Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary

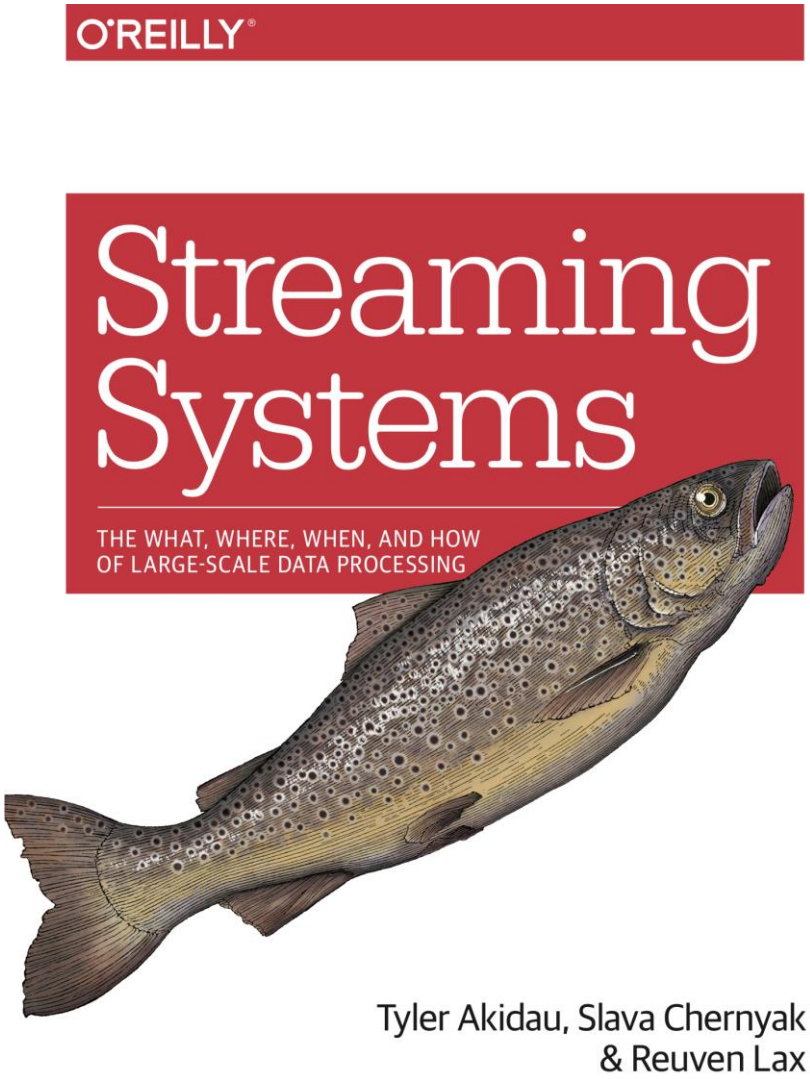
SKETCHES, FREQUENT PATTERNS AND SEQUENCES

Stream mining (SM)

