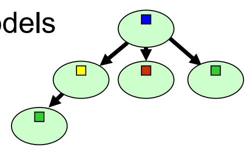
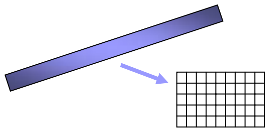
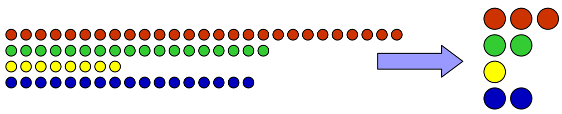
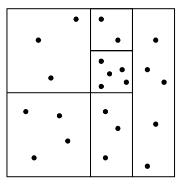
Fundamentals of Analyzing and Mining Data Streams

Graham Cormode
graham@research.att.com

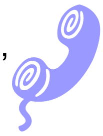
Outline

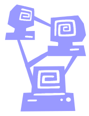
1. Streaming summaries, sketches and samples
   – Motivating examples, applications and models
   – Random sampling: reservoir and minwise
     ▪ Application: Estimating entropy
   – Sketches: Count-Min, AMS, FM
2. Stream Data Mining Algorithms
   – Association Rule Mining
   – Change Detection
   – Clustering

Data is Massive

- Data is growing faster than our ability to store or index it
- There are 3 Billion Telephone Calls in US each day, 30 Billion emails daily, 1 Billion SMS, IMs.
- Scientific data: NASA's observation satellites generate billions of readings each per day.
- IP Network Traffic: up to 1 Billion packets per hour per router. Each ISP has many (hundreds) routers!
- Whole genome sequences for many species now available: each megabytes to gigabytes in size



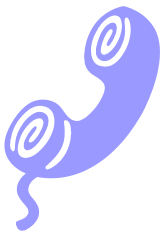


Massive Data Analysis

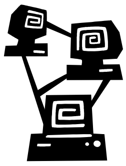
Must analyze this massive data:

- Scientific research (monitor environment, species)
- System management (spot faults, drops, failures)
- Customer research (association rules, new offers)
- For revenue protection (phone fraud, service abuse)

Else, why even measure this data?



Example: Network Data

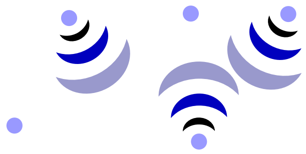


- Networks are sources of massive data: the metadata per hour per router is gigabytes
- Fundamental problem of data stream analysis:
  Too much information to store or transmit
- So process data as it arrives: one pass, small space: the *data stream* approach.
- Approximate answers to many questions are OK, if there are guarantees of result quality

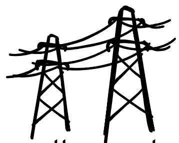
Streaming Data Questions

- Network managers ask questions requiring us to analyze and mine the data:
  - Find hosts with similar usage patterns (clusters)?
  - Which destinations or groups use most bandwidth?
  - Was there a change in traffic distribution overnight?
- Extra complexity comes from limited space and time
- Will introduce solutions for these and other problems

Other Streaming Applications



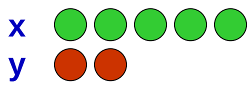
- Sensor networks
  - Monitor habitat and environmental parameters
  - Track many objects, intrusions, trend analysis…



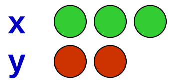
- Utility Companies
  - Monitor power grid, customer usage patterns etc.
  - Alerts and rapid response in case of problems

Data Stream Models

- We model data streams as sequences of simple tuples
- Complexity arises from massive length of streams
- Arrivals only streams:
  - Example: (x, 3), (y, 2), (x, 2) encodes the arrival of 3 copies of item x, 2 copies of y, then 2 copies of x.



  - Could represent eg. packets on a network; power usage
- Arrivals and departures:
  - Example: (x, 3), (y,2), (x, -2) encodes final state of (x, 1), (y, 2).



  - Can represent fluctuating quantities, or measure differences between two distributions

Approximation and Randomization

- Many things are hard to compute exactly over a stream
  - Is the count of all items the same in two different streams?
  - Requires linear space to compute exactly
- Approximation: find an answer correct within some factor
  - Find an answer that is within 10% of correct result
  - More generally, a $(1\pm \varepsilon)$ factor approximation
- Randomization: allow a small probability of failure
  - Answer is correct, except with probability 1 in 10,000
  - More generally, success probability $(1-\delta)$
- Approximation **and** Randomization: (ε, δ)-approximations

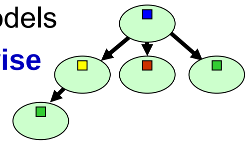
Structure

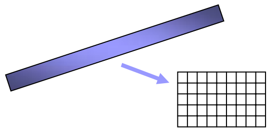
1. Stream summaries, sketches and samples
   - Answer simple distribution agnostic questions about stream
   - Describe properties of the distribution
   - E.g. general shape, item frequencies, frequency moments
2. Data Mining Algorithms
   - Extend existing mining problems to the stream domain
   - Go beyond simple properties to deeper structure
   - Build on sketch, sampling ideas

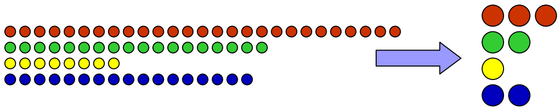
- Only a representative sample of each topic, many other problems, algorithms and techniques not covered

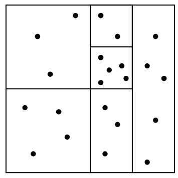
Outline

1. Streaming summaries, sketches and samples
   - Motivating examples, applications and models
   - **Random sampling: reservoir and minwise**
     - **Application: Estimating entropy**
   - Sketches: Count-Min, AMS, FM
2. Stream Data Mining Algorithms
   - Association Rule Mining
   - Change Detection
   - Clustering

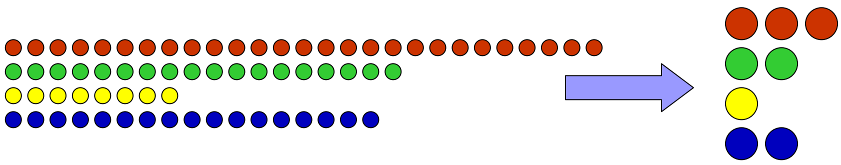






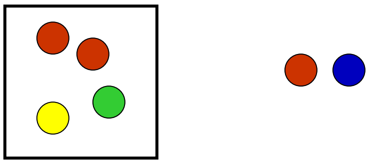


Sampling From a Data Stream



- Fundamental prob: sample m items uniformly from stream
  - Useful: approximate costly computation on small sample
- Challenge: don't know how long stream is
  - So when/how often to sample?
- Two solutions, apply to different situations:
  - Reservoir sampling (dates from 1980s?)
  - Min-wise sampling (dates from 1990s?)

