Online Mining Data Streams
Problems, Applications and Progress

Jian Pei[1] Haixun Wang[2] Philip S. Yu[2]
[1]Simon Fraser University, Canada & State University of New York at Buffalo, USA
[2]IBM T.J. Watson Research Center, USA

The latest version of the handouts is downloadable at
http://www.cse.buffalo.edu/faculty/jianpei
http://wis.cs.ucla.edu/~hxwang

Outline

- Introduction
  - Applications and Challenges
- Online mining data streams – problems and techniques
  - Synopsis maintenance; Classification, regression and learning; Stream data mining languages; Frequent pattern mining; Clustering; Change and novelty detection
- Summary: Mining data stream – inherent challenges and open problems

Motivations

- A large number of applications generate data streams
  - Telecommunication (call records)
  - System management (network events)
  - Surveillance (sensor network, audio/video)
  - Financial market (stock exchange)
  - Day to day business (credit card, ATM transactions, etc)
- Tasks: Real time pattern discovery, query answering, statistics maintenance on data streams

Classification on Data Sources

Volume (high/low)

Structured?

Structured Low-volume	Structured High-volume
Un-structured Low-volume	Un-structured High-volume

Data Sources: Structured

- Structured low-volume
  - Wire services , Phone call connection reports, Phone and organization directory, Badge access tracking, Customer Lists, Account History, Personal address book, Personal records, Payroll data bases, Expense reports, Logs of tunnel activities, Purchasing logs, Supplier relationships, Work logs/project history, Temperature in machine room for IS reliability, Active monitoring remote copy to disaster site, Disaster site monitoring, Credit reports, Biometric access control
- Structured high-volume
  - Stock Exchange Transactions, Web pages for news/weather, Access, audit records, CRM Data bases, Web access logs and network logs, Company Web site, Mutual fund valuation and transactions, "Financial product" sales, Credit/Debit card transactions, RFID Tracking Logs, Analog Signatures

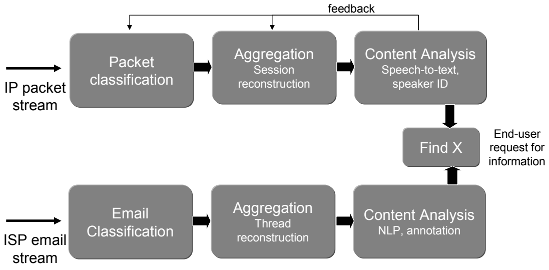
Data Sources: Un-Structured

- Unstructured low-volume
  - Email, Trading floor sound, Chat, Instant Messages, Reports – Internal, Printed reports, Hand phone logs, Courier records, Call Center Data & Logs, Pager, External proprietary reports and data, Customer enquiries, Customer complaints, Public records, Patents, FAX, Scanned checks, RF Monitoring (look for rogue hubs), Print stream monitoring, Calendars
- Unstructured high-volume
  - Phone calls (e.g. voip) content, Broadcast media (TV& Radio), Web Radio, Web cams, Web crawl, Surveillance cameras (internal), Video conferences, Phone conferences, Voice Mail, Satellite photos, Laptop Desktop contents, Cypher detection, Pervasive device communication (second path outside infrastructure), Baby monitor

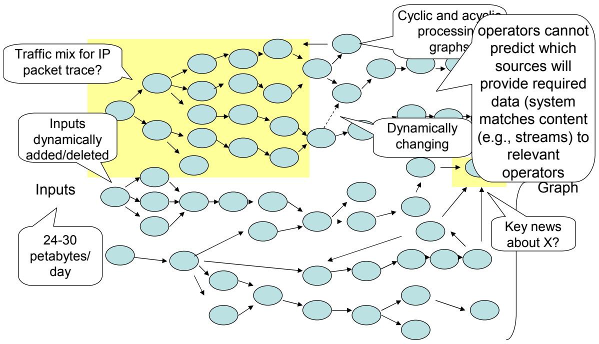
Data Stream Characteristics

- Records arrive at a rapid rate
- Huge volumes of continuous data, possibly infinite
- Fast changing
- Requires fast, real-time response
- Random access is expensive—single linear scan algorithm (*can only have one look*)
- Store only the summary of the data seen thus far

A Stream Application Example



Processing and Dataflow


Cyclic and acyclic processing graphs
operators cannot predict which sources will provide required data (system matches content (e.g., streams) to relevant operators
Traffic mix for IP packet trace?
Inputs dynamically added/deleted
Dynamically changing
Inputs
Graph
Key news about X?
24-30 petabytes/ day

Why Mining Data Streams?

- Summarization
  - Detailed streaming data is hard to understand
  - Patterns as summarization
- Prediction
- Change detection
  - Concept drifting
  - Exception and anomaly detection: novelty, intrusions, …

Challenges in Mining Data Streams

- Limited resources
  - Fixed main memory size, limited disk space
  - Limited CPU and I/O time per record for online processing
  - How to online compute on data streams?
- Unlimited streaming data – you have only one look
  - How to collect and summarize data streams?
- Unlimited user queries and exploration
  - How to answer various user queries from limited summarization data?

Ultimate Goals

- Effective knowledge delivered to users
- How to present patterns found in streams?
- How to visualize changes in data streams?
- Hands-off stream mining

Online Mining Data Streams

- Synopsis maintenance
- Classification, regression and learning
- Stream data mining languages
- Frequent pattern mining
- Clustering
- Change and novelty detection

Compute Synopses on Streams

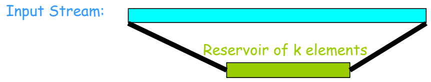
- Sampling
  - Find uniform random samples of an infinite data stream
- Approximate order statistics
  - Medians and quantiles
  - Approximate frequency counts
- Top-K monitoring on data streams

Sampling

- Input:
  - Stream of data that arrive online
  - Sample size k
  - Sample range
    - entire stream
    - most recent window (count-based or time-based)
- Output:
  - k elements chosen uniformly at random within the sample range

Reservoir Sampling

- Classical algorithm by Vitter (1985):
  - Size of data stream is not known in advance
  - Goal: maintains a *fixed-size uniform random sample*



  - Put the first k elements from the stream into the repository
  - When the i-th element arrives
    - Add it to reservoir S with probability p= k/i
    - If added, randomly remove an element from S
    - Instead of flipping a coin for each element, determine the number of elements to skip before the next to be added to S

Duplicates in Stream

- Observation:
  - Stream contains duplicate elements
  - e.g. Zipf distribution
  - Any value occurring frequently in the sample is a wasteful use of the available space

Concise Sampling

- By Gibbons and Matias, 1998
  - Represent an element in the sample by (value, count)
  - $\tau = 1$
  - Add new element with probability $1/\tau$ (increase count if element already in S)
  - If S is full
    - increase τ to τ'
    - evict each element (or decrease count) from S with probability τ / τ'

Sampling from a Moving Window

- Timeliness: old data are not useful
- Restrict samples to a window of recent data
  - As new data arrives, old data "expires"
- Reservoir sampling cannot handle data expiration
  - Replace an "expired" element in the reservoir with a random element in the current window
  - But we cannot access the window!

A Naïve Algorithm

- Use reservoir sampling to find a sample of size k for the first n elements in the stream
- Place a moving window on the stream
  - an old element *y* expires when a new element *x* arrives
- If *y* is in not in the reservoir, we do nothing, otherwise we replace *y* with *x*
- Problem: periodicity
  - If j-th element is in the sample, then any element with index j+cn is in the sample

Chain Sampling

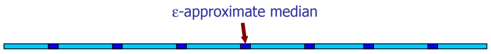
- Babcock, Datar, Motwani, 2002
- Motivation:
  - When an element x is added to the sample, decide immediately which future element y will replace x when x expires
  - Store y when y arrives (x has not expired yet)
  - Of course, we must decide which future element will replace y, …
  - Thus, we don't have to look back!

Chain Sampling

- Include each new element in the sample with probability 1/min(i,n)
- The i-th element is added to the sample …
  - because when it expires, the window will be (i+1,…,i+n)
  - so we randomly choose a future element whose index is in [i+1, i+n] to replace it when it expires
- Once the element with that index arrives, store it and choose the index that will replace it in turn, building a "chain" of potential replacements
- When an element is chosen to be discarded from the sample, discard its "chain" as well

Order Statistics

- Median
- φ-Quantile: element with rank $\lceil \varphi N \rceil$ $0 < \varphi < 1$
- Finding exact quantile requires linear space
- ε-Approximate φ-quantile: any element with rank $\lceil (\varphi \pm \varepsilon) N \rceil$ $0 < \varepsilon < 1$



Approximate Quantile

- **Task:** given ε, δ and φ, devise an online algorithm to compute, with probability at least 1-δ, an ε-approximate φ-quantile of a stream.
- Typical $\varepsilon = 0.01$
- Typical $\delta = 0.0001$

φ-Quantile and Sample Size

- Reservoir sampling generates a sample of size k without a priori knowledge of stream size
- Known fact: if sample size is $O(\varepsilon^{-2}\log\delta^{-1})$ then the φ-quantile of the sample is an ε-approximate quantile of the input with probability at least $1- \delta$
- However, ε^{-2} is too large!