Imre Lendák, PhD, Associate Professor
Chandresh Maurya, PhD, Assistant Professor

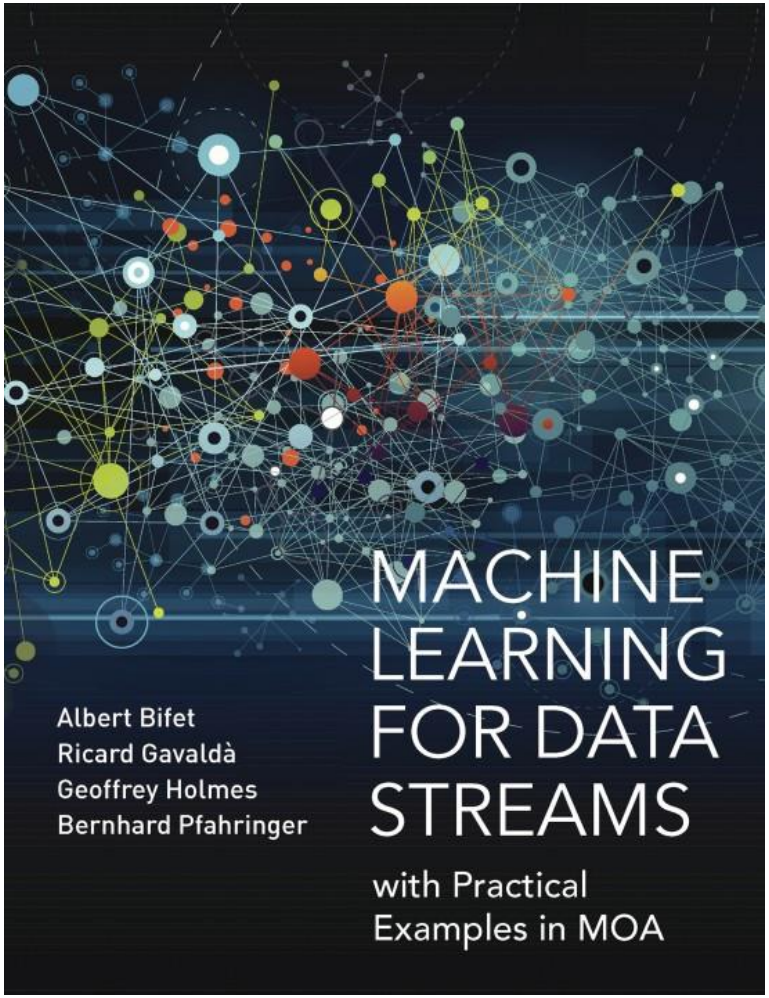
2020
Budapest, Hungary

Stream mining course status update

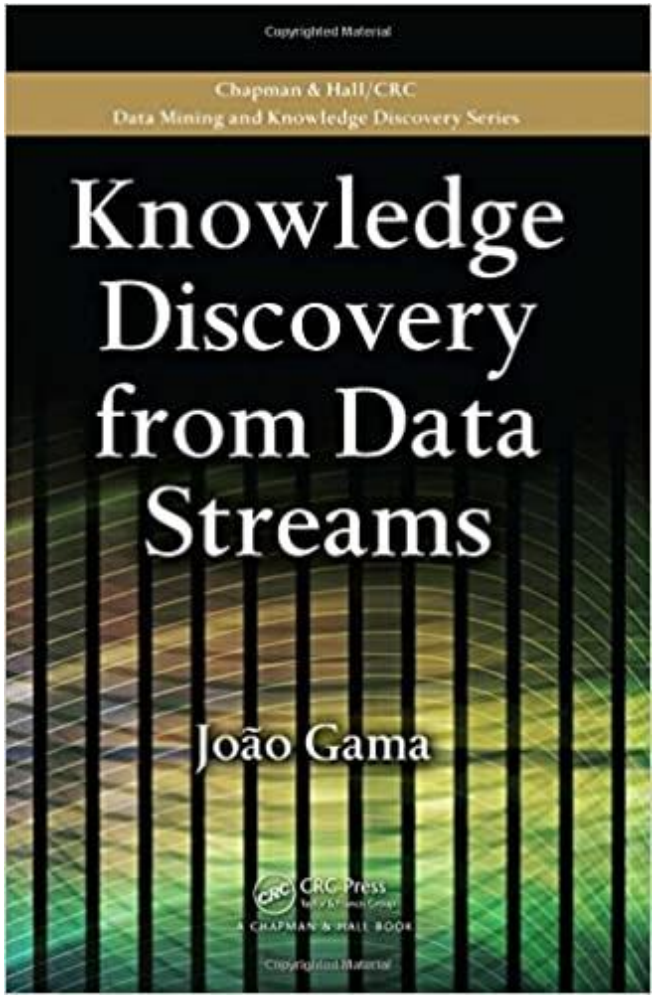


- Started with the Streaming Systems book by Google
- In the stream mining domain we covered:
  1. Sampling and filtering
  2. Distinct elements and moments
  3. Clustering
  4. Classification
- What comes next (maybe in different sequence)
  1. Frequent patterns
  2. Time series
  3. Novelty detection
  4. Change detection

