

These approaches are different from the one presented here in at least one of the following ways:

- The margin constraint favors longer patterns, whereas condensed representation focus on reconstruction of frequencies for patterns not reported or compression ratio of the complete pattern set.
- Patterns observed exactly as is, with exact frequencies, are reported, whereas approximate patterns represent observations that may differ (slightly).
- The pruning is integrated in the mining algorithm whereas compression is a postprocessing of the results after mining.

The presented approach therefore has merits in particular if the patterns are used in a context that requires interpretation, as opposed to automated post processing with other algorithms. Longer patterns are more interpretable because they offer more context to the analyst. While approximations that tolerate errors may be more robust, they may report approximate support values and possibly list itemsets that are not observed as such in the collection at all or with much smaller support. This might be misleading in exploratory applications.

A generalization of sequential patterns are partial orders [9]. Instead of requiring a full ordering of the itemsets in a pattern some order relation may be unspecified. This is typically represented by a directed graph of itemsets. In [34] it is shown that closed partial orders are also a generalization of Episodes [22] that are restricted to directed graph of itemsets. In [9] closed partial orders are mined by grouping of sequential patterns and generating directed graphs. In [26] it is shown that the grouping corresponds to an instance of the closed itemset mining problem. [36] describes an algorithm to mine arbitrary (not necessarily closed) groups of sequential patterns. Experiments on real life data in [29] show that the grouping can both reduce or increase the number of patterns found, depending on the dataset.

3. MINING MARGIN-CLOSED SEQUENTIAL PATTERNS

3.1 Preliminaries

Definition 3.1. An event sequence over a set of events $\Sigma$ is a sequence of pairs $(t_i, s_i)$ of event sets $s_i \subseteq \Sigma \forall i = 1, \ldots, n$ and time stamps $t_i \in \mathbb{R}^+$. The ordering is based on time, i.e. $\forall i < j : t_i \leq t_j$. The length of the event sequence is $n$.

For most of the discussion in our work (and in much of sequential pattern mining literature) the exact values of the time stamps are not as important as the ordering that they impose. We will therefore omit timestamps from discussion and will treat event sequences as just an ordered set of event sets $S = \{s_i\}$.

Definition 3.2. A sequence database, SDB, of size $N$ is a collection of event sequences $P_i, i = 1, \ldots, N$.

We now need to introduce a basic definition from order theory.

Definition 3.3. A partial order is a binary relation $\prec$ over a set $S$ which is reflexive, antisymmetric, and transitive, i.e., for all $a, b, c \in P$, we have that:

- $a \prec a$ (reflexivity);
- $a \prec b$ and $b \prec a$ imply $a = b$ (antisymmetry);
- $a \prec b$ and $b \prec c$ imply $a \prec c$ (transitivity).

A set $S$ with a partial order is a chain iff $\forall a, b \in S: a \prec b$ or $b \prec a$.

Definition 3.4. A partial order pattern $P$ is a set of event sets $\{p_i\}, i = 1, \ldots, n$ together with a partial order $\prec$ over them.

Definition 3.5. A sequential pattern $P$ is a partial order pattern that is a chain: $p_1 \prec p_2 \prec \ldots \prec p_n$.

Note that in Episod mining sequential patterns are called serial patterns.

Definition 3.6. A parallel pattern $P$ is a partial order pattern with no order relations among the event sets.

Definition 3.7. A sequence $S = \{s_i\}, i = 1, \ldots, k$ matches a sequential pattern $P = \{p_j\}, i = 1, \ldots, m$ (or a pattern occurs in the sequence) iff $\exists i_1, \ldots, i_m$ with $p_j \subseteq s_{i_j}$ for $j = 1, \ldots, m$, such that $\forall 1 \leq j, k \leq m: p_j \prec p_k$ implies $i_j < i_k$. We will denote such a match by $m_{i_1,\ldots,i_m}(P,S)$.

Definition 3.8. A match $m_{i_1,\ldots,i_m}(P,S)$ is the earliest match iff for any other match $m_{j_1,\ldots,j_m}(P,S)$ $i_k \leq j_k, \forall k = 1, \ldots, m$.

