Unit-V

Mining Data Streams

Dr. K.RAGHAVA RAO

Professor of CSE

Dept. of MCA

KL University

krraocse@gmail.com

http://datamining.blog.com

Characteristics of Data Streams

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

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)

Methodologies for Stream Data Processing

- Major challenges
  - Keep track of a large universe, e.g., pairs of IP address, not ages
- Methodology
  - Synopses (trade-off between accuracy and storage)
  - Use *synopsis data structure*, much smaller ($O(\log^k N)$ space) than their base data set ($O(N)$ space)
  - Compute an *approximate answer* within a *small error range* (factor ε of the actual answer)
- Major methods
  - Random sampling
  - Histograms
  - Sliding windows
  - Multi-resolution model
  - Sketches
  - Radomized algorithms

Stream Data Processing Methods

- Random sampling (but without knowing the total length in advance)
  - *Reservoir sampling*: maintain a set of *s* candidates in the reservoir, which form a true random sample of the element seen so far in the stream. As the data stream flow, every new element has a certain probability (*s*/N) of replacing an old element in the reservoir.
- Sliding windows
  - Make decisions based only on *recent data* of sliding window size *w*
  - An element arriving at time *t* expires at time *t* + *w*
- Histograms
  - Approximate the frequency distribution of element values in a stream
  - Partition data into a set of contiguous buckets
  - Equal-width (equal value range for buckets) vs. V-optimal (minimizing frequency variance within each bucket)
- Multi-resolution models
  - Popular models: balanced binary trees, micro-clusters, and wavelets

Stream Data Mining vs. Stream Querying

- Stream mining—A more challenging task in many cases
  - It shares most of the difficulties with stream querying
    - But often requires less "precision", e.g., no join, grouping, sorting
  - Patterns are hidden and more general than querying
  - It may require exploratory analysis
    - Not necessarily continuous queries
- Stream data mining tasks
  - Multi-dimensional on-line analysis of streams
  - Mining outliers and unusual patterns in stream data
  - Clustering data streams
  - Classification of stream data

Challenges for Mining Dynamics in Data Streams

- Most stream data are at pretty low-level or multi-dimensional in nature: needs ML/MD processing
- Analysis requirements
  - Multi-dimensional trends and unusual patterns
  - Capturing important changes at multi-dimensions/levels
  - Fast, real-time detection and response
  - Comparing with data cube: Similarity and differences
- Stream (data) cube or stream OLAP: Is this feasible?
  - Can we implement it efficiently?

Multi-Dimensional Stream Analysis: Examples

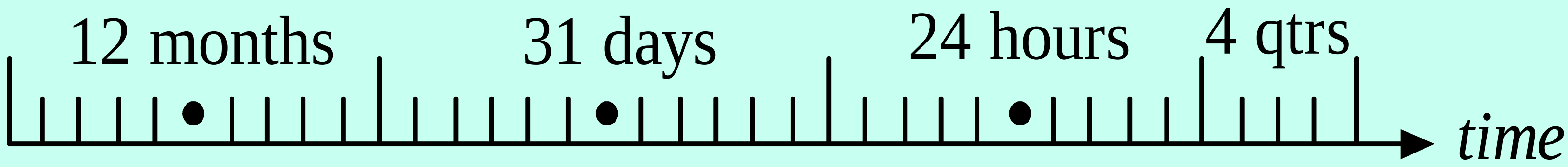
- Analysis of Web click streams
  - Raw data at low levels: seconds, web page addresses, user IP addresses, …
  - Analysts want: changes, trends, unusual patterns, at reasonable levels of details
  - E.g., *Average clicking traffic in North America on sports in the last 15 minutes is 40% higher than that in the last 24 hours."*
- Analysis of power consumption streams
  - Raw data: power consumption flow for every household, every minute
  - Patterns one may find: *average hourly power consumption surges up 30% for manufacturing companies in Chicago in the last 2 hours today than that of the same day a week ago*

