Abstract

Different from traditional positive sequential pattern mining, negative sequential pattern mining considers both positive and negative relationships between items. Negative sequential pattern mining doesn’t necessarily follow the Apriori principle, and the searching space is much larger than positive pattern mining. Giving definitions and some constraints of negative sequential patterns, this paper proposes a new method for mining negative sequential patterns, called Negative-GSP. Negative-GSP can find negative sequential patterns effectively and efficiently by joining and pruning, and extensive experimental results show the efficiency of the method.

Keywords: Negative Sequential Pattern, Sequence Mining, Data Mining

1 Introduction

The concept of discovering sequential patterns was firstly introduced in 1995 (Agrawal et al. 1995), aiming at discovering frequent subsequences as patterns in a sequence database, given a user-specified minimum support threshold. Some popular algorithms on sequential pattern mining include AprioriAll (Agrawal et al. 1995), GSP (Generalized Sequential Patterns) (Srikant et al. 1998) and PrefixSpan (Pei et al. 2004). GSP and AprioriAll are both Apriori-like methods based on breadth-first search, while PrefixSpan is based on depth-first search. Some other methods such as SPADE (Sequential Pattern Discovery using Equivalence classes) (Zaki 2001) and SPAM (Sequential Pattern Mining) (Jay et al. 2002) are also widely used in researches.

Different from traditional positive sequential patterns, negative sequential patterns focus on negative relationship between itemsets, in which case, absent items are taken into consideration. We give a simple example to illustrate the differences: Suppose \( p_1 = \langle a \ b \ c \ d \ f \rangle \) is a positive sequential pattern; \( p_2 = \langle a \ b \ \neg c \ e \ f \rangle \) is a negative sequential pattern; and each item \( a, b, c, \) etc, stands for a medical item code in the customer claim database of a private health care insurance company. By getting pattern \( p_1 \), we can tell that an insurer usually claimed for \( a, b, c, d \) and \( f \) in a row; but with pattern \( p_2 \), we are also able to find that given an insurer claim for medical items \( a \) and \( b \), and the customer does NOT claim \( c \), he/she would claim item \( e \) instead of \( d \) later. This kind of patterns can’t be described or discovered by positive sequential patterns like \( p_1 \).

However, while we tried to apply traditional frequent pattern mining algorithm to the negative patterns, two problems stand in the way:

1. Apriori principle doesn’t apply to negative sequential patterns. Apriori principle can be simply described as: a sequence can not be frequent if any of its sub-sequences is not. The Apriori principle is widely adopted to reduce the candidate subsequences in positive patterns (Agrawal et al. 1995, Srikant et al. 1998), but it is not necessarily true with patterns containing negative items. Take \( c_1 = \langle b \ \neg c \rangle \), \( c_2 = \langle b \ c \ a \rangle \) as two candidate patterns and \( s = \langle b \ d \ a \rangle \) as sequence data. We can see that \( s \) supports \( c_2 \) but doesn’t support \( c_1 \), which is to say, the pattern \( c_2 \) may have greater support than \( c_1 \) although \( c_2 \) has one more element. We are going to discuss this problem thoroughly in Section 3.

2. Due to the vast candidate space, how can we find frequent patterns efficiently and effectively? Take a 3-length sequence \( \langle a \ b \ c \rangle \) for instance, it can only support positive candidate patterns \( \langle a \rangle, \langle b \rangle, \langle c \rangle, \langle a \ b \rangle, \langle a \ c \rangle, \langle b \ c \rangle \) and \( \langle a \ b \ c \rangle \). But in the negative case, sequence \( \langle a \ b \ c \rangle \) is not only matched with the above positive patterns, but also can be matched with a large bunch of negative candidates, such as \( \langle a \ \neg a \rangle, \langle b \ \neg a \rangle, \langle b \ \neg b \rangle, \langle a \ c \rangle, \langle a \ \neg c \rangle \) etc, which makes the searching space much larger.

In this paper, we propose a new method, Negative-GSP, for mining negative sequential patterns based on the GSP algorithm. We also improve the joining and pruning steps for negative sequences to ensure search space integrity and reduce the number of negative candidates as well. Our experiments show that our method can find sequential patterns effectively.

2 Related Work

Before negative sequential pattern mining was proposed, a couple of methods have been designed to find negative association rules (Savasere et al. 1998, Wu et al. 2004, Antonie et al. 2004) and negative sequential rules (Zhao et al. 2008). Most early researches on sequential patterns focused on positive relationships, and in recent years, a few researches start to focus on negative sequential pattern mining. The following are some researches pressed in recent years.