Definition 3.9. A pattern $P$ has support($P$) = $s$ in an SDB $D$ if $D$ contains $s$ distinct event sequences that match $P$. A pattern is frequent iff its support is no less than some predefined minimum support value $\mu$, i.e. support($P$) $\geq \mu$.

In the following, when talking about patterns, we will always assume that they are frequent, with some minimum support $\mu$ defined.

Definition 3.10. A frequent pattern $P$ is considered closed in an SDB $D$, if there is no pattern $P' \neq P$ in $D$, such that $\exists m(P, P')$ (i.e. $P$ occurs in $P'$) and support($P'$) = support($P$).

Definition 3.11. A pattern $P$ is considered margin-closed in an SDB $D$, with margin $\alpha$, if there is no pattern $P' \neq P$ in $D$ such that $P$ occurs in $P'$ and support($P'$) $> (1 - \alpha)$ * support($P$).

In other words, $P$ is margin-closed if there is no pattern $P'$ that contains $P$ and is almost as frequent.

In order to describe the algorithms in the following sections, we need to introduce the notion of projected databases, which is extremely useful in constructing efficient algorithms for sequential pattern mining.
**Definition 3.12.** Given a pattern \( P = \{ p_i \}, i = 1, \ldots, |P| \) and a sequence \( S = \{ s_j \}, j = 1, \ldots, |S| \), with the earliest match \( m_{k_1,k_m}(P, S) \), a projection of \( S \) on \( P \) results in a projected sequence \( S' \mid P = \{ s_k \} \), where \( t = k_m + 1, \ldots, |S| \). We refer to \( k_m \) as an offset.

**Definition 3.13.** Given a pattern \( P = \{ p_i \}, i = 1, \ldots, |P| \) and an SDB \( D = \{ S_j \} \), \( j = 1, \ldots, |D| \), a projection of \( D \) on \( P \) is a projected database \( D' \mid P \), consisting of projected sequences \( S_j \mid P \), obtained by projecting \( S_j \mid P \) onto \( P \). Note that if a sequence does not match a pattern, it does not appear in the projected database.

Projected database \( D' \mid P \) can be efficiently represented with a list of pairs of indices \( (j_a, t_a) \), where \( j_a \) refers to \( S_j \mid P \) and \( t_a \) is the corresponding offset.

**Definition 3.14.** For a projected sequence \( S' \mid P \) we define two operations:

- original\( (S' \mid P) = S \); and
- prefix\( (S' \mid P) = \{ s_i \}, t = 1, \ldots, k_m, \) where \( m_{k_1,k_m}(P, S) \) is the earliest match.

In other words, original\( of a projection returns the whole sequence \( S \), while prefix\( of a projection returns the part of the sequence preceding the projection.

### 3.2 The BIDE algorithm

BIDE is an efficient algorithm for finding frequent closed sequential patterns in sequential databases [39]. We extend this algorithm to enforce the margin-closed constraints. In order to make this paper self-contained we provide a detailed description of BIDE using our own definitions.

BIDE is initially called with the full sequential database \( D \), minimum support \( \mu \) and an empty pattern \( P = \emptyset \). It returns a list of frequent closed sequential patterns. BIDE operates by recursively extending patterns, and, while their frequency is above the minimum support, checking closure properties of the extensions.

Consider a frequent pattern \( P = \{ p_i \}, i = 1, \ldots, n \). There are two ways to extend pattern \( P \) forward with item \( j \):

- Appending the set \( p_{n+1} = \{ j \} \) to \( P \) obtaining \( P' = p_1 \ldots p_n p_{n+1} \), called a forward-\( S(\text{equence}) \)-extension.
- Adding \( j \) to the last itemset of \( P \): \( P' = p_1 \ldots p_n j \), assuming \( j \notin p_n \), called a forward-\( I(\text{tem}) \)-extension.

Similarly, a pattern can be extended backward

- Inserting the set \( p_e = \{ j \} \) into \( P \) anywhere before the last set obtaining \( P' = p_1 \ldots p_i p_j p_{i+1} \ldots p_n \), for some \( 0 \leq i \leq n \), called a backward-\( S(\text{equence}) \)-extension.
- Adding \( j \) to any set in \( P \) obtaining \( P' = p_1 \ldots p_i j \ldots p_n \), with \( p_i' = p_i \cup j \), assuming \( j \notin p_n \), \( 1 \leq i \leq n \), called a backward-\( I(\text{tem}) \)-extension.