A Stream Cube Architecture

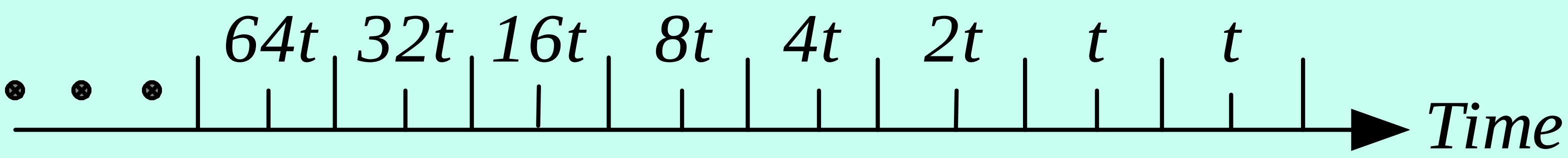
- A tilted time frame
  - Different time granularities
    - second, minute, quarter, hour, day, week, ...
- Critical layers
  - Minimum interest layer (m-layer)
  - Observation layer (o-layer)
  - User: watches at o-layer and occasionally needs to drill-down down to m-layer
- Partial materialization of stream cubes
  - Full materialization: too space and time consuming
  - No materialization: slow response at query time
  - Partial materialization: what do we mean "partial"?

A Titled Time Model

- Natural tilted time frame:
  - Example: Minimal: quarter, then 4 quarters → 1 hour, 24 hours → day, ...



- Logarithmic tilted time frame:
  - Example: Minimal: 1 minute, then 1, 2, 4, 8, 16, 32, ...



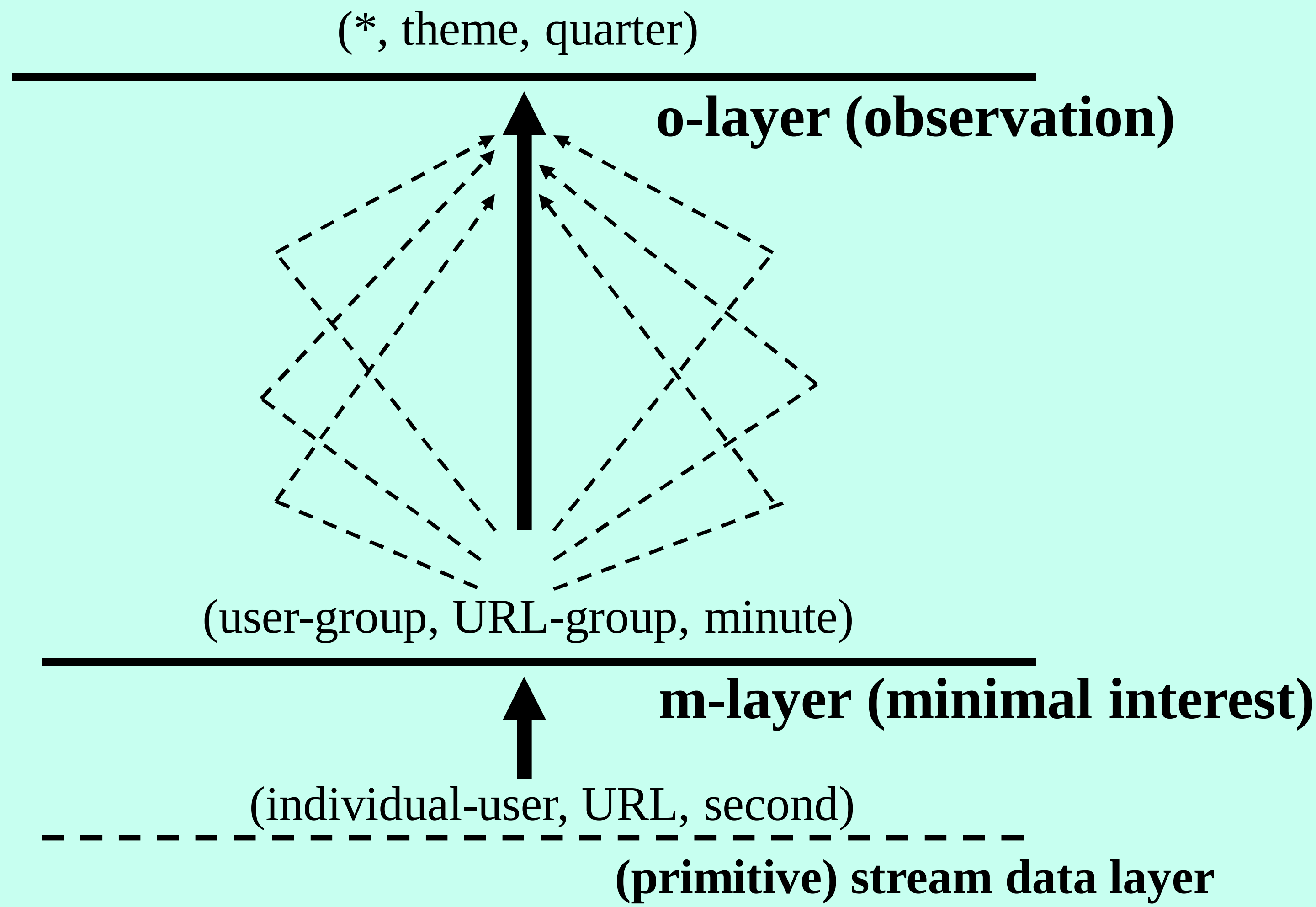
A Titled Time Model (2)

