Mining Frequent Patterns from Data Streams

Acknowledgement: Slides modified by Dr. Lei Chen based on the slides provided by Charu Aggarwal

And Jiawei Han, Jure Leskovec

OUTLINE

- **Data Streams**
  - Characteristics of Data Streams
  - Key Challenges in Stream Data
- **Frequent Pattern Mining over Data Streams**
  - Counting Itemsets
  - Lossy Counting
  - Extensions

Data Stream

- What is the data stream?
  - A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of items using limited computing and storage capabilities.

Data Streams

- Traditional DBMS
  - Data stored in finite, persistent data sets

- Data Streams
  - Continuous, ordered, changing, fast, huge amount
  - Managed by Data Stream Management System (DSMS)

DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- “Unbounded” disk store
- Only current state matters
- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

- Transient streams
- Continuous queries
- Sequential access
- Bounded main memory
- Historical data is important
- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics

Data Streams

- Data Streams
  - Data streams - continuous, ordered, changing, fast, huge amount
  - Traditional DBMS - data stored in finite, persistent data sets
- Characteristics of Data Streams
  - Fast changing and requires fast, real-time response

Characteristics of Data Streams

- Data Streams
  - Data streams—continuous, ordered, changing, fast, huge amount
  - Traditional DBMS—data stored in finite, persistent data sets
- Characteristics
  - Fast changing and requires fast, real-time response
  - Huge volumes of continuous data, possibly infinite
  - Data stream captures nicely our data processing needs of today

APPLICATIONS

Sensors

Weather

Stock Exchange

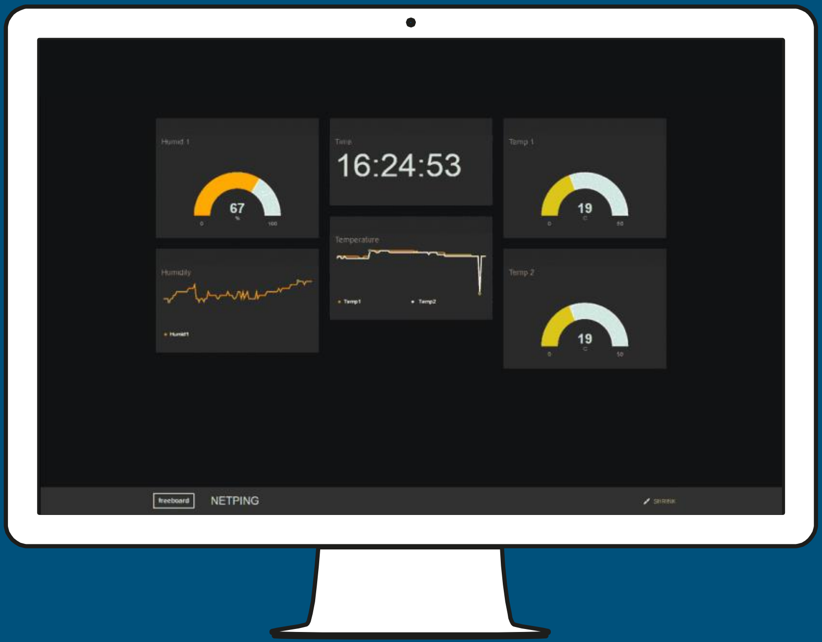
Self Driving Cars

Trends

Tweets

Logs

Articles/News
...



Wikipedia Edits

ATM Transactions

Chats

Television

Seisms

Music similarities

CO2 Level

Car Tracking
...

Example: Freeboard.io - Dashboards For the Internet Of Things - https://freeboard.io/

Stream Data Applications

- Telecommunication calling records
- Business: credit card transaction flows
- Network monitoring and traffic engineering
- Financial market: stock exchange
- Engineering & industrial processes: power supply & manufacturing
- Sensor, monitoring & surveillance: video streams, RFIDs
- Security monitoring
- Web logs and Web page click streams
- Massive data sets (even saved but random access is too expensive)

Characteristics of Data Streams

- Data Streams
  - Data streams—continuous, ordered, changing, fast, huge amount
  - Traditional DBMS—data stored in finite, persistent data sets
- Characteristics
  - Fast changing and requires fast, real-time response
  - Huge volumes of continuous data, possibly infinite
  - Data stream captures nicely our data processing needs of today
  - Random access is expensive—single scan algorithm (*can only have one look*)
  - Store only the summary of the data seen thus far
  - Most stream data are at pretty low-level or multi-dimensional in nature, needs multi-level and multi-dimensional processing

Key Challenges in Stream Data

- **Mining precise freq. patterns in stream data: unrealistic**
  - Infinite length
  - Concept-drift
  - Concept-evolution
  - Feature evolution

