Mining Data Streams

- What is stream data? Stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary
Data Streams and Their Characteristics

- Data Streams
  - Features: Continuous, ordered, changing, fast, huge volume
  - Contrast with traditional DBMS (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 low-level and multi-dimensional in nature, needs multi-level and multi-dimensional processing
**Streaming 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)
DBMS vs. DSMS (Data Stream Management Systems)

- 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

Ack. From Motwani's PODS'04 tutorial slides
Q: How can we perform cluster analysis effectively in data streams?

- Data Streams
  - Continuous, ordered, changing, fast, huge volume
  - Single-scan algorithm
Challenges of Stream Query Processing

- Multiple, continuous, rapid, time-varying, ordered streams
- Main memory computations
- Queries are often continuous
  - Evaluated continuously as stream data arrives
  - Answer updated over time
- Queries are often complex
  - Beyond element-at-a-time processing
  - Beyond stream-at-a-time processing
  - Beyond relational queries (scientific, data mining, OLAP)
- Multi-level/multi-dimensional query processing
  - Most stream data are at low-level or multi-dimensional in nature
Stream Data Mining Tasks

- Stream mining vs. stream querying
  - Stream mining shares many difficulties with stream querying
  - E.g., single-scan, fast response, dynamic, ...
  - But often requires less “precision”, e.g., no join, grouping, sorting
  - Patterns are hidden and more general than querying

- Stream data mining tasks
  - Multi-dimensional on-line analysis of streams
  - Pattern mining in data streams
  - Classification of stream data
  - Clustering data streams
  - Mining outliers and anomalies in stream data
Challenges of 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?
Mining Data Streams

- What is stream data? stream data management systems?
  and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary
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”?
Cube: A Lattice of Cuboids

- 0-D (apex) cuboid
- 1-D cuboids
- 2-D cuboids
- 3-D cuboids
- 4-D (base) cuboid
Time Dimension: A Tilted Time Model

- **Tilted time frames**: A trade-off between space and granularity of time
- Decide at what moments the snapshots of the statistical information are stored
- **Design**: *Natural*, logarithmic and pyramidal tilted time frames

- **Natural tilted time frame**:
  - Ex: Minimal: 15min, then 4 * 15mins → 1 hour, 24 hours → day, ...

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

![Diagram showing natural and logarithmic tilted time frames]
Two Critical Generalized Layers in the Stream Cube

- Raw data stream sits at the “primitive” stream data layer.
- Stream data is generalized to m-layer (minimal interest layer) and “stored” to facilitate flexible drilling.
- Stream data should be constantly summarized and presented at the o-layer (observation layer) for constant observation.

Diagram:
- (*, theme, quarter) o-layer (observation)
- (user-group, URL-group, minute) m-layer (minimal interest)
- (individual-user, URL, second) (primitive) stream data layer
OLAP Operation and Cube Materialization

- OLAP (Online Analytical Processing) operations:
  - Roll up (drill-up): summarize data
    - by climbing up hierarchy or by dimension reduction
  - Drill down (roll down): reverse of roll-up
    - from higher level summary to lower level summary or detailed data, or introducing new dimensions
  - Slice and dice: project and select
  - Pivot (rotate): reorient the cube, visualization, 3D to series of 2D planes

- Cube partial materialization
  - Store some pre-computed cuboids for fast online processing
On-Line Partial 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

Materialization on Popular Path
OLAP Processing Using Stream Cubes

- Online aggregation vs. query-based computation
  - Online computing while streaming: aggregating stream cubes
  - Query-based computation: Using computed cuboids

- An H-tree cubing Structure (Ref.: Han, et al., SIGMOD’01)
  - Space preserving
    - Intermediate aggregates can be computed incrementally and saved in tree nodes
  - Facilitate computing other cells and multi-dimensional analysis
  - H-tree with computed cells can be viewed as *stream cube*
Mining Data Streams

- What is stream data? Stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary
Mining Approximate Frequent Patterns

