Mining Stream, Time-Series, and Sequence Data

- Mining data streams
- Mining time-series data
- Mining sequence patterns in transactional databases
- Mining sequence patterns in biological data
Mining Data Streams

- Stream data and stream data processing
- Basic methodologies for stream data processing and mining
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
Characteristics of Data Streams

- **Data Streams**
  - A sequence of data in transmission
  - An ordered pair \((s, \Delta)\) where: \(s\) is a sequence of tuples, \(\Delta\) is the sequence of time intervals

- **Characteristics**
  - Continuous
  - Huge volumes, possibly infinite
  - Fast changing and requires fast, real-time response
  - Random access is expensive—single scan algorithm
  - Low-level or multi-dimensional in nature
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)
Architecture: Stream Query Processing and Mining

SDMS (Stream Data Management System)

User/Application

Continuous Query

Multiple streams

Stream Query Processor

Results

Scratch Space
(Main memory and/or Disk)
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

Ack. From Motwani’s PODS tutorial slides
Mining Data Streams

- Stream data and stream data processing
- Foundations for stream data mining
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
Methodologies for Stream Data Processing

- Major challenges
  - Keep track of a large universe
- Methodology
  - Choosing a subset of data
    - Sampling
    - Sliding windows
    - Load shedding
  - Summarizing the data
    - Synopses (trade-off between accuracy and storage)
Random Sampling: Uniform Sampling

- Uniform sampling
  - Data stream of size $N$
  - Assume all samples are equally likely

- Example
  - a data stream of size 4 (also called population)
  
  \[
  \begin{array}{cccc}
  1 & 2 & 3 & 4 \\
  \end{array}
  \]
  
  possible samples of size 2
  
  \[
  \begin{array}{cccc}
  1 & 2 & 1 & 3 & 1 & 4 & 2 & 3 & 2 & 4 & 3 & 4 \\
  16\% & 16\% & 16\% & 16\% & 16\% & 16\% \end{array}
  \]
Random Sampling: Reservoir Sampling

- Reservoir sampling
  - Single-scan algorithm
  - Compute a uniform sample of $M$ elements without $N$

- Idea
  - Maintain a reservoir, which form a random sample of the elements seen so far in the stream

- Algorithm
  - add the first $M$ elements
  - Afterwards at item $i$, flip a coin
    a) ignore the element (reject)
    b) replace a random element in the sample (accept)

$$P(t_i \text{ is accepted}) = \frac{\text{sample size}}{\text{current population size}} = \frac{M}{i}$$

Slides: R. Gemulla, W. Lehner, P. J. Haas
Random Sampling: Reservoir Sampling (Example)

- Example
  - data stream
  - sample size $M = 2$

$t_1 + t_2$

$t_3$

$t_4$
Sliding Windows

- Sliding Windows
  - Make decisions based only on *recent data* of sliding window size $w$
  - An element arriving at time $t$ expires at time $t + w$

- Why?
  - Approximation technique for bounded memory
  - Natural in applications (emphasizes recent data)
  - Well-specified and deterministic semantics
Load Shedding

- **Load shedding**
  - Discards some data so the system can flow

- **Techniques**
  - Filters (semantic drop)
    - Chooses what to shed based on QoS, selectivity
  - Drops (random drop)
    - Eliminates a random fraction of input

- **Hospital example**
  - Load shedding based on condition

```
Patients  Condition Filter  Join  Doctors who can work on a patient
Doctors
```

```
Patients  Join  Doctors who can work on a patient
Doctors
```
Synopsis

- Synopsis
  - Summaries for data
  - Can be used to return approximate answers
  - Trade off between space and accuracy

- Techniques
  - Histograms
  - Wavelets
  - Sketching

- May require multiple passes

Synopses/Data Structures
Mining Data Streams

- Stream data and stream data processing
- Foundations for stream data mining
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues
Frequent Pattern Mining for Data Streams

- **Issues**
  - Multiple scans for training not feasible
  - Memory/space management
  - Concept drift

- **Methods**
  - Approximate frequent patterns (Manku & Motwani VLDB’02)
  - Mining evolution of freq. patterns (C. Giannella, J. Han, X. Yan, P.S. Yu, 2003)
  - Space-saving computation of frequent and top-k elements (Metwally, Agrawal, and El Abbadi, ICDT'05)
Mining Approximate Frequent Patterns