Key Challenges: Infinite Length

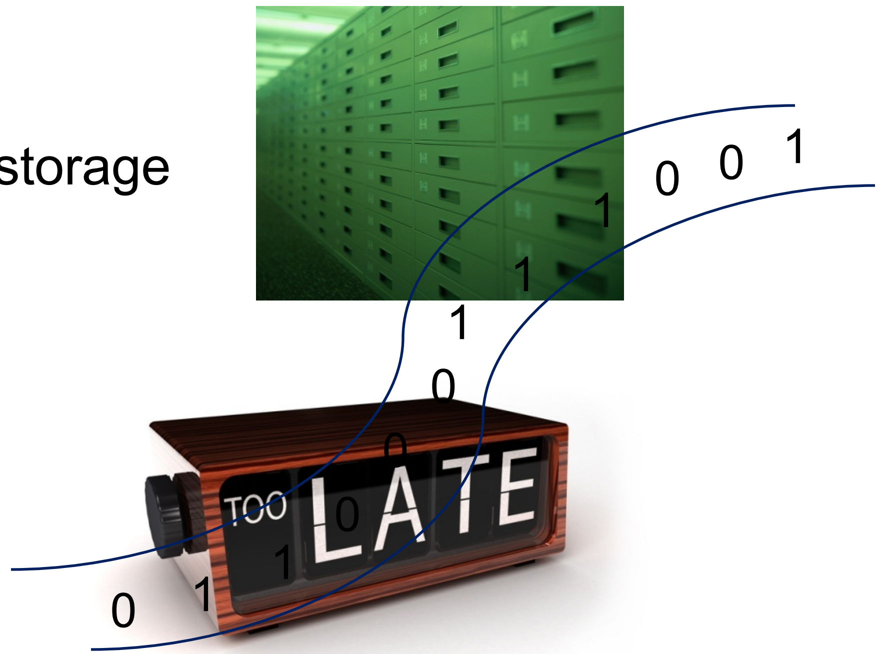
- **Infinite length**
- In many data mining situations, we do not know the entire data set in advance. Stream management is important when the input rate is controlled externally
- Examples: Google queries, Twitter or Facebook status updates

Key Challenges: Infinite Length

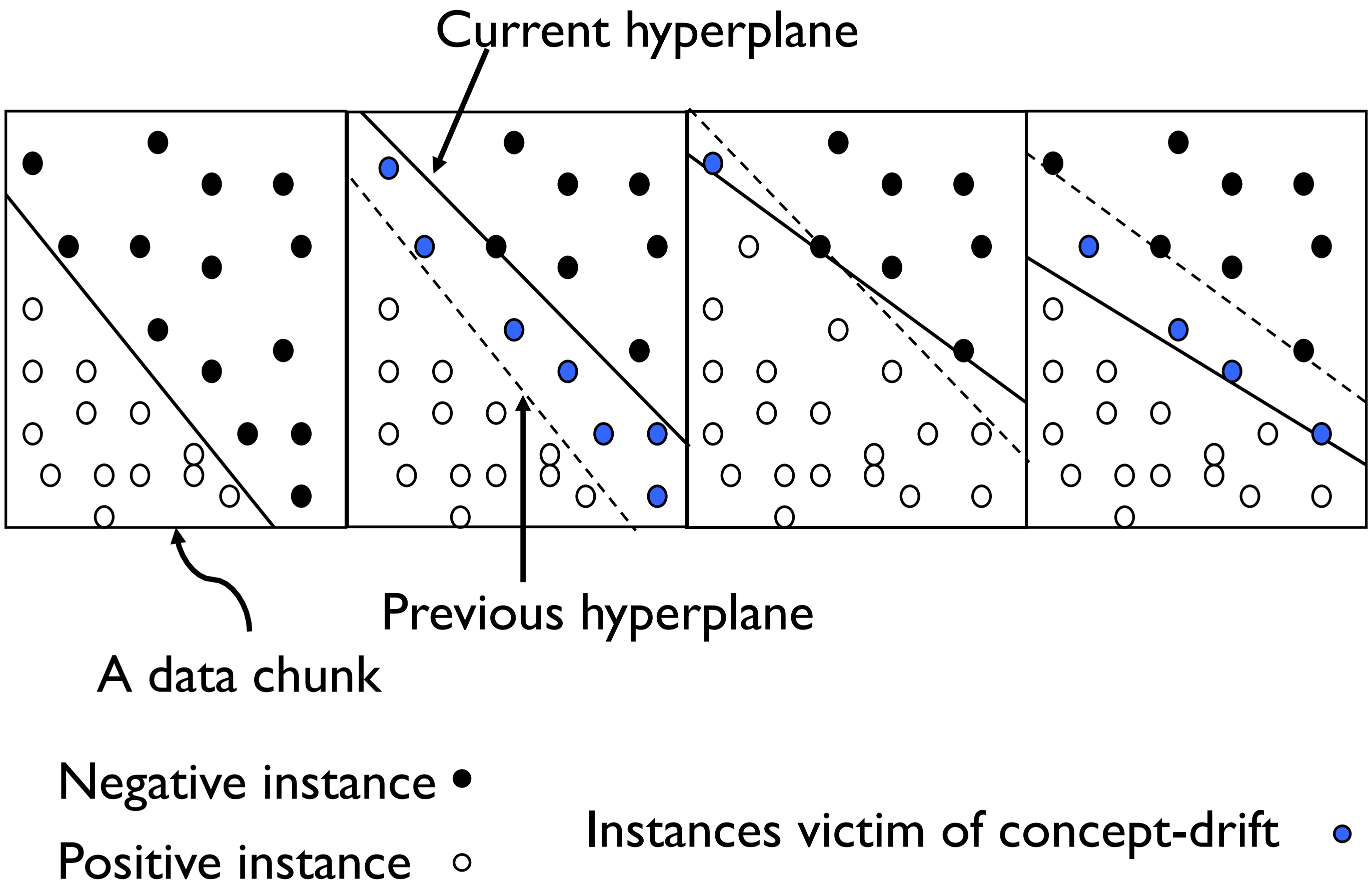
- **Infinite length**: Impractical to store and use all historical data