- Mining precise frequent patterns in stream data: Unrealistic
  - Cannot even store them in a compressed form (e.g., FPtree)
- Approximate answers are often sufficient for pattern analysis
  - Ex.: A router
    - is interested in all flows whose frequency is at least 1% (\(\sigma\)) of the entire traffic stream seen so far
    - and feels that 1/10 of \(\sigma\) (\(\varepsilon = 0.1\%\)) error is comfortable
- How to mine frequent patterns with good approximation?
  - Lossy Counting Algorithm (Manku & Motwani, VLDB’02)
    - Major ideas: Not to keep the items with very low support count
    - Advantage: Guaranteed error bound
    - Disadvantage: Keeping a large set of traces
Lossy Counting for Frequent Single Items

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

First Bucket of the Stream

Empty (summary)

At bucket boundary, decrease all counters by 1

Next Bucket of the Stream
Approximation Guarantee

- Given: (1) support threshold: $\sigma$, (2) error threshold: $\varepsilon$, 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 $\leq$ # of buckets
    - $= N/bucket-size = N/(1/\varepsilon) = \varepsilon N$
- Approximation guarantee
  - No false negatives
  - False positives have true frequency count at least $(\sigma - \varepsilon)N$
  - Frequency count underestimated by at most $\varepsilon N$
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, then decrease each frequency count by 3
- Update summary data structure
  - Itemset (■) is deleted. That’s why we choose a large number of buckets – delete more
- Pruning Itemsets – Apriori Rule
  - If we find itemset (■) is not frequent, we needn’t consider its superset
Other Issues and Recommended Readings

- Other issues on pattern discovery in data streams
  - Space-saving computation of frequent and top-$k$ elements (Metwally, Agrawal, and El Abbadi, ICDT'05)
  - Mining approximate frequent $k$-itemsets in data streams
  - Mining sequential patterns in data streams

- Recommended Readings
  - G. Manku and R. Motwani, “Approximate Frequency Counts over Data Streams”, VLDB’02
  - A. Metwally, D. Agrawal, and A. El Abbadi, “Efficient Computation of Frequent and Top-$k$ Elements in Data Streams”, ICDT'05
Mining Data Streams

- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary
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)
    - Classifying skewed stream data (Gao, Fan, and Han, SDM'07)
- Evolution modeling: Tilted time framework, incremental updating, dynamic maintenance, and model construction
- Comparing of models to find changes
Very Fast Decision Tree for Data Streams

- Very Fast Decision Trees (VFDT) (Domingos, et al., KDD’00)
- Hoeffding's inequality: A result in probability theory that gives an upper bound on the probability for the sum of random variables to deviate from its expected value
- Based on Hoeffding Bound principle, classifying different samples leads to the same model with high probability—can use a small set of samples
- Hoeffding Bound (Additive Chernoff Bound)
  - Given: r: random variable, R: range of r, N: # independent observations
  - True mean of r is at least \( r_{\text{avg}} - \varepsilon \), with probability \( 1 - \delta \)
    \[
    \varepsilon = \sqrt{\frac{R^2 \ln(1 / \delta)}{2N}}
    \]
    (where \( \delta \) is user-specified)

Ack. From Gehrke’s SIGMOD tutorial slides
Hoeffding Tree: How to Handle Concept Drifts?

- Hoeffding Tree: strengths and weakness
  - 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

- Concept Drift
  - Time-changing data streams
  - Incorporate new and eliminate old
  - CVFDT (Concept-adapting VFDT)
    - Increments count with new example
    - Decrement old example
    - Sliding window
    - Nodes assigned monotonically increasing IDs
  - Grows alternate subtrees
    - When alternate more accurate: Replace the old one
  - $O(w)$ better runtime than VFDT-window
Ensemble of Classifiers

- Ensemble is a better way to handle concept drift than single trees
- 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
Issues in Stream Classification

- Descriptive model vs. generative model
  - Generative models assume data follows some distribution while descriptive models make no assumptions
  - Distribution of stream data is unknown and may evolve, so descriptive model is better