- Lossy Counting Algorithm (Manku & Motwani, VLDB’02)
- Motivation
  - Mining precise freq. patterns in stream data: unrealistic
  - Approximate answers are often sufficient (e.g., trend/pattern analysis)
  - Example: a router interested in all flows whose frequency is at least 1% (\(\sigma\)) of the entire traffic stream seen so far;
    - 1/10 of \(\sigma\) (\(\varepsilon = 0.1\%\)) error is comfortable
- Major ideas: approximation by tracing only “frequent” items
  - Adv: guaranteed error bound
  - Disadv: keep a large set of traces
Lossy Counting for Frequent Items

- **Input variables**
  - \( \varepsilon: \text{min\_support} \), \( \varepsilon: \text{error bound} \)

- **Fixed variables**
  - \( w = 1/\varepsilon \): window size

- **Running variables**
  - \( N: \text{current stream length} \)
  - \( b_{\text{current}} = \varepsilon N: \text{the current bucket} \)
  - \( f_e: \text{the real frequency count of element } e \)
  - Set of \((e, f, \Delta): (\text{element, approximate frequency, max error})\)
Lossy Counting for Frequent Items

- For each new element e
  - If an entry for e exists, then incrementing its frequency f by 1
  - Otherwise, create a new entry (e, 1, bcurrent -1)
- At bucket boundaries
  - Decrement frequency of all entries by 1
  - Delete entries with f+Δ ≤ bcurrent
Illustration

\[ b_{\text{current}} = 1 \]

\( (e, f, \Delta) \)

Empty (summary) + \[
\begin{array}{c}
\text{+}
\end{array}
\]

\[ b_{\text{current}} \]

\( (e, f, \Delta) \)

\[ \rightarrow \]

March 26, 2008  Data Mining: Concepts and Techniques
Approximation Guarantee

- Output: items with frequency counts exceeding \((\sigma - \varepsilon)N\)

- Error analysis: how much do we undercount?
  
  If stream length seen so far \(= N\) and bucket-size \(= 1/\varepsilon\)
  
  then frequency count error \(\leq \#\text{buckets} = \varepsilon N\)

- 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 itemsets

- Set of (set, f, ∆): (itemset, approximate frequency, max error)
Update of Summary Data Structure

summary data + Processing 3 buckets summary data in memory
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
Mining Evolution of Frequent Patterns for Stream Data

- 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
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, ...

---

**Diagram:**

- **Time:**
  - 12 months
  - 31 days
  - 24 hours
  - 4 qtrs
  - 64t
  - 32t
  - 16t
  - 8t
  - 4t
  - 2t
  - t
  - t

---

March 26, 2008 Data Mining: Concepts and Techniques
Two Structures for Mining Frequent Patterns with Tilted-Time Window (1)

- FP-Trees store Frequent Patterns
- Tilted-time major: An FP-tree for each tilted time frame
Frequent Pattern & Tilted-Time Window (2)

- The second data structure:
  - Observation: FP-Trees of different time units are similar
  - Pattern-tree major: each node is associated with a tilted-time window
Mining Data Streams

- Stream data and stream data processing
- Foundations for stream data mining
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
Classification for Dynamic Data Streams

- Issues
  - Multiple scans for training not feasible
  - Concept drift
- Methods
  - VFDT (Very Fast Decision Tree) and CVFDT (Concept-adapting Very Fast Decision Tree) (Domingos, Hulten, Spencer, KDD00/KDD01)
  - Ensemble (Wang, Fan, Yu, Han. KDD’03)
  - K-nearest neighbors (Aggarwal, Han, Wang, Yu. KDD’04)
Basic idea
- Consider only a small subset of training examples to find best split attribute at a node given a split evaluation measure $G$
- How many examples are necessary at each node?

