Frequent Pattern Mining

Albert Bifet

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

May 2012

COMP423A/COMP523A Data Stream Mining

Outline

1. Introduction
2. Stream Algorithmics
3. Concept drift
4. Evaluation
5. Classification
6. Ensemble Methods
7. Regression
8. Clustering
9. **Frequent Pattern Mining**
10. Distributed Streaming

Data Streams

Big Data & Real Time

Frequent Patterns

Suppose $\mathcal{D}$ is a dataset of patterns, $t \in \mathcal{D}$, and *min_sup* is a constant.

Frequent Patterns

Suppose $\mathcal{D}$ is a dataset of patterns, $t \in \mathcal{D}$, and *min_sup* is a constant.

Definition

Support (t): number of patterns in $\mathcal{D}$ that are superpatterns of t.

Frequent Patterns

Suppose $\mathcal{D}$ is a dataset of patterns, $t \in \mathcal{D}$, and *min_sup* is a constant.

Definition

Support (t): number of patterns in $\mathcal{D}$ that are superpatterns of t.

Definition

Pattern t is *frequent* if *Support* $(t) \geq$ *min_sup*.

Frequent Patterns

Suppose $\mathcal{D}$ is a dataset of patterns, $t \in \mathcal{D}$, and *min_sup* is a constant.

Definition

Support (t): number of patterns in $\mathcal{D}$ that are superpatterns of t.

Definition

Pattern t is *frequent* if *Support* $(t) \geq$ *min_sup*.

Frequent Subpattern Problem

Given $\mathcal{D}$ and *min_sup*, find all frequent subpatterns of patterns in $\mathcal{D}$.

Pattern Mining

Dataset Example

Document	Patterns
d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**
d1,d2,d3,d4,d5,d6	c
d1,d2,d3,d4,d5	e,ce
d1,d3,d4,d5	a,ac,ae,ace
d1,d3,d5,d6	b,bc
d2,d4,d5,d6	d,cd
d1,d3,d5	ab,abc,abe be,bce,abce
d2,d4,d5	de,cde

minimal support = 3

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent
6	c
5	e,ce
4	a,ac,ae,ace
4	b,bc
4	d,cd
3	ab,abc,abe be,bce,abce
3	de,cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed
6	c	c	c
5	e,ce	e	ce
4	a,ac,ae,ace	a	ace
4	b,bc	b	bc
4	d,cd	d	cd
3	ab,abc,abe	ab	
	be,bce,abce	be	abce
3	de,cde	de	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**	**Gen**	**Closed**	**Max**
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe be,bce,abce	ab be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

e → ce

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

$a \rightarrow ace$

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**	**Gen**	**Closed**	**Max**
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe be,bce,abce	ab be	abce	abce
3	de,cde	de	cde	cde

Closed Patterns

Usually, there are too many frequent patterns. We can compute a smaller set, while keeping the same information.

Example

A set of 1000 items, has $2^{1000} \approx 10^{301}$ subsets, that is more than the number of atoms in the universe $\approx 10^{79}$

Closed Patterns

A priori property

If t' is a subpattern of t, then $\mathit{Support}(t') \geq \mathit{Support}(t)$.

Definition

A frequent pattern t is *closed* if none of its proper superpatterns has the same support as it has.

Frequent subpatterns and their supports can be generated from closed patterns.

Maximal Patterns

Definition

A frequent pattern t is *maximal* if none of its proper superpatterns is frequent.

Frequent subpatterns can be generated from maximal patterns, but not with their support.

All maximal patterns are closed, but not all closed patterns are maximal.

Non streaming frequent itemset miners

Representation:

- Horizontal layout

T1: a, b, c
T2: b, c, e
T3: b, d, e

- Vertical layout

a: 1 0 0
b: 1 1 1
c: 1 1 0

Search:

- Breadth-first (levelwise): Apriori
- Depth-first: Eclat, FP-Growth

The Apriori Algorithm

APRIORI ALGORITHM

```
Initialize the item set size k = 1
Start with single element sets
Prune the non-frequent ones
while there are frequent item sets
      do create candidates with one item more
         Prune the non-frequent ones
         Increment the item set size k = k + 1

Output: the frequent item sets
```