Additional sources



https://mitpress.mit.edu/books/machine-learning-data-streams



https://www.amazon.com/Knowledge-Discovery-Streams-Chapman-Mining/dp/1439826110

SKETCHES

Stream mining axioms (re-iterated)

- **One pass** – each item in the data stream is observed only once
- **Low processing time** – per item processing times must be short
- **Low memory use** – per item memory use must be low
- **Anytime query execution** – the stream mining solution must be capable to answer queries at any moment and with limited latency
- **Non-stationary data streams** – stream mining solutions must be designed with input data change in mind

Sketches defined

- DEF: Set I consists of all records in a data stream
  - Data streams are unbounded data structures
  - It is (often) not feasible to store all elements of I in limited memory, neither random access, nor storage.
- DEF: An **item/record** is an element of set I
  - An item is stored in X units of memory
- DEF: A **sketch** is a (data structure, algorithm) pair which read a stream and store sufficient information to be able to answer one or more predefined queries about the data stream

Sketch creation and use

- **Initialize** → initialize the data structure and constants necessary for the sketch
- **Update** → called when a new item is observed in the data stream
  - Optionally modifies the underlying data structure of the sketch
- **Query** → return the current value of the pre-defined query
  - The return value is calculated based on the items read
  - A single sketch can implement multiple queries → multiple useful pieces of information can be extracted from the data structure maintained in the Update phase

Accuracy and confidence (re-visited)

- For accuracy value ε we define
  - Absolute or additive ε-approximation as
    $$\forall x\colon |f(x) - g(x)| < \varepsilon$$
  - Relative or multiplicative ε-approximation as
    $$\forall x\colon |f(x) - g(x)| < \varepsilon |f(x)|$$

- Confidence is defined as $1 - \delta$

- An (ε, δ)-approximation requires that an ε-approximation holds with confidence $1 - \delta$
  - For (0.1, 0.01) the approximation error should be less than 0.1 for 99% of calls

Sampling

RESERVOIR SAMPLING

```
1  Init(k):
2      create a reservoir (array[0...k − 1]) of up to k items
3      fill the reservoir with the first k items in the stream
4      t ← k        ▷ t counts the number of items seen so far
5  Update(x):
6      select a random number r between 0 and t − 1
7      if r < k
8          replace the rth item in the reservoir with item x
9      t ← t + 1
10 Query():
11     return the current reservoir
```

- **Name:** Reservoir Sampling
- **Init:** create data structure for k items
- **Update:** replace the r-th element in the reservoir with x (i.e. the new item)
- **Query:** return the contents of the reservoir reservoir

https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html

Counting total items – 1

MORRIS'S COUNTER

1 *Init*():
2 $c \leftarrow 0$
3 *Update*(*item*):
4 increment c with probability 2^{-c}
5 ▷ and do nothing with probability $1 - 2^{-c}$
6 *Query*():
7 **return** $2^c - 1$

- **Name:** Morris's counter
- **Init:** set count c to 0
- **Update:** increment c with a probability
- **Query:** return the estimation

https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html

Counting distinct items – 1

- Counting distinct elements → maintain a count of distinct (i.e. different) items observed in the data stream so far