Sampling with Unknown N

```
Given
    $b$ buffers of $k$ elements each.
Repeatedly
    If there is an empty buffer
        Fill buffer with elements in the stream
        Assign buffer with weight 1
    Reclaim space with COLLAPSE.
Finally
    Output the $\varphi$-quantiles of last buffer.
```

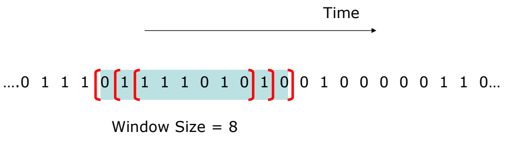
Memory Requirement

- Memory size = b·k
  - b: # of buffers
  - k: buffer size
- Output is an ε-approximate quantile of the input with probability at least $1-\delta$ if b and k satisfy certain constraints
- For $\varepsilon = 0.01$ and $\delta = 0.0001$, b·k can be an order of magnitude smaller than $O(\varepsilon^{-2}\log\delta^{-1})$

Statistics from a Moving Window

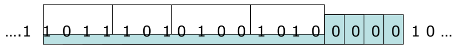
- Maintaining statistics
  - Count/Sum of non-zero elements
  - Variance
  - K-Mean
  - …
- Moving window
  - add/delete operation on synopsis data structure
  - but the exact expiring element is not known!

Moving Window



Moving Window

- **Challenge**: how to update synopsis data structure when we don’t have the exact value of the expired element?
- **Solution**: Exponential Histogram, by Datar et al
- **Example**: sum of elements in a window on a bit stream

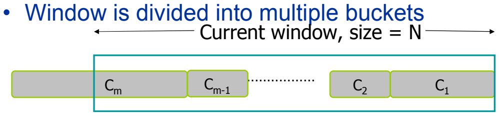


Bucket Sums = {3,2,1,2}

Bucket Sums = {2,1,2}

Histograms

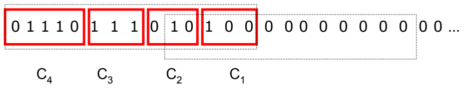
- Window is divided into multiple buckets



- Let C_i = count of 1's in the i-th bucket
- True count is at least $1 + \sum_{i=1}^{m-1} C_i$
- Estimate = $C_m / 2 + \sum_{i=1}^{m-1} C_i$
- Relative error < $C_m / 2(1 + \sum_{i=1}^{m-1} C_i)$

Error Bound

- Guarantee $C_m / 2(1 + \sum_{i=1}^{m-1} C_i) \le \varepsilon$
- As the window moves …



$C_4/2(1+C_3+C_2+C_1) < \varepsilon$
$C_3/2(1+C_2+C_1) < \varepsilon$
$C_2/2(1+C_1) < \varepsilon$

- C_m, C_{m-1}, …, C_2, C_1 must be exponentially decreasing

Exponential Histogram

- If a new element is '1', create a new bucket of size 1
- If there are *k* (depends on error bound) buckets of the same size, merge the oldest two into a single bucket of double the size

Exponential Histogram

- Example (k=3):
  32, 16, 8, 8, 4, 4, 2, 1, 1 + 1
  32, 16, 8, 8, 4, 4, 2, 2, 1 + 1
  32, 16, 8, 8, 4, 4, 2, 2, 1, 1 + 1
  32, 16, 8, 8, 4, 4, 2, 2, 2, 1
  32, 16, 8, 8, 4, 4, 4, 2, 1
  32, 16, 8, 8, 8, 4, 2, 1
  32, 16,16, 8, 4, 2, 1

Exponential Histogram

- Using O($\varepsilon^{-1}\log^2 N$) bits of memory, we can estimate the count to within a factor of 1+ ε
- EH can estimate any function *f* defined over windows that satisfies:
  - Positive: $f(X) \geq 0$
  - Polynomially bounded: $f(X) \leq poly(|X|)$
  - Composable: Can compute $f(X + Y)$ from $f(X)$, $f(Y)$ and little additional information
  - Weakly Additive: $(f(X) + f(Y)) \leq f(X + Y) \leq c(f(X) + f(Y))$

Variance over Moving Window

- Problem: last bucket cannot be ignored
- Solution (Babcock et al 2003):
  - Merge is triggered by variances between adjacent buckets
  - Ensure variance of merged bucket is small compared to combined variance of later buckets
  - Relative error ≤ ε, provided $V_m \leq (\varepsilon^2/9)\ V_{m*}$

Online Mining Data Streams

- Synopsis maintenance
- Classification, regression and learning
- Stream data mining languages
- Frequent pattern mining
- Clustering
- Change and novelty detection

Classification of Data Streams

- Challenges
- The Decision Tree Classifier
- Hoeffding Trees
- VFDT and CVFDT
- Ensemble of Classifiers

What are the Challenges?

- Data Volume
  - impossible to mine the entire data at one time
  - can only afford **constant memory** per data sample
- Concept Drifts
  - previously learned models are invalid
- Cost of Learning
  - model updates can be costly
  - can only afford **constant time** per data sample

The Decision Tree Classifier

- Learning (Training) :
  - Input: a data set of (a, b), where a is a vector, b a class label
  - Output: a model (decision tree)
- Testing:
  - Input: a test sample (x, ?)
  - Output: a class label prediction for x

The Decision Tree Classifier

- A divide-and-conquer approach
  - Simple algorithm, intuitive model
  - No ‘optimal’ model
- Compute information gain for data in each node
  - Super-linear complexity
- Typically a decision tree grows one level for each scan of data
  - Multiple scans are required
- The data structure is not ‘stable’
  - Subtle changes of data can cause global changes in the data structure

Challenge #1

- Task:
  - Given enough samples, can we build a tree in constant time that is ***nearly identical*** to the tree a batch learner (C4.5, Sprint, etc.) would build?
- Intuition:
  - With increasing # of samples, the # of possible decision trees becomes smaller
- Forget about concept drifts for now.

Hoeffding Bound

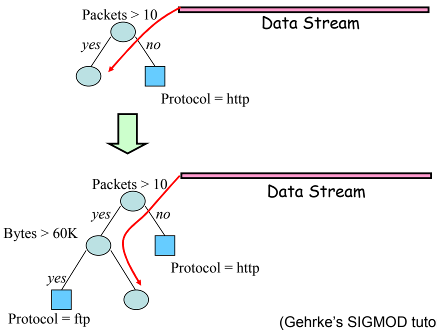
- Also known as additive Chernoff Bound
- Given
  - r : real valued random variable
  - n : # independent observations of r
  - R : range of r
- Mean of r is at least $r_{avg}-\varepsilon$, with probability $1-\delta$, or:
- $P(\mu_r \geq r_{avg} - \varepsilon) = 1-\delta$ and $\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$

Hoeffding Bound

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

- Properties:
  - Hoeffding bound is independent of data distribution
  - Error ε decreases when n (# of samples) increases
- At each node, we shall accumulate enough samples (n) before we make a split

Building a Hoeffding Tree



(Gehrke's SIGMOD tutorial)

Nearly Identical?

- Categorical attributes
  - with a high probability, the attribute we choose for split is the same attribute as would be chosen by a batch learner
  - identical decision tree
- Continuous attributes
  - discretize them into categorical ones

Building a Hoeffding Tree

- $G(X_i)$: the heuristic measure used to split a node (X_i is a discrete attribute)
- X_a, X_b : the attributes with the highest and second-highest G() after n examples
- $\Delta G = G(X_a) - G(X_b) \geq 0$

Building a Hoeffding Tree

- If $\Delta G > \varepsilon$, the Hoeffding bound states that :

$$P(\mu_{\Delta G} \geq \Delta G - \varepsilon > 0) = 1 - \delta$$

- $\mu_{\Delta G} > 0 \Rightarrow \mu_{G(X_a)} - \mu_{G(X_b)} > 0 \Rightarrow \mu_{G(X_a)} > \mu_{G(X_b)}$

- **Conclusion**: we have found a best attribute for split (X_a) with probability 1- δ

Hoeffding Tree: Pros and Cons

- Scales better than traditional DT algorithms
  - Incremental
  - Sub-linear with sampling
  - Small memory requirement
- Cons:
  - Only consider top 2 attributes
  - Tie breaking takes time
  - Grow a deep tree takes time
  - Discrete attribute only

VFDT

- Very Fast Decision Tree
  - Domingos, Hulten, 2000
- Various Improvement over Hoeffding Tree
  - Break near-ties more aggressively
  - G computed every n_{min} tuples (instead of for every tuple)
  - Deactivating unpromising leaf nodes
  - Dropping poor attributes
  - …
  - **Better time and memory performance**
- Still does not handle concept drifts

Concept Drifts

- Time-changing data streams
- Incorporate new samples and eliminate effect of old samples
- Naïve approach
  - Place a sliding window on the stream
  - Reapply C4.5 or VFDT whenever window moves
  - Time consuming!

