Mining Maximal Sequential Patterns without Candidate Maintenance

Artificial
Intelligence
Seminar
10 March 2016

Syméon
Malengreau
Mhd Ayad
Aldayeh

Table of contents

Authors presentation

- Reference
- Authors

Concepts

- Sequence database
- Sequential pattern
- Closed Sequential pattern
- Maximal Sequential pattern

Algorithm

- PrefixSpan
- MaxSP

Measures

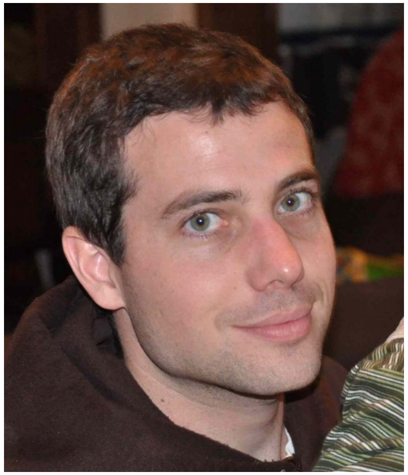
Reference : Pierre Schaus

Let's take a comparison for the article's authors

Pierre Schaus is a UCL professor, you might know him

H-index : 10

Citations : 283



Authors

Vincent S. Tseng

Philippe Fournier-Viger

Cheng-Wei Wu

Authors : Vincent S. Tseng

Ph.D. in Computer Science

Professor, Dept. Computer Science, National Chiao Tung University, Taiwan

H-index : 32

- 3,2x more

Citations : 3308

- 11,7x more

Authors : Philippe Fournier-Viger

- Ph.D. in Computer Science
- Associate Professor, Harbin Institute of Technology, Shenzhen Graduate School
- He created the SPMF library
- H-index : 17
  - 1,7x more
- Citations : 712
  - 2,5x more

Authors : Cheng-Wei Wu

H-index : 12

1,2x more

Citations : 518

1,83x more

About the article

We can trust the author, but

- We found errors in some examples,
- Some part of the articles were mathematicaly not clear enough, and some definition were simply wrong,
- In the end, the article was a complete mess, we cannot understand who accepted to publish it !

After some search else where we made it possible to understand the content of the articles. But based only on it, it wouldn't have be possible.

It was published in a book : *Advanced Data Mining and Applications*

This particular article has (when the slides where made) 10 citations

First Concepts

Sequence Database

Sequential pattern

Closed sequential pattern

Maximal sequential pattern

Sequence Database

A sequence database consist of :

- A set of items
  - {1, 2, 3, 4, ..., N}
- Itemset (set of item, distinct and unordered)
  - {1, 2, 3, 5} or {4, 5} or {3, 7} or ...
- Sequence (set of itemsets)
  - <{1,2}, {3}, {5}> or <{4}, {6}> or ...

The sequence database is a set of sequences

What do theses concepts represents ?

Sequence Database : Illustration

Let's take as an example a book

- Set of item → The words
  - {He, nice, the, is, a, guy, sun, shine, ...}
- Itemset → A sentence (where words are distinct and unordered)
  - {He, a, nice, guy, is}
  - {The, sun, shine, in, the, sky}
- Sequence → A chapter of the book
- Sequence Database → The book

Sequential pattern

Synonyms are *sub-sequence* **or** *frequent* ***sequence***

It is a sequence of item that appears a certain number of time, that number is the *minimum support threshold* (or *minsup*)

Sequence database

- <{1,2},{3},{4},{6}>
- <{2},{5},{6}>
- <{1,3},{5},{6}>

With minsup = 2, some examples of sequential pattern

- {5},{6}
- {1}
- {3},{6}
- ...

Closed sequential pattern

A closed sequential pattern is a sequential pattern not included in another closed pattern <u>having the same frequency</u>.

<{1},{1 2 3},{1 3},{4},{3 6}>

<{1 4},{3},{2 3},{1 5}>

<{5 6},{1 2},{4 6},{3},{2}>

<{5},{7},{1 6},{3},{2},{3}>

With support 2 (or 2/4 entry → 50 %), here are some closed sequential pattern

{1}, {3}	100 % (4/4)
{1}, {3}, {2}	75 % (3/4)
{5}, {1}, {3}, {2}	50% (2/4)
{5}	75 % (3/4)

And this one is <u>NOT</u>

{1} 100 % (4/4)

Maximal sequential pattern

The same as the closed sequential pattern, but if one sequence is in another one, it is not maximal.