LINEAR COUNTING

1 *Init*(D, ρ):
2 $\triangleright$ D is an upper bound on the number of distinct elements
3 $\triangleright$ $\rho > 0$ is a load factor
4 $s \leftarrow D/\rho$
5 choose a hash function h from items to $\{0, \ldots, s-1\}$
6 build a bit vector B of size s
7 *Update*(x):
8 $B[h(x)] \leftarrow 1$
9 *Query*():
10 let w be the fraction of 0s in B
11 **return** $s \cdot \ln(1/w)$

https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html

- **Name:** Linear Counting
- **Init:** choose hash function, init vector B
- **Update:** set B[h(x)]=1
  - Calculate hash of x
  - Use hash as index
  - Set value at index to 1
- **Query:** calculate & return current count

Counting distinct items – 2

PROBABILISTIC COUNTER

```
1  Init(D):
2      ▷ D is an upper bound on the number of distinct elements
3      L ← log D
4      choose a hash function h from items to {0, ..., L − 1}
5      p ← L
6  Update(x):
7      let i be such that h(x) in binary starts with 0^i 1
8      p ← min(p, i)
9  Query():
10     return 2^p/0.77351
```

Flajolet-Martin

- **Name:** Flajolet-Martin
- **Init:** set up the necessary constants
- **Update:**
  - Divide stream in $m = 2^b$ substreams
  - Use first b = log(m) bits of h(x) to choose substream
- **Query:** calculate & return value

https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html

Frequency problems

- Often it is necessary to separately count different types of items
- **DEF:** The **absolute frequency** of item x over (time) period t is the number of times x appeared (within period t)
- **DEF:** The **relative frequency** of item x over period t is the number of times it appeared divided by t, i.e. count(x)/t

- The **heavy hitter** problem: Given a threshold ϵ the set of heavy hitters consists of items whose relative frequency exceeds ϵ
- Naïve solution for the heavy hitter problem: use sampling and calculate relative frequency for the sample

Frequency problems

SPACESAVING

```
1  Init(k):
2      create an empty set S of pairs (item,count) able to hold up to k pairs
3  Update(x):
4      if item x is in S
5          increase its count by 1
6      else if S contains less than k entries
7          add (x, 1) to S
8      else
9          let (y, count) be an entry in S with lowest count
10         replace the entry (y, count) with (x, count + 1) in S
11 Query():
12     return the list of pairs (x, count(x)) currently in S
```

- **Init:** Creates empty data structure for a set of k (item, count) pairs
- **Update** (when item x is received):
  - Add (x,1) if below capacity
  - Increment (x, count) by one if x already ‘in’
  - Evict element (y, count) with lowest count and add (x, count+1)
- **Query:** return list (x, count(x))

https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html

Exponential histogram for sliding windows

- **DEF:** Sliding windows in the stream mining context allow the algorithm to consider only the last W items where W is the window size
  - Windows are usually created by item count W (i.e. the number of elements), but it is feasible to consider items received in time period T, where T can be a second, hour, day or other time period
- Naïve window implementation: keep a circular buffer for the last W items
  - When a new item is received, evict the oldest element → circular buffer
  - Uses memory complexity W
- The exponential histogram approach optimizes memory use to O(log W) memory

Exponential histogram

Bucket:	1011100	10100101	100010	11	10	1000
Capacity:	4	4	2	2	1	1
Timestamp:	$t-24$	$t-14$	$t-9$	$t-6$	$t-5$	$t-3$

EXPONENTIAL HISTOGRAMS

```
1   Init(k, W):
2       t ← 0
3       create a list of empty buckets
4   Update(b):
5       t ← t + 1
6       if b = 1    ▷ do nothing with 0s
7           let t be the current time
8           create a bucket of capacity 1 and timestamp t
9           i ← 0
10          while there are more than k buckets of capacity 2^i
11              merge the two oldest buckets of capacity 2^i into a
12                  bucket of capacity 2^(i+1): the timestamp of the new bucket
13                  is the largest (most recent) of their two timestamps
14              i ← i + 1
15          remove all buckets whose timestamp is ≤ t − W
16  Query():
17      return the sum of the capacities of all existing buckets
18          minus half the capacity of the oldest bucket
```

