

Fundamentals of Analyzing and Mining Data Streams

Graham Cormode

AT&T Labs–Research, 180 Park Avenue, Florham Park, NJ 07932, USA

Abstract. Many scenarios, such as network analysis, utility monitoring, and financial applications, generate massive streams of data. These streams consist of millions or billions of simple updates every hour, and must be processed to extract the information described in tiny pieces. This survey provides an introduction to the problems of *data stream monitoring*, and some of the techniques that have been developed over recent years to help mine the data while avoiding drowning in these massive flows of information. In particular, this tutorial introduces the fundamental techniques used to create compact summaries of data streams: sampling, sketching, and other synopsis techniques. It describes how to extract features such as clusters and association rules. Lastly, we see methods to detect when and how the process generating the stream is evolving, indicating some important change has occurred.

Keywords: data streams, sampling, sketches, association rules, clustering, change detection.

1 Introduction

In recent years there has been growing interest in the study and analysis of *data streams*: flows of data that are so large that it is usually impractical to store them completely. Instead, they must be analyzed as they are produced, and high quality results guaranteed, no matter what outcomes are observed as the stream progresses. This tutorial surveys some of the key ideas and techniques that have been developed to analyze and mine such massive data streams. See [13] for a longer survey from an algorithmic perspective.

Motivation for studying data streams comes from a variety of areas: scientific data generation, from satellite observation to experiments on subatomic particles can generate terabytes of data in short amounts of time; sensor networks may have many hundreds or even thousands of nodes, each taking readings at a high rate; and communications networks generate huge quantities of *meta-data* about the traffic passing across them. In all cases, this information must be processed and analyzed for a variety of reasons: to monitor a system, analyze an experiment, or to ensure that a service is running correctly. However, given the massive size of the input, it is typically not feasible to store it all for convenient access. Instead, we must operate with resources much smaller than the size of the input (“sublinear”), and still guarantee a good quality answer for particular computations over the data.

From these disparate settings we can abstract a general framework within which to study them: the streaming model. In fact, there are several variations of this model, depending on what form the input may take and how an algorithm must respond.

Models: Arrivals only, or Arrivals and Departures. The basic model of data streams is an arrivals-only one. Here, the stream consists of a quantity of tuples, or items, which describe the input. Typically each tuple is a simple, small object, which might indicate, for example, the identity of a particular object of interest, and a weight or value associated with this arrival. In a network, the observation of a packet could be interpreted as a tuple indicating the intended destination of the packet, and the size of the packet payload in bytes. For another application, the same packet could be interpreted as a tuple whose identity is the concatenation of the source and destination of the packet, with a weight of 1, indicating that it is a single packet. Typically, we can interpret these streams as defining massive implicit vectors, indexed by item names, and whose entries are (usually) the sum of the associated counts (although many other interpretations may be possible). A richer model allows departures: in addition to positive updates to entries in this implicit vector, they may be negative. This captures more general situations in which earlier updates might be revoked, or observations for which negative values are feasible. In either case, the assumption is that each tuple in the input stream must be processed as it is seen, and cannot be revisited later unless it is stored explicitly by the stream algorithm within its limited internal memory.

Randomization and Approximation. Within these models, many natural and fundamental questions can be shown to require space linear in the input to answer exactly. For example, to test whether two separate

streams are the same (i.e. they encode the same number of occurrences of each item) requires us to store space linear in the number of distinct items, which could be immense. To be able to make progress, we typically allow *approximation*: returning an answer that is correct within some small fraction, ϵ of error; and *randomization*: allowing our algorithms to make random choices and to fail with some small probability δ . Algorithms which use both randomization and approximation we refer to as (ϵ, δ) approximations.

Update time, query time and space usage. To evaluate algorithms that operate on streams, we typically look at their behavior with respect to three additional features:

- *Update time*: the time to process each stream update.
- *Query time*: the time to use the information stored to answer the question of interest.
- *Space Usage*: the amount of memory used by the algorithm to keep information.

Typically, these three are measured in terms of parameters of the stream: the number of tuples, n and the number of different items m ; and the parameters ϵ and δ . To be an effective streaming algorithm these measures, particularly the space used, should be sublinear in m and n , and ideally poly-logarithmic (i.e. $O((\log m \log n)^c)$ for some constant c).

2 Streaming, sketches and summaries

In this section we outline two fundamental approaches to coping with streaming data: drawing a representative sample, and creating a compact “sketch” of the stream.

2.1 Random Sampling: reservoir and minwise

Many mining algorithms can be applied if only we can draw a representative sample of the data from the stream. The question is, how to ensure such a sample is drawn uniformly, given that the stream is continuously growing? For example, if we want to draw a sample of 100 items and the stream has length only 1000, then we want to sample roughly one in ten items. But if a further million items arrive, we must ensure that the probability of any item being sampled is more like one in a million. If we retain the same 100 items, then this is very skewed to the prefix of the stream, which is unlikely to be representative.