- Requires infinite storage

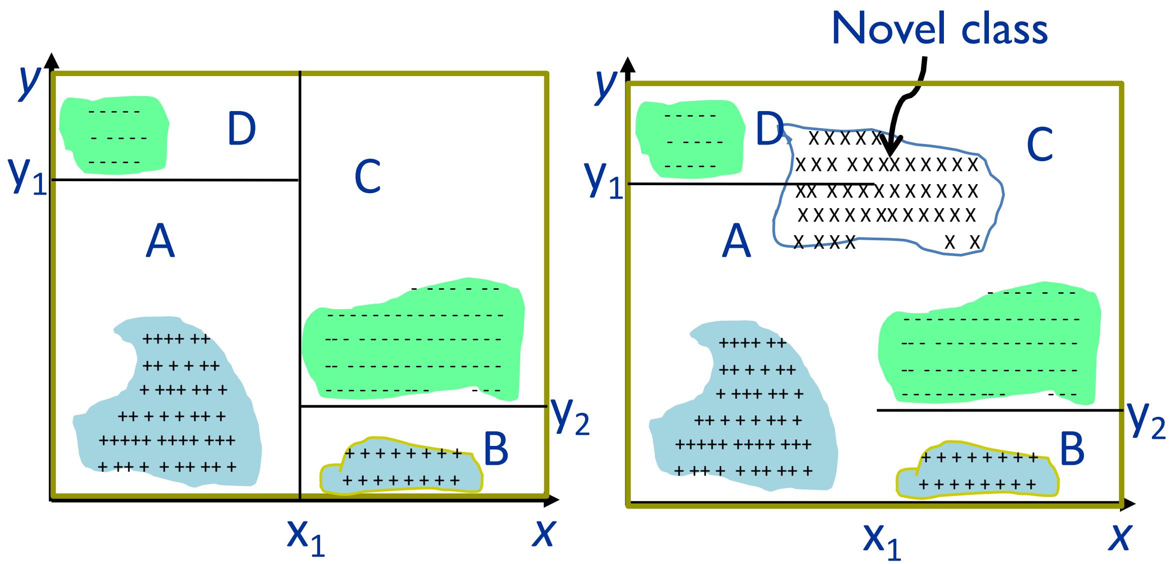
- And running time



Key Challenges: Concept-Drift



Key Challenges: Concept-Evolution



Classification rules:

R1. if ($x > x_1$ and $y < y_2$) or ($x < x_1$ and $y < y_1$) then class = +

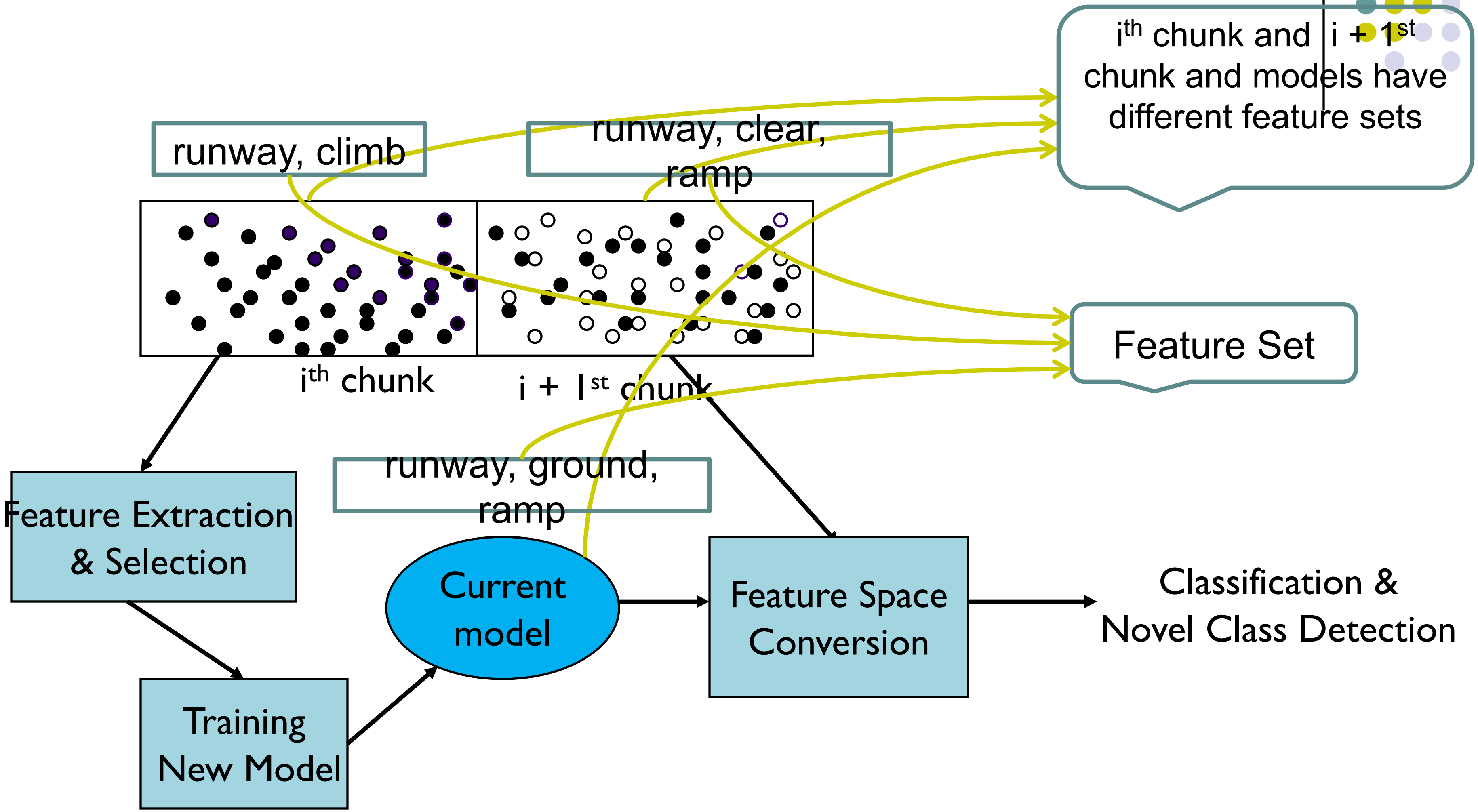
R2. if ($x > x_1$ and $y > y2$) or ($x < x_1$ and $y > y_1$) then class = -

- Concept-evolution occurs when a new class arrives in the stream.
- In this example, we again see a data chunk having two dimensional data points.
- There are two classes here, + and -. Suppose we train a rule-based classifier using this chunk
- Suppose a new class x arrives in the stream in the next chunk.
- If we use the same classification rules, all novel class instances will be misclassified as either + or -.