Mergeability

- **DEF:** A sketch is **mergeable** if sketches built from data streams D1,D2,…,Dn can be combined into a sketch of the same type which can correctly answer queries about the interleavings of the input streams
- Item frequency sketches are trivial to merge → the answer should be the same for all interleavings
- Item counting sketches are also trivial to merge → the integer counts can be added
- Other possible operators for merging sketches:
  - OR for linear counters
  - MIN for Cohen and Flajolet-Martin

FREQUENT PATTERNS

Introduction

- DEF: Frequent pattern mining is the identification of the most frequent patterns in a collection of data

Transactions	Itemsets
t_1	milk, bread, butter
t_2	milk, beer, diper
t_3	diper, shampoo coke
t_4	beer, diper, milk, bread
t_5	beer, diper, coke
t_6	milk, diper, shampoo, beer

Definitions

- A collection of **items** is denoted by $\mathrm{I} = \{i_1, i_2, \ldots, i_m\}$
  - E.g. items you buy from supermarket, pages you visit over internet in one session, etc.
- Any subset S of items I is called an **itemset**

$$S \subseteq I$$

- A set of **transactions** is denoted by $\mathrm{T} = \{t_1, t_2, \ldots, t_n\}$ and it is called the transaction database
- The **support** of an itemset is defined as the number of transactions containing it and denoted by σ

- NOTE: Bifet re-defines graphs and trees in the frequent pattern mining chapter → we will refrain from that

Frequent itemset mining defined

- **DEF:** Given a set of items $\mathrm{I} = \{i_1, i_2, \ldots, i_m\}$ and a vector of transactions $\mathrm{T} = \{t_1, t_2, \ldots, t_n\}$ and minimum support σ_{min} the **frequent itemset mining problem** aims to find the set of frequent itemsets

$$\mathrm{FI} = \{S \subseteq I | \sigma(S) \geq \sigma_{min}\}$$

- **Anti-monotone** property: If $X, Y \subseteq S$ are two itemsets such that $X \subseteq Y \Rightarrow \sigma(Y) \leq \sigma(\mathrm{X})$
  - If an itemset is infrequent, then all its supersets will also be infrequent.
- Question: How many itemsets are possible?

Patterns

Patterns and Support

- **DEF:** **Patterns** are entities whose presence or absence, or frequency, indicate ways in which data deviates from randomness
- **Sub-pattern** and **super-pattern** correspond to the notions of subset and superset
  - A sub-pattern is a subset of items identified as a pattern
  - A super-pattern is a superset of items identified as a pattern
- **DEF:** A transaction t **supports** a pattern p if p is a sub-pattern of the pattern in transaction t

Closed and Maximal

- **DEF:** A pattern is **closed** for a dataset D if it has higher support in D than every one of its super-patterns
- **DEF:** A pattern is **maximal** if it is frequent and none of its superpatterns are frequent
  - Note: Every maximal pattern is closed.

Itemset types

ID	Transaction
t1	abce
t2	cde
t3	abce
t4	acde
t5	abcde
t6	bcd

Transactions

Support		Frequent	Gen	Closed	Max
6	t1,t2,t3,t4,t5,t6	c	c	c	
5	t1,t2,t3,t4,t5	e,ce	e	ce	
4	t1,t3,t4,t5	a,ac,ae,ace	a	ace	
4	t1,t2,t3,t5	b,bc	b	bc	
4	t2,t4,t5,t6	d,cd	d	cd	
3	t1,t3,t5	ab,abc,abe, be,bce,abce	ab,be	abce	abce
3	t2,t4,t5	de,cde	de	cde	cde

Itemset characteristics for the above set of transactions

Batch algorithms

- **Naïve pattern mining:** one-pass the dataset, keep track of every (pattern, support) and output those with support > threshold
  - Challenge: the number of possible patterns grows too fast with dataset size
- **FP-growth algorithm:** two-passes, FP-Tree data structure, no candidate generation phase
  - Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov., 8(1):53–87, 2004.
- **Eclat algorithm:** depth-first search, collect transactions supporting each itemset & intersect them
  - Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In Proceedings of the Third ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 97), Newport Beach, California, USA, August 14–17, 1997, pages 283–286, 1997.