According to a Theorem 3 of [40], a pattern is closed if there exists no forward-\( S \)-extension item, forward-\( I \)-extension item, backward-\( S \)-extension item, nor backward-\( I \)-extension item with the same support.

Furthermore, if there is a backwards extension item, then the resulting extension and all of its future extension are explored in a different branch of recursion, meaning that it can be pruned from current analysis. These insights are combined in BIDE, leading to a very memory-efficient algorithm, because the patterns found do not need to be kept in memory while the algorithm is running.

Specifically, consider pseudo-code for BIDE (Algorithm 1). In Lines 3-4 items that can be used in forward extension of the current pattern are found. If there is no forward extension with the same support (Line 5), the backward closure is checked (Line 6) using function backScan. If the pattern is also backwards-closed, it can be added to the set of closed frequent patterns (Line 7).

Then, we check every item in forward and I extensions (in the two for-loops) to see whether it is explored in a different branch of recursion, again via backScan function (Lines 12 and 18). If not, then we project the database on the extension and call BIDE recursively on the extension and the new projected database.

Pseudo-code for sStepFrequentItems and iStepFrequentItems is shown in Algorithms 3 and 4. Algorithm 2 shows the FrequencyCheck function. These functions are rather straightforward.

It remains to discuss the backScan function (Algorithm 5). The backScan function has two uses. The first time it is called in function BIDE, closure check flag is set to true (Line 6). Then backScan returns true if and only if pattern \( P \) is backwards closed, i.e. if there is no backwards extension with the same support. This is Case I. The other calls from BIDE are with closure check set to false. In these situations backScan needs to check if a pattern extension is backwards closed in its projected database i.e. that it can’t be
reached in a different way, via a different recursion branch. This is Case II.

In order to check for backward extensions of a pattern \( P \), we need to know which parts of sequences in \( D \) need to be looked at. If \( P \) has a backward-S-extension item \( t \) between \( p_i \) and \( p_i + 1 \), it means that in each sequence \( S \in D \) matching \( P \) there is a particular match \( m_{k_1 \ldots k_m}(P,S) \), such that \( t \) occurs between \( s_{k_1} \) and \( s_{k_1 + 1} \). In order to check for an existence of such an item, we can find the earliest and the latest matches \( m_{k_1 \ldots k_m}(P,S) \) and \( m_{j_1 \ldots j_m}(P,S) \), and examine the itemsets \( s_{k_1 + 1}, \ldots, s_{j_1 + 1} \). In other words, we can check for backwards-I-extension item \( t \) by looking at all possible occurrences of \( t \) together with \( p_i \), starting from earliest and ending with latest match occurrences of \( p_i \). Function \( \text{FindMaximumGaps} \) (Algorithm 6) is used exactly for finding and storing the earliest and latest indices of consecutive itemsets of pattern \( P \) in a match \( m(P,S) \). Finding of the latest match is most efficiently found by searching for a reverse of \( P \) in a reverse of sequence \( S \), and transforming the indices appropriately.

We can now discuss the two Cases mentioned above. In Case I closure check flag is set to true. Since we want to check if pattern \( P \) is closed, we need to fully examine all sequences in \( D/P \) for potential extensions. Therefore, function \( \text{FindMaximumGaps} \) is called on original sequences in \( D/P \), not on the projections. In Case II, we only care if there is a backward extension in order to prune the current pattern. Therefore, when \( \text{closedCheck} \) is false, \( \text{FindMaximumGaps} \) is called on prefixes of sequences in \( D/P \).

Once array \( G \) is computed, we check for I-expansions and for S-expansions (Algorithms 7 and 8). Consider \( \text{BackwardExpansionCheck} \). We want to detect if an item can be inserted into any itemset of \( P \), while maintaining the same support. Therefore, for each position in \( P \), we examine all sequences in \( D/P \), in the intervals specified by array \( G \). If \( \text{closedCheck} \) is true, we look at the full sequence \( S \), otherwise we look at the prefix of \( S/P \). The ‘end’ indices need to be computed differently for the two cases, because when \( \text{closedCheck} \) is false, the last occurrence of the last itemset of \( P \) is also the first potential point for forward expansion and does not need to be considered, but when \( \text{closedCheck} \) is true,
we check for closedness of $P$ and all potential expansion locations need to be examined.  