Statistical foundation: Hoeffding Bound (Additive Chernoff Bound)
- $r$: random variable
- $R$: range of $r$
- $n$: # independent observations

True mean of $r$ is at least $r_{\text{avg}} - \varepsilon$, with probability $1 - \delta$

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

Given observed best attribute $X_a$ and second best attribute $X_b$
- if $\Delta G = G(X_a) - G(X_b) > \varepsilon$, then $\Delta G \geq \Delta G - \varepsilon > 0$ with probability $1 - \delta$
Hoeffding Tree Algorithm

Hoeffding Tree Input
- S: sequence of examples
- X: attributes
- G: split evaluation function (info gain, Gini index)
- δ: 1 - desired probability of choosing correct attribute

Hoeffding Tree Algorithm
for each example in S
  retrieve G(X_a) and G(X_b)  //two highest G(X_i)
  compute ε
  if ( G(X_a) – G(X_b) > ε )
    split on X_a
    recursive to next node
  break
Decision-Tree Induction with Data Streams

Data Stream

Packets > 10

yes → Protocol = http

no

Protocol = http

Bytes > 60K

yes

no

Protocol = ftp
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 utilization issues with tree expansion and large number of candidate attributes
VFDT (Very Fast Decision Tree)

- Modifications to Hoeffding Tree
  - Near-ties broken more aggressively
  - G computed every $n_{\text{min}}$
  - Deactivates certain leaves to save memory
  - Poor attributes dropped
  - Initialize with traditional learner (helps learning curve)
- Compare to traditional decision tree
  - Similar accuracy
  - Better runtime with 1.61 million examples
    - 21 minutes for VFDT
    - 24 hours for C4.5
CVFDT (Concept-adapting VFDT)

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

- Stream data and stream data processing
- Foundations for stream data mining
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
Stream Cluster Analysis

- Issues
  - Multiple scan not feasible
  - Memory and time constraints
  - Concept drift

- Methods
  - STREAM based on k-medians [GMMO01]
  - CLuStream based on microclustering and macroclustering
    (Agarwal, Han, Wang, Yu, VLDB’03)
Problem: find k clusters in the stream s.t. the sum of distances from data points to their closest center is minimized (k-median method)

Basic idea: divide-and-conquer

Approximation algorithm

1. For each set of M records, \( S_i \), perform k-median clustering and find \( O(k) \) centers
   - Only retain center information (weighted by # points assigned to the cluster)

2. When there are enough centers, cluster the weighted centers
Hierarchical Clustering Tree

- **data points**
- **level-i medians**
- **level-(i+1) medians**
Hierarchical 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

Drawbacks:
- Low quality for evolving data streams (register only \( k \) centers)
- Limited functionality in discovering and exploring clusters over different portions of the stream over time
CluStream: A Framework for Clustering Evolving Data Streams

- Basic idea
  - Tilted time framework
  - Two stages: micro-clustering and macro-clustering

- Algorithm
  - Online/micro-clustering: periodically computes microclusters
    - Given Multi-dimensional points \( \bar{X}_1, \ldots, \bar{X}_k \) at time stamps \( T_1, \ldots, T_k \)
    - Cluster-feature vector (temporal extension of BIRCH)
      \[
      \left( CF2^x, CF1^x, CF2^t, CF1^t, n \right)
      \]
  - Offline/macro-clustering: compute macroclusters using the k-means algorithm
    - based on user-specified time-horizon
Summary: Stream Data Mining

- Stream data mining: A rich and on-going research field

- Current research focus in database community:
  - DSMS system architecture, continuous query processing, supporting mechanisms

- Stream data mining
  - Powerful tools for finding general and unusual patterns
  - Effectiveness, efficiency and scalability: lots of open problems
References on Stream Data Mining (1)

- C. Aggarwal, J. Han, J. Wang, P. S. Yu. A Framework for Clustering Data Streams, VLDB'03
- C. C. Aggarwal, J. Han, J. Wang and 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, R. Rastogi. Processing Complex Aggregate Queries over Data Streams, SIGMOD’02
- J. Gehrke, F. Korn, D. Srivastava. On computing correlated aggregates over continuous data streams. SIGMOD’01
- C. Giannella, J. Han, J. Pei, X. Yan and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities, Kargupta, et al. (eds.), Next Generation Data Mining’04
References on Stream Data Mining (2)

- 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 2001
- S. Madden, M. Shah, J. Hellerstein, V. Raman, Continuously Adaptive Continuous Queries over Streams, SIGMOD02
- 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