Interesting property :

- You can derive every closed sequential patterns from the maximal sequential patterns

Question 1 : Closed and Maximal pattern

Considering the database

1. Which one of these is not a closed sequential pattern ? Why ?
   - → <{b},{f}>
   - → <{b}>
   - → <{a,b}>
   - → <{a},{b},{e}>
2. Which one of these is a maximal sequential pattern ? Why ?
   - → <{a},{e}>
   - → <{b}, {b}>
   - → <{b}, {f}, {e}>
   - → <{a}, {f}>

<{a,b},{c},{f,g},{g},{e}>

<{a,d},{c},{b},{a,b,e,f}>

<{a},{b},{f,g},{e}>

<{b},{f,g}>

MaxSP Algorithm

Find the maximal sequential pattern

It is build uppon the PrefixSpan Algorithm

Why the need for a new algorithm ?

→ Less memory usage

→ Faster to find sequential pattern

PrefixSpan : Start

First let's explain the PrefixSpan Algorithm

It's the most efficient pattern mining algorithm

We start with a sequence database

<{1},{1 2 3},{1 3},{4},{3 6}>

<{1 4},{3},{2 3},{1 5}>

<{5 6},{1 2},{4 6},{3},{2}>

<{5},{7},{1 6},{3},{2},{3}>

PrefixSpan : Pattern-growth

It works by pattern-growth, which does not generate any candidates (saving memory)

1. Scan : *Calculate support for each item and existing itemset*
2. Output : *Output item that have enough support*
3. Projection : *Recursively project the database with every item that have enough support*

PrefixSpan : Scan

MinSup 75 % (3)

- <{1},{1 2 3},{1 3},{4},{3 6}>
- <{1 4},{3},{2 3},{1 5}>
- <{5 6},{1 2},{4 6},{3},{2}>
- <{5},{7},{1 6},{3},{2},{3}>

Item	Support
1	
2	
3	
4	
5	
6	
7	

PrefixSpan : Scan

MinSup 75 % (3)

- <{1},{1 2 3},{1 3},{4},{3 6}>
- <{1 4},{3},{2 3},{1 5}>
- <{5 6},{1 2},{4 6},{3},{2}>
- <{5},{7},{1 6},{3},{2},{3}>

Item	Support
1	100 % (4)
2	
3	
4	
5	
6	
7	

PrefixSpan : Scan

MinSup 75 % (3)

- <{ },{ 2 3},{ 3},{4},{3 6}>
- <{ 4},{3},{2 3},{ 5}>
- <{5 6},{ 2},{4 6},{3},{2}>
- <{5},{7},{ 6},{3},{2},{3}>

Item	Support
1	100 % (4)
2	100 % (4)
3	
4	
5	
6	
7	

PrefixSpan : Scan

MinSup 75 % (3)

- <{ },{ 3},{ 3},{4},{3 6}>
- <{ 4},{3},{ 3},{ 5}>
- <{5 6},{ },{4 6},{3},{ }>
- <{5},{7},{ 6},{3},{ },{3}>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	
5	
6	
7	

PrefixSpan : Scan

MinSup 75 % (3)

- <{ },{ },{ },{4},{ 6}>
- <{ 4},{ },{ },{ 5}>
- <{5 6},{ },{4 6},{ },{ }>
- <{5},{7},{ 6},{ },{ },{ }>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	
6	
7	

PrefixSpan : Scan

MinSup 75 % (3)

- <{ },{ },{ },{ },{ 6}>
- <{ },{ },{ },{ 5}>
- <{5 6},{ },{ 6},{ }, { }>
- <{5},{7},{ 6},{ },{ }, { }>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	75 % (3)
6	
7	

PrefixSpan : Scan

MinSup 75 % (3)

- <{ },{ },{ },{ },{ 6}>
- <{ },{ },{ },{ }>
- <{ 6},{ },{ 6},{ }, { }>
- <{ },{7},{ 6},{ },{ }, { }>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	75 % (3)
6	75 % (3)
7	

PrefixSpan : Scan

MinSup 75 % (3)

<{ },{ },{ },{ },{ }>

<{ },{ },{ },{ }>

<{ },{ },{ },{ }, { }>

<{ },{7},{ },{ },{ }, { }>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	75 % (3)
6	75 % (3)
7	25 % (1)

PrefixSpan : Scan

MinSup 75 % (3)

- <{1},{1 2 3},{1 3},{4},{3 6}>
- <{1 4},{3},{2 3},{1 5}>
- <{5 6},{1 2},{4 6},{3},{2}>
- <{5},{7},{1 6},{3},{2},{3}>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	75 % (3)
6	75 % (3)
7	25 % (1)