Challenges and some solutions

- Frequent itemset mining in data streams is challenging due to several reasons:
  - Low memory
  - Single pass
  - Excessively large search space
  - Change → previously infrequent item become frequent and vice versa
- Three stream mining approaches for frequent patterns:
  - Approaches which don't distinguish old and new itemsets
  - Approaches which give more weights to recent items
  - Itemset mining over multiple time granularity

Approaches and criteria for streams

- **All or closed/maximal:** Mine all frequent patterns or only the closed or maximal ones.
- **Recent or all:** Consider the pattern frequency from the beginning of the stream, or give more importance to recent items, either via sliding windows or some importance decay scheme.
- **Fast update:** Update the set of frequent items after each item arrives, or perform batch updates: that is, collect a batch of stream items, then update the data structure that represents frequent patterns using the batch. The choice of batch size is key to performance.
- **Exact or approximate:** return exactly the set of patterns that satisfy the frequency condition, or just some approximation of the set. Approximate algorithms may have false positives (infrequent patterns reported as frequent), false negatives (frequent patterns not reported), or both.

Coresets of Closed Patterns

- **DEF:** A coreset of a set C is a small subset such that solving the problem on the coreset provides an approximate solution for the problem on C
  - supp(p) – absolute support for pattern p
  - σ(p) – relative support of pattern p
- p is δ-closed in D if every super-pattern of p has support less than $(1 - \delta)$ suppD(p)
  - Note: Closed patterns are the 0-closed patterns and maximal patterns are the 1-closed patterns.
- (σ, δ)-coreset of D is dataset $D' \subseteq D$ such that, for every pattern p,
  - every pattern p occurring in D' is σ-frequent and δ-closed in D, and
  - if $p \in D'$ then p occurs as many times in D' as in D.
- (σ, δ)-coresets are lossy, compressed summaries
  - Minimum support σ excludes infrequent patterns, and
  - δ-closure excludes patterns whose support is very similar to that of some subpattern.

Heavy hitters

- In frequent pattern mining **heavy hitters** are those frequent itemsets whose counts exceed a (pre-defined) threshold
  - Re-visit: Heavy hitters in sketches calculate items with relative frequency above a threshold ϵ
- Naïve heavy hitter solution for frequent itemsets: group stream items into batches, identify frequent itemsets and memorize them in an adequately chosen data structure
- Other solutions which can be adapted for the heavy hitter itemset problem: Lossy Counting, SpaceSaving

Moment

- Moment is an exact closed frequent itemset miner operating on a sliding window
- Uses the Closed Enumeration Tree data structure to store all itemsets needed at the moment
- Distinguishes node types:
  - **Infrequent** gateway node: Contains an itemset x that is not frequent, but whose parent node contains an itemset y that is frequent and has a frequent sibling z such that $x = y \cup z$.
  - **Unpromising** gateway node: Contains an itemset x that is frequent and such that there is a closed frequent itemset y that is a superset of x, has the same support, and is lexicographically before x.
  - **Intermediate** node: A node that is not an unpromising gateway and contains a frequent itemset that is non-closed but has a child node that contains an itemset with the same support.
  - **Closed** node: Contains a closed frequent itemset.

Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window. *Knowl. Inf. Syst.*, 10(3):265–294, 2006.

Moment: Closed Enumeration Tree

- Moment adds and removes transactions from the tree while maintaining these five properties:
  1. All supersets of itemsets in infrequent gateway nodes are infrequent.
  2. All itemsets in unpromising gateway nodes and all their descendants are non-closed.
  3. All itemsets in intermediate nodes are non-closed but have some closed descendant.
  4. When adding a transaction, closed itemsets remain closed.
  5. When removing a transaction, infrequent items remain infrequent and non-closed itemsets remain non-closed.

Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window. *Knowl. Inf. Syst.*, 10(3):265–294, 2006.

FP-Stream

- Tilted-time window types:
  - Natural tilted-time window: Maintains the most recent 4 quarters of an hour, the last 24 hours, and the last 31 days. It needs only 59 counters to keep this information.
  - Logarithmic tilted-time window: Maintains slots with the last quarter of an hour, the next 2 quarters, 4 quarters, 8 quarters, 16 quarters, and so on. As in Exponential Histograms, the memory required is logarithmic in the length of time being monitored.
- A pattern is **subfrequent** if its support is more than σ' but less than σ, for a value $\sigma' < \sigma$.
- FP-STREAM maintains a global FP-Tree structure and processes transactions in batches. Every time a new batch is collected, it computes a new FP-Tree and adds it to the global FP-Tree.
  - FP-STREAM computes frequent and subfrequent patterns.

C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Proceedings of the NSF Workshop on Next Generation Data Mining*, pages 191–212, 2002.

IncMine

- The IncMine algorithm was designed to identify frequent closed itemsets (FCIs) in windows
  - Besides FCIs, it stores candidate itemsets as well → maintains set C of semi-FCIs
  - Stream elements are collected in batches B of size b
  - Batch B is analyzed and identified FCIs are merged into set C
  - w = number of batches stored in the window → represents wb transactions
  - Number of itemsets is minimized by early dropping pattern p if it does not occur at least σib times in the first i batches and with minimum support σ
- Pro: considerably faster than Moment and FP-STREAM
- Contra: approximate solution, allows false negatives

James Cheng, Yiping Ke, and Wilfred Ng. Maintaining frequent closed itemsets over a sliding window. *J. Intell. Inf. Syst.*, 31(3):191–215, 2008.

SEQUENTIAL PATTERN MINING

Introduction

- **DEF:** A **sequence** is an ordered list of items. Formally, a sequence of items $A = \{a_1 a_2, a_3, \ldots, a_n\}$ is such that items in the set are ordered according to some rule. Rule could be timestamp based on arrival, or values of the items.
- Transactions vs sequences (they are not the same!)

ID	Transaction
t1	abce
t2	cde
t3	abce
t4	acde
t5	abcde
t6	bcd

ID	Sequence(s)
s1	<a(abc)(ac)d(cf)>
s2	<(ad)c(bc)(ae)>
s3	<(ef)(ab)(df)>
s4	<eg(af)>
s5	<af>
s6	<(eg)>

Sequence mining defined

- **DEF: Sequential pattern mining** aims to find the complete set of frequent subsequences given a set of sequences and support threshold

ID	Sequence(s)
s1	<a(abc)(ac)d(cf)>
s2	<(ad)c(bc)(ae)>
s3	<(ef)(ab)(df)>
s4	<eg(af)>
s5	<af>
s6	<(eg)>

A sequence → **<a(abc)(ac)d(cf)>**

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

Sub-sequence → **<a(abc)(ac)>**

- Given **support** threshold min_sup =2, <(ab)c> is a sequential pattern

Challenges

- A huge number of possible sequential patterns are hidden in datasets and especially in unbounded data streams
- A sequential pattern mining algorithms should therefore
  - find the complete set of patterns, when possible, satisfying the minimum support (frequency) threshold
  - be highly efficient, scalable, involving only a small number of database scans
  - be able to incorporate various kinds of user-specific constraints

Applications

- Sequences of commands frequently used by users of a software product
- Shopping of items: first buy computer, then pen-drive, then keyboard over a period of time.
- Some event sequence: earthquake → tsunami → medical treatment → (optionally) fission core melts
- DNA/protein sequences etc.

Summary

- Sketches
- Frequent patterns and itemsets
- Sequential pattern mining

Thank you for your attention!