Several solutions are possible to ensure that we continuously maintain a uniform sample from the stream. The idea of reservoir sampling dates back to the eighties and before [15]. It is easiest to describe if we wish to draw a sample of size 1. Here, we initialize the sample with the first item from the stream. We replace the current sample with the i th item in the stream (when it is seen) by throwing some random bits to simulate a coin whose probability of landing “heads” is $1/i$, and keeping the i th item as the sampled item if we observe heads. It is a simple exercise to prove that, after seeing n items, the probability that any of those is retained as the sample is precisely $1/n$. One can generalize this technique to draw a sample of k items, either with replacement (by performing k independent repetitions of the above algorithm) or without replacement (by picking k items and replacing one with the i th with probability $1/i$).

A drawback of this approach is it does not easily generalize when we have many, distributed streams, and wish to sample uniformly from their union. For example, consider trying to sample from a stream formed by network traffic crossing the Atlantic and Pacific oceans. It is not feasible to operate jointly on both streams. Instead we use an alternative sampling algorithm, which we refer to as “min-wise sampling” (by analogy with an alternate technique known as min-wise hashing [3]). For each item in the stream we pick a random label as a real number in the range 0 to 1. We retain the item with the smallest random label seen so far. It is straightforward to observe that each item has an equal chance of getting the smallest tag, due to the symmetry of the procedure, and therefore it picks uniformly from the stream. Moreover, we can run the same algorithm across distributed streams and merge the results to get an item picked uniformly from the (disjoint) union of the streams, by picking the retained item with the smallest label.

Application: Estimating Entropy. The empirical entropy of a sequence of characters is computed by finding the number of occurrences, f_i , for each character i and computing $H = -\sum_{i=1}^m \frac{f_i}{n} \log_2 \frac{f_i}{n}$. This entropy is often used in network monitoring applications to detect anomalies. When the number of possible

items m is very large, we need a different approach to approximate the entropy. We build an estimator for entropy as follows based on sampling a position j in the stream (using the above min-wise sampling), and counting the number of subsequent occurrences in the stream of the character at position j as r . We can build an unbiased estimate of H as $r \log \frac{m}{r} - (r-1) \log \frac{m}{r-1}$. This estimate is not very reliable; it can be improved by taking the average of many repetitions using different random samples. This can be shown to give an $(\frac{\epsilon}{H^2}, \frac{1}{4})$ estimator; by taking the median of $O(\log 1/\delta)$ repetitions we form an $(\frac{\epsilon}{H^2}, \delta)$ estimator. This works well when H is large, but H can be very small, which results in a less reliable estimator. Further modifications of this technique can be used to generate an (ϵ, δ) estimator; see [4] for details.

2.2 Sketches for Estimation

Many data stream problems cannot be solved with just a sample. Instead, we can make use of data structures which, in effect, include a contribution from the entire input, rather than just the items picked in the sample. For example, consider trying to count the number of distinct objects in a stream. It is easy to see that unless almost all items are included in the sample, then we cannot tell whether they are the same or distinct. Since a streaming algorithm gets to see each item in turn, it can do better, as we shall see later. We refer to a “sketch” as a compact data structure which summarizes the stream for certain types of query. Typically it is a linear transformation of the stream: we can imagine the stream as defining a vector, and the algorithm computes the product of a matrix with this vector (to be effective, the matrix must have a very small representation, e.g. being defined implicitly by hash functions). We highlight three popular sketch algorithms:

Count-Min Sketch. The count-min sketch [6] is an array of counters of size $\frac{2}{\epsilon} \times \log \frac{1}{\delta}$, and $\log \frac{1}{\delta}$ hash functions. Each update is mapped to $\log \frac{1}{\delta}$ counters, one in each row, which are incremented to reflect the update. From this data structure, one can estimate the frequency f_i of any item, with error at most ϵn with probability at least $1 - \delta$, in space $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$.

Flajolet-Martin Sketch. The Flajolet-Martin sketch [10] is a bitmap of length approximately $\log m$. Each item is mapped by a hash function into an entry of the bitmap: with probability $\frac{1}{2}$ it maps into entry 1, $\frac{1}{4}$ to entry 2, $\frac{1}{8}$ to entry 3 and so on. For each item in the stream, we map to its bit under the hash function, and set the bit to 1. The position of the least significant 0 in the bitmap indicates the logarithm of the number of distinct items seen, D ; taking repetitions with randomly chosen hash functions improves the accuracy. Space $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ is sufficient for an (ϵ, δ) approximation of D .