Ouyang & Huang (Ouyang et al. 2007) extended traditional sequential pattern definition \( (a,b) \) to include negative element like \( (\neg a,b), (a,\neg b) \) and \( (\neg a,\neg b) \). They put forward an algorithm which finds both frequent and infrequent sequences and then obtains negative sequential patterns from infrequent sequences. A drawback of the algorithm is that a large amount of space is required in order to find both frequent and infrequent sequences.
Nancy et al. (Nance et al. 2007) designed an algorithm named PNSPM (Positive and Negative sequential pattern mining) for mining negative sequential patterns. They applied the Apriori principle to prune redundant candidates, and extracted meaningful negative sequences using the interestingness measure. According to their pattern definition, all elements must be positive except for the last one.

Ouyang et al. (Ouyang et al. 2008) presented the definitions of three types of negative fuzzy sequential patterns: k-sequence (−(a, b)) and (−a, b), and then described a method for mining native fuzzy sequential patterns from quantitative valued transactions.

The GSP algorithm (Srikant et al. 1998) is a classical and widely recognized algorithm for sequential pattern mining. GSP makes multiple passes over the dataset to generate frequent sequential patterns. The first pass starts from calculating the support of each single item. At the end of the first pass, all of the 1-item patterns are obtained, which are then used as seeds to generate new candidates for the next pass. Each new candidate has one more item than its seed. The candidates are then pruned to remove infrequent ones. After pruning, the supports of the new candidates are counted by another pass over the dataset and frequent patterns become the seeds for the next pass. The algorithm terminates when there are no frequent patterns at the end of a pass, or when no candidates are generated.

Sue-Chen et al. (Sue-Chen et al. 2008) proposed an algorithm called PNSP (Positive and Negative sequential pattern mining). They presented more comprehensive definitions about negative sequential patterns and extended GSP algorithm to deal with mining negative sequential patterns. Two concepts called n-cover and n-contain are employed to guide the method. It was claimed that if a n-cover value of a candidate is less than the min_support, any of its super-sequence is not going to be frequent so the searching of candidate is ended. However, we found out it may cause a loss of some candidates, and so we proposed new measurement to generate and prune candidates.

3 Problem Statement

3.1 Negative Sequence

Definition 1: Sequence

A sequence s is an ordered list of elements, s =< e1 e2 ... en >, where each ei, 1 ≤ i ≤ n, is an element. An element ei (1 ≤ i ≤ k) includes one or more items. For example, sequence a b (c, d) e includes 4 elements and (c, d) is an element which includes two items.

The length of a sequence is the number of items in the sequence. A sequence with k items is called a k-sequence or k-item sequence.

Definition 2: Positive/Negative Sequence

A sequence s =< e1 e2 ... en > is a positive sequence, when each element ei, 1 ≤ i ≤ n, is a positive element. A sequence s =< e1 e2 ... en > is a negative sequence, when ∃i, 1 ≤ i ≤ n, ei is a negative element representing the absence of an element. For example, −c and −(c, d) are negative elements, so < a b −c f > and < a b −(c, d) f > are both negative sequences.

Definition 3: Subsequence

A sequence s =< r1 r2 ... rm > is a subsequence of another sequence s =< p1 p2 ... pm >, if there exists 1 ≤ i1 ≤ i2 ≤ ... ≤ im ≤ n, ri ∈ pi, ri ∈ pi, ..., ri ∈ pi. Definition 4: Maximum Positive Subsequence

A sequence sα is a maximum positive subsequence of another sequence sβ, if sα is a subsequence of sβ, and sα includes all positive elements of sβ. For example, < a b f > is maximum positive subsequence of < a b f > and < a b −(c, d) f >.

Definition 5: Negative Sequential Pattern

If the support of a negative sequence is greater than a pre-defined support threshold min_sup, and it meets the following constraints, then we call it a negative sequential pattern.

1) Items in a single element should be all positive or all negative. For example, < a (a, b) c > is not allowed because item a and item b are in a same element.

2) Two or more continuous negative elements are not accepted in the negative sequence. This constraint is also used by other researchers (Sue-Chen et al. 2008).

3) For each negative item in a negative pattern, its corresponding positive item is required to be frequent. For example, if < −c > is a negative item, its corresponding positive item < c > is required to be frequent.

3.2 Negative Pattern Matching

In order to calculate the support of a negative sequential pattern against the sequence data in database, we need clarify pattern-sequence matching method and criterion. We need to define which patterns can a sequence support.