Key Challenges: Dynamic Features

- Why new features evolving
  - Infinite data stream
    - Normally, global feature set is unknown
    - New features may appear
  - Concept drift
    - As concept drifting, new features may appear
  - Concept evolution
    - New type of class normally holds new set of features

Key Challenges: Dynamic Features



Existing classification models need complete fixed features and apply to all the chunks. Global features are difficult to predict. One solution is using all English words and generate vector. Dimension of the vector will be too high.

Frequent Pattern Mining over Data Stream

- **Items Counting**
- Lossy Counting
- Extensions

Items Counting

Counting Bits – (1)

- Problem: given a stream of 0's and 1's, be prepared to answer queries of the form "how many 1's in the last k bits?" where $k \leq N$.
- Obvious solution: store the most recent N bits.
  - When new bit comes in, discard the $N+1^{st}$ bit.

Counting Bits – (2)

- You can't get an exact answer without storing the entire window.
- Real Problem: what if we cannot afford to store N bits?
  - E.g., we're processing 1 billion streams and $N = 1$ billion
- But we're happy with an approximate answer.

DGIM* Method

- Store $O(\log^2 N)$ bits per stream.
- Gives approximate answer, never off by more than 50%.
  - Error factor can be reduced to any fraction > 0, with more complicated algorithm and proportionally more stored bits.

*Datar, Gionis, Indyk, and Motwani

Something That Doesn’t (Quite) Work

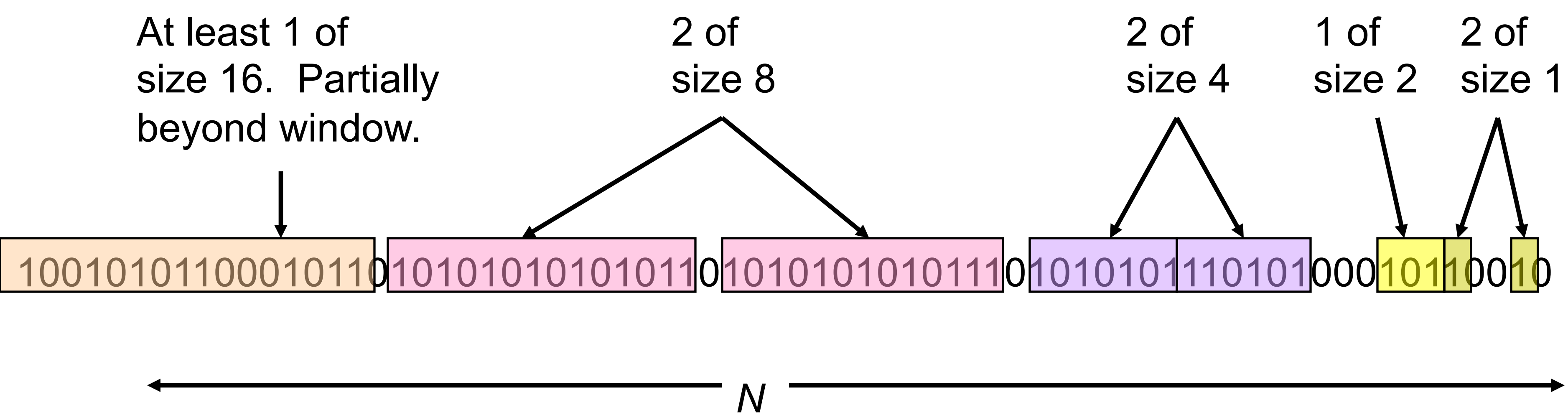
- Summarize exponentially increasing regions of the stream, looking backward.
- Drop small regions if they begin at the same point as a larger region.

Key Idea

- Summarize blocks of stream with specific numbers of 1's.

- Block *sizes* (number of 1's) increase exponentially as we go back in time

Example: Bucketized Stream



Timestamps

- Each bit in the stream has a *timestamp*, starting 1, 2, …
- Record timestamps modulo N (the window size), so we can represent any relevant timestamp in $O(\log_2 N)$ bits.

Buckets

- A *bucket* in the DGIM method is a record consisting of:
  1. The timestamp of its end [O(log N) bits].
  2. The number of 1's between its beginning and end [O(log log N) bits].
- Constraint on buckets: number of 1's must be a power of 2.
  - That explains the log log N in (2).

Representing a Stream by Buckets

- Either one or two buckets with the same power-of-2 number of 1's.
- Buckets do not overlap in timestamps.
- Buckets are sorted by size.
  - Earlier buckets are not smaller than later buckets.
- Buckets disappear when their end-time is > *N* time units in the past.

Updating Buckets – (1)

- When a new bit comes in, drop the last (oldest) bucket if its end-time is prior to *N* time units before the current time.
- If the current bit is 0, no other changes are needed.