For each $s_j$, if $p_i \in s_j$, we add all items in $s_j$ to set $C$, except for items already in $p_i$. After processing a sequence, we update frequency counts of items in $C$ that are stored a hash map $M$, and keep track of the maximum frequency value. Once we have processed all sequences for a particular $p_i$, we check if the maximum frequency is equal to support of $P$ ($\text{support}(P) = |D|P|$). If so, that means that there is some backward-1-expansion item for $P$, and therefore $P$ is not closed and, there is another recursion branch that will examine this expansion, so BackwardExpansionCheck returns false. If maximum frequency is below support of $P$, BackwardExpansionCheck return true.

BackwardExpansionCheck operates similarly.

**Algorithm 7 BackwardExpansionCheck**

Require: Pattern $P$, Projected Database $D|P$, $\mu$,closedCheck, gap array $G$

1: for $i = 1, \ldots, |P|$ do
2: Initialize Hash Map $M$
3: for $j = 1, \ldots, |D|$ do
4: $\text{start} = G[j][i][0]+1$
5: if closedCheck then
6: $\text{end} = G[j][i][1]$
7: $S' = \text{original}(S_j)$
8: else
9: $\text{end} = G[j][i][1] - 1$
10: $S' = \text{prefix}(S_j)$
11: end if
12: $C = \emptyset$
13: for $k = \text{start}, \ldots, \text{end}$ do
14: if $P_i \in S_k$ then
15: $C = C \cup S_k$
16: end if
17: end for
18: if $M(s_i) = M(s_i) + 1$
19: if $M(s_i) > max$ then
20: max = $M(s_i)$
21: end if
22: end if
23: end if
24: end for
25: end for
26: if max = support($P$) then
27: return false;
28: end if
29: end for
30: return true

3.3 The BIDE-Margin algorithm

We now describe the changes required to enforce margin-closedness in BIDE leading to the BIDE-Margin algorithm. The flag marginCheck is used in the backScan function instead of closedCheck and there is the additional margin parameter $\alpha$. There are three changes to the functions described in the following sections.

When Forward Expansion is considered, rather than checking if there are items with the same support as the current pattern, one instead checks for presence of items that are within margin $\alpha$ of the pattern’s support. The function FrequencyCheck for BIDE-Margin is shown in Algorithm 9.

**Algorithm 8 BackwardSExpansionCheck**

Require: Pattern $P$, Projected Database $D|P$, $\mu$,closedCheck, gap array $G$

1: for $i = 1, \ldots, |P|$ do
2: Initialize Hash Map $M$
3: for $j = 1, \ldots, |D|$ do
4: $\text{start} = G[j][i][0]+1$
5: if closedCheck then
6: $\text{end} = G[j][i][1]$ - 1
7: $S' = \text{original}(S_j)$
8: else
9: $S' = \text{prefix}(S_j)$
10: end if
11: $C = \emptyset$
12: for $k = \text{start}, \ldots, \text{end}$ do
13: if $P_i \in S_k$ then
14: $C = C \cup S_k$
15: end if
16: if $M(s_i) = M(s_i) + 1$
17: if $M(s_i) > max$ then
18: max = $M(s_i)$
19: end if
20: end if
21: end for
22: end for
23: if max = support($P$) then
24: return false;
25: end if
26: end for
27: return true

3.4 Computational Efficiency

The BIDE-Margin algorithm has the same complexity as BIDE, since it still generates all frequent sequential patterns, in exactly the same fashion. However, unlike BIDE it searches for margin-
predicted pattern will be margin-closed, rather than just closed, patterns and therefore it need to check closeness less frequently. In other words, due to a "looser" frequency check used by BIDE-Margin (Algorithm 9), the call to backscan algorithm in Line 6 of Algorithm 1 will occur less often in BIDE-Margin than in BIDE. Thus, while the overall algorithm complexity is the same, BIDE-Margin may perform slightly faster. The extent of this depends on the nature of the data and the value of margin specified.