Reservoir Sampling



- Sample first m items
- Choose to sample the i'th item (i>m) with probability m/i
- If sampled, randomly replace a previously sampled item

- Optimization: when i gets large, compute which item will be sampled next, skip over intervening items. [Vitter 85]

Reservoir Sampling - Analysis

- Analyze simple case: sample size m = 1
- Probability i'th item is the sample from stream length n:
  - Prob. i is sampled on arrival × prob. i survives to end

$$\frac{1}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \ldots \frac{\cancel{n-2}}{\cancel{n-1}} \times \frac{\cancel{n-1}}{n}$$

$$= 1/n$$

- Case for m > 1 is similar, easy to show uniform probability
- Drawbacks of reservoir sampling: hard to parallelize

Min-wise Sampling

- For each item, pick a random fraction between 0 and 1
- Store item(s) with the smallest random tag [Nath et al.'04]

- Each item has same chance of least tag, so uniform
- Can run on multiple streams separately, then merge

Application of Sampling: Entropy

- Given a long sequence of characters

  $S = \langle a_1, a_2, a_3 \ldots a_m \rangle$ each $a_j \in \{1 \ldots n\}$
- Let f_i = frequency of i in the sequence
- Compute the empirical entropy:

  $$H(S) = - \sum_i f_i/m \log f_i/m = - \sum_i p_i \log p_i$$

- Example: $S = \langle$ **a, b, a, b, c, a, d, a** $\rangle$
  - $p_a = 1/2$, $p_b = 1/4$, $p_c = 1/8$, $p_d = 1/8$
  - $H(S) = ½ + ¼ \times 2 + 1/8 \times 3 + 1/8 \times 3 = 7/4$
- Entropy promoted for anomaly detection in networks

Sampling Based Algorithm

- Simple estimator:
  - Randomly sample a position j in the stream
  - Count how many times a_j appears subsequently = r
  - Output $X = -(r \log r/m – (r-1) \log(r-1)/m)$
- Claim: Estimator is unbiased – $E[X] = H(S)$
  - Proof: prob of picking $j = 1/m$, sum telescopes correctly
- Variance is not too large – $Var[X] = O(\log^2 m)$
  - Can be proven by bounding $|X| \leq \log m$

17

Fundamentals of Analyzing and Mining Data Streams

Analysis of Basic Estimator

- A general technique in data streams:
  - Repeat in parallel an unbiased estimator with bounded variance, take average of estimates to improve result
  - Apply Chebyshev bounds to guarantee accuracy
  - Number of repetitions depends on ratio $Var[X]/E^2[X]$
  - For entropy, this means space $O(\log^2 m/H^2(S))$
- Problem for entropy: when $H(S)$ is very small?
  - Space needed for an accurate approx goes as $1/H^2$!

18

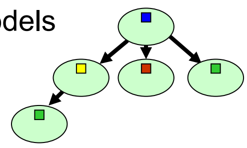
Fundamentals of Analyzing and Mining Data Streams

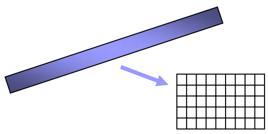
Outline of Improved Algorithm

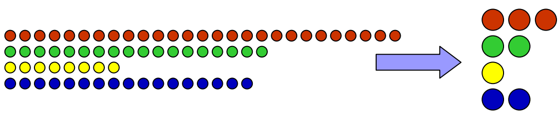
- Observation: only way to get $H(S) = o(1)$ is to have only one character with p_i close to 1
  - **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaa**
- If we can identify this character, and make an estimator on stream without this token, can estimate H(S)
- How to identify and remove all in one pass?
- Can do some clever tricks with ‘backup samples’ by adapting the min-wise sampling technique
- Full details and analysis in [Chakrabarti, C, McGregor 07]
  - Total space is $O(\varepsilon^{-2} \log m \log 1/\delta)$ for (ε,δ) approx

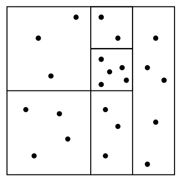
Outline

1. Streaming summaries, sketches and samples
   - Motivating examples, applications and models
   - Random sampling: reservoir and minwise
     - Application: Estimating entropy
   - **Sketches: Count-Min, AMS, FM**
2. Stream Data Mining Algorithms
   - Association Rule Mining
   - Change Detection
   - Clustering



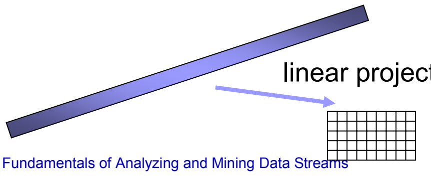




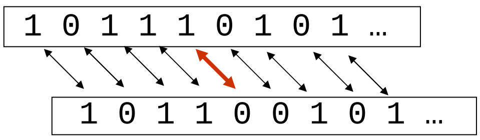


Sketches

- Not every problem can be solved with sampling
  - Example: counting how many distinct items in the stream
  - If a large fraction of items aren't sampled, don't know if they are all same or all different
- Other techniques take advantage that the algorithm can "see" all the data even if it can't "remember" it all
- (To me) a sketch is a linear transform of the input
  - Model stream as defining a vector, sketch is result of multiplying stream vector by an (implicit) matrix