AMS Sketches. The Alon-Matias-Szegedy sketch [2] can be described in terms of the Count-Min sketch. Now, when we go to update a counter, we multiply the value of the update by a hash function g on the item being updated: half the items are mapped to $+1$ by this hash function, and half to -1 . Taking the sum of the squares of all counters in each row gives a high-quality estimate for $F_2 = \sum_{i=1}^m f_i^2$, the sum of the squares of the frequency counts. This computation, or variations thereof, is at the heart of many data stream analyses. An (ϵ, δ) approximation for F_2 can be formed in space $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$.

A common feature of these sketch algorithms is that they rely on hash functions on item identifiers, which are relatively easy to implement and fast to compute. Indeed, many practical streaming data management systems implement such sketches, such as Sprint’s CMON system [14] and AT&T’s Gigascope [8], both of which operate on network data streams at gigabit speeds. Implementations of sketches can be found on the web, including <http://www.cs.rutgers.edu/~muthu/massdal-code-index.html>.

3 Stream Data Mining Algorithms

Building on ideas of sampling and sketching, we can design algorithms for specific analysis and data mining tasks. We discuss three popular problems: association rule mining, change detection, and clustering.

3.1 Association Rule Mining

A classic problem in data mining is Association Rule Mining [1]. Given a large collection of transactions t_i , each of which is a subset of possible items, for example sets of items bought from a supermarket, the

goal is to find rules of the form $X \rightarrow y$. The support of the rule is the fraction of the input which contains all members of the rule, i.e. $|\{i | X \cup \{y\} \subseteq t_i\}| / |\{t_i\}|$. The confidence of the rule is the number of input transactions which contains all members of the rule divided by the number containing the conditions (left side), i.e. $|\{i | X \cup \{y\} \subseteq t_i\}| / |\{t_i | X \subseteq t_i\}|$. In general, one seeks to find all rules with support and confidence both exceeding specific thresholds. There are exponentially many possible rules, and so careful strategies are designed to search through them efficiently. Typically, the problem is reduced to one of finding all *frequent itemsets*: subsets of items with high support (above some threshold ϕ). From these itemsets, the association rules can be determined.

Clearly this problem is especially challenging when the input transactions are observed in a streaming fashion, and limited resources are available to process them. Indeed, even the question of finding the frequent 1-itemsets (sets of size 1) — a necessary precursor to solving the general problem — is a challenge when the set of possible items is large, and has attracted significant interest. Sketching techniques as outlined above can be applied, but here we describe deterministic (non-randomized) approaches. The SpaceSaving algorithm [12] shows that the problem can be $(\epsilon, 0)$ approximated using space $O(\frac{1}{\epsilon})$. It tracks a set of $k = 1/\epsilon$ items and associated counts. For each item, if it has an associated counter then the counter is incremented; else, the item replaces the item with the smallest count, and that count is incremented. It can be shown that this simple algorithm gives the desired accuracy, and can be implemented efficiently.

Given ways to find frequent items, they can be extended to frequent itemsets. The method outlined by Manku and Motwani [11] attempts to use the available space as fully as possible. For each new transaction, it generates all the subsets, and stores them in a compact trie-based structure. When the space is full, it uses a pruning algorithm based on frequent items algorithms to delete the least frequent itemsets, and track the error in the estimated counts of each item. This gives an efficient and somewhat scalable solution, although in general there is no convenient non-trivial worst case bound on the space required for a given accuracy. Many variations of the problem have been studied, based on finding itemsets which correspond to ordered subsequences, or sequential patterns (substrings) of the input transactions.

3.2 Change Detection

As we are monitoring a stream of values, a fundamental question is “has the distribution of values changed recently?”. We want to know if things have changed so that we can detect anomalies — some deviation from what is expected — and trigger an alert if it has. It can tell us if there has been some problem with a data feed which has caused the distribution to shift. If we have built some data mining algorithm based on a particular model, a change indicated that the model may no longer be valid and we need to rebuild. But what is a “change”? It can be the change in behaviour (frequency) of some subset of items, or a change in other patterns. Here, we take a definition where the underlying distribution (of frequencies) changes. We aim to do this non-parametrically: that is, without explicitly fixing a model that we expect the data to fit.

Dasu *et al.* propose a technique based on statistical bootstrapping to identify when a change has occurred [9]. They consider the case when the input consists of a series of points from a high dimensional space (value-based or categorical). Because we do not expect to see the exact same points many times, instead we use a space-partitioning algorithm over a “reference window” to define regions, and compute the relative frequencies within each region: a set of empirical probabilities $p(i)$ for the reference window and $q(i)$ for the sliding window. This is applied both to a fixed reference window, and a sliding window, both of size n points. To test for change, they compute the Kullback-Leibler distance (KL) as $D(p||q) = \sum_i p(i) \log_2 p(i)/q(i)$.