CVFDT

- Concept-adapting VFDT
  – Hulten, Spencer, Domingos, 2001
- Goal
  – Classifying concept-drifting data streams
- Approach
  – Make use of Hoeffding bound
  – Incorporate “windowing”
  – Monitor changes of information gain for attributes.
  – If change reaches threshold, generate alternate subtree with new “best” attribute, but keep on background.
  – Replace if new subtree becomes more accurate.

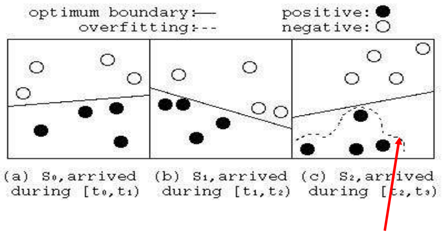
Sliding Windows

- Mining data streams = Mining windows of static data ?
- To design of a knowledge discovery system, we should be concerned with:
  – Time it takes to learn the model
  – Memory space for computation, model, and data
  – Sample size must be large enough (no longer a problem in the streaming environment?)

Drawbacks of Sliding Windows

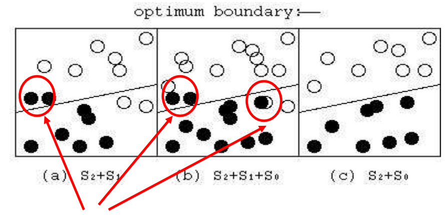
- Window-based incremental algorithm
  – incorporate new samples and eliminate effects of old samples
- Old samples = samples outside window = samples arrived t time units ago
- How to decide t?

Data Distribution and Optimal Decision Boundaries


optimum boundary:—
overfitting:--
positive: ●
negative: ○
(a) S0,arrived during [t0,t1)
(b) S1,arrived during [t1,t2)
(c) S2,arrived during [t2,t3)

Overfitting!

Data Distribution and Optimal Decision Boundaries

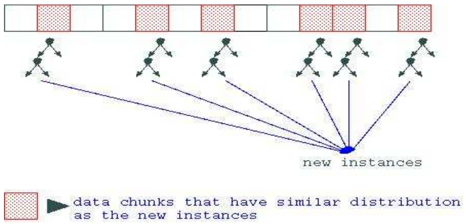


Conflicting Concepts!

Challenges

- How to 'forget' old samples?
  - Discard instances after a fixed time period T
  - T is too large: conflicting concepts
  - T is too small: overfitting
- Other issues of a single model approach
  - Runtime performance
  - Ease of use
  - Parallelizability
  - …

Classifier Ensemble Method


new instances
data chunks that have similar distribution as the new instances

Basic Idea

- Steam data is partitioned into sequential chunks
- Train a weighted classifier from each chunk
- The weight is based on the expected prediction accuracy on the current test examples
- Only top K classifiers are kept

Bias Variance Decomposition

- The expected added error of a classifier is expressed by:

$$Err = \frac{\delta^2_{\eta_c}}{s}$$

  - s is a constant independent of training model
  - $\delta^2_{\eta_c}$ denotes the variance

Accuracy Weighted Ensemble

$$\boxed{\begin{aligned} Err_i &= \frac{\delta^2_{\eta_c^i}}{s} \\ w_i &= \frac{c}{\delta^2_{\eta_c^i}} \end{aligned}}$$

Single Classifier

- Probability Output:

$$f_c^g(y) = p(c \mid y) + \eta_c^g(y)$$

- Assuming each partition is of the same size

$$\sigma^2_{\eta_c^g(y)} \geq \frac{1}{k^2} \sum_{i=n-k+1}^{n} \sigma^2_{\eta_c^i}$$

Ensemble Classifier

- Probability Output:

$$f_c^E(y) = p(c \mid y) + \eta_c^E(y)$$

- Naïve assumption: the variances of different classifiers are independent.

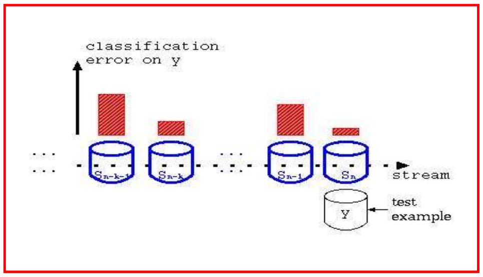
$$\sigma^2_{\eta_c^E(y)} = \sum_{i=n-k+1}^{n} w_i^2 \sigma^2_{\eta_c^i} \Big/ \left(\sum_{i=n-k+1}^{n} w_i \right)^2$$

- Conclusion: ensemble has smaller error

But in Reality …

- We do not know
  - Error or variance or the function being learned
- Solution:
  - Use estimation!
  - Apply the i-th classifier on the current training data to estimate the error of the classifier

Weight Estimation



Learning Cost Is Still High

- Tree construction has super-linear complexity
  - Much effort has been made to build a decision tree faster (CLOUD, Rainforest, etc.)
- Approximate methods: Hoeffding bound
  - We still need to evaluate G()
  - Incremental updating a non-stable structure
- Learn/update a model is costly

“Optimal” Decision Tree?

- The Hoeffding method tries to build decision trees “**nearly identical**” to trees that a batch classifier (ID3, C4.5, Sprint) would build
- But batch learner builds decision trees in a divide-and-conquer greedy manner

Ensemble of Random Decision Trees

- Build an ensemble of N random trees
- At each node, randomly choose an attribute to split
- Throw samples into each tree
- Use class label distribution in the leaf nodes as probability output
- Constant cost of tree construction
- Maximize structural diversity in N trees

Online Mining Data Streams

- Synopsis maintenance
- Classification, regression and learning
- Stream data mining languages
- Frequent pattern mining
- Clustering
- Change and novelty detection

Hancock

- Hancock: a language for data mining on streams [Cortes et al, KDD 2002]
- Maintain *signatures* of stream
  - signature = aggregate = synopsis data structure
- Goal: to replace hard-to-maintain, hand-written C code with Hancock code, so that users can shift focus from *how* to manage the data to *what* to compute.

Hancock's Computation Model

- Iterating over a sorted stream of transaction records

```
Iterate (over stream variable
          filteredby filter predicate
          sortedby sorting order
          withevents event detection function)
{
      event event-name (parameter) {
          operations;
      }
      …
};
```

Stream Data Mining and SQL

- Relational DBMSs have been a huge success since 1970s
- Data are in the relational form
  - (real time) data warehouses
  - Data generated on the fly by OLAP and other query tools
- Decades of efforts in building SQL engine
  - Parallelization
  - Performance
  - …
- A case study: association rule mining in SQL
  - SIGMOD test-of-the-time award 2003

ATLAS: Minimalist’s Extension of SQL

- Wang, Zaniolo et al, VLDB00, SIAM DM 02, VLDB04
- Computation model:

```
AGGREGATE avg (next INT)
{       TABLE memo(sum INT, cnt INT);
        INITIALIZE: {
                INSERT INTO memo VALUES (next, 1);
        }
        ITERATE: {
                UPDATE memo SET sum=sum+next, cnt = cnt + 1;
                INSERT INTO return SELECT sum/cnt FROM memo;
        }
        EXPIRE:{
                UPDATE memo SET sum=sum-next, cnt = cnt -1;
        }
}
```

Sticky Sampling in SQL

```
AGGREGATE basicCount (next INT, t TIMESTAMP, k INT)
{
    WINDOW hist(h INT, t TIMESTAMP);
    TABLE memo(last INT, total INT) AS VALUES (0,0);
    INITIALIZE: ITERATE: {
          INSERT INTO hist VALUES(next, t) WHERE next >0;
          UPDATE memo SET total = total + next;
          SELECT merge(h, t, k) OVER (ORDER BY t DESC) FROM hist;
          /* Update last pointer due to merge */
          UPDATE memo SET last = (SELECT max(h) FROM hist);
   }
   EXPIRE:{
          UPDATE memo SET last = h/2
                  WHERE (SELECT count(1) FROM hist h WHERE h.h = last) =1;
          UPDATE memo SET total = total - h/2 - last/2;
  }
}
```