- Pyramidal tilted time frame:
  - Example: Suppose there are 5 frames and each takes maximal 3 snapshots
  - Given a snapshot number N, if N mod $2^d = 0$, insert into the frame number d. If there are more than 3 snapshots, “kick out” the oldest one.

Frame no.	Snapshots (by clock time)
0	69 67 65
1	70 66 62
2	68 60 52
3	56 40 24
4	48 16
5	64 32

Two Critical Layers in the Stream Cube

Fig. Two critical layers in “power supply station” stream data cube.



On-Line Partial Materialization vs. OLAP Processing

- On-line materialization
  - Materialization takes precious space and time
    - Only incremental materialization (with tilted time frame)
  - Only materialize "cuboids" of the critical layers?
    - Online computation may take too much time
  - Preferred solution:
    - *popular-path* approach: Materializing those along the popular drilling paths
    - *H-tree structure*: Such cuboids can be computed and stored efficiently using the H-tree structure
- Online aggregation vs. query-based computation
  - Online computing while streaming: aggregating stream cubes
  - Query-based computation: using computed cuboids

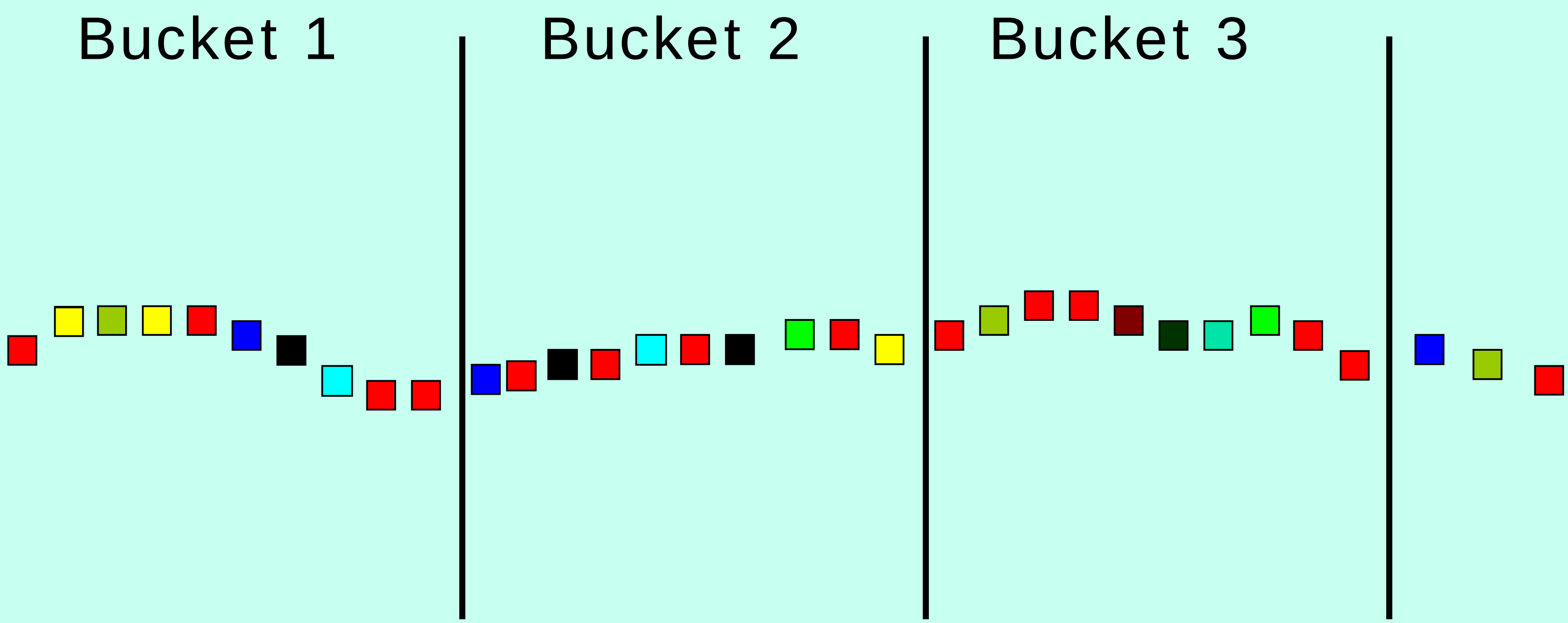