Updating Buckets – (2)

- If the current bit is 1:
  1. Create a new bucket of size 1, for just this bit.
     - End timestamp = current time.
  2. If there are now three buckets of size 1, combine the oldest two into a bucket of size 2.
  3. If there are now three buckets of size 2, combine the oldest two into a bucket of size 4.
  4. And so on …

Example

100101011000101100101010101010101101010101010101110101010111010100010110010

0010101100010110110101010101010110110101010101011101101010111010100011011001011

0010101100010110110101010101010110110101010101011101101010111010100011011001011

0101100010110110101010101010110110101010101011101101010111010100011011001011

0101100010110110101010101010110110101010101011101101010111010100011011001011

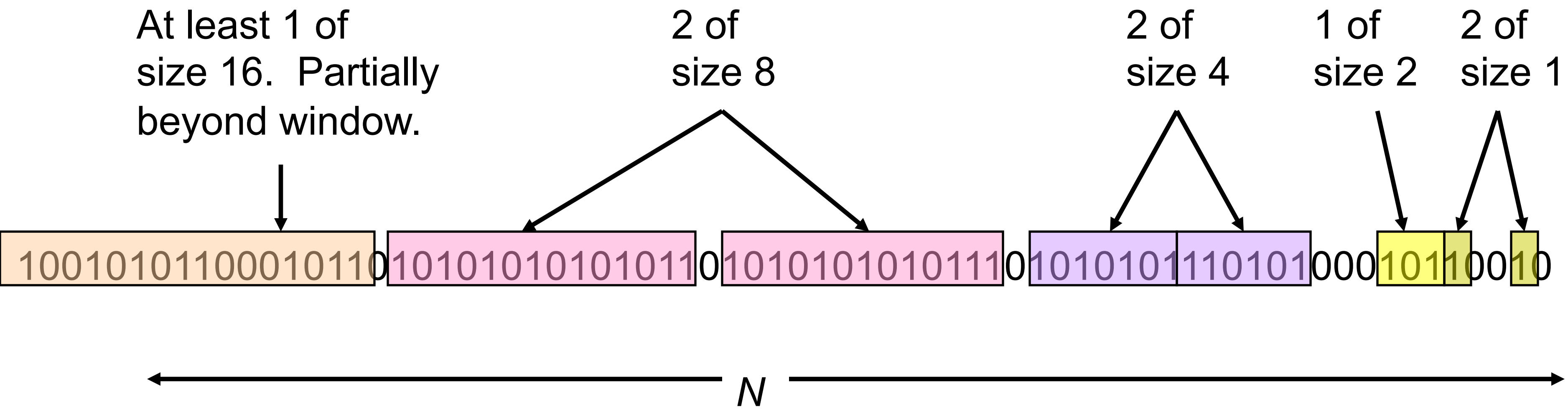
0101100010110110101010101010110101010101010111011010101110101000110110010110111

Querying

- To estimate the number of 1's in the most recent *N* bits:
  1. Sum the sizes of all buckets but the last.
  2. Add half the size of the last bucket.

- Remember: we don't know how many 1's of the last bucket are still within the window.

Example: Bucketized Stream



Error Bound

- Suppose the last bucket has size 2^k.
- Then by assuming 2^{k-1} of its 1's are still within the window, we make an error of at most 2^{k-1}.
- Since there is at least one bucket of each of the sizes less than 2^k, the true sum is at least $1 + 2 + .. + 2^{k-1} = 2^k - 1$.
- Thus, error at most 50%.

Frequent Pattern Mining over Data Stream

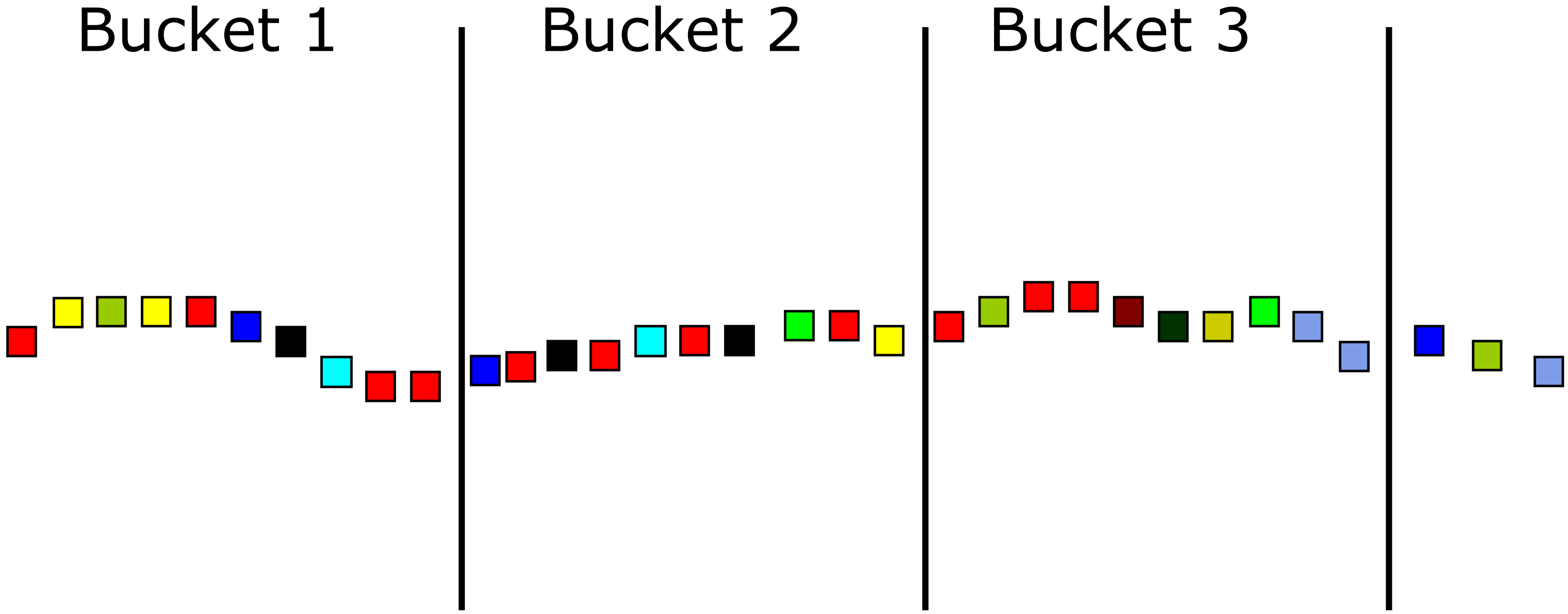
- Items Counting
- **Lossy Counting**
- Extensions