Decision Tree Classifier in SQL

```
AGGREGATE classify(iNode Int, iRecId Int, iCol Int, iValue Int, iYorN Int)
{   TABLE treenodes(RecId Int, Node Int, Col Int, Value Int, YorN Int);
    TABLE mincol(Col Int);
    TABLE summary(Col Int, Value Int, Yc Int, Nc Int) INDEX (Col,Value);
    TABLE ginitable(Col Int, Gini Int);
    INITIALIZE : ITERATE : {
            INSERT INTO treenodes VALUES(iRecId, iNode, iCol, iValue, iYorN);
            UPDATE summary SET Yc=Yc+iYorN, Nc=Nc+1-iYorN
                    WHERE Col = iCol AND Value = iValue;
            INSERT INTO summary
                    SELECT iCol, iValue, iYorN, 1-iYorN WHERE SQLCODE<>0;
    }
TERMINATE : {
            INSERT INTO ginitable
                    SELECT Col, sum((Yc*Nc)/(Yc+Nc))/sum(Yc+Nc) FROM summary
                    GROUP BY Col HAVING count(Value) > 1 AND sum(Yc)>0 AND sum(Nc)>0;
            INSERT INTO mincol SELECT argmax(Col, Gini) FROM ginitable;
            INSERT INTO result SELECT iNode, Col FROM mincol;
            SELECT classify(t.Node*MAXVALUE+m.Value+1, t.RecId, t.Col, t.Value, t.YorN)
                    FROM treenodes AS t,
                    ( SELECT tt.RecId RecId, tt.Value Value
                    FROM treenodes AS tt, mincol AS m WHERE tt.Col=m.Col) AS m
                    WHERE t.RecId = m.RecId
                    GROUP BY m.Value;
    }
  }
```

Why SQL Matters to Data Streams and Data Mining

- Why SQL Matters to Data Streams and Data Mining?
  - The code is very short, which saves programming efforts
  - It is implemented in a declarative language, which means optimization opportunity
- It is a tightly-coupled approach, where you don't
  - move the data outside a DBMS
  - write ad-hoc programs to mine the data
  - lose the support of DBMS, or worry about data generated on the fly
- Extended SQL for stream processing has strong expressive power
  - Law, Wang, Zaniolo VLDB 2004

Mining Data Streams in SQL

```
AGGREGATE classifystream(col1, ..., coln, label Int) : Int
{   TABLE state(cnt Int) AS VALUES (0);
    INITIALIZE : ITERATE : {}
    REVISE : {
        SELECT learn(W.*) FROM WINDOW AS W
                WHERE ((SELECT cnt FROM state) = 0
                OR ((SELECT cnt FROM state) % 1000 = 0 AND
                (SELECT sum(|classify(W.*)-W.label|) FROM WINDOW AS W
                WHERE W.label=NULL)⩾ threshold))
                AND W.label <> NULL;
        UPDATE state SET cnt=cnt+1;
        INSERT INTO RETURN
        SELECT classify(V.*) FROM VALUES(col1,...,coln) AS V
        WHERE label = NULL;
    }
}
```

The Beauty of SQL

```
SELECT classifystream(S.*)
OVER (ROWS 10000 PRECEDING)
FROM stream AS S;
```

- Our implementation has taken into consideration:
  - Code optimization;
  - Parallelization;
  - Distributed databases;
  - Ease of use;
  - ...

Online Mining Data Streams

- Synopsis maintenance
- Classification, regression and learning
- Stream data mining languages
- Frequent pattern mining
- Clustering
- Change and novelty detection

Frequent Pattern Mining

- Frequent patterns: patterns (set of items, sequence, etc.) that occur frequently in a database [AIS93]
- Frequent pattern mining: finding regularities in data
  - What products were often purchased together?
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we classify web documents based on key-word combinations?

Why Is Frequent Pattern Mining Essential?

- Foundation for many data mining tasks
  - Association rules, correlation, causality, sequential patterns, spatial and multimedia patterns, associative classification, cluster analysis, iceberg cube, …
- Broad applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log (click stream) analysis, …

Basics

- Itemset: a set of items
  - E.g., acm={a, c, m}
- Support of itemsets
  - Sup(acm)=3
- Given min_sup = 3, acm is a frequent pattern
- Frequent pattern mining: find all frequent patterns in a database

Transaction database TDB

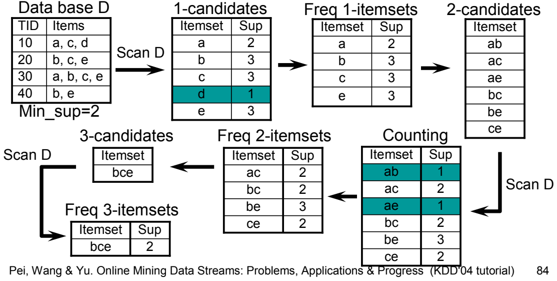
TID	Items bought
100	f, a, c, d, g, I, m, p
200	a, b, c, f, l, m, o
300	b, f, h, j, o
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

A Priori: Candidate Generation-and-test

- Any subset of a *frequent* itemset must be also *frequent* – an anti-monotone property
  - A transaction containing {beer, diaper, nuts} also contains {beer, diaper}
  - {beer, diaper, nuts} is frequent → {beer, diaper} must also be frequent
- In other words, any superset of an *infrequent* itemset must also be *infrequent*
  - No superset of any infrequent itemset should be generated or tested
  - Many item combinations can be pruned!

Apriori Algorithm

- A level-wise, candidate-generation-and-test approach (Agrawal & Srikant 1994)



Data base D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=2

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

3-candidates

Itemset
bce

Freq 3-itemsets

Itemset	Sup
bce	2

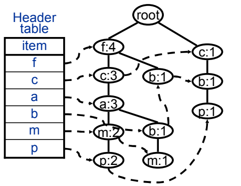
The Apriori Algorithm

- C_k: Candidate itemset of size k
- L_k : frequent itemset of size k

- L_1 = {frequent items};
- for (k = 1; L_k !=∅; k++) do
  - C_{k+1} = candidates generated from L_k;
  - for each transaction t in database do increment the count of all candidates in C_{k+1} that are contained in t
  - L_{k+1} = candidates in C_{k+1} with min_support
- return $\cup_k L_k$;

Compress Database by FP-tree

- 1st scan: find freq items
  - Only record freq items in FP-tree
  - F-list: f-c-a-b-m-p
- 2nd scan: construct tree
  - Order freq items in each transaction w.r.t. f-list
  - Explore sharing among transactions



TID	Items bought	(ordered) freq items
100	f, a, c, d, g, I, m, p	f, c, a, m, p
200	a, b, c, f, l,m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Partition Frequent Patterns

- Frequent patterns can be partitioned into subsets according to f-list: f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, or p
  - Pattern f
- The partitioning is complete and without any overlap

Find Patterns Having Item "*p*"

- Only transactions containing *p* are needed
- Form *p*-projected database
  - Starting at entry *p* of the header table
  - Follow the side-link of frequent item *p*
  - Accumulate all transformed prefix paths of *p*

p-projected database TDB|$_p$

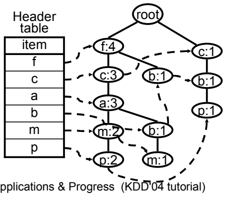
fcam: 2

cb: 1

Local frequent item: *c*:3

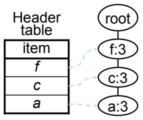
Frequent patterns containing *p*

p: 3, *pc*: 3

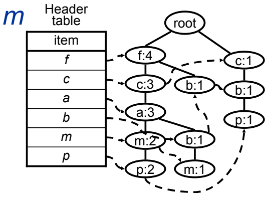


Find Patterns Having Item *m* But No *p*

- Form *m*-projected database TDB|*m*
  - Item *p* is excluded (why?)
  - Contain *fca*:2, *fcab*:1
  - Local frequent items: *f*, *c*, *a*
- Build FP-tree for TDB|*m*

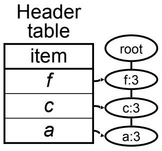


m-projected FP-tree



Recursive Mining

- Patterns having m but no p can be mined recursively
- Optimization: enumerate patterns from single-branch FP-tree
  - Enumerate all combination
  - Support = that of the last item
    - *m*, *fm*, *cm*, *am*
    - *fcm*, *fam*, *cam*
    - *fcam*



m-projected FP-tree

FP-growth

- Pattern-growth: recursively grow frequent patterns by pattern and database partitioning
- Algorithm
  - For each frequent item, construct its projected database, and then its projected FP-tree
  - Repeat the process on each newly created projected FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path — single path generates all the combinations, each of which is a frequent pattern

Mining Data Streams – Challenges

- Maintaining exact counts for all (frequent) itemsets needs multiple scans of the stream
  - Maintain approximation of counts
- Finding the exact set of frequent itemsets from data streams cannot be online
  - Have to scan data streams multiple times
  - Space overhead
  - Finding approximation of set of frequent itemsets

Mining Data Streams – A Roadmap

- Basic extensions
  - Finding frequent items/itemsets from a stream
- Advanced extensions – mining time-sensitive frequent patterns
  - Finding frequent patterns in sliding window
  - Mining recently frequent patterns
  - Mining temporal frequent patterns
- Applications
  - Hierarchical heavy hitters and semi-structured patterns

