Sequence Data Mining

Young-Rae Cho, Ph.D.
Associate Professor
Division of Software / Division of Digital Healthcare
Yonsei University – Mirae Campus

Topics

- **Single Sequence Mining**
  - Frequent string pattern mining
    - Finding substrings that frequently occur in a single sequence

ACTTCGATGGAGCCAGTCGCGAAATTCGACTAGATCG

- **Sequence Dataset Mining**
  - Frequent string/sequence pattern mining
    - Finding substrings/sub-sequences that frequently occur among sequences
  - Sequence data clustering
    - Grouping similar sequences
  - Sequence data classification
    - Classifying a new sequence

id	sequence
1	ACTTCG
2	GGAGC
3	CGTCGAT
4	ATCGATCGC
5	TCGACT
6	AGATCGC

Applications

- **Examples**
  - Customer shopping sequential patterns

    e.g., First buy a computer, then a CD-ROM, and then a printer within 3 months
  - Stock market changes
  - Web-log patterns
  - Medical treatment records
  - Gene or protein sequences
  - Natural disaster records

- **Challenges**
  - Finding the complete set satisfying the minimum support threshold
  - Developing efficient and scalable algorithms
  - Incorporating various kinds of user-specific constraints

Frequent String Pattern Minding

- **Problem Definition**
  - Finding all substrings that occur frequently among multiple sequences
  - Sequence: an ordered list of items
  - Substring of a sequence x: a part of consecutive items from x
  - Frequent string pattern ?
    - Suppose minimum support of 75%

```
T1= cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
T2= agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
T3= aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaatttt
T4= agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtgcgtataca
T5= ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

    - “acgtacgt” occurs in 4 out of 5 sequences (80%) – more frequent than minimum support

Properties of String Patterns

- **Properties**
  - Anti-monotonic property → Apriori algorithm
  - If a string S is not frequent, then none of the super-strings of S are frequent
  - Example: If < a b > is infrequent, < a b c > and < d a b > are infrequent too.

- **Process to Solve Frequent String Pattern Mining**
  - Iteratively increase length of substrings that occur more frequently than minimum support

- **Problems?**
  - Selective joining?
  - Apriori pruning?
  - Support computing?

Frequent Sequence Pattern Minding

❑ Scope

- Transaction data → Sequential transaction data
- Frequent itemset patterns → (Frequent) Sequential patterns

❑ Problem Definitions

- Sequence: an ordered list of items
  - e.g., x = <a c f e c d>
- Sub-sequence of x: an ordered sequence of items from x
  - Not necessarily consecutive (different from a substring)
  - <a e c> <c f d> <a c> <c e c> <a f c d> ~~<c f c e>~~
- Frequent sequence pattern ?
  - Suppose minimum support of 75%
  - < a c d > is one of the frequent sequence patterns
  - Any others?

SID	sequences
10	<acadcf>
20	<abcaed>
30	<efadfcb>
40	<egafcbd>

Frequent Sequence Pattern Minding – cont’

❑ Extended Problem Definitions

- Consider “or” at a single position
- Sequence: an ordered list of elements and each element is a set of items,
  - e.g., x = <(ab) c (df) (ef) d>
- Sub-sequence of x
  - <(ab) d> <a c (ef)> <a f e> ~~<(ab) e f>~~ <b f d> <c d f>
- Frequent sequence pattern
  - Suppose minimum support of 75%

SID	sequences
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

  - <(ab) c> is one of the frequent sequence patterns
  - Any others?

Properties of Sequence Patterns

❑ Properties

- Anti-monotonic property → Apriori algorithm
- If a sequence S is not frequent, then none of the super-sequences of S are frequent

❑ Example

- If <(ab)d> is infrequent, so do <(ab)de> and <(abc)d> and <a(ab)d>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup = 75%

GSP (Generalized Sequential Pattern mining) Algorithm

Process

1) Initially, find all frequent length-1 sequences (= frequent 1-itemset)
2) Generate candidate length-(k+1) sequences from frequent length-k sequences
3) Count support for each candidate sequence to select frequent sequences
4) Repeat (2) and (3) until no frequent sequence or no candidate is found

Example

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup = 75%

Candidate & Frequent Length-1 Sequences

- <a>, <b>, <c>, <d>, <e>, <f>, <g>

GSP Algorithm – Length-2 Sequences

❑ **Candidate & Frequent Length-2 Sequences**

	<a>	<b>	<c>	<e>	<f>
<a>	<aa>	<ab> <ba>	<ac> <ca>	<ae> <ea>	<af> <fa>