In order to test whether this distance is significant, they use a bootstrapping idea: compute distances based on randomly assigning points from the two windows to two sets, and computing the distance. A high quantile (e.g. the 99th percentile) of the distances is used as a boundary: if the measured KL distance exceeds this for several steps, we declare that a change has occurred. The whole procedure can be implemented efficiently in a streaming fashion by keeping appropriate data structures, and observing that as the sliding window advances, we do not have to recompute the KL distance from scratch, but rather can compute it incrementally from the previous value with only a few operations. This technique turns out to be quite efficient in practice, requiring only tens of microseconds per update. Many extensions and variations are possible, based on variant formulations, and the use of other change tests, kernel based methods etc.

3.3 Clustering

The notion of a cluster is a familiar one: we often talk of “cancer clusters”, or “crime clusters”, indicating a high local density of events. Formally, given a set of items, a good clustering places those items that are similar together in clusters, and ensures that the items in different clusters are different. It is natural to try to extend clustering to a stream, but what does it mean when the stream is so large we cannot store for each point which cluster it is allocated to? Typically, we seek a number of clusters, k , which is much smaller than the number of points, n to be clustered. After seeing the stream, the output is just the k clusters, from which the mapping of points to clusters is implicit (e.g. each point is mapped to its closest cluster).

We give a simple example of clustering the stream based on optimizing the k -center objective: attempting to minimize the diameter (the maximum distance between any two points in the same cluster). The algorithm arises by guessing the diameter of the clustering is some value d . The first point is allocated a cluster of its own. For each subsequent point in the stream, if it is far from any existing cluster, a new cluster containing the new point is created, else it is allocated to an existing cluster. If the guess of d was good, then no more than k clusters will be created. Moreover, if d was reasonably close to the true diameter, then the diameter of the stream clustering will be within a factor of 2 of the best possible cluster radius. By trying different guesses of d in parallel, and discarding any that generate more than k clusters, we can build a $(1 + \epsilon, 0)$ (i.e. deterministic) clustering algorithm [5, 7].

Other stream clustering algorithms get more complex. Some are based on the notion of “core-sets”: a small subset of the input such that solving the problem on the subset gives a good approximation to the solution on the full input. Yet more use a hierarchical approach: solving the problem exactly on a small subset of data that fits in memory, then merging such solutions to get an approximate solution to the full problem. Different techniques are needed to guarantee good results for other clustering objective functions, such as k -median, k -means and so on.

References

1. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, 1993.
2. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.
3. A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC*, 1998.
4. A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *SODA*, 2007.
5. M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *STOC*, 1997.
6. 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.
7. G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *ICDE*, 2007.
8. C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, 2003.
9. T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information theoretic approach to detecting changes in multi-dimensional data streams. In *Interface*, 2006.
10. P. Flajolet and G. N. Martin. Probabilistic counting algorithms for database applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.
11. G.S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.
12. A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, 2005.
13. S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, 2005.
14. K. To, T. Ye, and S. Bhattacharyya. CMON: A general purpose continuous IP backbone traffic analysis platform. Research Report RR04-ATL-110309, Sprint ATL, 2004.
15. J. S. Vitter. Random sampling with a reservoir. *ACM Trans. on Mathematical Software*, 11(1):37–57, March 1985.

Fundamentals of Analyzing and Mining Data Streams

Graham Cormode
graham@research.att.com

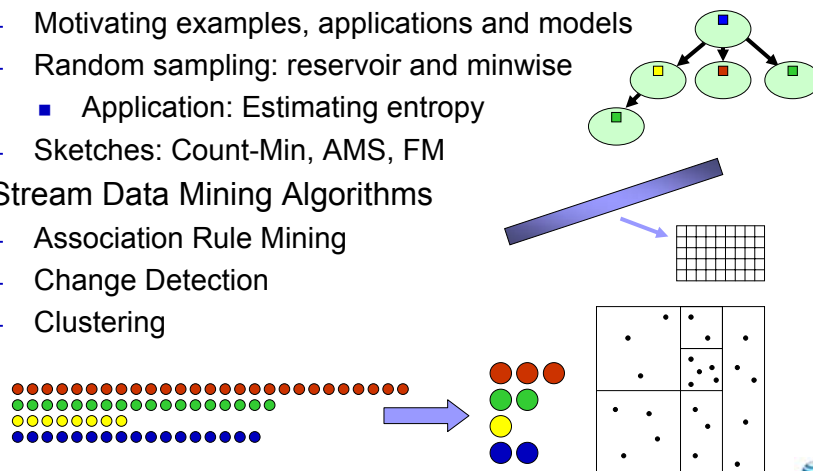
Outline

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



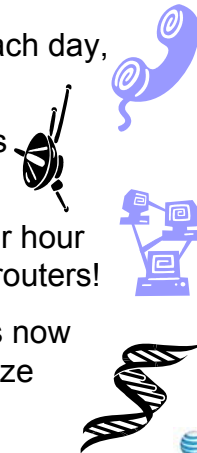
2

Fundamentals of Analyzing and Mining Data Streams



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



3

Fundamentals of Analyzing and Mining Data Streams



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?