Definition 6: Negative Base-matching

A negative sequence sn =< c1 c2 ... ck > base-matches a data sequence s =< d1 d2 ... dm >, iff, for every negative element c1, there exists integers p, q, r (p < q < r) such that the two conditions hold:

1) s contains the maximum positive subsequence of sn.

2) ∃e1−1 ≤ dp and ei+1 ≤ dp, and ∃dq, ei ≤ dq.

That is, each positive element in sn matches with the same elements in s with same order, while each negative element of sn can find a match element in s at the corresponding position.

For example, sn =< b −c a >, it base-matches < b d a >, and also base-matches < b d c a >, since the element d, which matches −c, can be found between the element b and a.

If a sequence base-match a pattern, then we should count it in base_support value. We use base-match to find seed sequences, which will ensure the integrity of result patterns.

Definition 7: Negative Matching

A negative sequence sn =< e1 e2 ... en > matches a data sequence s =< d1 d2 ... dm >, iff, for every negative element e1, there exist integers p, q, r (p < q < r) such that the two conditions hold: (1) s contains the maximum positive subsequence of sn.

2) ∃e1−1 ≤ dp and ei+1 ≤ dp, and ∀dq, ei ≤ dq.

For example, sn =< b −c a > matches < b d a >, but does not match < b d c a >, since the negative element c appears between the element b and a in sn.

The above match definition is same as n-cover concept in (Sue-Chen et al. 2008). (Sue-Chen et al. 2008) also defined n-contain, which is a more restricted match criterion.

A pattern-sequence matching example is given in Table 1. Based on the matching method we have, Apriori-property is not suitable for this case, as we

<table>
<thead>
<tr>
<th>Pattern</th>
<th>base-match</th>
<th>match</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; b − c a &gt;</td>
<td>√</td>
<td>√</td>
<td>&lt; b d a &gt;</td>
</tr>
<tr>
<td>&lt; b − c a &gt;</td>
<td>√</td>
<td>√</td>
<td>&lt; b d a c &gt;</td>
</tr>
<tr>
<td>&lt; b c a &gt;</td>
<td>√</td>
<td>×</td>
<td>&lt; b d c a &gt;</td>
</tr>
</tbody>
</table>
Table 2: Examples for explain Apriori-Principle

<table>
<thead>
<tr>
<th>Pattern</th>
<th>match</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_{n1} = \langle \lessdot b \prec c , a \rangle$</td>
<td>$\times$</td>
<td>$\langle b , d , c , a \rangle$</td>
</tr>
<tr>
<td>$s_{n2} = \langle \lessdot b \prec c , d , a \rangle$</td>
<td>$\sqrt{\times}$</td>
<td>$\langle b , d , c , a \rangle$</td>
</tr>
</tbody>
</table>

Figure 1: Process Flow of Negative-GSP

pointed out in Section 1. Table 2 gives out a straightforward example. $s_{n1} = \langle \lessdot b \prec c \, a \rangle$ and $s_{n2} = \langle \lessdot b \prec c \, d \, a \rangle$ are two candidate patterns and $\langle b \, d \, c \, a \rangle$ is a sequence in the dataset. The table clearly shows that $s_{n1}$ does not match $s$, while $s_{n2}$ matches $s$ even if $s_{n1}$ is a sub-sequence of $s_{n2}$.

Property: Negative Frequent Pattern Property
Based on the above definitions, apparently we have a property: if a negative sequence $s$ is frequent, all its positive subsequence must be frequent; or if the maximum positive subsequence of a sequence $s$ is not frequent, $s$ can not be frequent.

4 Negative-GSP Algorithm

4.1 The Idea
Assume in a sequence data set $D_s = \{d_1, d_2, d_3, ..., d_n\}$, where $d_i \ (1 \leq i \leq n)$ is a sequence, and our objective is to find frequent negative sequential patterns in $D_s$, with a minimum support threshold $\text{min_sup}$. The basic process flow as Fig. 1.

First, we utilize the GSP algorithm to generate all positive sequential patterns. Assume $L_{pos} = \{L_{pos,1}, L_{pos,2}, ..., L_{pos,l}\}$, where $L_{pos,i}$ represents the $i$-item frequent positive sequential pattern set.