Frequent Patterns for Stream Data

- Frequent pattern mining is valuable in stream applications
  - e.g., network intrusion mining (Dokas, et al'02)
- Mining precise freq. patterns in stream data: unrealistic
  - Even store them in a compressed form, such as FPtree
- How to mine frequent patterns with good approximation?
  - Approximate frequent patterns (Manku & Motwani VLDB'02)
  - Keep only current frequent patterns? No changes can be detected
- Mining evolution freq. patterns (C. Giannella, J. Han, X. Yan, P.S. Yu, 2003)
  - Use tilted time window frame
  - Mining evolution and dramatic changes of frequent patterns
- Space-saving computation of frequent and top-k elements (Metwally, Agrawal, and El Abbadi, ICDT'05)

Mining Approximate Frequent Patterns

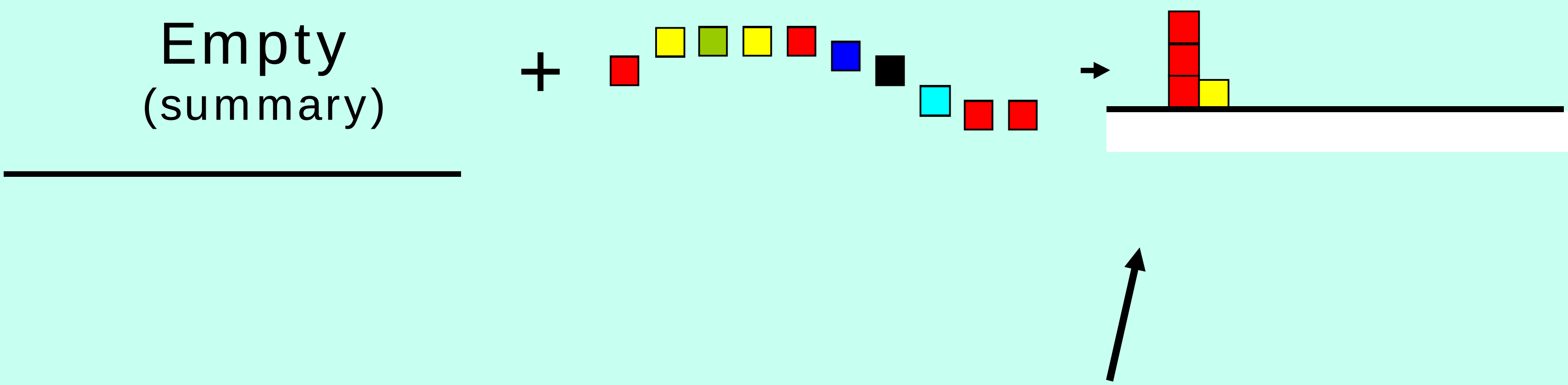
- Mining precise freq. patterns in stream data: unrealistic
  - Even store them in a compressed form, such as FPtree
- Approximate answers are often sufficient (e.g., trend/pattern analysis)
  - Example: a router is interested in all flows:
    - whose frequency is at least 1% (σ) of the entire traffic 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 Items