- Label prediction vs. probability estimation
  - Classify test examples into one class or estimate \( P(y|x) \) for each \( y \)
  - Probability estimation is better:
    - Stream applications may be stochastic (an example could be assigned to several classes with different probabilities)
    - Probability estimates provide confidence information and could be used in post processing
Classifying Data Streams with Skewed Distribution

- Problems of typical classification methods on skewed data:
  - Tend to ignore positive examples due to the small number
  - The cost of misclassifying positive examples is usually huge, e.g., misclassifying credit card fraud as normal
- Classify data stream with skewed distribution (i.e., rare events)
  - Employ both biased sampling and ensemble techniques
  - Reduce classification errors on the minority class
Concept Drifts

- Changes in P(x, y) x-feature vector y-class label P(x,y) = P(y|x)P(x)

- Four possibilities:
  - No change: P(y|x), P(x) remain unchanged
  - Feature change: only P(x) changes
  - Conditional change: only P(y|x) changes
  - Dual change: both P(y|x) and P(x) changes

- Expected error:
  \[ Err = \int_{(x,y) \in P(x,y)} P(x)(1 - P(y|x)) dx \]

- No matter how concept changes, the expected error could increase, decrease, or remain unchanged

- Training on the most recent data could help reduce expected error
Stream Ensemble Approach

- **Biased Sampling**: Save only the positive examples in the streams
- **Ensemble**: Partition negative examples of $S_m$ into $k$ portions to build $k$ classifiers
Experiments: Mean Squared Error on Synthetic & Real Data

- Test on concept-drift streams (synthetic data)

<table>
<thead>
<tr>
<th>Changes</th>
<th>Decision Trees</th>
<th>Naive Bayes</th>
<th>Logistic Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SE</td>
<td>NS</td>
<td>SS</td>
</tr>
<tr>
<td>Feature</td>
<td>0.1275</td>
<td>0.9637</td>
<td>0.6446</td>
</tr>
<tr>
<td>Conditional</td>
<td>0.0943</td>
<td>0.9805</td>
<td>0.5500</td>
</tr>
<tr>
<td>Dual</td>
<td>0.0854</td>
<td>0.9521</td>
<td>0.5174</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Changes</th>
<th>Decision Trees</th>
<th>Naive Bayes</th>
<th>Logistic Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SE</td>
<td>NS</td>
<td>SS</td>
</tr>
<tr>
<td>Feature</td>
<td>0.0847</td>
<td>0.6823</td>
<td>0.4639</td>
</tr>
<tr>
<td>Conditional</td>
<td>0.0552</td>
<td>0.6421</td>
<td>0.4463</td>
</tr>
<tr>
<td>Dual</td>
<td>0.0684</td>
<td>0.6758</td>
<td>0.4107</td>
</tr>
</tbody>
</table>

- Test on real data

Stream Ensemble always has lower error rate
Experiments: Model Accuracy and Training Efficiency

- **Model accuracy**

<table>
<thead>
<tr>
<th></th>
<th>SF</th>
<th>NS</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic1</td>
<td>0.9404</td>
<td>0.5175</td>
<td>0.6944</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>0.9337</td>
<td>0.4840</td>
<td>0.6611</td>
</tr>
<tr>
<td>Thyroid1</td>
<td>1.0000</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>Thyroid2</td>
<td>0.9998</td>
<td>0.9998</td>
<td>0.9996</td>
</tr>
<tr>
<td>Opt</td>
<td>0.9942</td>
<td>0.9495</td>
<td>0.9777</td>
</tr>
<tr>
<td>Letter</td>
<td>0.9931</td>
<td>0.9467</td>
<td>0.9782</td>
</tr>
<tr>
<td>Covtype</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.9999</td>
</tr>
</tbody>
</table>

- **Training time**

![Training Time Graph]