4

Fundamentals of Analyzing and Mining Data Streams



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

5

Fundamentals of Analyzing and Mining Data Streams



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



6

Fundamentals of Analyzing and Mining Data Streams



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



7

Fundamentals of Analyzing and Mining Data Streams





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 x 
the arrival of 3 copies of item x , y 
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 x 
final state of $(x, 1), (y, 2)$. y 

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

8

Fundamentals of Analyzing and Mining Data Streams



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 $(1 \pm \epsilon)$ 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

9

Fundamentals of Analyzing and Mining Data Streams



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

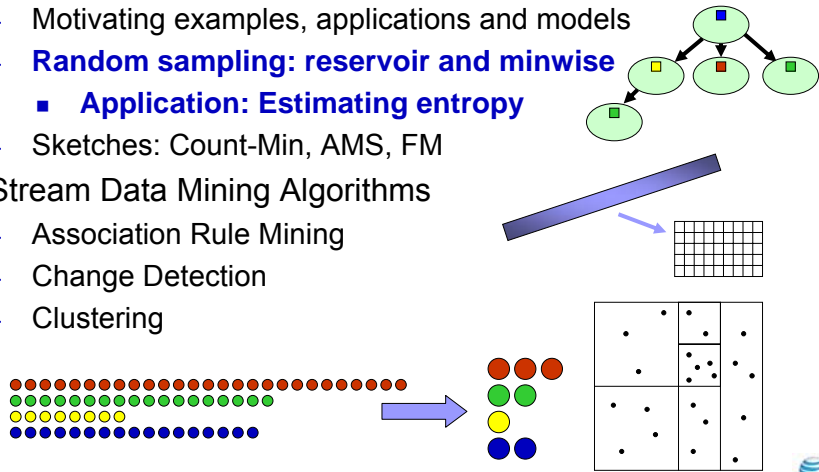
10

Fundamentals of Analyzing and Mining Data Streams



Outline

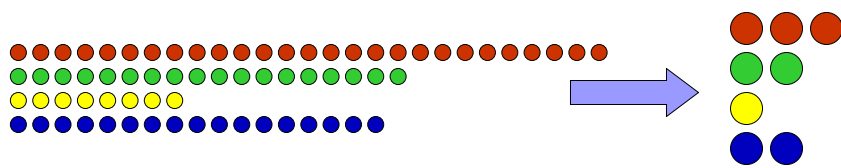
1. Streaming summaries, sketches and samples
 - Motivating examples, applications and models
 - **Random sampling: reservoir and minwise**
 - **Application: Estimating entropy**
 - Sketches: Count-Min, AMS, FM
2. Stream Data Mining Algorithms
 - Association Rule Mining
 - Change Detection
 - Clustering



11



Sampling From a Data Stream

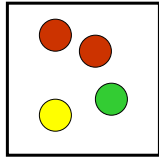


- Fundamental prob: sample m items uniformly from stream
 - Useful: approximate costly computation on small sample
- **Challenge:** don't know how long stream is
 - So when/how often to sample?
- Two solutions, apply to different situations:
 - Reservoir sampling (dates from 1980s?)
 - Min-wise sampling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

12



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]

13

Fundamentals of Analyzing and Mining Data Streams



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 \times prob. i survives to end

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

$$= 1/n$$

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

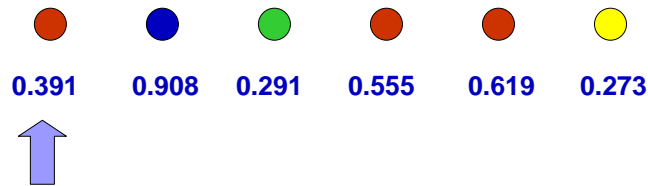
14

Fundamentals of Analyzing and Mining Data Streams



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

15

Fundamentals of Analyzing and Mining Data Streams



Application of Sampling: Entropy

- Given a long sequence of characters
 $S = \langle a_1, a_2, a_3 \dots a_m \rangle$ each $a_j \in \{1 \dots n\}$

- 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: $S = \langle a, b, a, b, c, a, d, a \rangle$