LOSSY COUNTING

Mining Approximate Frequent Patterns

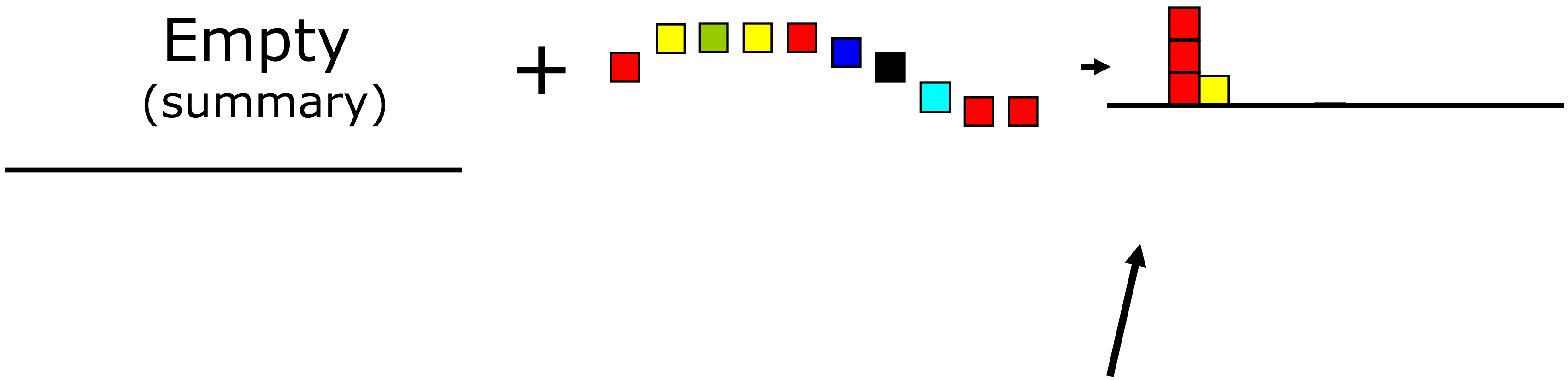
- Mining precise freq. patterns in stream data: unrealistic
  - Even store them in a compressed form, such as FPtree
- Approximate answers are often sufficient (e.g., trend/pattern analysis)
  - Example: A router is interested in all flows:
    - whose frequency is at least 1% (σ) of the entire traffic stream seen so far
    - and feels that 1/10 of σ (ε = 0.1%) error is comfortable
- How to mine frequent patterns with good approximation?
  - Lossy Counting Algorithm (Manku & Motwani, VLDB'02)
    - Major ideas: not tracing items until it becomes frequent
    - Adv: guaranteed error bound
    - Disadv: keep a large set of traces

Lossy Counting for Frequent Single Items



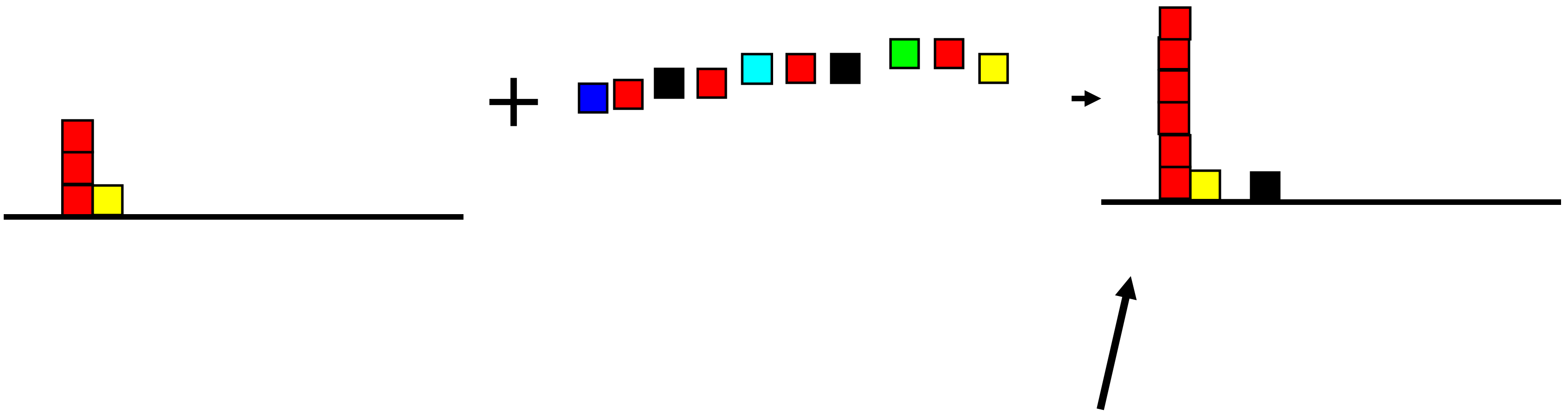
Divide stream into 'buckets' (bucket size is $1/\varepsilon = 1000$)

First Bucket of Stream



At bucket boundary, decrease all counters by 1

Next Bucket of Stream



At bucket boundary, decrease all counters by 1

Approximation Guarantee

- Given: (1) support threshold: σ, (2) error threshold: ε, and (3) stream length N
- Output: items with frequency counts exceeding $(\sigma - \varepsilon)$ N
- How much do we undercount?