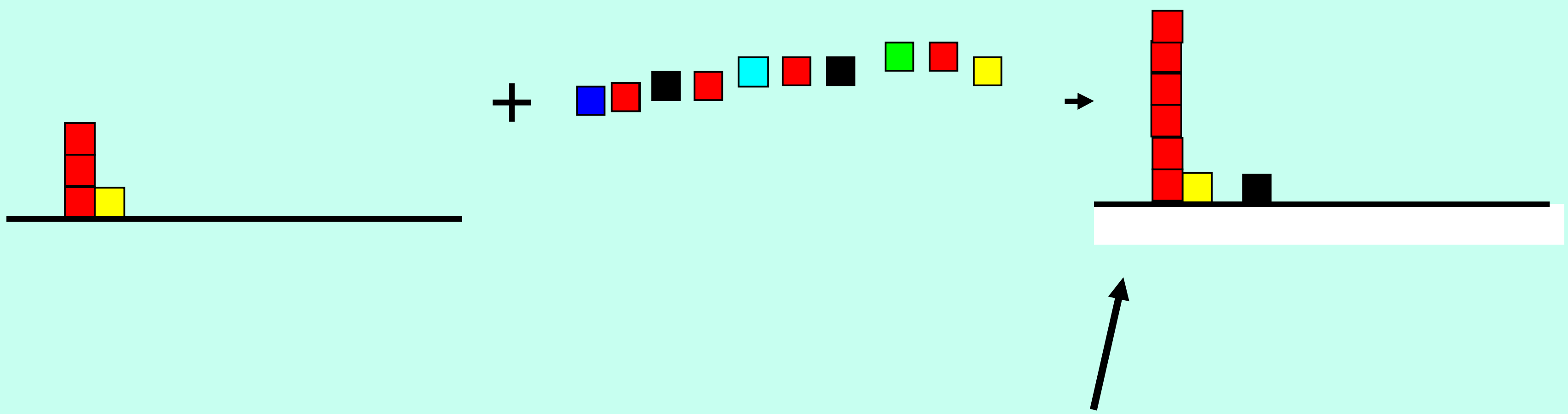
Divide Stream into 'Buckets' (bucket size is $1/\varepsilon = 1000$)

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 (σ – ε) N
- How much do we undercount?