- $p_a = 1/2, p_b = 1/4, p_c = 1/8, p_d = 1/8$

- $H(S) = 1/2 + 1/4 \times 2 + 1/8 \times 3 + 1/8 \times 3 = 7/4$

- Entropy promoted for anomaly detection in networks

16

Fundamentals of Analyzing and Mining Data Streams



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 – $\text{Var}[X] = O(\log^2 m)$
 - 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 $\text{Var}[X]/E^2[X]$
 - 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
 - aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaa
- 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 $O(\epsilon^{-2} \log m \log 1/\delta)$ for (ϵ, δ) approx

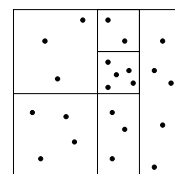
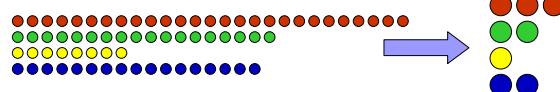
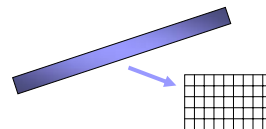
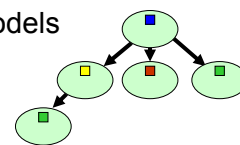
19

Fundamentals of Analyzing and Mining Data Streams



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



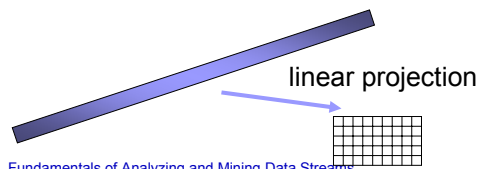
20

Fundamentals of Analyzing and Mining Data Streams



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

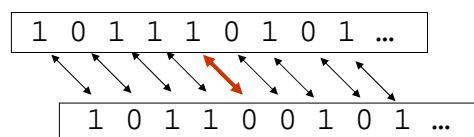


21

Fundamentals of Analyzing and Mining Data Streams



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: $h(x) = x_i 2^i \text{ mod } p$)
 - **Exercise:** extend to real valued vectors in update model

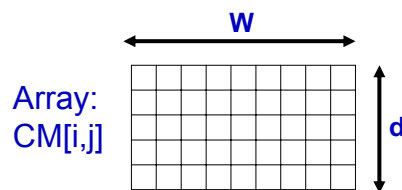
22

Fundamentals of Analyzing and Mining Data Streams



Count-Min Sketch

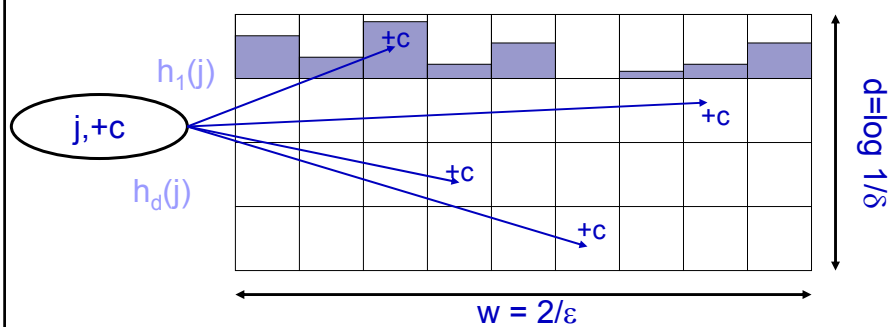
- Simple sketch idea, can be used for as the basis of many different stream mining tasks.
- Model input stream as a vector x of dimension U
- Creates a small summary as an array of $w \times d$ in size
- Use d hash function to map vector entries to $[1..w]$
- Works on arrivals only and arrivals & departures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



23



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 $\epsilon \|x\|_1$ in size $O(1/\epsilon \log 1/\delta)$
 - Probability of more error is less than $1-\delta$

[C, Muthukrishnan '04]

24



Approximation

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

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

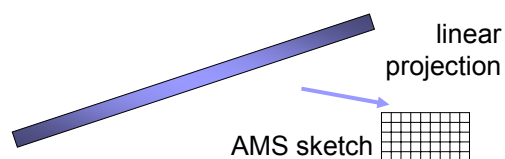
25

Fundamentals of Analyzing and Mining Data Streams



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 $g_1 \dots g_{\log 1/\delta} \{1 \dots U\} \rightarrow \{+1, -1\}$
- Now, given update $(j, +c)$, set $CM[k, h_k(i)] += c * g_k(j)$