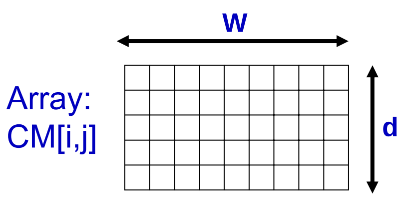
Trivial Example of a Sketch



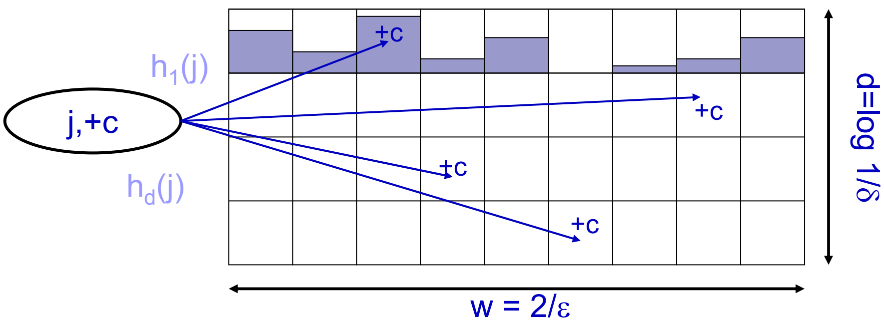
- Test if two (asynchronous) binary streams are equal
  $d_=(x,y) = 0$ iff $x=y$, 1 otherwise
- To test in small space: pick a random hash function h
- Test $h(x)=h(y)$: small chance of false positive, no chance of false negative.
- Compute $h(x)$, $h(y)$ incrementally as new bits arrive (Karp-Rabin: $h(x) = x_i 2^i \bmod p$)
  - Exercise: extend to real valued vectors in update model

Count-Min Sketch

- Simple sketch idea, can be used for as the basis of many different stream mining tasks.
- Model input stream as a vector x of dimension U
- Creates a small summary as an array of $w \times d$ in size
- Use d hash function to map vector entries to [1..w]
- Works on arrivals only and arrivals & departures streams



CM Sketch Structure



- Each entry in vector x is mapped to one bucket per row.
- Merge two sketches by entry-wise summation
- Estimate $x[j]$ by taking $\min_k CM[k,h_k(j)]$
  - Guarantees error less than $\varepsilon||x||_1$ in size $O(1/\varepsilon \log 1/\delta)$
  - Probability of more error is less than $1-\delta$

[C, Muthukrishnan '04]

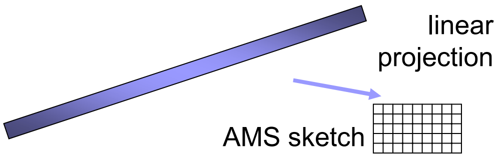
Approximation

Approximate $x'[j] = \min_k CM[k,h_k(j)]$

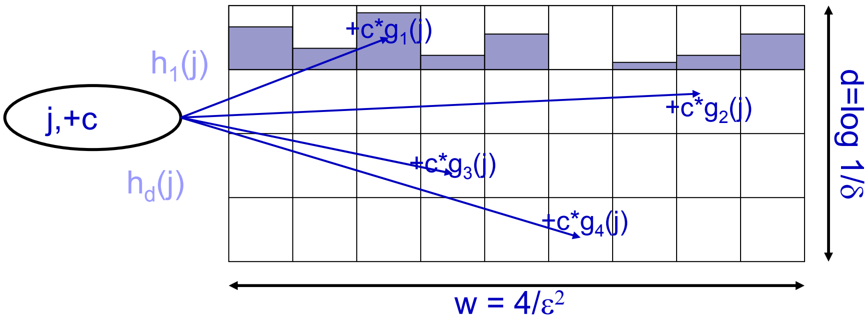
- Analysis: In k'th row, $CM[k,h_k(j)] = x[j] + X_{k,j}$
  - $X_{k,j} = \Sigma\ x[i] \mid h_k(i) = h_k(j)$
  - $E(X_{k,j}) = \Sigma\ x[k]^*Pr[h_k(i)=h_k(j)]$
    $\leq Pr[h_k(i)=h_k(k)]\ ^*\ \Sigma\ a[i]$
    $= \varepsilon\ ||x||_1/2$ by pairwise independence of h
  - $Pr[X_{k,j} \geq \varepsilon||x||_1] = Pr[X_{k,j} \geq 2E(X_{k,j})] \leq 1/2$ by Markov inequality
- So, $Pr[x'[j] \geq x[j] + \varepsilon\ ||x||_1] = Pr[\forall\ k.\ X_{k,j} > \varepsilon\ ||x||_1] \leq 1/2^{\log 1/\delta} = \delta$
- Final result: with certainty $x[j] \leq x'[j]$ and with probability at least $1-\delta$, $x'[j] < x[j] + \varepsilon\ ||x||_1$

L_2 distance

- AMS sketch (for Alon-Matias-Szegedy) proposed in 1996
  - Allows estimation of L_2 (Euclidean) distance between streaming vectors, $||\ x - y\ ||_2$
  - Used at the heart of many streaming and non-streaming mining applications: achieves dimensionality reduction
- Here, describe AMS sketch by generalizing CM sketch.
- Uses extra hash functions $g_1 \ldots g_{\log 1/\delta}\ \{1 \ldots U\} \rightarrow \{+1,-1\}$
- Now, given update $(j,+c)$, set $CM[k,h_k(i)] \mathrel{+}= c^*g_k(j)$



L_2 analysis



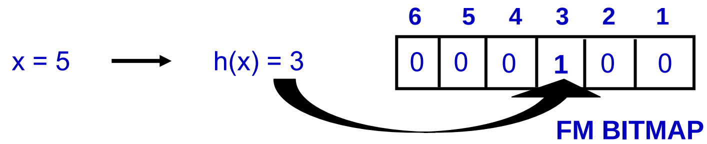
- Estimate $||x||_2^2 = \text{median}_k \sum_i CM[k,i]^2$
- Each row's result is $\sum_k g(i)^2 x_i^2 + \sum_{h(i)=h(j)} 2\, g(i)\, g(j)\, x_i\, x_j$
- But $g(i)^2 = -1^2 = +1^2 = 1$, and $\sum_i x_i^2 = ||x||_2^2$
- $g(i)g(j)$ has 1/2 chance of +1 or –1 : expectation is 0 …