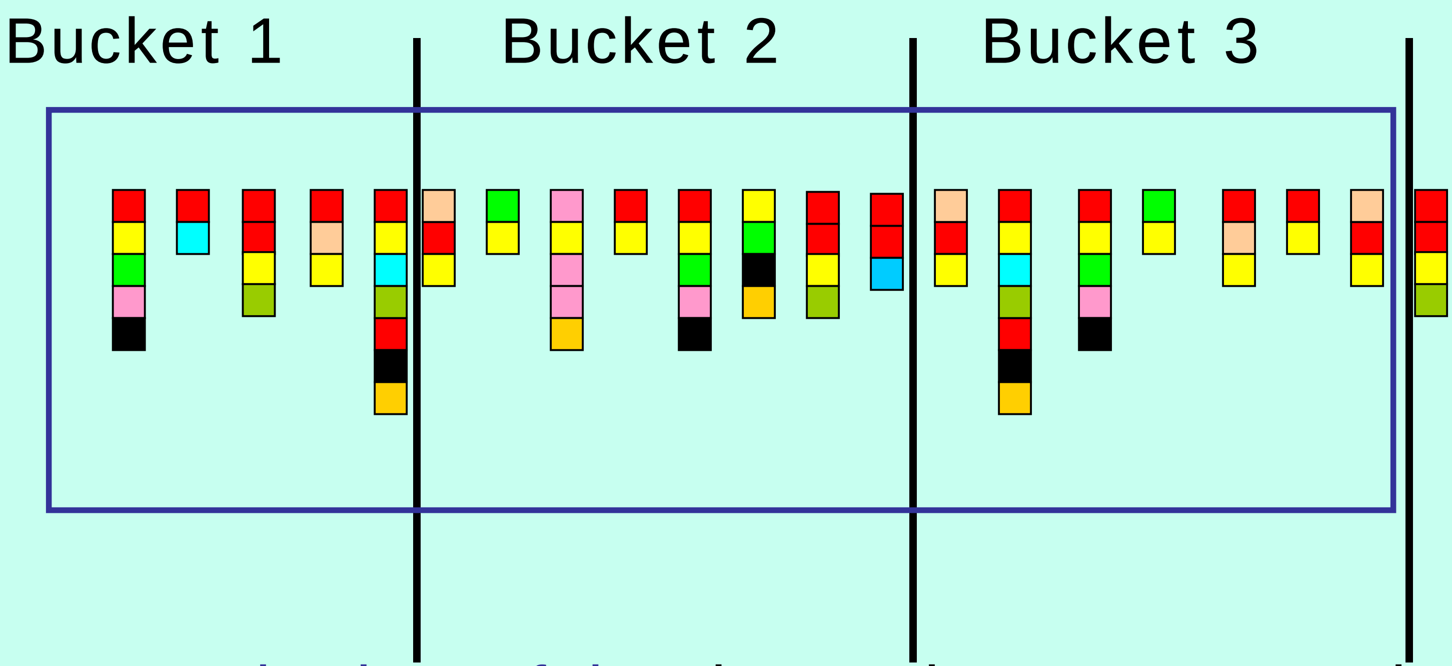
  If stream length seen so far = N

  and bucket-size = 1/ε

  then frequency count error ≤ #buckets = εN
- Approximation guarantee
  - No false negatives
  - False positives have true frequency count at least (σ–ε)N
  - Frequency count underestimated by at most εN