The Eclat Algorithm

Depth-First Search

- divide-and-conquer scheme : the problem is processed by splitting it into smaller subproblems, which are then processed recursively
  - **conditional database for the prefix a**
    - transactions that contain a
  - **conditional database for item sets without a**
    - transactions that not contain a
- Vertical representation
- Support counting is done by intersecting lists of transaction identifiers

The FP-Growth Algorithm

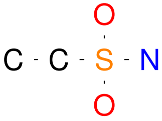
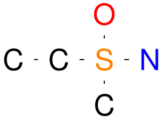
Depth-First Search

- divide-and-conquer scheme : the problem is processed by splitting it into smaller subproblems, which are then processed recursively
  - **conditional database for the prefix a**
    - transactions that contain a
  - **conditional database for item sets without a**
    - transactions that not contain a
- Vertical and Horizontal representation : FP-Tree
  - prefix tree with links between nodes that correspond to the same item
- Support counting is done using FP-Tree

Mining Graph Data

Problem

Given a data set of graphs, find frequent graphs.

Transaction Id	Graph
1	
2	
3	

The gSpan Algorithm

GSPAN(g, D, *min_sup*, S)

Input: A graph g, a graph dataset D, *min_sup*.
Output: The frequent graph set S.

```
if g ≠ min(g)
    then return S
insert g into S
update support counter structure
C ← ∅
for each g′ that can be right-most
        extended from g in one step
      do if support(g) ≥ min_sup
            then insert g′ into C
for each g′ in C
      do S ← GSPAN(g′, D, min_sup, S)
return S
```

Mining Patterns over Data Streams

Requirements: fast, use small amount of memory and adaptive

- Type:
  - Exact
  - Approximate
- Per batch, per transaction
- Incremental, Sliding Window, Adaptive
- Frequent, Closed, Maximal patterns

LOSSYCOUNTING

- Extension of LOSSYCOUNTING to Itemsets
- Keeps a structure with tuples $(X, \overline{freq}(X), error(X))$
- For each batch, to update an itemset:
  - Add the frequency of X in the batch to $\overline{freq}(X)$
  - If $\overline{freq}(X) + error(X) < bucketID$, delete this itemset
  - If the frequency of X in the batch in the batch is at least β, add a new tuple with $error(X) = bucketID - \beta$
- Uses an implementation based in :
  - Buffer: stores incoming transaction
  - Trie: forest of prefix trees
  - SetGen: generates itemsets supported in the current batch using apriori

Moment

- Computes **closed** frequents itemsets in a sliding window
- Uses Closed Enumeration Tree
- Uses 4 type of Nodes:
  - Closed Nodes
  - Intermediate Nodes
  - Unpromising Gateway Nodes
  - Infrequent Gateway Nodes
- Adding transactions: closed items remains closed
- Removing transactions: infrequent items remains infrequent

FP-Stream

- Mining Frequent Itemsets at Multiple Time Granularities
- Based in FP-Growth
- Maintains
  - pattern tree
  - tilted-time window
- Allows to answer time-sensitive queries
- Places greater information to recent data
- Drawback: time and memory complexity

Tree and Graph Mining: Dealing with time changes

- Keep a window on recent stream elements
  - Actually, just its lattice of closed sets!
- Keep track of number of closed patterns in lattice, N
- Use some change detector on N
- When change is detected:
  - Drop stale part of the window
  - Update lattice to reflect this deletion, using deletion rule

Alternatively, sliding window of some fixed size

Graph Coresets

Coreset of a set P with respect to some problem

Small subset that approximates the original set P.

- Solving the problem for the coreset provides an approximate solution for the problem on P.

Graph Coresets

Coreset of a set P with respect to some problem

Small subset that approximates the original set P.

- Solving the problem for the coreset provides an approximate solution for the problem on P.

δ-tolerance Closed Graph

A graph g is δ-*tolerance closed* if none of its proper frequent supergraphs has a weighted support $\geq (1 - \delta) \cdot \mathit{support}(g)$.

- Maximal graph: 1-tolerance closed graph
- Closed graph: 0-tolerance closed graph.

Graph Coresets

Relative support of a closed graph

Support of a graph minus the relative support of its closed supergraphs.

- The sum of the closed supergraphs' relative supports of a graph and its relative support is equal to its own support.