We would also like to note that BIDE-Margin is significantly more efficient than a brute force post-processing of BIDE results would be. If $N$ is the number of patterns produced by BIDE for a particular value of support, the postprocessing would require $O(N)$ memory and $O(N^2)$ time to order the patterns by their support and then to check for each pattern $P$ if it is margin-closed.

4. EXPERIMENTS

We performed experiments on real life sequential data sets to compare BIDE and BIDE-Margin in two ways: 1) The number of patterns produced. By definition BIDE-Margin with $\alpha > 0$ produces the same or less patterns than BIDE. The goal of the experiment is to quantify the extent of this reduction on real life data. 2) Predictiveness of patterns found: We compare the classification performance of the sets of patterns when used as features in SVM training. Since BIDE-Margin suppresses only features that are very similar in frequency to reported features, we expect to see only minor performance decrease, if any. With SVM being a classifier that can deal with high dimensional data and redundancy among features this is a tough test.

We did not perform run-time comparisons between BIDE and BIDE-Margin, since the differences are expected to be small. Similarly, the scalability with the number of sequences in the database is inherited directly from BIDE.

4.1 Data

We performed experiments on six interval datasets, previously used in [29], summarized in Table 1. While technically databases of intervals, they can be interpreted as sequential databases by treating start and end boundaries of an interval as separate events [42]. Specifically, each symbolic interval, a triple $(t_s, t_e, \sigma)$ with event $\sigma \in \Sigma$ and time stamps $t_s \leq t_e$, is converted into two symbolic time points $(t_s, \sigma^*)$ and $(t_e, \sigma^*)$, and then all time points with the same time stamp are aggregated into itemsets, resulting in a standard event sequence as in Definition 3.1. Further details are given in [29].

The advantage of this collection is that class labels are available for each sequence that allows an automated evaluation of patterns using a classifier, while the categorical sequential data available in the UCI Machine Learning Repository [3] is largely unlabeled such as web log data.
The intervals label describe which blocks touch and the actions of the hand (contacts blue red, attached hand red). Each sequence represents one of 8 different scenarios from atomic actions (pick-up) to complete scenarios (assemble).

The intervals describe visual primitives obtained from videos of a human hand stacking colored blocks provided by [15]. The interval labels describe which blocks touch and the actions of the hand (contacts blue red, attached hand red). Each sequence represents one of 8 different scenarios from atomic actions (pick-up) to complete scenarios (assemble).

The intervals were derived from the high quality Aus- tralian Sign Language dataset in the UCI repository [3] donated by Kadous [19]. The x,y,z dimensions were discretized using Persist with 2 bins, 5 dimensions representing the fingers were discretized into 2 bins using the median as the divider. Each sequence represents a word like girl or right.

Blocks2 The intervals describe visual primitives obtained from videos of a human hand stacking colored blocks provided by [15]. The interval labels describe which blocks touch and the actions of the hand (contacts blue red, attached hand red). Each sequence represents one of 8 different scenarios from atomic actions (pick-up) to complete scenarios (assemble).

Context1 The intervals were derived from categoric and numeric data describing the context of a mobile device carried by humans in different situations [23]. Numeric sensors were discretized using 2-3 bins chosen manually based on exploratory data analysis. Each sequence represents one of five scenarios such as street or meeting.

Pioneer The intervals were derived from the Pioneer-1 datasets in the UCI repository [3]. The numerical time series were discretized into 2-4 bins by choosing thresholds manually based on exploratory data analysis. Each sequence describes one of three scenarios: gripper, move, turn.

Skating The intervals were derived from 14 dimensional numerical time series describing muscle activity and leg position of 6 professional In-Line Speed Skaters during controlled tests at 7 different speeds on a treadmill [27]. The time series were discretized into 2-3 bins using Persist and manually chosen thresholds. Each sequence represents a complete movement cycle and is labeled by skater or speed.

4.2 Numerosity

By construction, the number of patterns produced by BIDE-Margin is always less than or equal to that produced by BIDE. Figure 1 shows the number of patterns (on a log10 scale) found by these methods using different support thresholds and margin values. For all datasets except ASL-BU, BIDE-Margin produces significantly fewer patterns. The reduction is strongest for the Context and Skating data and for Auslan2 for large minimum supports.