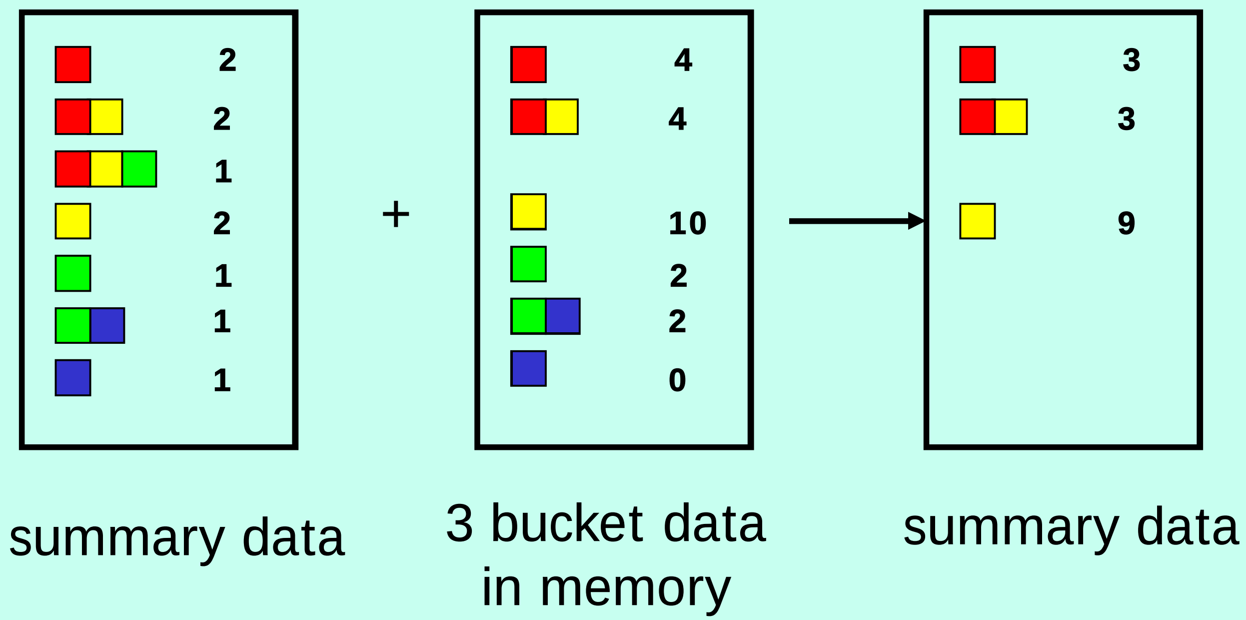
Lossy Counting For Frequent Itemsets

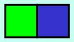
Divide Stream into 'Buckets' as for frequent items
But fill as many buckets as possible in main memory one time



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

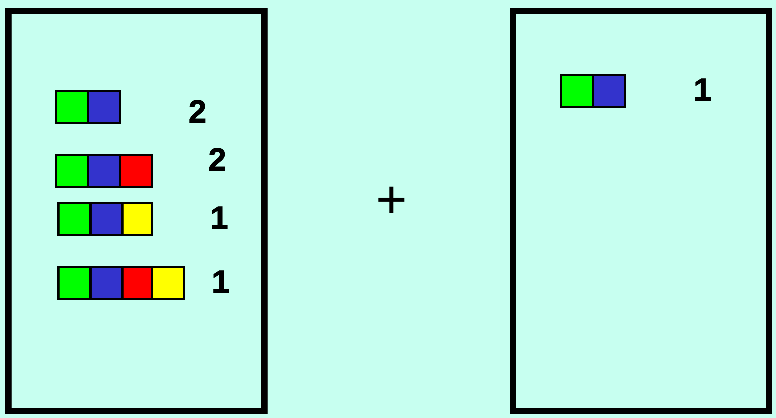
Update of Summary Data Structure



Itemset () is deleted.

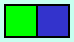
That's why we choose a large number of buckets – delete more

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

Classification for Dynamic Data Streams

- Decision tree induction for stream data classification
  - VFDT (Very Fast Decision Tree)/CVFDT (Domingos, Hulten, Spencer, KDD00/KDD01)
- Is decision-tree good for modeling fast changing data, e.g., stock market analysis?
- Other stream classification methods
  - Instead of decision-trees, consider other models
    - Naïve Bayesian
    - Ensemble (Wang, Fan, Yu, Han. KDD'03)
    - K-nearest neighbors (Aggarwal, Han, Wang, Yu. KDD'04)
  - Tilted time framework, incremental updating, dynamic maintenance, and model construction
  - Comparing of models to find changes

Hoeffding Tree

- With high probability, classifies tuples the same
- Only uses small sample
  - Based on Hoeffding Bound principle
- Hoeffding Bound (Additive Chernoff Bound)

  r: random variable

  R: range of r

  n: # independent observations

  Mean of r is at least $r_{avg} - \varepsilon$, with probability 1 – d

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$$

Hoeffding Tree Algorithm

- Hoeffding Tree Input

  S: sequence of examples

  X: attributes

  G( ): evaluation function

  d: desired accuracy

- Hoeffding Tree Algorithm

  for each example in S

    retrieve $G(X_a)$ and $G(X_b)$ //two highest $G(X_i)$

    if ($G(X_a) - G(X_b) > \varepsilon$)

      split on X_a

      recurse to next node

      break

Hoeffding Tree: Strengths and Weaknesses

- Strengths
  - Scales better than traditional methods
    - Sublinear with sampling
    - Very small memory utilization
  - Incremental
    - Make class predictions in parallel
    - New examples are added as they come
- Weakness
  - Could spend a lot of time with ties
  - Memory used with tree expansion
  - Number of candidate attributes