L_2 accuracy

- Formally, one can show an (ε, δ) approximation
  - Expectation of each estimate is exactly $||x||_2^2$ and variance is bounded by ε^2 times expectation squared.
  - Using Chebyshev's inequality, show that probability that each estimate is within $\pm\, \varepsilon\, ||x||_2^2$ is constant
  - Take median of $\log (1/\delta)$ estimates reduces probability of failure to δ (using Chernoff bounds)
- Result: given sketches of size $O(1/\varepsilon^2 \log 1/\delta)$ can estimate $||x||_2^2$ so that result is in $(1\pm\varepsilon)||x||_2^2$ with probability at least $1-\delta$ ❑
  - Note: same analysis used many time in data streams
- In Practice: Can be very fast, very accurate!
  - Used in Sprint 'CMON' tool

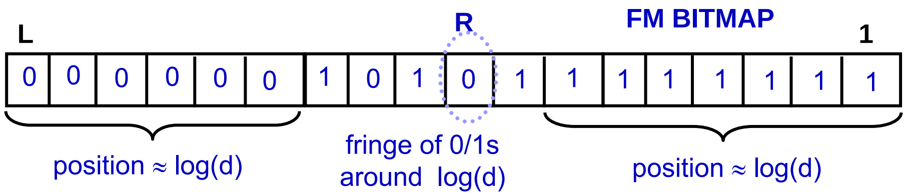
FM Sketch

- Estimates number of distinct inputs (**`count distinct`**)
- Uses hash function mapping input items to i with prob 2^{-i}
  - i.e. Pr[h(x) = 1] = ½, Pr[h(x) = 2] = ¼, Pr[h(x)=3] = 1/8 …
  - Easy to construct h() from a uniform hash function by counting trailing zeros
- Maintain FM Sketch = bitmap array of L = log U bits
  - Initialize bitmap to all 0s
  - For each incoming value x, set FM[h(x)] = 1



FM Analysis

- If d distinct values, expect d/2 map to FM[1], d/4 to FM[2]…



  - Let R = position of rightmost zero in FM, indicator of log(d)
  - Basic estimate $d = c2^R$ for scaling constant $c \approx 1.3$
  - Average many copies (different hash fns) improves accuracy
- With $O(1/\varepsilon^2 \log 1/\delta)$ copies, get (ε,δ) approximation
  - 10 copies gets ≈ 30% error, 100 copies < 10% error

Sketching and Sampling Summary

- Sampling and sketching ideas are at the heart of many stream mining algorithms
  - Entropy computation, association rule mining, clustering (still to come)
- A sample is a quite general representative of the data set; sketches tend to be specific to a particular purpose
  - FM sketch for count distinct, AMS sketch for L_2 estimation

Practicality

- Algorithms discussed here are quite simple and very fast
  - Sketches can easily process millions of updates per second on standard hardware
  - Limiting factor in practice is often I/O related
- Implemented in several practical systems:
  - AT&T's Gigascope system on live network streams
  - Sprint's CMON system on live streams
  - Google's log analysis
- Sample implementations available on the web
  - `http://www.cs.rutgers.edu/~muthu/massdal-code-index.html`
  - or web search for 'massdal'

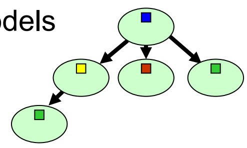
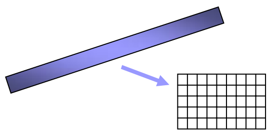
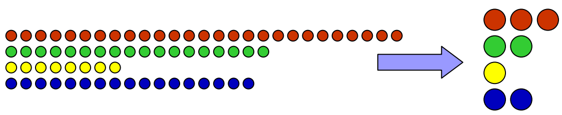
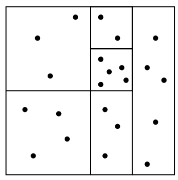
Other Streaming Algorithms

Many fundamentals have been studied, not covered here:

- Different streaming data types
  - Permutations, Graph Data, Geometric Data (Location Streams)
- Different streaming processing models
  - Sliding Windows, Exponential and other decay, Duplicate sensitivity, Random order streams, Skewed streams
- Different streaming scenarios
  - Distributed computations, sensor network computations

Outline

1. Streaming summaries, sketches and samples
   - Motivating examples, applications and models
   - Random sampling: reservoir and minwise
     - Application: Estimating entropy
   - Sketches: Count-Min, AMS, FM
2. Stream Data Mining Algorithms
   - **Association Rule Mining**
   - Change Detection
   - Clustering

Data Mining on Streams

- Pattern finding: finding common patterns or features
  - Association rule mining, Clustering, Histograms, Wavelet & Fourier Representations
- Data Quality Issues
  - Change Detection, Data Cleaning, Anomaly detection, Continuous Distributed Monitoring
- Learning and Predicting
  - Building Decision Trees, Regression, Supervised Learning
- Putting it all together: Systems Issues
  - Adaptive Load Shedding, Query Languages, Planning and Execution

Association Rule Mining

- Classic example: supermarket wants to discover correlations in buying patterns [Agrawal, Imielinski, Swami 93]
  - (bogus) result: diapers → beer
- **Input**: transactions t_1 = {eggs, milk, bread}, t_2 = {milk} ...t_n
- **Output**: rules of form {eggs, milk} → bread
  - Support: proportion of input containing {eggs, milk, bread}
  - Confidence: $\frac{\text{proportion containing \{eggs, milk, bread\}}}{\text{proportion containing \{eggs, milk\}}}$
- **Goal**: find rules with support, confidence above threshold

Frequent Itemsets

- Association Rule Mining (ARM) algorithms first find all frequent itemsets: subsets of items with support > ϕ
  - m-itemset: itemset with size m, i.e. |X| = m
- Use these frequent itemsets to generate the rules
- Start by finding all frequent 1-itemsets
  - Even this is a challenge in massive data streams