Finding Frequent Items in Streams

- $S=x_1x_2\ldots x_n$ is a stream of items, where $x_i \in I$, and I is the set of items
  - An item x can appear multiple times in the stream
  - Assumption: n » $|I|$ »$1/\theta$
- For a support threshold θ ($0 \leq \theta \leq 1$), find the set of items that appears more than θn times in S?
  - Small average processing time of each item in S
  - Worst-case time: the maximal time among all items in S
  - Number of passes: one scan for online algorithms
  - Space overheads: should be bounded

Space Requirement

- Any online algorithm for computing frequent items needs in the worse case $\Omega(|I|log(n/|I|))$ bits
  - $|I|$: the number of distinct items
  - n: the length of the stream
  - [Karp et al. 03] and [Demaine et al. 02]
- Intuition: in the middle of a stream S, if no item so far has a count over θn, then the count of each item has to be remembered
  - Keep the count combinations i.e., the set of all sequences of n integers between 0 and $\theta n-1$ adding to $\lfloor n/2 \rfloor$

Idea

- Given a stream of two symbols: x and y, find the dominating symbol by using only one counter
- Set count=0
- If we see x, count=count+1
- If we see y, count=count-1
- After all
  - If count>0, then x is the dominant symbol
  - If count=0, then tie
  - If count <0, then y is the dominant symbol

Generalization for Multiple Items

- Observation: # of frequent items <= *1/θ*
- Compute top-$\lfloor 1/\theta \rfloor$ most frequent items using *O(1/θ)* memory cells

```
let K be a set of $\lfloor 1/\theta \rfloor$ counters, each counter can have a label
initially, the counters are set  to 0;
for i=1 to n do
    if $x_i$ is in K then increase its count
    else // i.e., $x_i$ is not in K
        if |K| < $\lfloor 1/\theta \rfloor$ then insert $x_i$ into K and set its count to 1
        else // i.e., K is full at the current moment
            decrease each counter by 1;
            delete all items from K whose count is 0;
Output K // the counts in K are not the real counts
```

Completing the Job

- K tells the frequent items, but not the counts
- Another scan over the bag tells the counts
  - Another scan over the stream may not be feasible in practice
  - One scan algorithms obtaining both frequent items and the estimated counts will be discussed soon

Extension: Mining Frequent Itemsets

- Use the KPS algorithm to find frequent 2-itemsets
  - Treat each 2-itemset as an “item” in KPS algorithm
- Use Apriori to generate and test candidates, and get frequent k-itemsets (k>2)
  - Still need two scans to find estimated counts
- [Jin & Agrawal, 04]

Approximating Frequency Counts

- [Manku & Motwani, 02]
- Input
  - Support threshold s, 0 < s < 1
  - Error parameter e, 0 < e < 1, e « s
  - The length of the stream s is n
- Features
  - No false negatives: all frequent item(set)s are output
  - No item(set)s whose true frequency is less than *(s-e)n* is output
  - Estimated frequencies are less than the true frequencies by at most *en*

Sticky Sampling – Ideas

- Central idea: frequent items and their supports can be estimated by a good sample
- One sample rate cannot handle a potentially infinite stream – the sample is also a stream
  - Adjust (decrease) sample rate progressively to handle more and more new data
  - The first t items, take them; the next 2t items, sample using rate 0.5; the next 4t items, sample using rate 0.25, and so on
- How to keep counts from samples of different rates consistent?
  - Adjust counts according to the sampling rate

Sticky Sampling – Algorithm

- Maintain a set S of entries (x, f), where x is an item and f is the estimated count
- Initially, S is empty, sampling rate r=1
  - An element has a probability of 1/r to be sampled/counted
  - If an item is in S, increment the frequency
  - Otherwise, add an entry (x, 1) into S

Sticky Sampling – Algorithm

- Adjust sampling rate to handle more data
  - $t = e^{-1}\log(s^{-1}\delta^{-1})$, δ is the probability of failure
  - First 2t elements, r=1; next 2t elements, r=2, next 4t elements, r=4, …
- Update estimated counts for adjusted sampling rates
  - Diminishing f by a random variable in geometric distribution,
  - After adjustment, f is as if counted with the adjusted sampling rate
- Frequent items: entries in S where $f \geq (s-e)n$

Sticky Sampling – Properties

- Compute frequent items with error bound e
  - With probability at least 1- δ using at most $2/e\log(s^{-1}\delta^{-1})$ expected number of entries
- Space complexity is independent of n
- May still have chance to fail – violate one of the following three requirements
  - No false negatives: all frequent item(set)s are output
  - No item(set)s whose true frequency is less than *(s-e)n* is output
  - Estimated frequencies are less than the true frequencies by at most *en*

Lossy Counting – Ideas

- Divide the stream into buckets, maintain a global count of buckets seen so far
- For any item, if its count is less than the global count of buckets, then its count does not need to be maintained
  - How to divide buckets so that the possible errors are bounded?
  - How to guarantee the number of entries needed to be recorded is also bounded?

Lossy Counting – Algorithms

- Divide a stream into buckets of width $w=\lceil 1/e \rceil$
  - The current bucket id $b=\lceil n/w \rceil$
- Maintain a set D of entries (x, f, Δ), where Δ is the maximum possible error in f
- Whenever a new x arrives, lookup D
  - If x is in D, update f
  - Otherwise, add (e, 1, b-1) into D
- After a bucket, remove entries where $f+\Delta \leq b$
- At most $e^{-1}\log(en)$ entries in S
  - Practically better than Sticky Sampling

From Frequent Items to Itemsets

- Maintain a set D of entries (X, f, Δ), where X is an itemset
- Divide the incoming transaction stream into buckets, each has $w=\lceil 1/e \rceil$ transactions
  - The current bucket id is b

Finding Frequent Itemsets

- Fill available main memory with as many transactions as possible
  - B: # buckets in main memory in the current batch, must be a large number
  - For each entry (X, f, Δ) in D, update f; if $f+\Delta \leq b$, delete the entry
  - If an itemset X not in D has frequency $f \geq B$ in the current batch, create a new entry (X, f, b-B)
- Efficient implementation techniques in [Manku & Motwani, 02]

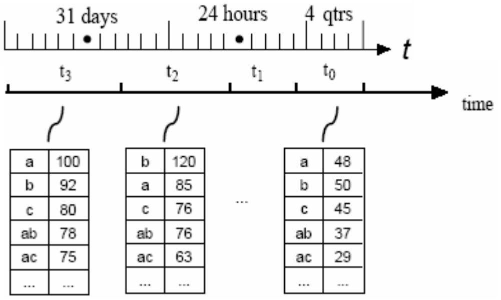
More Interesting Patterns from Streams

- People sometimes are more interested in recently frequent patterns or frequent patterns in some specific periods, instead of patterns in the stream having seen so far
- Challenges
  - How to answer queries about patterns in different periods?
    - Space and time overheads
  - How to summarize data and forget details?

Time-sensitive Frequent Patterns

- Mining time-sensitive frequent patterns
  - Maintain frequent patterns in different periods
- More recent patterns are more interesting – should be in finer granularity
  - Use tilted-time window to record changes in various granularity over different periods

Example



a	100
b	92
c	80
ab	78
ac	75
...	...

b	120
a	85
c	76
ab	76
ac	63
...	...

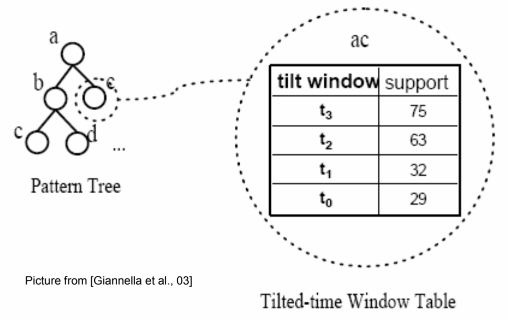
a	48
b	50
c	45
ab	37
ac	29
...	...