<b>		<bb>	<bc> <cb>	<be> <eb>	<bf> <fb>
<c>			<cc>	<ce> <ec>	<cf> <fc>
<e>				<ee>	<ef> <fe>
<f>					<ff>

	<a>	<b>	<c>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ae)>	<(af)>
<b>			<(bc)>	<(be)>	<(bf)>
<c>				<(ce)>	<(cf)>
<e>					<(ef)>
<f>					

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

GSP Algorithm – Length-3 Sequences

❑ Candidate & Frequent Length-3 Sequences

	<ab>	<ac>	<bc>	<(ab)>
<ab>	<aab> <abb>	<abc> <acb>	~~<abc>~~	<(ab)b> <a(ab)>
<ac>		<aac> <acc>	~~<abc>~~ <bac>	<(ab)c>
<bc>			<bbc> <bcc>	~~<(ab)c>~~
<(ab)>				<(ab)(ab)>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Summary of GSP Algorithm

- **Strength**
  - Apriori pruning

- **Weakness**
  - Generates a huge set of candidate sequences
  - Requires multiple scans of database
  - Inefficient for mining long sequential patterns

- **References**

  - Agrawal, R. and Srikant, R., "Mining sequential patterns." In Proceedings of ICDE (1995)
  - Srikant, R. and Agrawal, R., "Mining sequential patterns: Generalizations and performance improvements." In Proceedings of EDBT (1996)

Prefix

- **General Definition**
  - When a sequence S is a list of items,
  - S[1 .. j] is a prefix of a string S[1 .. n] where $j \le n$
  - X is a prefix of Y if X·Z=Y for some string Z

- **Extended Definition**
  - When a sequence is a list of elements and each element is a set of items,
  - Sort the items of each element in an alphabetic order
  - Given $x=<e_1, e_2, \ldots , e_n>$, $y=<e'_1, e'_2, \ldots , e'_m>$ $(m \le n)$ is a **prefix** of x if and only if
    1) $e_i = e'_i$ for $(i \le m-1)$
    2) $e'_m \subseteq e_m$
    3) all items in $(e_m - e'_m)$ are alphabetically after those in e'_m
  - Examples: <a>, <aa>, <a(ab)> and <a(abc)> are prefixes of <a(abc)(ac)d(cf)>

PrefixSpan (Prefix-projected Sequential Pattern mining) Algorithm

Main Idea

- Keep track of prefixes (instead of all candidate sequences) from the sequence database
- Project their suffixes into projected databases

Projected Database

- A set of maximal-length suffixes with a given prefix

Example

- <a>-projected database has
  - <(abc)(ac)d(cf)>
  - <(_b)a(bc)(ae)>
  - <(_b)(df)cb>
  - <(_f)cbc>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

PrefixSpan Algorithm – cont'

❑ Process

1) Find all frequent length-1 sequences
   - <a>, <b>, <c>, <e>, <f>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

2) Divide search space to
   - Maximal-length suffixes of a prefix ending with <a>
   - Maximal-length suffixes of a prefix ending with <b>
   - Maximal-length suffixes of a prefix ending with <c>
   - Maximal-length suffixes of a prefix ending with <e>
   - Maximal-length suffixes of a prefix ending with <f>
3) Construct a projected database for each search space
4) Find sequential patterns recursively

PrefixSpan Algorithm – cont’

❑ Projected Databases in the 1st Recursion

min_sup = 75%

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

pattern	projected database
<a>	<(abc)(ac)d(cf)>, <(_b)a(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
<b>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>, <c>
<c>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<e>	<(_f)(ab)(df)cb>, <g(af)cbc>
<f>	<(ab)(df)cb>, <cbc>

PrefixSpan Algorithm – cont'

- **Projected Databases in the 2nd Recursion**

pattern	projected database
<a>	<(abc)(ac)d(cf)>, <(_b)a(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
<b>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>, <c>
<c>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<e>	<(_f)(ab)(df)cb>, <g(af)cbc>
<f>	<(ab)(df)cb>, <cbc>

pattern	projected database
<ab>	<(_c)(ac)d(cf)>, <(_c)(ae)>, <c>
<ac>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<(ab)>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>

PrefixSpan Algorithm – cont'

❑ Projected Databases in the 2nd Recursion

pattern	projected database
<ab>	<(_c)(ac)d(cf)>, <(_c)(ae)>, <c>
<ac>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<(ab)>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>

pattern	projected database
<(ab)c>	<d(cf)>, <(ae)>, <b>

Summary of PrefixSpan Algorithm

- **Strength**
  - Efficient (major cost is to construct projected databases)
  - Project databases keep shrinking rapidly

- **Weakness**
  - Searching frequency redundantly

- **References**
  - Pei, J., et al., "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth." In Proceedings of ICDE (2001)

Questions?

- Lecture Slides on the Course Website, "https://ads.yonsei.ac.kr/faculty/data_mining"