PrefixSpan : Scan

MinSup 75 % (3)

- <{1},{1 2 3},{1 3},{4},{3 6}>
- <{1 4},{3},{2 3},{1 5}>
- <{5 6},{1 2},{4 6},{3},{2}>
- <{5},{7},{1 6},{3},{2},{3}>

It has to be done for itemsets too…

- {1 2 3}, {1 2}, {1 3}, {3 6}, {1 4}, {2 3},
- {1 5}, {5 6}, {4 6}, {1 6}

Item	Support	Item	Support
1	100 %	1 3	50 %
2	100 %	4 6	25 %
3	100 %	3 6	25 %
4	75 %	1 4	25 %
5	75 %	2 3	25 %
6	75 %	1 5	25 %
7	25 %	5 6	25 %
1 2 3	25 %	1 6	25 %
1 2	50 %		

PrefixSpan : Scan

We take each item with the support ≥ minsup, and output it as a sequence with one item.

Here the output :

- <{1}>
- <{2}>
- <{3}>
- <{4}>
- <{5}>
- <{6}>

New Concept : Projection

We need to define a new concept → *Projection*

If we project a sequence <{1},{2},{3}> by a prefix <{1}>, we take the part of the sequence that follow the prefix. Here <{2},{3}>

Some examples :

<{1},{2},{1},{3}> by <{1}> → <{2},{1},{3}>

<{3},{4},{5}> by <{3},{4}> → <{5}>

<{1}, {3, 4}, {5}, {6}> by <{3}> → <{5},{6}>

<{2}, {3}, {4}, {5}, {6}> by <{3}, {5}> → <{6}>

→ Projecting a database, means to project every sequence

PrefixSpan : Projection

First we will output the result we found with enough support.

Then we will recursively project the database with every of those items.

Lets take the result we have found so far to make it clearer.

We keep *minsup* of 75 % (3)

PrefixSpan : Projection

1. Scan Database

<{1},{1 2 3},{1 3},{4},{3 6}>

<{1 4},{3},{2 3},{1 5}>

<{5 6},{1 2},{4 6},{3},{2}>

<{5},{7},{1 6},{3},{2},{3}>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	75 % (3)
6	75 % (3)
7	25 % (1)

PrefixSpan : Projection

2. Output first item

<{1}> → Support : 100 % (4)

3. Project first item (2)

<{1},{1 2 3},{1 3},{4},{3 6}>

<{1 4},{3},{2 3},{1 5}>

<{5 6},{1 2},{4 6},{3},{2}>

<{5},{7},{1 6},{3},{2},{3}>

Item	Support
1	100 % (4)
2	100 % (4)
3	100 % (4)
4	75 % (3)
5	75 % (3)
6	75 % (3)
7	25 % (1)

PrefixSpan : Projection

1. Scan Again

<{1 2 3},{1 3},{4},{3 6}>

<{3},{2 3},{1 5}>

<{4 6},{3},{2}>

<{3},{2},{3}>

Item	Support
1	50 % (2)
2	100 % (4)
3	100 % (4)
4	50 % (2)
5	25 % (1)
6	50 % (2)
7	25 % (1)

PrefixSpan : Projection

2. Output the sequence

<{1},{2}> → Support : 100 % (4)

3. Project first item (2)

<{1 2 3},{1 3},{4},{3 6}>

<{3},{2 3},{1 5}>

<{4 6},{3},{2}>

<{3},{2},{3}>

Item	Support
1	50 % (2)
2	100 % (4)
3	100 % (4)
4	50 % (2)
5	25 % (1)
6	50 % (2)
7	25 % (1)

PrefixSpan : Projection

1. Scan again

<{1 3},{4},{3 6}>

<{1 5}>

<{3}>

Item	Support
1	50 % (2)
2	0 % (0)
3	50 % (2)
4	25 % (1)
5	25 % (1)
6	25 % (1)
7	0 % (0)

PrefixSpan : Projection

Operation are over

We continue with other items

Item	Support
1	50 % (2)
2	0 % (0)
3	50 % (2)
4	25 % (1)
5	25 % (1)
6	25 % (1)
7	0 % (0)

PrefixSpan : Projection

2. Output the sequence

<{1},{3}> → Support : 100 % (4)

3. Project second item (2)

<{1 2 3},{1 3},{4},{3 6}>

<{3},{2 3},{1 5}>

<{4 6},{3},{2}>

<{3},{2},{3}>

Item	Support
1	50 % (2)
~~2~~	~~100 % (4)~~
3	100 % (4)
4	50 % (2)
5	25 % (1)
6	50 % (2)
7	25 % (1)