Next, we begin to generate negative sequential patterns based on $L_{pos}$. We transform all 1-item positive sequential patterns $L_{pos,1}$ to their corresponding 1-item negative sequences, which are taken as the initial 1-item candidates set $C_{neg,1}$. Then we also need prune unnecessary candidates from candidates set $C_{neg,1}$ before calculating the support of every candidates by passing over $D_s$, which will be discussed in Section 4.3.

Because Apriori Principle doesn’t work for negative sequence, the 1-item candidates need to be divided into two classes: one is valid to join with other candidates to generate 2-item negative candidates, and the other is invalid. To verify whether they are valid in the next pass, we use a base-match method to verify it (refer to Section 4.4). After this step, we get a processed 1-item seed set $S_{neg,1}$, which is then used as seed to generate a 2-item candidate set $C_{neg,2}$. The candidates with support higher than $\text{min_sup}$ are outputted to 1-item frequent patterns $L_{neg,1}$.

Based on the k-item seed set $S_{neg,k}$, we join them with the joining method presented in Section 4.2 to produce a (k+1)-item candidates set $C_{neg,k+1}$. The new candidates may include many invalid candidates, so pruning invalid candidates is necessary and helpful for further search. Our idea is to verify whether the maximum positive subsequence of the candidate is frequent. Invalid candidates are pruned, and valid ones are kept in the seed set. Then, by passing over the data set $D_s$, we get the supports of all (k+1)-item candidates. Again, the (k+1)-item frequent candidates with support higher than $\text{min_sup}$ are outputted to $L_{neg,k+1}$ as (k+1)-item frequent patterns.

After the above operation, we get the (k+1)-item frequent result $L_{neg,k+1}$ and (k+1)-item seed set $S_{neg,k+1}$ for next pass. Longer patterns are generated by repeating the above process until the candidate set becomes empty. Each iteration will generate and output frequent negative sequential patterns into $L_{neg} = \{L_{neg,1}, L_{neg,2}, ..., L_{neg,l}\}$, which is the final set of negative sequential patterns.

4.2 Joining to Generate Candidates
The (k+1)-item candidates are generated by joining all k-item seed sequences. Given two k-item seed sequences $c_i = \langle e_{i_1}, e_{i_2}, e_{i_3}, ..., e_{i_{k-1}}, e_{i_k} \rangle$ and $c_j = \langle e_{j_1}, e_{j_2}, e_{j_3}, ..., e_{j_{k-1}}, e_{j_k} \rangle$, assume $c_i = \langle e_{i_1}, e_{i_2}, e_{i_3}, ..., e_{i_{k-1}}, e_{i_k}\rangle$, $c_j = \langle e_{j_2}, e_{j_3}, ..., e_{j_{k-1}}, e_{j_k}\rangle$. If $e_{i_l} = e_{j_1}$, $c_i$ and $c_j$ are joined to get a (k+1)-item candidate: $\langle e_{i_1}, e_{i_2}, e_{i_3}, ..., e_{i_{k-1}}, e_{j_k}\rangle$.

The above joining method is similar to but different from the joining method of GSP. The GSP algorithm gets all (k+1)-item candidates by joining k-item frequent sequential patterns since positive sequences obey the Apriori principle. However, when we mine negative sequential patterns, we need to join not only k-item frequent patterns but also some infrequent k-item sequences, as demonstrated by Section 4.3.

Another point is that, while performing the joining operation, we ignore the joining of positive patterns with themselves, since they have been done in the positive sequential pattern mining at the first step of our algorithm.

4.3 Pruning Unnecessary Candidates
Since the candidates of negative sequential patterns generated during joining step are much more than positive sequential patterns, it is necessary to design an effective pruning method for negative sequential pattern mining. This step prunes some unnecessary candidates from candidates set.

With the GSP algorithm, while pruning in k-item candidates, it prunes all the candidates whose (k-1)-subsequences are not frequent. However, the above pruning method does not work for negative sequential pattern mining in the following ways.

Firstly, given a candidate $C = \langle a \prec b \prec c \, d \prec e \rangle$. Note that we don’t allow two continuous negative elements in a sequential pattern, so $C = \langle a \prec b \prec c \rangle$ is an invalid negative sequential pattern. As a result, $C$ will be pruned. However, in fact, the candidate $C$ may be a valid candidate of negative sequential pattern. Secondly, $C$ may be frequent even when its subsequence is not frequent. So we can’t prune $C$ simply even its subsequence is infrequent.