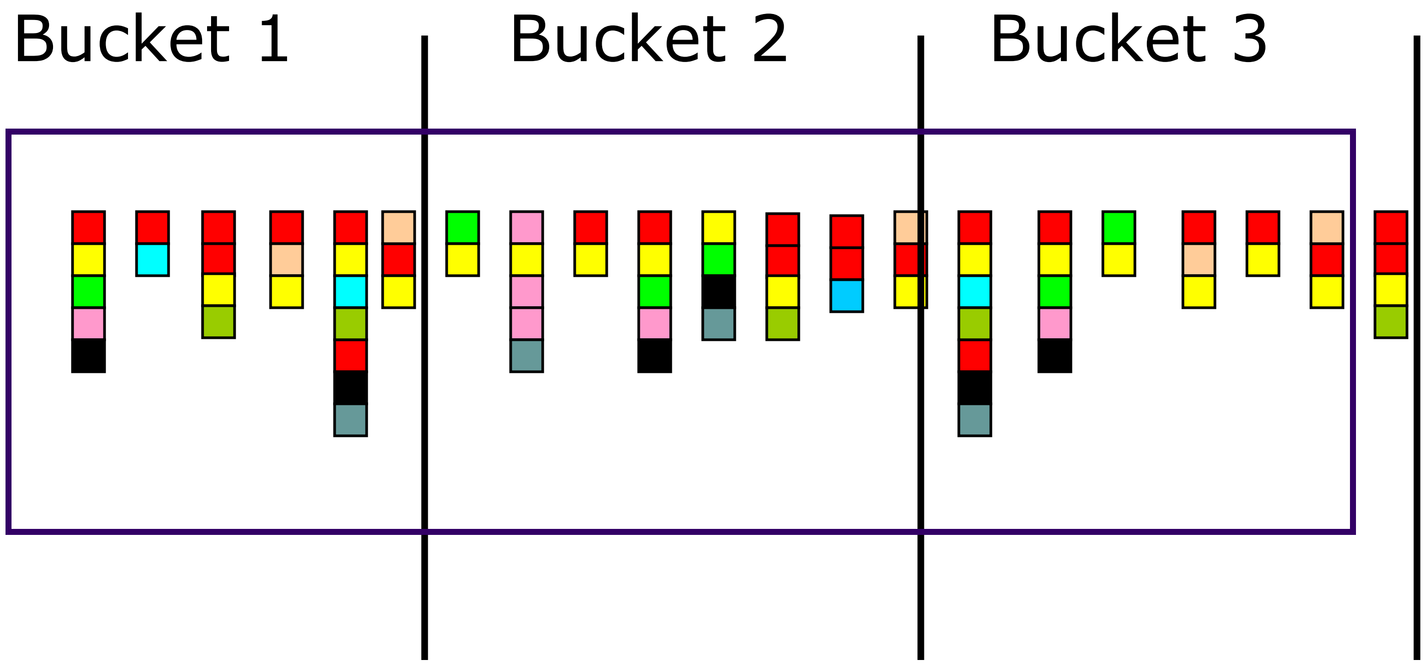
  If stream length seen so far = N and bucket-size = $1/\varepsilon$

  then frequency count error $\leq$ #buckets

  $= N/\text{bucket-size} = N/(1/\varepsilon) = \varepsilon N$
- Approximation guarantee
  - No false negatives
  - False positives have true frequency count at least $(\sigma-\varepsilon)N$
  - Frequency count underestimated by at most εN

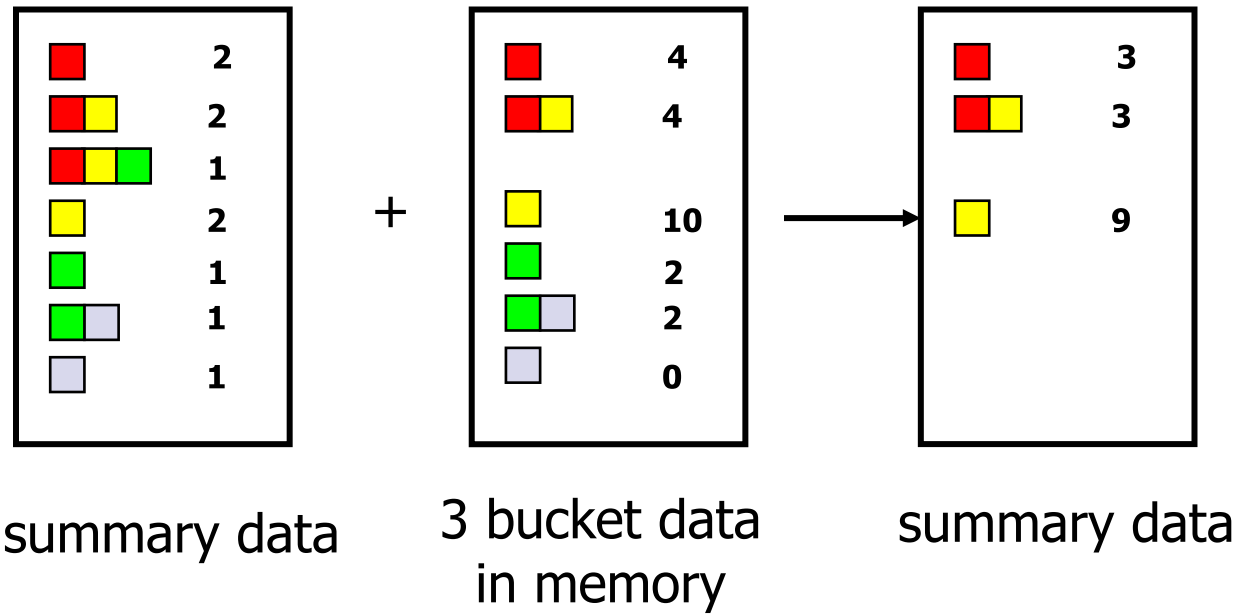
Lossy Counting For Frequent Itemsets

Divide Stream into 'Buckets' as for frequent items
But fill as many buckets as possible in main memory one time