Picture from [Giannella et al., 03]

FP-stream

- Use FP-tree to mine frequent itemsets at the current batch of transactions
  - Soft-online: each transaction is mined only once
- Maintain frequent patterns in an FP-tree-like FP-stream structure
  - Provide approximate answers to queries about frequent patterns in any specific periods
- Details in [Giannella et al., 03]

Example



tilt window	support
t_3	75
t_2	63
t_1	32
t_0	29

Picture from [Giannella et al., 03]

Finding Recently Frequent Itemsets

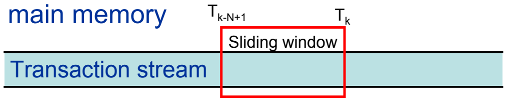
- Modeling preference on recently frequent patterns by weighting transactions
  - Recent transactions have higher weights
  - Use a decay factor d (0 < d < 1)
  - A just arrived transaction has weight 1
  - When a new transaction arrives, each previous transaction’s weight is decayed by d
- Recently frequent itemsets: frequent in the weighted transactions [Chang & Lee, 03]

Online Maintenance of Recently Frequent Patterns

- For an itemset X, use the counts of its subsets to estimate its count
  - Only potential itemsets are tested and maintained
- Maintain an upper part of the itemset lattice (the set of possible frequent itemsets)
  - Update the counts in the lattice for every new transaction

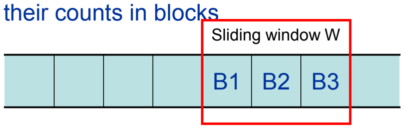
Mining Frequent Items in Sliding Windows

- Can we maintain counts of frequent items in a sliding window?
  - Fixed size window of width N
  - Variable size window of maximal width N
- Assumption: the sliding window can be scanned only once and cannot be held into main memory



Intuitions

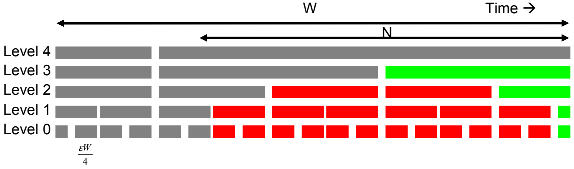
- Divide a stream into blocks and get frequent items and their counts in blocks



Frequent patterns in W can be derived approximately from B1, B2 and B3

- Quality problem: The errors will be accumulated from blocks
  - The maximal error in W is the sum of the maximal errors in B1, B2 and B3

Multiple Levels of Blocks



- High level blocks have frequent patterns over longer periods
- Error-bound in each level $\varepsilon_l = \frac{\varepsilon}{2(2L+2)} 2^{(L-l)}$
  - Higher level blocks have lower percentage of errors

How to Derive Frequent Items in Blocks?

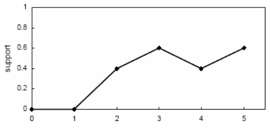
- Use the algorithms similar to [Manku, Motwani, 02]
- Can be extended to maintain quantiles
  - Φ-quantile (0<Φ≤1) of a sliding window is the set of items with rank ⌈ΦN⌉ in count, where N is the number of items in the window
  - 0.5-quantile is the median
- Can be extended to compute sketches for unbounded-windows under some condition

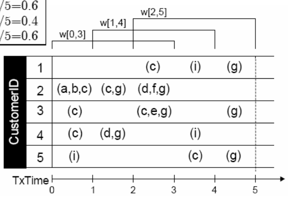
Temporal Patterns over Streams

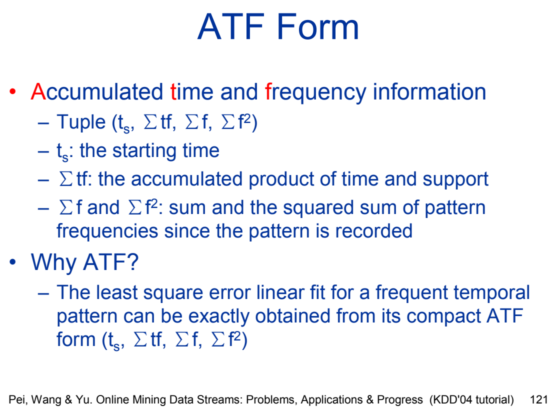
- Inter-transaction frequent itemsets in a sliding window [Teng, Chen, Yu, 03]

TxTime		Occurrence(s) of {c, g}	Support
t=1	w[0,1]	none	0
t=2	w[0,2]	CustomerID={2, 4}	2/5=0.4
t=3	w[0,3]	CustomerID={2, 3, 4}	3/5=0.6
t=4	w[1,4]	CustomerID={2, 3}	2/5=0.4
t=5	w[2,5]	CustomerID={1, 3, 5}	3/5=0.6

Picture from [Teng, Chen, Yu, 03]

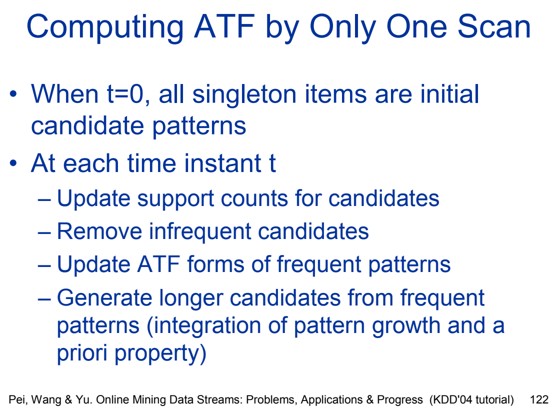






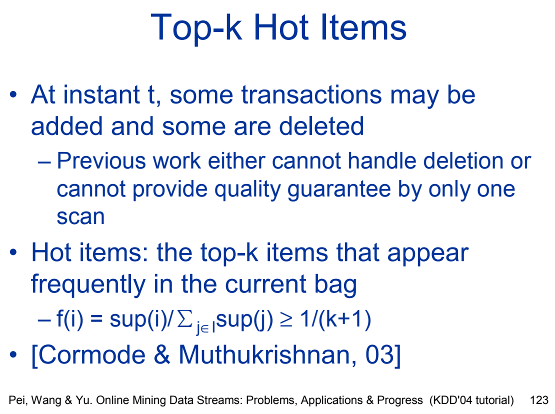
ATF Form

- Accumulated time and frequency information
  - Tuple (t_s, Σtf, Σf, Σf^2)
  - t_s: the starting time
  - Σtf: the accumulated product of time and support
  - Σf and Σf^2: sum and the squared sum of pattern frequencies since the pattern is recorded
- Why ATF?
  - The least square error linear fit for a frequent temporal pattern can be exactly obtained from its compact ATF form (t_s, Σtf, Σf, Σf^2)



Computing ATF by Only One Scan

- When t=0, all singleton items are initial candidate patterns
- At each time instant t
  - Update support counts for candidates
  - Remove infrequent candidates
  - Update ATF forms of frequent patterns
  - Generate longer candidates from frequent patterns (integration of pattern growth and a priori property)

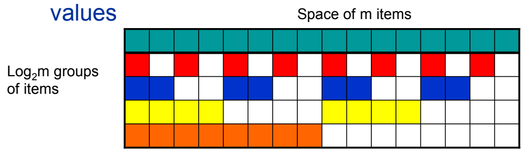


Top-k Hot Items

- At instant t, some transactions may be added and some are deleted
  - Previous work either cannot handle deletion or cannot provide quality guarantee by only one scan
- Hot items: the top-k items that appear frequently in the current bag
  - $f(i) = sup(i)/\Sigma_{j \in I} sup(j) \geq 1/(k+1)$
- [Cormode & Muthukrishnan, 03]

Tracking Hot Items

- Idea: use group testing
- Deploy log m counters
  - By overlap of counters, determine hot items
  - Each test includes half of the range [1 . . .m], corresponding to the binary representation of values



More Space Efficient by Hashing

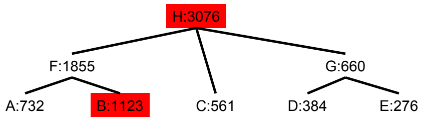
- Instead of using group test, we can maintain a group of h hashing functions
  - Less space overheads
  - Comparable accuracy
- [Jin et al., 03]

```
Function hCount
 if insertion then N++;
 else N--;
 for j=1 to h do
    pos=((a_j k + b_j) mod P) mod m;
    if insertion then S[pos][j]++;
    else S[pos][j]--;
```

```
Function eFreq(s)
  for k=1 to M do
    c = min_{1≤j≤h}(S[H_j(k)][j])
    if c < sN then output(k, c/N);
```

Hierarchical Heavy Hitters

- A frequent item is also called a heavy hitter
- Given a hierarchy on items, an element is a hierarchical heavy hitter if the frequency of the element except for its heavy hitter descendants is frequent
  - Example: a bag of 100,000 items, Φ=1%



Naïve Approaches

- Treat each element independently
- Use lossyCount [Manku & Motwani, 02] to find frequent items
- Remove false positives to find HHH
- Problem: the naïve approach is not "hierarchy-aware"
  - A heavy hitter makes all of its ancestors frequent
  - There can be many false positives

Hierarchy-Aware Methods

- Maintain a trie of samples from the stream
  - For each node in the trie, maintain the lower- and upper-bounds on the frequencies
- Divide the stream into buckets of $w=\lceil 1/\epsilon \rceil$
- Two phases and four strategies
  - Insertion: read the current bucket
  - Compression: merging auxiliary values and remove unnecessary nodes
- [Cormode et al., 03]

From Frequent Itemsets to Semi-structured Patterns

- [Asai et al., 02]
- Input: a stream of labeled trees (e.g., XML documents)
- Output: at each instant, a set of frequent tree patterns
- Steps: at each instant
  - Update the support counts of previous patterns
  - Remove infrequent patterns
  - Generate new candidates

Summary – Frequent Patterns

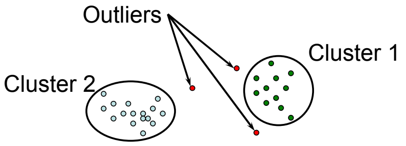
- Mining frequent patterns from data streams is challenging
  - A data stream can be scanned only once
  - Limited space for storing support counts
  - Quality warrant expected
- Solutions
  - Carefully designed sketches to capture the critical counting information
- Applications
  - With various constraints, different types of data

Online Mining Data Streams

- Synopsis maintenance
- Classification, regression and learning
- Stream data mining languages
- Frequent pattern mining
- Clustering
- Change and novelty detection

What Is Clustering?