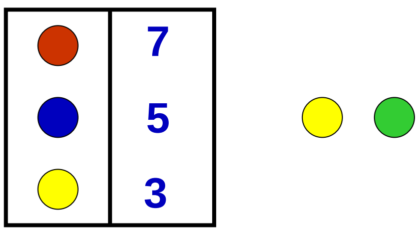
Heavy Hitters Problem

- The 'heavy hitters' are the frequent 1-itemsets
- Many, many streaming algorithms proposed:
  - Random sampling
  - Lossy Counting [Manku, Motwani 02]
  - Frequent [Misra, Gries 82, Karp et al 02, Demaine et al 02]
  - Count-Min, Count Sketches [Charikar, Chen, Farach-Colton 02]
  - And many more...
- 1-itemsets used to find, e.g heavy users in a network
  - The basis of general frequent itemset algorithms
  - A non-uniform kind of sampling

Space Saving Algorithm

- "SpaceSaving" algorithm [Metwally, Agrawal, El Abaddi 05] merges 'Lossy Counting' and 'Frequent' algorithms
  - Gets best space bound, very fast in practice
- Finds all items with count $\geq \phi n$, none with count $< (\phi - \varepsilon)n$
  - Error $0 < \varepsilon < 1$, e.g. $\varepsilon = 1/1000$
  - Equivalently, estimate each frequency with error $\pm\varepsilon n$
- Simple data structure:
  - Keep $k = 1/\varepsilon$ item names and counts, initially zero
  - Fill counters by counting first k distinct items exactly

SpaceSaving Algorithm



- On seeing new item:
  - If it has a counter, increment counter
  - If not, replace item with least count, increment count

SpaceSaving Analysis

- Smallest counter value, min, is at most εn
  - Counters sum to n by induction
  - $1/\varepsilon$ counters, so average is εn: smallest cannot be bigger
- True count of an uncounted item is between 0 and min
  - Proof by induction, true initially, min increases monotonically
  - Hence, the count of any item stored is off by at most εn
- Any item x whose true count $> \varepsilon n$ is stored
  - By contradiction: x was evicted in past, with count $\leq min_t$
  - Every count is an overestimate, using above observation
  - So est. count of $x > \varepsilon n \geq min \geq min_t$, and would not be evicted

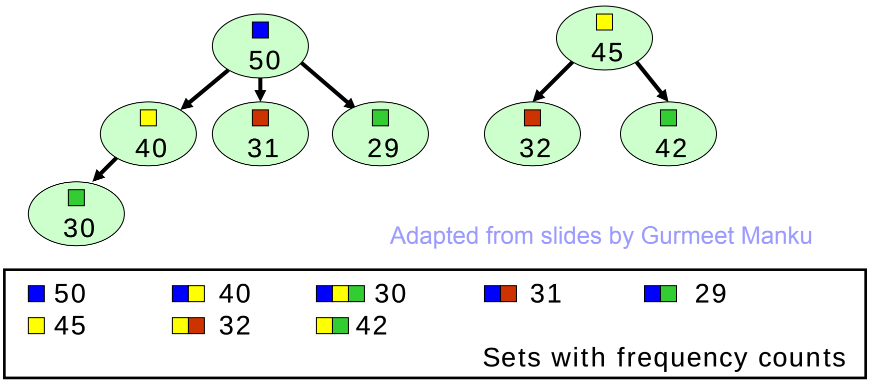
So: Find all items with count $> \varepsilon n$, error in counts $\leq \varepsilon n$

Extending to Frequent Itemsets

- Use similar "approximate counting" ideas for finding frequent itemsets [Manku, Motwani 02]
  - From each new transaction, generate all subsets
  - Track potentially frequent itemsets, prune away infrequent
  - Similar guarantees: error in count at most εn
- Efficiency concerns:
  - Buffer as many transactions as possible, generate subsets together so can prune early
  - Need compact representation of itemsets

Trie Representation of subsets

Compact representation of itemsets in lexicographic order.



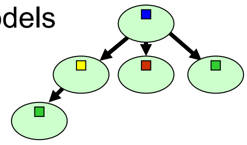
Use '*a priori*' rule: if a subset is infrequent, so are all of its supersets – so whole subtrees can be pruned

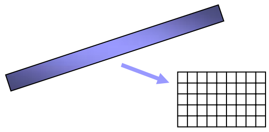
ARM Summary

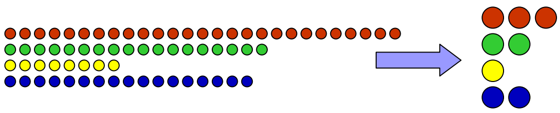
- [Manku, Motwani 02] gives details on when and how to prune
- Final Result: can monitor and extract association rules from frequent item sets with high accuracy
- Many extensions and variations to study:
  - Space required depends a lot on input, can be many potential frequent itemsets
  - How to mine when itemsets are observed over many sites (e.g. different routers; stores) and guarantee discovery?
  - Variant definitions: frequent subsequences, sequential patterns, maximal itemsets etc.
  - Sessions later in workshop…

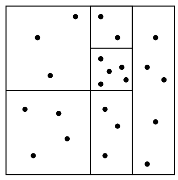
Outline

1. Streaming summaries, sketches and samples
   - Motivating examples, applications and models
   - Random sampling: reservoir and minwise
     - Application: Estimating entropy
   - Sketches: Count-Min, AMS, FM
2. Stream Data Mining Algorithms
   - Association Rule Mining
   - **Change Detection**
   - Clustering









Change Detection

Basic question: monitor a stream of events (network, power grid, sensors etc.), detect "changes" for:
- Anomaly detection – trigger alerts/alarms
- Data cleaning – detect errors in data feeds
- Data mining – indicate when to learn a new model

- What is "change"?
  - Change in behaviour of some subset of items
  - Change in patterns and rules detected
  - Change in underlying distribution of frequencies

Approaches to Change Detection

General idea: compare a reference distribution to a current window of events

- Item changes: individual items with big frequency change
  - Techniques based on sketches
- Fix a distribution (eg. mixture of gaussians), fit parameters
  - Not always clear which distribution to fix *a priori*
- Non-parametric change detection
  - Few parameters to set, but must specify when to call a change significant