Our pruning method is described as follows. For a candidate $C = \langle c_1, c_2, c_3, ..., c_n \rangle$, suppose $C' = \langle e_1, e_2, ..., e_k \rangle$ is the maximum positive subsequence of $C$. If $C'$ is not frequent, $C$ must be infrequent and should be pruned. This method is simple but effective to prune invalid candidates without cutting off possible valid candidates by mistake.
4.4 Generating Seed Set for Next Pass

This step keeps all necessary k-item seed sequences in seed set for next pass, and then (k+1)-item candidates are generated by joining both frequent and infrequent k-item seed sequences.

Given an infrequent 2-item sequence <b ¬c>, 3-item candidate <b ¬c d> may still be frequent. So we need count <b ¬c> as seed sequence for joining and generate 3-item candidate <b ¬c d>.

It is proposed that the k-item sequences will be regarded as k-item seed sequences if their base-supports are greater than min_sup. So we check each candidate’s base-support value to see whether it is greater than min_sup. If not, it means that the sequence is impossible to generate (k+1)-item frequent pattern and we don’t need count it in the seed set again.

4.5 Algorithm Description

Our proposed algorithm for negative sequential pattern mining is given below.

Step 1: Find all positive sequential patterns by the traditional GSP algorithm(Srikant et al. 1998).

Step 2: Transform 1-item positive patterns to 1-item negative patterns as candidates set, and then get 1-item seed set and 1-item patterns.

Step 3: Use all (k-1)-item seeds, perform the joining operation with each other and get k-item candidates and also join (k-1)-item positive patterns with (k-1)-item seed sequences.

Step 4: Prune unnecessary candidates to get a smaller candidate set.

Step 5: Count all candidates’ supports and base-supports.

Step 6: If the base-support is greater than min_sup, then the candidate is added to k-item seed set. If the support is greater than min_sup, then the candidate is frequent and is outputted as a k-item pattern.

Step 7: If k-item seed set is not empty, increase k by one and loop back to Step 3 until the next candidate set is empty.

The procedure is illustrated with an example in Fig 2.

5 Experiments and Results

5.1 Datasets

Two synthetic datasets generated by IBM data generator(Agrawal et al. 1995) are used to test our algorithm in the experiments.

Dataset1(DS1) is C8.T8.S4.I8.DB10k.N1k, which contains 10k sequences and the number of items is 1000, the average number of elements in a sequence is 8, the average number of items in an element is 8, average length of maximal pattern consists of 4 elements and each element is composed of 8 items averagely.

Dataset2(DS2) is C10.T2.5.S4.I2.5.DB100k.N10k, which contains 100k sequences and the number of items is 10k, the average number of elements in a sequence is 10, the average number of items in an element is 2.5, average length of maximal pattern consists of 4 elements and each element is composed of 2.5 items averagely.

5.2 Experiments Results

We compared the runtime of negative sequential pattern mining with different support thresholds (see Fig 3), and also compared counts of patterns with different support thresholds (see Fig 4). Negative pattern mining spends much more runtime than positive pattern mining because the candidates counts are not of the same magnitude, especially when the support threshold is set very low.

5.3 Comparison with PNSP Algorithm

We compared our method with PNSP algorithm(Sue-Chen et al. 2008). The results of runtime (see Fig 5)
show that our method outperforms PNSP. The reason is that PNSP generates more unnecessary candidates. Especially when the number of negative frequent items increased, its negative candidates may increase sharply. For our algorithm, when there are a lot of negative candidates, it will cost much running time in the joining process for new candidates. So when \( \min sup \) is very low, Negative-GSP can't get very good performance as when \( \min sup \) is high.

6 Conclusions and Future Work

In this paper, we presented the definitions for negative sequential patterns, and proposed a method for negative sequential pattern mining based on the GSP algorithm. We also designed effective pruning method to reduce the number of candidates. The efficiency and effectiveness of our algorithm is shown in our experiments on two synthetic datasets.

In our future research, we will focus on selecting interesting rules from the discovered negative sequential patterns. How to find interesting and interpretable rules from a lot of negative sequential patterns is valuable in business applications. Another research topic is to find more effective pruning method that can reduce candidates more effectively and avoid unnecessary candidates.

Acknowledgements

This work was supported by the Australia Research Council (ARC) Linkage Project LP0775041 and Discovery Projects DP0667060 & DP0773412, and by the Early Career Researcher Grant from University of Technology, Sydney, Australia. The author of this paper acknowledge the financial support of the Capital Markets CRC.

References


Figure 3: Runtime on DS1 and DS2

Figure 4: Patterns counts on DS1 and DS2

Figure 5: Comparison with PNSP Algorithm