4.3 Predictiveness

Patterns obtained by unsupervised mining can be used for knowledge discovery by ranking and analyzing them directly, for gener-

<table>
<thead>
<tr>
<th>Data</th>
<th>Intervals</th>
<th>Labels</th>
<th>Sequences</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASL-BU</td>
<td>18250</td>
<td>154</td>
<td>441</td>
<td>7</td>
</tr>
<tr>
<td>Auslan2</td>
<td>900</td>
<td>12</td>
<td>200</td>
<td>10</td>
</tr>
<tr>
<td>Blocks</td>
<td>1207</td>
<td>8</td>
<td>210</td>
<td>8</td>
</tr>
<tr>
<td>Context</td>
<td>12916</td>
<td>54</td>
<td>240</td>
<td>5</td>
</tr>
<tr>
<td>Pioneer</td>
<td>4883</td>
<td>92</td>
<td>160</td>
<td>3</td>
</tr>
<tr>
<td>Skating</td>
<td>18953</td>
<td>41</td>
<td>530</td>
<td>6/7</td>
</tr>
</tbody>
</table>

Table 1: Interval data: Seven databases consisting of many sequences of labeled intervals with class labels for each sequence.

In our experiments we have used the Spider Toolbox for Matlab4 As classifier, we focused on Support Vector Machines. We have also experimented with decision trees and random forests, obtaining qualitatively similar results.

Once patterns were generated, for a particular value of support and margin, we have performed 10-fold cross-validation with linear SVM, setting parameter \( C \) in turn to \( 2^k, k = -10, -9, \ldots, 9, 10 \). The best value over all values of \( C \) is reported. Note, that this is done purely for the purpose of comparing the properties of two unsupervised pattern mining techniques, hence we did not interleave the pattern mining with the cross validation, as would be needed if the goal were to train a classifier with good generalization performance.

The results are shown in Figure 2. The y-axis is the lowest classification error, while the x-axis is the minimum support. The results with different margin values are shown as different lines. Examination of these results suggests that using margin 0.05 or 0.1 barely affects the classification error rate. Margin of 0.2 does lead to noticeably worse results on Pioneer dataset, and on Auslan2 with support 20, but not on the other datasets. The differences in performance tend to become smaller as support increases and the number of patterns decreases.

Figure 3 shows results obtained with J48, using the default settings. The results are qualitatively similar to those obtained with SVM, i.e. classification error does not increase for small values of the margin. Results with random forests are omitted due to space constraints.

5. CONCLUSION

We presented a new constraint for reducing the output of sequential pattern mining and an efficient algorithm for mining such patterns. We have demonstrated that the number of margin-closed patterns can be a lot smaller than that of closed patterns, but that these patterns are just as useful, as evidenced by performance of classifiers built using these patterns.

Using data mining in real life systems often requires the analyst to understand and trust the reported results to take appropriate action. We believe that reporting of exact patterns with exact frequency and favoring longer patterns while pruning similar shorter patterns are all advantageous for interpretation of mining results by an analyst. For some domains, however, error tolerance [45] may be more important than interpretability.

Mining closed sequential patterns is an important task in temporal data mining from time point and time interval data [28] and it is a substep in the process of mining partial orders [9]. In future work, we intend to conduct an experimental evaluation of the run-time of BIDE-Margin compared with BIDE using both actual pattern mining time and the time needed by follow up data mining tasks such as classification or grouping of sequences into partial orders.

6. REFERENCES


1http://www.bu.edu/asllrp/
3http://www.cis.purdue.edu/gobi/ama.tar.Z
4http://www.kyb.mpg.de/bs/people/spider/main.html
Figure 1: Comparison of the number of patterns (log10 scale) for different minimum support thresholds and margin values.


Figure 2: SVM classification errors achieved with different minimum support thresholds and margin values.
Figure 3: J48 classification errors achieved with different minimum support thresholds and margin values.

[40] Jianyong Wang, Jiawei Han, and Chun Li. Frequent closed sequence mining without candidate maintenance. IEEE Transactions on Knowledge and Data Engineering, 19(8):1042–1056, 2007.