<table>
<thead>
<tr>
<th></th>
<th>SE</th>
<th>NS</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic1</td>
<td>0.9532</td>
<td>0.8220</td>
<td>0.9525</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>0.9558</td>
<td>0.8355</td>
<td>0.9556</td>
</tr>
<tr>
<td>Thyroid1</td>
<td>0.9982</td>
<td>0.9979</td>
<td>0.9982</td>
</tr>
<tr>
<td>Thyroid2</td>
<td>0.9551</td>
<td>0.9054</td>
<td>0.9145</td>
</tr>
<tr>
<td>Opt</td>
<td>0.9926</td>
<td>0.9722</td>
<td>0.9898</td>
</tr>
<tr>
<td>Letter</td>
<td>0.9305</td>
<td>0.9389</td>
<td>0.9389</td>
</tr>
<tr>
<td>Covtype</td>
<td>0.9907</td>
<td>0.9995</td>
<td>0.9907</td>
</tr>
</tbody>
</table>

![Logistic Regression Graph]

<table>
<thead>
<tr>
<th></th>
<th>SE</th>
<th>NS</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic1</td>
<td>0.8801</td>
<td>0.8363</td>
<td>0.8737</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>0.8992</td>
<td>0.8102</td>
<td>0.8854</td>
</tr>
<tr>
<td>Thyroid1</td>
<td>0.9977</td>
<td>0.9774</td>
<td>0.9909</td>
</tr>
<tr>
<td>Thyroid2</td>
<td>0.9949</td>
<td>0.9593</td>
<td>0.9930</td>
</tr>
<tr>
<td>Opt</td>
<td>0.9971</td>
<td>0.9940</td>
<td>0.9953</td>
</tr>
<tr>
<td>Letter</td>
<td>0.9545</td>
<td>0.9448</td>
<td>0.9517</td>
</tr>
<tr>
<td>Covtype</td>
<td>0.9995</td>
<td>0.9989</td>
<td>0.9994</td>
</tr>
</tbody>
</table>

Figure 4: Training Time
Mining Data Streams

- What is stream data? stream data management systems?
  and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary
Stream Clustering: A K-Median Approach

- O'Callaghan et al. Streaming-Data Algorithms for High-Quality Clustering (ICDE'02)
- Base on the \( k \)-median method
  - Data stream points are from metric space
  - Find \( k \) clusters in the stream such that the sum of distances from data points to their closest centers is minimized
- A constant factor approximation algorithm
  - In small space, a simple two-step algorithm
  - 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
  - Let \( S' \) be centers for \( S_1, \ldots, S_l \) with each center weighted by the number of points assigned to it
    - Cluster \( S' \) to find \( k \) centers
Hierarchical Clustering Tree Method:
- Maintain at most $m$ level-$i$ medians
- On seeing $m$ of them, generate $O(k)$ level-$(i+1)$ medians of weight equal to the sum of the weights of the intermediate medians assigned to them

Concerns:
- Quality will suffer for evolving data streams (maintaining only $m$ level-$i$ medians)
- Limited functionality in discovering and exploring clusters over different portions of the stream over time
CluStream: A Framework for Clustering Evolving Data Streams

- C. Aggarwal, J. Han, J. Wang, P. S. Yu, A Framework for Clustering Data Streams, VLDB'03

- Design goal of CluStream
  - High quality for clustering evolving data streams with rich functionality
  - Stream mining: One-pass over the stream data, limited space usage, high efficiency

- The CluStream Methodology
  - **Tilted time frame work**: otherwise, will lose dynamic changes
  - **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
Pyramidal Tilted Time Frame Adopted by CluStream

- **Pyramidal tilted time frame:**
  - Example: Suppose there are six frames $(d = 5)$ and each takes a maximal of three snapshots
  - Given a snapshot number $N$
    - If $N \mod 2^d = 0$, insert into the frame number $d$
    - If there are more than three snapshots, eliminate the oldest one

- Snapshots of a set of micro-clusters are stored following the pyramidal pattern
  - They are stored at differing levels of granularity depending on the recency

- Snapshots are classified into different orders varying from 1 to $\log(T)$
  - The $i$-th order snapshots occur at intervals of $a^i$ where $a \geq 1$
  - Only the last $(\alpha + 1)$ snapshots are stored