PrefixSpan : Projection

1. Scan again

<{1 3},{4},{3 6}>

<{2 3},{1 5}>

<{2}>

<{2},{3}>

Item	Support
1	50 % (2)
2	75 % (3)
3	75 % (3)
4	25 % (1)
5	25 % (1)
6	25 % (1)
7	0 % (0)

PrefixSpan : Projection

2. Output the sequence

<{1}, {3}, {2}> → Support : 75 % (3)

3. Project first item (2)

<{1 3},{4},{3 6}>

<{2 3},{1 5}>

<{2}>

<{2},{3}>

Item	Support
1	50 % (2)
2	75 % (3)
3	75 % (3)
4	25 % (1)
5	25 % (1)
6	25 % (1)
7	0 % (0)

PrefixSpan : Projection

Do all recursive projection would take a certain amount of times

We will skip the projection, as the process repeat itself

Pattern	Support (≥ 75%)	Pattern	Support (≥ 75%)
<{1}>	100 %	<{3},{2}>	75 %
<{1},{2}>	100 %	<{3},{3}>	75 %
<{1},{3}>	100 %	<{4}>	75 %
<{1},{3},{2}>	75 %	<{4},{3}>	75 %
<{1},{3},{3}>	75 %	<{5}>	75 %
<{2}>	100 %	<{6}>	75 %
<{2},{3}>	75 %		
<{3}>	100 %		

Question 2 : Projection

Considering the database

1. What is the result of the projection of <{b},{f}> on the database ?
2. In previous sequence, which are not closed and which are maximal ?

<{a,b},{c},{f,g},{g},{e}>

<{a,d},{c},{b},{a,b,e,f}>

<{a},{b},{f,g},{e}>

<{b},{f,g}>

Pattern	Support (≥ 75%)	Pattern	Support (≥ 75%)
<{1}>	100 %	<{3},{2}>	75 %
<{1},{2}>	100 %	<{3},{3}>	75 %
<{1},{3}>	100 %	<{4}>	75 %
<{1},{3},{2}>	75 %	<{4},{3}>	75 %
<{1},{3},{3}>	75 %	<{5}>	75 %
<{2}>	100 %	<{6}>	75 %
<{2},{3}>	75 %		
<{3}>	100 %		

PrefixSpan : Projection

Here are the closed.

Pattern	Support (≥ 75%)	Pattern	Support (≥ 75%)
<{1}>	100 %	<{3},{2}>	75 %
<{1},{2}>	100 %	<{3},{3}>	75 %
<{1},{3}>	100 %	<{4}>	75 %
<{1},{3},{2}>	75 %	<{4},{3}>	75 %
<{1},{3},{3}>	75 %	<{5}>	75 %
<{2}>	100 %	<{6}>	75 %
<{2},{3}>	75 %		
<{3}>	100 %		

PrefixSpan : Projection

Here are the closed and maximal.

Pattern	Support (≥ 75%)	Pattern	Support (≥ 75%)
<{1}>	100 %	<{3},{2}>	75 %
<{1},{2}>	100 %	<{3},{3}>	75 %
<{1},{3}>	100 %	<{4}>	75 %
<{1},{3},{2}>	75 %	<{4},{3}>	75 %
<{1},{3},{3}>	75 %	<{5}>	75 %
<{2}>	100 %	<{6}>	75 %
<{2},{3}>	75 %		
<{3}>	100 %		

MaxSp : Basic Idea

MaxSP extends the PrefixSpan

A naïve approach would be to keep all sequence in memory and to check every time a new sequence arrives if it is maximal.

That is CloSpan

→ Inefficient

→ Memory consuming

MaxSp : Basic Idea

The question is

How to know if a pattern P is maximal, without maintening pattern in memory ?

The solution : Can P be extended by appending items ?

→ YES ? It isn't a maximal sequential pattern

Two check :

1. Maximal backward extension check
2. Maximal forward extension check

MaxSp : Maximal backward extension check

With sequence database and minsup = 75 % (3/4)

<{1}, {2}, {3}, {4}>

<{1}, {3}, {5}, {2}>

<{1}, {2}, {5, 6}>

<{1}, {4, 5}, {4}>

1. Maximal backward extension check

MaxSp : Maximal forward extension check

With the same idea but going in the other direction

MaxSp Algorithm

Concretly, how does one implement MaxSP based on PrefixSpan ?

Question 3 : Checks