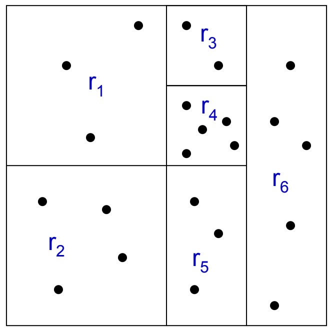
Non-parametric Change Detection

Technique due to [Dasu et al 06]

- Measure change using Kullback-Leibler divergence (KL)
  - Standard measure in statistics
  - Many desirable properties, generalizes t-test and χ^2
- KL divergence = $D(p||q) = \Sigma_x\, p(x) \log_2 p(x)/q(x)$
  - for probability distributions p, q
  - If p, q are distributions over high dimensional spaces, no intersection between samples – need to capture density

Space Division Approach

- Use a hierarchical space division (kd-tree) to define r regions r_i of (approximately equal) weight for the reference data
- Compute discrete probability p over the regions
- Apply same space division over a window of recent stream items to create q
- Compute KL divergence D(p||q)



49 Fundamentals of Analyzing and Mining Data Streams

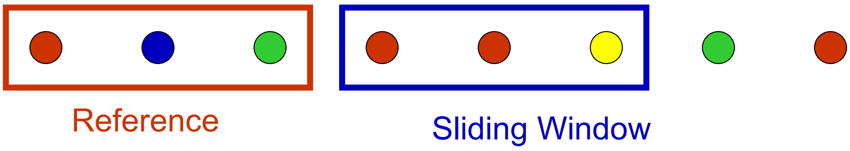
Bootstrapping

How to tell if the KL divergence is significant?

- Statistical bootstrapping approach: use the input data to compute a distribution of distances
- Pool reference and first sliding window data, randomly split into two pieces, measure KL divergence
- Repeat k times, find e.g. 0.99 quantile of divergences
- If KL distance between reference and window > 0.99 quantile of distances for several steps, declare “change”

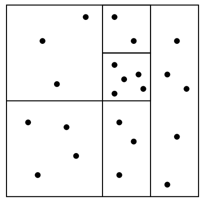
50 Fundamentals of Analyzing and Mining Data Streams

Streaming Computation



For each update:

- Slide window, update region counts
- Update KL divergence between reference p and window q, size w
- Test for significance



Efficient Implementation

- Don't have to recompute KL divergence from scratch
  - Can write normalized KL divergence in terms of

$$\Sigma_i \left(p(r_i) + 1/(2w)\right) \log \frac{p(r_i) + 1/(2w)}{q(r_i) + 1/(2w)}$$

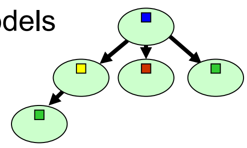
  - Only two terms change per update
- Total time cost per update:
  - Update two regional counts in kd-tree, O(log w)
  - Update KL divergence, in time O(1)
  - Compare to stored divergence cut off for significance test
  - Overall, O(log w)
- Space cost: store tree and counts, O(w)

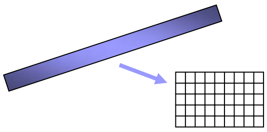
Change Detection Summary

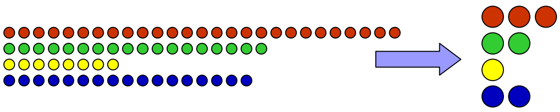
- Proposed technique is pretty efficient in practice
  - Competitive in accuracy with custom, application-aware change detection
  - Pretty fast – tens of microseconds per update
  - Produces simple description of change based on regions
- Extensions and open problems:
  - Other approaches – histogram or kernel based?
  - Better bootstrapping: quantile approach is only first order accurate…

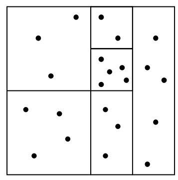
Outline

1. Streaming summaries, sketches and samples
   - Motivating examples, applications and models
   - Random sampling: reservoir and minwise
     - Application: Estimating entropy
   - Sketches: Count-Min, AMS, FM
2. Stream Data Mining Algorithms
   - Association Rule Mining
   - Change Detection
   - **Clustering**



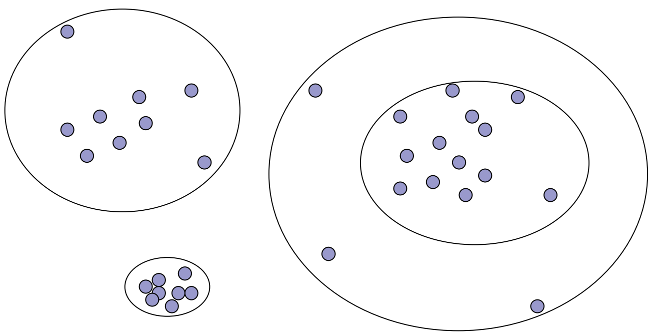




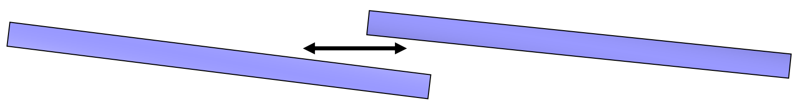


Clustering Data Streams

- We often talk informally about "clusters": 'cancer clusters', 'disease clusters' or 'crime clusters'
- Clustering has an intuitive appeal. We see a bunch of items... we want to discover the clusters...