VFDT (Very Fast Decision Tree)

- Modifications to Hoeffding Tree
  - Near-ties broken more aggressively
  - G computed every n_{min}
  - Deactivates certain leaves to save memory
  - Poor attributes dropped
  - Initialize with traditional learner (helps learning curve)
- Compare to Hoeffding Tree: Better time and memory
- Compare to traditional decision tree
  - Similar accuracy
  - Better runtime with 1.61 million examples
    - 21 minutes for VFDT
    - 24 hours for C4.5
- Still does not handle concept drift

CVFDT (Concept-adapting VFDT)

- Concept Drift
  - Time-changing data streams
  - Incorporate new and eliminate old
- CVFDT
  - Increments count with new example
  - Decrement old example
    - Sliding window
  - Grows alternate subtrees
  - When alternate more accurate => replace old

Ensemble of Classifiers Algorithm

- H. Wang, W. Fan, P. S. Yu, and J. Han, “Mining Concept-Drifting Data Streams using Ensemble Classifiers”, KDD'03.
- Method (derived from the ensemble idea in classification)
  - train K classifiers from K chunks
  - for each subsequent chunk

    train a new classifier

    test other classifiers against the chunk

    assign weight to each classifier

    select top K classifiers

Clustering Data Streams [GMMO01]

- Base on the k-median method
  - Data stream points from metric space
  - Find k clusters in the stream s.t. the sum of distances from data points to their closest center is minimized
- Constant factor approximation algorithm
  - In small space, a simple two step algorithm:
  1. For each set of M records, S_i, find O(k) centers in $S_1, \ldots, S_l$
     - Local clustering: Assign each point in S_i to its closest center
  2. Let S' be centers for $S_1, \ldots, S_l$ with each center weighted by number of points assigned to it
     - Cluster S' to find k centers

Clustering for Mining Stream Dynamics

- Network intrusion detection: one example
  - Detect bursts of activities or abrupt changes in real time—by on-line clustering
- Our methodology (C. Agarwal, J. Han, J. Wang, P.S. Yu, VLDB'03)
  - Tilted time frame work: o.w. dynamic changes cannot be found
  - Micro-clustering: better quality than k-means/k-median
    - incremental, online processing and maintenance)
  - Two stages: micro-clustering and macro-clustering
  - With limited "overhead" to achieve high efficiency, scalability, quality of results and power of evolution/change detection

CluStream: A Framework for Clustering Evolving Data Streams

- Design goal
  - High quality for clustering evolving data streams with greater functionality
  - While keep the stream mining requirement in mind
    - One-pass over the original stream data
    - Limited space usage and high efficiency
- CluStream: A framework for clustering evolving data streams
  - Divide the clustering process into online and offline components
    - Online component: periodically stores summary statistics about the stream data
    - Offline component: answers various user questions based on the stored summary statistics

The CluStream Framework

- Micro-cluster
  - Statistical information about data locality
  - Temporal extension of the *cluster-feature vector*
    - Multi-dimensional points $\overline{X}_1 \ldots \overline{X}_k \ldots$ with time stamps $T_1 \ldots T_k \ldots$
    - Each point contains *d* dimensions, i.e., $\overline{X}_i = \left(x_i^1 \ldots x_i^d\right)$
    - A micro-cluster for *n* points is defined as a (2.*d* + 3) tuple

$$\left(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n\right)$$

- Pyramidal time frame
  - Decide at what moments the snapshots of the statistical information are stored away on disk

CluStream: Clustering On-line Streams

- Online micro-cluster maintenance
  - Initial creation of q micro-clusters
    - q is usually significantly larger than the number of natural clusters
  - Online incremental update of micro-clusters
    - If new point is within max-boundary, insert into the micro-cluster
    - O.w., create a new cluster
    - May delete obsolete micro-cluster or merge two closest ones
- Query-based macro-clustering
  - Based on a user-specified time-horizon h and the number of macro-clusters K, compute macroclusters using the k-means algorithm

End of unit-V part-1