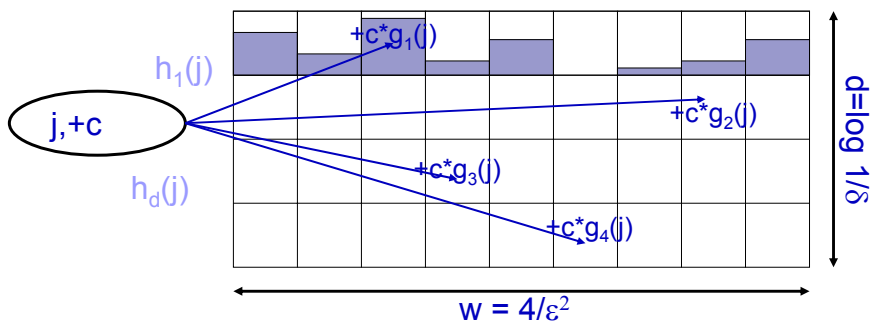


26

Fundamentals of Analyzing and Mining Data Streams



L₂ analysis



- Estimate $\|x\|_2^2 = \text{median}_k \sum_i \text{CM}[k, i]^2$
- 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 ...

27

Fundamentals of Analyzing and Mining Data Streams



L₂ 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 $\pm \epsilon \|x\|_2^2$ is constant
 - Take median of $\log(1/\delta)$ estimates reduces probability of failure to δ (using Chernoff bounds)
- **Result:** given sketches of size $O(1/\epsilon^2 \log 1/\delta)$ can estimate $\|x\|_2^2$ so that result is in $(1 \pm \epsilon)\|x\|_2^2$ 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

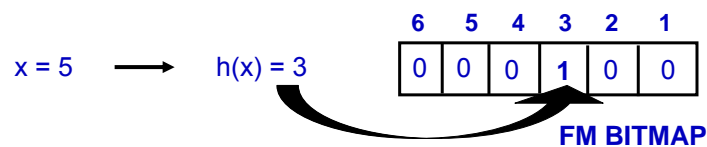
28

Fundamentals of Analyzing and Mining Data Streams



FM Sketch

- Estimates number of distinct inputs (**count distinct**)
- Uses hash function mapping input items to i with prob 2^{-i}
 - i.e. $\Pr[h(x) = 1] = 1/2$, $\Pr[h(x) = 2] = 1/4$, $\Pr[h(x)=3] = 1/8 \dots$
 - 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 $FM[h(x)] = 1$



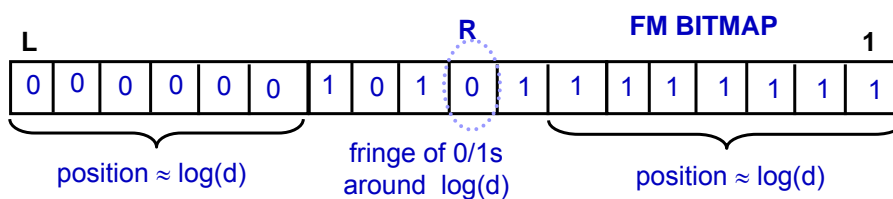
29

Fundamentals of Analyzing and Mining Data Streams



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 $O(1/\epsilon^2 \log 1/\delta)$ copies, get (ϵ, δ) approximation
 - 10 copies gets $\approx 30\%$ error, 100 copies $< 10\%$ error

30

Fundamentals of Analyzing and Mining Data Streams



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

31

Fundamentals of Analyzing and Mining Data Streams



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'

32

Fundamentals of Analyzing and Mining Data Streams



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

33

Fundamentals of Analyzing and Mining Data Streams



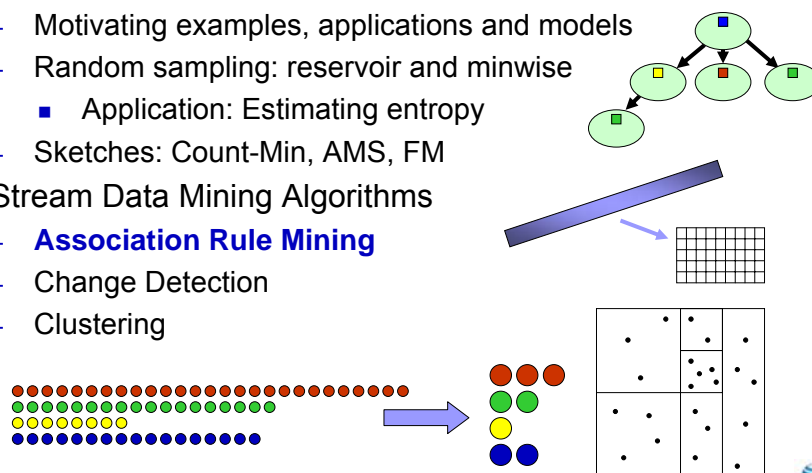
Outline

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



34

Fundamentals of Analyzing and Mining Data Streams



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

35

Fundamentals of Analyzing and Mining Data Streams



Association Rule Mining

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

36

Fundamentals of Analyzing and Mining Data Streams



Frequent Itemsets

- Association Rule Mining (ARM) algorithms first find all frequent itemsets: subsets of items with support $> \phi$
 - 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