Stream Clustering Large Points

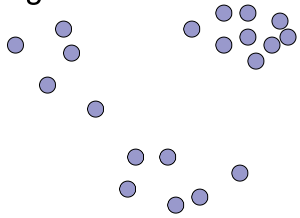


For clustering, need to compare the points. What happens when the points are very high dimensional?
- Eg. trying to compare whole genome sequences
- comparing yesterday's network traffic with today's
- clustering huge texts based on similarity

- If each point is size d, d very large， cost is very high (at least O(d). O(d^2) or worse for some metrics)
- We can do better: create a sketch for each point
- Do clustering using sketched approximate distances

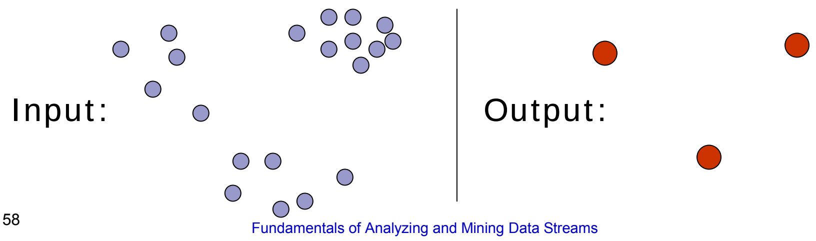
Stream Clustering Many Points

- What does it mean to cluster on the stream when there are too many points to store?
- We see a sequence of points one after the other, and we want to output a clustering for this observed data.
- Moreover, since this clustering changes with time, for each update we maintain some summary information, and at any time can output a clustering.
- Data stream restriction: data is assumed too large to store, so we do not keep all the input, or any constant fraction of it.



Clustering for the stream

- What should output of a stream clustering algorithm be?
- Classification of every input point?
  Too large to be useful?
  Might this change as more input points arrive?
  - Two points which are initially put in different clusters might end up in the same one
- An alternative is to output k cluster centers at end
  - any point can be classified using these centers.



Approximation for k-centers

k-center: minimize diameter (max dist) of each cluster.

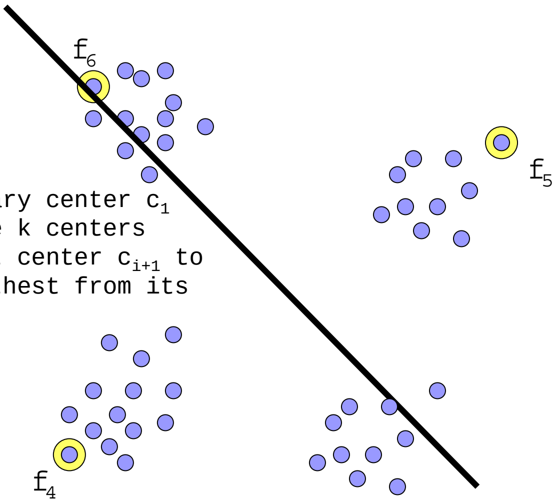
- Pick some point from the data as the first center.
  Repeat:
  - For each data point, compute distance d_{min} from its closest center
  - Find the data point that maximizes d_{min}
  - Add this point to the set of centers

  Until k centers are picked

- If we store the current best center for each point, then each pass requires O(1) time to update this for the new center, else O(k) to compare to k centers.
- So time cost is O(kn), but k passes [Gonzalez, 1985].

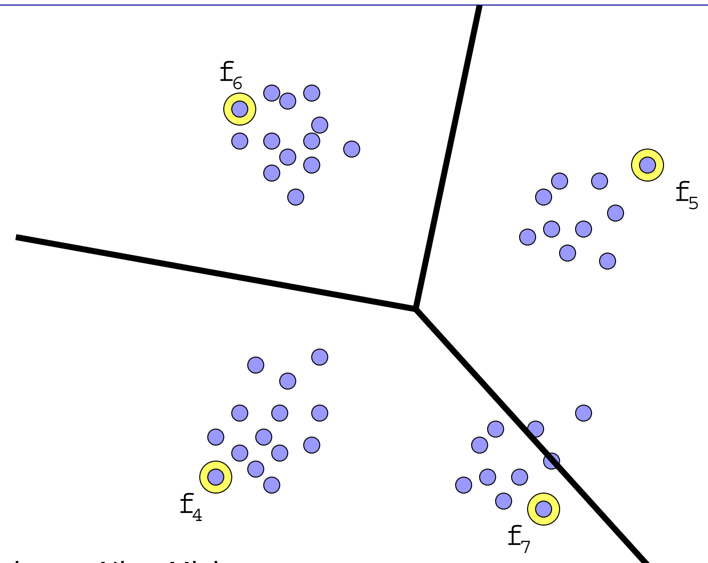
Gonzalez Clustering k=4

ALG:
Select an arbitrary center c_1
Repeat until have k centers
  Select the next center c_{i+1} to be the one farthest from its closest center



Slide due to Nina Mishra

Gonzalez Clustering k=4



Slide due to Nina Mishra

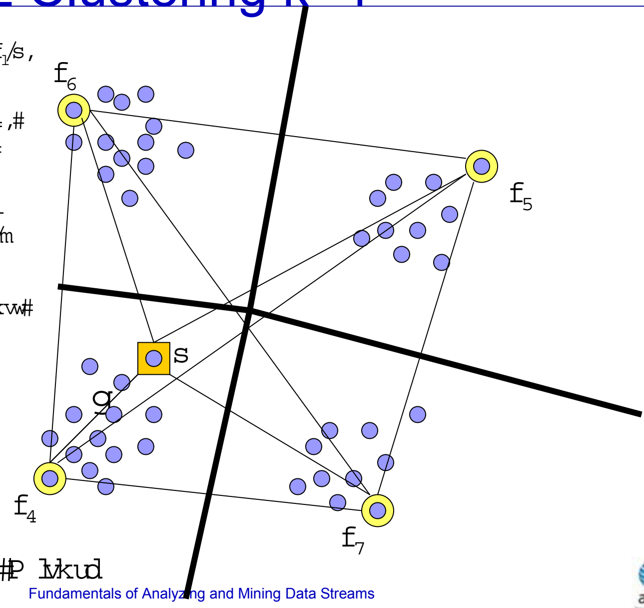
Gonzalez Clustering k=4

Ohw#g#@#p d{ glvw+f_l/s,

Fodlp =#Wkhuh#h{lvw#d#n.4,#
fdtxh#zkhuh#hdfk#sdlu#ri#
srlqw#lv#glvwdqfh#2g1
0glvw+f_l/s,#2g####iru#doo#l
0glvw+f_l/f_m,#2g###iru#doo#l/m

Qrwh=#Dq|#0fox vwhulqj#pxvw#
sxw#dw#ohdvw#wzr#ri#wkhvh#
n.4#srlqwv#lq#wkh#vdph#
fox vwhu1
0e|#sljhrqkroh