- Group data into clusters
  - Similar to one another within the same cluster
  - Dissimilar to the objects in other clusters
  - Unsupervised learning: no predefined classes



Application Examples

- A stand-alone tool: explore data distribution
- A preprocessing step for other algorithms
- Pattern recognition, spatial data analysis, image processing, market research, WWW, intrusion detection, …
  - Cluster documents
  - Cluster web log data to discover groups of similar access patterns

What Is Good Clustering?

- High intra-class similarity and low inter-class similarity
  - Depending on the similarity measure
- The ability to discover some or all of the hidden patterns

Requirements of Clustering

- Scalability
- Ability to deal with various types of attributes
- Discovery of clusters with arbitrary shape
- Minimal requirements for domain knowledge to determine input parameters

Requirements of Clustering

- Can deal with noise and outliers
- Insensitive to the order of input records
- Can handle high dimensionality
- Incorporation of user-specified constraints
- Interpretability and usability

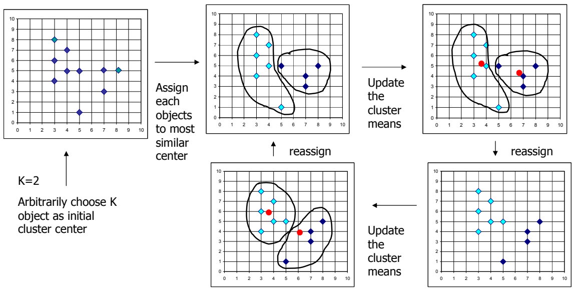
Categories of Clustering Approaches (1)

- Partitioning algorithms
  - Partition the objects into k clusters
  - Iteratively reallocate objects to improve the clustering
- Hierarchy algorithms
  - Agglomerative: each object is a cluster, merge clusters to form larger ones
  - Divisive: all objects are in a cluster, split it up into smaller clusters

Categories of Clustering Approaches (2)

- Density-based methods
  - Based on connectivity and density functions
  - Filter out noise, find clusters of arbitrary shape
- Grid-based methods
  - Quantize the object space into a grid structure
- Model-based methods
  - Use a model to find the best fit of data
- Pattern-based methods
  - A group of objects approximately following a pattern form a cluster

K-Means: Example



K=2

Arbitrarily choose K object as initial cluster center

Assign each objects to most similar center

Update the cluster means

reassign

reassign

Update the cluster means

K-means Clustering on Streams

- A data stream arrives in chunks X_1, …, X_n, each chunk fits in main memory
- Assumptions
  - Each chunk can be loaded into main memory only once
  - Limited main memory
- Challenges
  - Can we find a simple, fast, constant-factor-approximation k-median algorithm on one chunk?
  - How to derive global clusters?
- [O'Callaghan et al., 02]

Finding Good Initial Clustering

- Applied on the first chunk, 8-approximation to optimum
- Parameter: z – facility cost
- Steps
  - Reorder data points randomly
  - Create a cluster center at the first point
  - For every point after the first
    - Let *d* be the distance from the current data point to the nearest existing cluster center
    - With probability *d/z* create a new cluster center at the current data point; otherwise, add the current point to the best current cluster

LOCALSEARCH

- Starting from the initial clustering
- If the number of clusters in the initial clustering is far from k, adjust z
- Otherwise, adjust centers and assignments of points to clusters
- Details in [O'Callaghan et al., 02]

Deriving Global Clusters

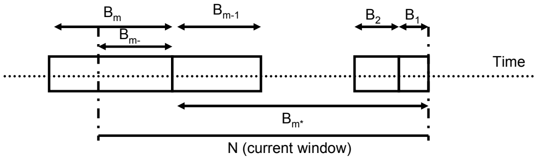
- For each chunk
  - Determine whether the chunk consists of mostly of a set of fewer than k points repeated over and over. If so, re-represent the chunk as a weighted data set such that each distinct point appears only once with a weight
  - Cluster the chunk using LOACLSEARCH, each cluster center has the weight of number of points it has
- Apply LOCALSEARCH to the weighted centers we have retained from the chunks so far

Clustering a Whole Stream Or a Sliding Window?

- In many applications, very old data is considered less useful and relevant
  - In some applications, it is required to cluster a sliding window instead of the whole stream
  - Network management, telecommunication, financial services, …
- How to "forget" stale data?
  - Aging – data items are associated with weights decaying over time
  - Sliding window – only consider the last n items

General Idea

- Divide the stream into buckets
  - Maintain statistics (e.g., # elements, mean, and variance) or local clustering in the buckets
- Derive global variance and clustering from locals
- Details in [Babcock et al., 03]



Other Than One Pass?

- One pass clustering algorithms are scalable, but may not be capable on streams evolving considerably
- How to explore clusters over different portions of the stream?
- Instead of one pass, clustering can be done in two phases
  - Online component: periodically summarize statistics
  - Offline component: in depth analysis based on the statistics
- [Aggarwal et al., 03]

Summarizing Using Micro-Clusters

- Micro-clusters using temporal extension of cluster feature vector
- For n d-dimensional points, a micro-cluster is defined as a (2d+3) tuple ($CF2^x$, $CF1^x$, $CF2^t$, $CF1^t$, n)
  - $CF2^x$ and $CF1^x$ are vectors of d entries, recording the sum of the squares and sum of the data values in each dimension, respectively
  - $CF2^t$ and $CF1^t$ record the sum of the squares and the sum of the time stamps, respectively

Additivity of Micro-Clusters

- The cluster feature vector of a larger micro-cluster can be derived from the cluster feature vectors of the sub-micro-clusters
- A natural choice for data stream processing

Pyramidal Time Frame

- The micro-clusters are stored at snapshots in time following a pyramidal pattern
  - Different levels of granularity depending on the recency
  - Snapshots are classified into different orders from 1 to log(T), where T is the clock time elapsed since the beginning of the stream
- For any user-specified time window of h, at least one stored snapshot can be found within 2h units of the current time
  - An effective trade-off between the storage requirements and the ability to recall summary statistics from different time horizons

An Example

- For any user-specified time window, let t_c be the current time and t_s be the time of the last stored snapshot of any order just before the time t_c-h. Then $(t_c – t_s) \leq 2h$

Order of snapshots	Clock time (last 5 snapshots)
0	55 54 53 52 51
1	54 52 50 48 46
2	52 48 44 40 36
3	48 40 32 24 16
4	48 32 16
5	32

Summary – Clustering

- Clustering data stream with one scan and limited main memory
  - Clustering the whole stream
  - Clustering in a sliding window
- How to handle evolving data?
  - Online summarization and offline analysis
- Applications and extensions
  - Outlier detection, nearest neighbor search, reverse nearest neighbor queries, …

Online Mining Data Streams

- Synopsis maintenance
- Classification, regression and learning
- Stream data mining languages
- Frequent pattern mining
- Clustering
- Change and novelty detection

Difference Between Static and Streaming Data

- "If the data distribution is stable, mining a data stream is largely the same as mining a large data set, since statistically we can draw and mine a sufficient sample"
- What are the expectations of mining data streams?
  - Assumption: the data is evolving
  - Finding and understanding changes!
  - Maintaining an updated model

Challenges in Change Detection

- How to measure the changes?
- How to describe and visualize the changes?
- How to characterize different types of changes?
- How to conduct reasoning on changes?
  - Why do we see the change?
  - What will follow?
  - …
- A new problem inherent to data streams

Diagnosing Changes in Streams

- The distribution of data can be measured by kernel density estimation
- Velocity density estimation: the rate of change in data density at each spatial location
- Visualization by temporal/spatial velocity profiles
- [Aggarwal, 03]

Burst Detection

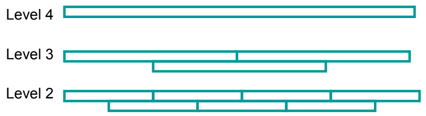
- Finding abnormal aggregates in data streams
- Monitor many sliding window sizes simultaneously and report those windows with aggregates significantly different from other periods
- Applications
  - Astronomy: Gamma ray burst
  - Network management: number of packages lost within a short period exceeds some threshold
  - Finance: stocks with unusually high trading volumes or with usually high price fluctuations within a short period

Three Types of Windows

- Landmark windows: the average stock price of IBM from Jan 1st, 2002 to today
- Sliding windows: the average stock price of IBM in the last 5 days
- Damped window: the weights of data decrease exponentially into the past
- Elastic window model – a generalization
  - Parameters: the range of the sliding window sizes [Zhu & Shasha, 03]