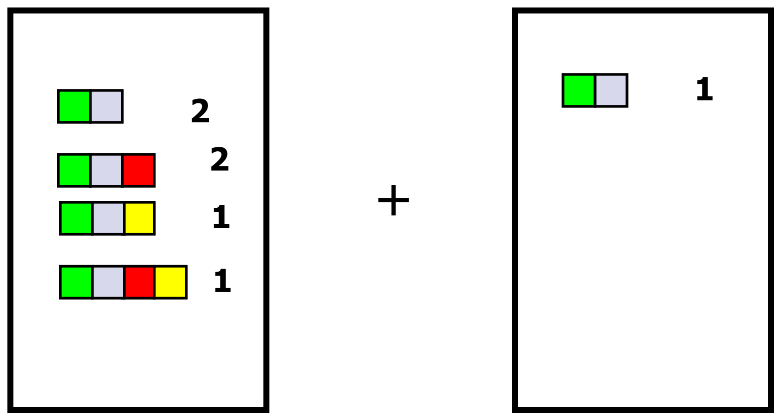
If we put 3 buckets of data into main memory one time, then decrease each frequency count by 3

Update of Summary Data Structure



Itemset ( ) is deleted.
That's why we choose a large number of buckets
– delete more

Pruning Itemsets – Apriori Rule



summary data

3 bucket data in memory

If we find itemset ( ) is not frequent itemset, then we needn't consider its superset

Summary of Lossy Counting

- Strength
  - A simple idea
  - Can be extended to frequent itemsets
- Weakness:
  - Space bound is not good
  - For frequent itemsets, they do scan each record many times
  - The output is based on all previous data. But sometimes, we are only interested in recent data
- A space-saving method for stream frequent item mining
  - Metwally, Agrawal, and El Abbadi, ICDT'05

Extensions

Extensions

- Lossy Counting Algorithm (Manku & Motwani, VLDB'02)
- Mine frequent patterns with Approximate frequent patterns.
- **Keep only current frequent patterns. No changes can be detected.**
- FP-Stream (C. Giannella, J. Han, X. Yan, P.S. Yu, 2003)
- Use tilted time window frame.
- **Mining evolution and dramatic changes of frequent patterns.**
- Moment (Y. Chi, ICDM '04)
- Very similar to FP-tree, except that keeps a dynamic set of items.
- **Maintain closed frequent itemsets over a Stream Sliding Window**

Lossy Counting versus FP-Stream

- Lossy Counting (Manku & Motwani VLDB'02)
  - Keep only current frequent patterns—No changes can be detected
- FP-Stream: Mining evolution and dramatic changes of frequent patterns (Giannella, Han, Yan, Yu, 2003)
  - Use tilted time window frame
  - Use compressed form to store significant (approximate) frequent patterns and their time-dependent traces

Summary of FP-Stream

- Mining Frequent Itemsets at Multiple Time
- Granularities Based in FP-Growth
- Maintains
  - Pattern Tree
  - Tilted-time window
- Advantages
  - Allows to answer time-sensitive queries
  - Places greater information to recent data
- Drawback
  - Time and memory complexity

Moment

- Regenerate frequent itemsets from the entire window whenever a new transaction comes into or an old transaction leaves the window
- Store every itemset, frequent or not, in a traditional data structure such as the prefix tree, and update its support whenever a new transaction comes into or an old transaction leaves the window
- Drawback
  - Mining each window from scratch - too expensive
    - Subsequent windows have many freq patterns in common
  - Updating frequent patterns every new tuple, also too expensive

Summary of 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

References

[Agrawal' 94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In VLDB, pages 487–499, 1994.

[Cheung' 03] W. Cheung and O. R. Zaiane, "Incremental mining of frequent patterns without candidate generation or support," in *DEAS*, 2003.

[Chi' 04] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, "Moment: Maintaining closed frequent itemsets over a stream sliding window," in *ICDM*, November 2004.

[Evfimievski' 03] A. Evfimievski, J. Gehrke, and R. Srikant, "Limiting privacy breaches in privacy preserving data mining," in *PODS*, 2003.

[Han' 00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In SIGMOD, 2000.

[Koh' 04] J. Koh and S. Shieh, "An efficient approach for maintaining association rules based on adjusting fp-tree structures." in *DASFAA*, 2004.

[Leung' 05] C.-S. Leung, Q. Khan, and T. Hoque, "Cantree: A tree structure for efficient incremental mining of frequent patterns," in *ICDM*, 2005.

[Toivonen' 96] H. Toivonen, "Sampling large databases for association rules," in *VLDB*, 1996, pp. 134–145.

Barzan Mozafari, Hetal Thakkar, Carlo Zaniolo: Verifying and Mining Frequent Patterns from Large Windows over Data Streams. ICDE 2008: 179-188

Hetal Thakkar, Barzan Mozafari, Carlo Zaniolo. Continuous Post-Mining of Association Rules in a Data Stream Management System. Chapter VII in Post-Mining of Association Rules: Techniques for Effective Knowledge Extraction, Yanchang Zhao; Chengqi Zhang; and Longbing Cao (eds.), ISBN: 978-1-60566-404-0.