Wkxv#g#≤ 5R SW



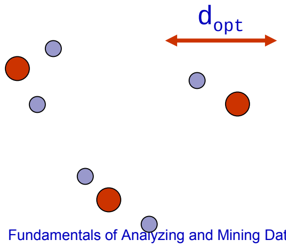
Vdgh#gxh#wr#Qlqd#Plvkud

Gonzalez is 2-approximation

- After picking k points to be centers, find next point that would be chosen. Let distance from closest center = d_{opt}
- We have k+1 points, every pair is separated by at least d_{opt}. Any clustering into k sets must put some pair in same set, so any k-clustering must have diameter d_{opt}
- For any two points allocated to the same center, they are both at distance at most d_{opt} from their closest center
- Their distance is at most $2d_{opt}$, using triangle inequality.
- Diameter of any clustering must be at least d_{opt}, and is at most $2d_{opt}$ – so we have a 2 approximation.
- Lower bound: NP-hard to *guarantee* better than 2

Gonzalez Restated

- Suppose we knew d_{opt} (from Gonzalez algorithm for k-centers) at the start
- Do the following procedure:
- `Select the first point as the first center`
- `For each point that arrives:`
  - `Compute` d_{min}, `the distance to the closest center`
  - `If` $d_{min} > d_{opt}$ `then set the new point to be a new center`



Analysis Restated

- d_{opt} is given, so we know that there are k+1 points separated by $\geq d_{opt}$ and d_{opt} is as large as possible
- So there are $\leq$ k points separated by $> d_{opt}$
- New algorithm outputs at most k centers: only include a center when its distance is $> d_{opt}$ from all others. If $>$ k centers output, then $>$ k points separated by $> d_{opt}$, contradicting optimality of d_{opt}.
- Every point not chosen as a center is $< d_{opt}$ from some center and so at most $2d_{opt}$ from any point allocated to the same center (triangle inequality)
- So: given d_{opt} we find a clustering where every point is at most twice this distance from its closest center

Guessing the optimal solution

- Hence, a 2-approximation – but, we aren't given d_{opt}
  - If we knew $d < d_{opt} < 2d$ then we could run the algorithm. If we find more than k centers, we guessed d_{opt} too low
  - So, in parallel, guess d_{opt} = 1, 2, 4, 8...
  - We reject everything $< d_{opt}$, so best guess is $< 2d_{opt}$: our output will be $< 2*2d_{opt}/d_{opt} = 4$ approx
- Need $\log_2 (d_{max}/d_{smallest})$ guesses, $d_{smallest}$ is minimum distance between any pair of points, as $d_{smallest} < d_{opt}$
- $O(k \log(d_{max} / d_{smallest}))$ may be high, can we reduce more?
- [Charikar et al 97]: doubling alg uses only O(k) space, gives 8-approximation. Subsequent work studied other settings

Clustering Summary

- General techniques: keeping small subset ("core-set") of input; guessing a key value; combining subproblems
- Many more complex solutions from computational geometry
- Variations and extensions:
  - When few data points but data points are high dimensional, use sketching techniques to represent
  - Different objectives: k-median, k-means, etc.
  - Better approximations, different guarantees (e.g. outputs 2k clusters, quality as good as that of best k-clustering)

Summary

- We have looked at
  - Sampling from streams and applications (entropy)
  - Sketch summaries for more advanced computations
  - Association Rule Mining to find interesting patterns
  - Change Detection for anomaly detection and alerts
  - Clustering to pick out significant clusters
- Many other variations to solve the problems discussed here, many other problems to study on data streams
  - See more over the course of this workshop.
  - Other tutorials and surveys: [Muthukrishnan '05] [Garofalakis, Gehrke, Rastogi '02]

References

[Agrawal, Imielinski, Swami '93] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. Proceedings of the ACM SIGMOD Conference on Management of Data, 1993.

[Alon, Matias, Szegedy '96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In Proceedings of the ACM Symposium on Theory of Computing, pages 20–29, 1996.

[Chakrabarti, Cormode, McGregor '07] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 2007.

[Charikar, Chen, Farach-Colton '02] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In Procedings of the International Colloquium on Automata, Languages and Programming (ICALP), 2002.

[Cormode, Muthukrishnan '04] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. Journal of Algorithms, 55(1):58–75, 2005.

References

[[Dasu et al '06] T. Dasu, S. Krishnan, S. Venkatasubramanian, K. Yi. An Information Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams. Proceedings of the 38th Symposium on the Interface of Statistics, Computing Science, and Applications (Interface), 2006.

[Demaine et al '03] E. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In Proceedings of the 10th Annual European Symposium on Algorithms, volume 2461 of Lecture Notes in Computer Science, pages 348–360, 2002.

[Garofalakis, Gehrke, Rastogi '02] M. Garofalakis and J. Gehrke and R. Rastogi. Querying and Mining Data Streams: You Only Get One Look. ACM SIGMOD Conference on Management of Data, 2002

[Gonzalez '85] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38(2-3):293–306, 1985.

[Karp, Papadimitriou, Shenker '03] R. Karp, C. Papadimitriou, and S. Shenker. A simple algorithm for finding frequent elements in sets and bags. ACM Transactions on Database Systems, 2003.

References

[Manku, Motwani '02] G.S. Manku and R. Motwani. Approximate frequency counts over data streams. In Proceedings of International Conference on Very Large Data Bases, pages 346–357, 2002.

[Metwally, Agrawal, El Abbadi '05] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In Proceedings of ICDT, 2005.

[Misra, Gries '82] J. Misra and D. Gries. Finding repeated elements. Science of Computer Programming, 2:143–152, 1982.

[Muthukrishnan '05] S. Muthukrishnan. Data Streams: Algorithms and Applications. Now Publishers, 2005.

[Nath et al.'04] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggrgation in sensor networks. In ACM SenSys, 2004.

[Vitter '85] J. S. Vitter. Random Sampling with a Reservoir, ACM Transactions on Mathematical Software, 11(1), March 1985, 37-57.