Graph Coresets

Relative support of a closed graph

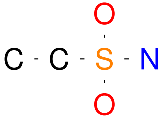
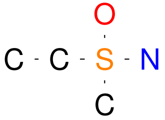
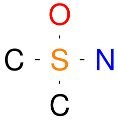
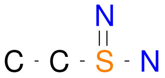
Support of a graph minus the relative support of its closed supergraphs.

- The sum of the closed supergraphs' relative supports of a graph and its relative support is equal to its own support.

(s, δ)-coreset for the problem of computing closed graphs

Weighted multiset of frequent δ-tolerance closed graphs with minimum support s using their relative support as a weight.

Graph Dataset

Transaction Id	Graph	Weight
1		1
2		1
3		1
4		1

Graph Coresets

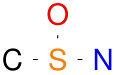
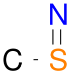
Graph	Relative Support	Support
C - C - S - N	3	3
	3	3
	3	3

Table : Example of a coreset with minimum support 50% and $\delta = 1$

Graph Coresets

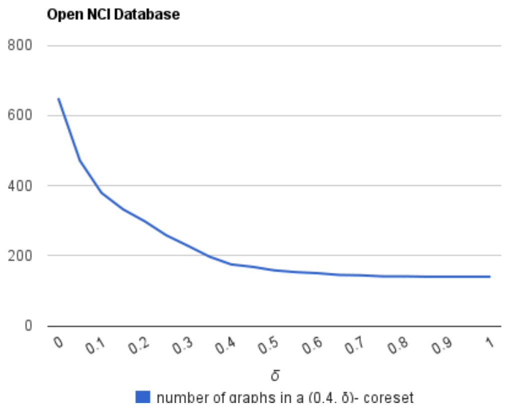


Figure : Number of graphs in a $(40\%, \delta)$-coreset for NCI.

IncGraphMiner

IncGraphMiner(D, min_sup)

Input: A graph dataset D, and min_sup.
Output: The frequent graph set G.

```
G ← ∅
for every batch b_t of graphs in D
    do C ← CoreSet(b_t, min_sup)
       G ← CoreSet(G ∪ C, min_sup)
return G
```

WinGraphMiner

WinGraphMiner(D, W, min_sup)

Input: A graph dataset D, a size window W and min_sup.
Output: The frequent graph set G.

$G \leftarrow \emptyset$
for every batch b_t of graphs in D
    do C $\leftarrow$ Coreset(b_t, min_sup)
        Store C in sliding window
        if sliding window is full
          then $\overline{R} \leftarrow$ Oldest C stored in sliding window,
                    negate all support values
          else $\overline{R} \leftarrow \emptyset$
        $G \leftarrow$ Coreset($G \cup C \cup \overline{R}$, min_sup)
return G

AdaGraphMiner

AdaGraphMiner(D, $Mode$, min_sup)

$G \leftarrow \emptyset$
Init ADWIN
for every batch b_t of graphs in D
    do $C \leftarrow$ Coreset(b_t, min_sup)
        $\overline{R} \leftarrow \emptyset$
        if Mode **is** Sliding Window
            then Store C in sliding window
                if ADWIN detected change
                    then $\overline{R} \leftarrow$ Batches to remove in sliding window with negative support
        $G \leftarrow$ Coreset($G \cup C \cup \overline{R}$, min_sup)
        if Mode **is** Sliding Window
            then Insert # closed graphs into ADWIN
            else for every g in G update g's ADWIN
return G

AdaGraphMiner

AdaGraphMiner(D, $Mode$, min_sup)

$G \leftarrow \emptyset$
Init ADWIN
for every batch b_t of graphs in D
  do $C \leftarrow$ Coreset(b_t, min_sup)
    $\overline{R} \leftarrow \emptyset$
      $G \leftarrow$ Coreset($G \cup C \cup \overline{R}$, min_sup)
      for every g in G update g's ADWIN
return G

AdaGraphMiner

AdaGraphMiner(D, $Mode$, min_sup)

```
G ← ∅
Init ADWIN
for every batch b_t of graphs in D
    do C ← CoreSet(b_t, min_sup)
        R̄ ← ∅
        if Mode is Sliding Window
          then Store C in sliding window
               if ADWIN detected change
                 then R̄ ← Batches to remove
                          in sliding window
                          with negative support
        G ← CoreSet(G ∪ C ∪ R̄, min_sup)
        if Mode is Sliding Window
          then Insert # closed graphs into ADWIN

return G
```