<table>
<thead>
<tr>
<th>Frame no.</th>
<th>Snapshots (by clock time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>69 67 65</td>
</tr>
<tr>
<td>1</td>
<td>70 66 62</td>
</tr>
<tr>
<td>2</td>
<td>68 60 52</td>
</tr>
<tr>
<td>3</td>
<td>56 40 24</td>
</tr>
<tr>
<td>4</td>
<td>48 16</td>
</tr>
<tr>
<td>5</td>
<td>64 32</td>
</tr>
</tbody>
</table>
The CluStream Framework: A Micro-Clustering Approach Using the BIRCH CF-Tree Structure

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

- A CF tree: A height-balanced tree that stores the clustering features (CFs)
- The non-leaf nodes store sums of the CFs of their children
CluStream: Clustering Evolving On-Line Data Streams

- Divide the clustering process into *online* and *offline* components
  - **Online component (micro-cluster maintenance)**
    - Periodically store summary statistics about the stream data
      - Initially, create $q$ 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
        - Otherwise, create a new cluster
        - May delete obsolete micro-clusters or merge two closest ones
  - **Offline component (query-based macro-clustering)**
    - Answers various user questions based on the stored summary statistics
    - Based on a user-specified time-horizon $h$ and the number of macro-clusters $k$, compute macro-clusters using the $k$-means algorithm
Mining Data Streams

- What is stream data? stream data management systems?
  and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary
Summary: Stream Data Mining

- Stream data mining and stream OLAP analysis:
  - Real life problem: Effectiveness, efficiency and scalability
- Stream OLAP
  - A multi-dimensional stream analysis framework
  - Time is a special dimension: Tilted time frame
  - What to compute and what to save? — Critical layers
  - Partial materialization and precomputation
- Stream data mining
  - Mining frequent patterns
  - Stream classification
  - Stream cluster analysis
References on Stream Data Mining (I)

- C. Aggarwal, J. Han, J. Wang, P. S. Yu, “A Framework for Clustering Data Streams”, VLDB'03
- C. Aggarwal, J. Han, J. Wang, P. S. Yu, “On-Demand Classification of Evolving Data Streams”, KDD'04
- C. Aggarwal, J. Han, J. Wang, and P. S. Yu, “A Framework for Projected Clustering of High Dimensional Data Streams”, VLDB'04
- S. Babu and J. Widom, “Continuous Queries over Data Streams”, SIGMOD Record, Sept. 2001
- Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, “Multi-Dimensional Regression Analysis of Time-Series Data Streams”, VLDB'02
- P. Domingos and G. Hulten, “Mining high-speed data streams”, KDD'00
- A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi, “Processing Complex Aggregate Queries over Data Streams”, SIGMOD’02
- J. Gehrke, F. Korn, and D. Srivastava, “On computing correlated aggregates over continuous data streams”, SIGMOD'01
- J. Gao, W. Fan, and J. Han, “A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions”, SDM'07
References on Stream Data Mining (II)

- S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, “Clustering Data Streams”, FOCS'00
- G. Hulten, L. Spencer and P. Domingos, “Mining time-changing data streams”, KDD’01
- S. Madden, M. Shah, J. Hellerstein, V. Raman, “Continuously Adaptive Continuous Queries over Streams”, SIGMOD’02
- G. Manku, R. Motwani, “Approximate Frequency Counts over Data Streams”, VLDB’02
- A. Metwally, D. Agrawal, and A. El Abbadi. “Efficient Computation of Frequent and Top-k Elements in Data Streams”. ICDT'05
- S. Viglas and J. Naughton, “Rate-Based Query Optimization for Streaming Information Sources”, SIGMOD’02
- Y. Zhu and D. Shasha. “StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time”, VLDB’02
- H. Wang, W. Fan, P. S. Yu, and J. Han, “Mining Concept-Drifting Data Streams using Ensemble Classifiers”, KDD'03