Shifted Wavelet Tree

- The adjacent windows of the same level are half overlapping



Building a Shifted Wavelet Tree

- Input: x[1..n], $n=2^a$ ($a=\log_2 n$)
- Output: shifted wavelet tree SWT[1..a][1..]
- Method:

```
– b ← x;
– for i=1 to a
–   // merge consecutive windows and form level i of SWT
–    for j=1 to size(b)-1 step 2
–       SWT[i][j]=b[j]+b[j+1];
–    for j=1 to size(SWT[i])/2
–       b[j]=SWT[i][2*j-1];
```

Detecting Burst

- If the sum at a high level window fails the threshold for the lowest level, no low level window will have a burst
- Can be extended to k-d shifted wavelet tree

Burst Detection in Text Streams

- Applications
  - Emails, published literature, …
- Modeling the stream using an infinite-state automaton
  - Busts: state transitions
  - Details in [Kleinberg, 02]

Novelty Detection

- Novelty: the newly arrive value x_t is substantially away from the prediction using x_1, …, x_{t-1}, or part of the sequence
- Use support vector regression to generate a model to predict x_t
  - Details in [Ma & Perkins, 03]
- Generalization
  - Online prediction and model maintenance
  - Novelty detection: outliers that the model fails

Summary – Change Detection

- An inherent problem for data streams
- Challenges
  - How to define meaningful changes?
    - Changes, burst, novelty, …
  - How to mine changes efficiently?
  - How to summarize/visualize changes?
- Good news: not much work yet!

Summary

- Mining data streams
  - Challenging problems
  - Exciting progress
- What is the next?
  - What have been done?
  - What have been started?
  - What should be done in the future?

What Have Been Done?

- Stream data management
  - Join, SQL query answering, …
- Synopsis monitoring
  - Basic aggregates, with or without sliding windows
- Critical tools: statistics + database techniques
- Achievements: accuracy + scalability

Challenges

- Standard for stream data management and processing
  - Schema?
  - Query language?
  - Benchmark?
- Real large applications of stream data management and processing systems
  - Killer applications?

What Have Been Just Started?

- Mining data streams
- Extensions of data mining tasks
  - Frequent pattern mining, classification, clustering, …
  - Extensions of conventional methods
- Stream-oriented mining
  - Change detection

Challenges

- Standard
  - Mining query language?
  - Benchmark?
- Large applications
- Stream mining systems – putting pieces together
  - Conducting multiple mining tasks on one stream?

What Should Be Done?

- Applications
  - If we have a system fast enough, do we still need stream processing/mining systems?
  - Killer applications are deadly wanted!
- Stream-oriented data mining
  - New types of patterns/knowledge from streams
- Interactive stream mining
  - Support interactive exploration

References – Synopsis and Classification (1)

- Arvind Arasu, Gurmeet Singh Manku. Approximate Counts and Quantiles over Sliding Windows. In the ACM Symposium on Principles of Database Systems (PODS) 2004.
- Brian Babcock, Chris Olston. Distributed Top-k Monitoring. In the ACM International Conference on Management of Data (SIGMOD) 2003.
- Brian Babcock, Mayur Datar, Rajeev Motwani, LiadanO O'Callaghan. Maintaining Variance and k-Medians over Data Stream Windows. In the ACM Symposium on Principles of Database Systems (PODS) 2003.
- Yunyue Zhu, Dennis Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In the International Conference on Very Large Data Bases (VLDB) 2002.
- Diane Lambert, Jose C. Pinheiro. Mining A Stream of Transactions for Customer Patterns. In the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) 2001.

References – Synopsis and Classification (2)

- Gurmeet Singh Manku, Sridhar Rajagopalan, Bruce G. Lindsay. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In the ACM International Conference on Management of Data (SIGMOD) 1998.
- Gurmeet Singh Manku, Sridhar Rajagopalan, Bruce G. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In the ACM International Conference on Management of Data (SIGMOD) 1999.
- Yan-Nei Law, Haixun Wang, Carlo Zaniolo. Query Languages and Data Models for Database Sequences and Data Streams. In the International Conference on Very Large Data Bases (VLDB) 2004.
- Haixun Wang, Carlo Zaniolo. ATLaS: A Native Extension of SQL for Data Mining. In the SIAM International Conference on Data Mining (SIAM DM) 2003.
- Wei Fan, Yi-an Huang, Haixun Wang, Philip S Yu. Active Mining of Data Streams. In the SIAM International Conference on Data Mining (SIAM DM) 2004.

References – Synopsis and Classification (3)

- Wei-Guang Teng, Ming-Syan Chen, Philip S. Yu. A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In the International Conference on Very Large Data Bases (VLDB) 2003.
- Haixun Wang, Wei Fan, Philip S. Yu, Jiawei Han. Mining Concept Drifting Data Streams using Ensemble Classifiers. In the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) 2003.
- Pedro Domingos, Geoff Hulten. Mining High Speed Data Streams. In the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) 2000.
- Geoff Hulten, Laurie Spencer, Pedro Domingos. Mining Time-Changing Data Streams. In the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) 2001.

References – Frequent Patterns (1)

- R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD'93, 207-216, Washington, D.C.
- R. Agrawal and R. Srikant. Fast algorithms for mining association rules. VLDB'94 487-499, Santiago, Chile.
- A. Arasu and G.S. Manku. Approximate counts and quantiles over sliding windows. PODS’04.
- T. Asai, H. Arimura, K. Abe, S. Kawasoe, and S. Arikawa. Online algorithms for mining semi-structured data stream. ICDM’02.
- J.H. Chang and W.S. Lee. Finding recent frequent itemsets adaptively over online data streams. In KDD’03.

References – Frequent Patterns (2)

- G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. VLDB’03.
- G. Cormode and S. Muthukrishnan. What’s hot and what’s not: Tracking most frequent items dynamically. PODS’03.
- C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities. in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), *Next Generation Data Mining*, AAAI/MIT, 2003**.**
- J. Han, J. Pei, and Y. Yin: “Mining frequent patterns without candidate generation”. In Proc. ACM-SIGMOD’2000, pp. 1-12, Dallas, TX, May 2000.

References – Frequent Patterns (3)

- R. Jin and G. Agrawal. An algorithm for in-core frequent itemset mining on streaming data. Submitted for publication
- C. Jin, W. Qian, C. Sha, J.X. Yu, A. Zhou. Dynamically maintaining frequent items over a data stream. CIKM’03.
- R.M. Karp and S. Shenker. A simple algorithm for finding frequent elements in streams and bags. In ACM TODS, Vol. 28, No. 1, Marge 2003, pages 51-55.
- G.S. Manku and R. Motwani. Approximate frequency counts over data streams. VLDB’02.
- W-G. Teng, M-S. Chen, and P.S. Yu. A regression-based temporal pattern mining scheme for data streams. VLDB’03

References – Clustering (1)

- R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. SIGMOD'98
- M. R. Anderberg. Cluster Analysis for Applications. Academic Press, 1973.
- M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure, SIGMOD’99.
- P. Arabie, L. J. Hubert, and G. De Soete. Clustering and Classification. World Scietific, 1996
- M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. KDD'96.
- M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. SSD'95.
- D. Fisher. Knowledge acquisition via incremental conceptual clustering. Machine Learning, 2:139-172, 1987.

References – Clustering (2)

- D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamic systems. In Proc. VLDB’98.
- S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. SIGMOD'98.
- A. K. Jain and R. C. Dubes. Algorithms for Clustering Data. Printice Hall, 1988.
- D. Jiang, J. Pei and A. Zhang. Interactive Exploration of Coherent Patterns in Time-Series Gene Expression Data. In KDD'03.
- L. Kaufman and P. J. Rousseeuw. Finding Groups in Data: an Introduction to Cluster Analysis. John Wiley & Sons, 1990.
- E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. VLDB’98.
- G. J. McLachlan and K.E. Bkasford. Mixture Models: Inference and Applications to Clustering. John Wiley and Sons, 1988.
- P. Michaud. Clustering techniques. Future Generation Computer systems, 13, 1997.

Reference – Clustering (3)

- R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. VLDB'94.
- J. Pei, X. Zhang, M. Cho, H. Wang and P.S. Yu. MaPle: A Fast Algorithm for Maximal Pattern-based Clustering. In ICDM'03.
- E. Schikuta. Grid clustering: An efficient hierarchical clustering method for very large data sets. Proc. 1996 Int. Conf. on Pattern Recognition, 101-105.

References – Clustering Streams

- C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A framework for clustering evolving data streams. In VLDB'03.
- B. Babcock, M. Datar, R. motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In PODS'03.
- F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse nearest neighbor aggregates over data streams. In VLDB'02.
- L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In ICDE'02.

References – Change Detection

- C.C. Aggarwal. A framework for diagnosing changes in evolving data streams. SIGMOD'03.
- J. Kleinberg. Bursty and hierarchical structure in streams. KDD'02.
- J. Ma and S. Perkins. Online novelty detection on temporal sequences. KDD'03.
- Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. KDD'03.