37

Fundamentals of Analyzing and Mining 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

38

Fundamentals of Analyzing and Mining Data Streams



Space Saving Algorithm

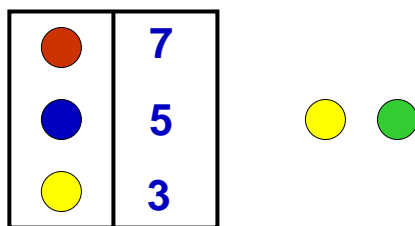
- “SpaceSaving” algorithm [Metwally, Agrawal, El Abaddi 05] merges ‘Lossy Counting’ and ‘Frequent’ algorithms
 - Gets best space bound, very fast in practice
- Finds all items with count $\geq \phi n$, none with count $< (\phi - \epsilon)n$
 - Error $0 < \epsilon < 1$, e.g. $\epsilon = 1/1000$
 - Equivalently,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

39

Fundamentals of Analyzing and Mining Data Streams



SpaceSaving Algorithm



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

40

Fundamentals of Analyzing and Mining Data Streams



SpaceSaving Analysis

- Smallest counter value, \min , is at most ϵn
 - Counters sum to n by induction
 - $1/\epsilon$ 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 $> \epsilon n$ 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 $x > \epsilon n \geq \min \geq \min_t$, and would not be evicted

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

41

Fundamentals of Analyzing and Mining Data Streams



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

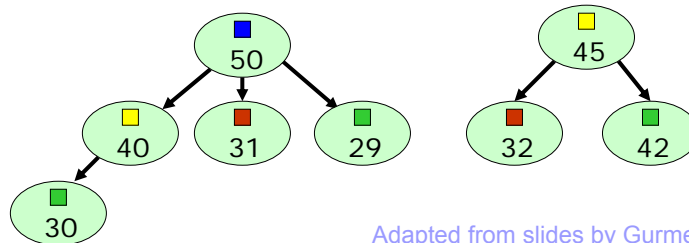
42

Fundamentals of Analyzing and Mining Data Streams

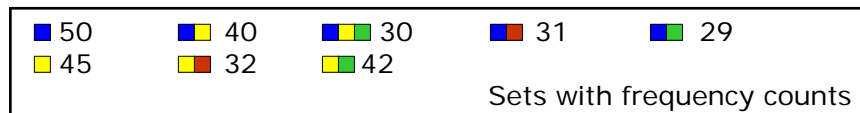


Trie Representation of subsets

Compact representation of itemsets in lexicographic order.



Adapted from slides by Gurmeet Manku



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

43

Fundamentals of Analyzing and Mining Data Streams



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...

44

Fundamentals of Analyzing and Mining Data Streams



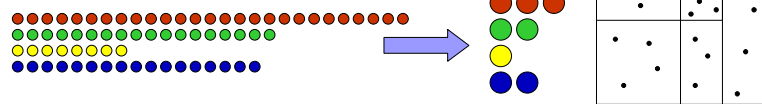
Outline

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



45

Fundamentals of Analyzing and Mining Data Streams



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

46

Fundamentals of Analyzing and Mining Data Streams



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

47

Fundamentals of Analyzing and Mining Data Streams



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 = $D(p||q) = \sum_x p(x) \log_2 p(x)/q(x)$
 - for probability distributions p, q
 - If p, q are distributions over high dimensional spaces, no intersection between samples – need to capture density

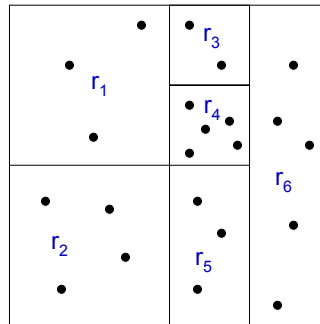
48

Fundamentals of Analyzing and Mining Data Streams



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?

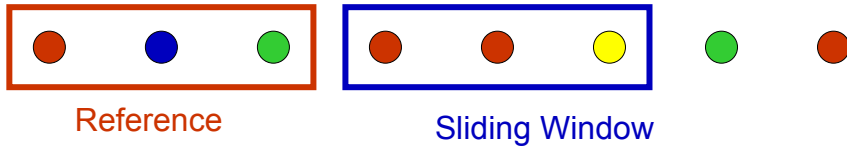
- Statistical bootstrapping approach: use the input data to compute a distribution of distances
- Pool reference and first sliding window data, randomly split into two pieces, measure KL divergence
- Repeat k times, find e.g. 0.99 quantile of divergences
- If KL distance between reference and window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

50

Fundamentals of Analyzing and Mining Data Streams

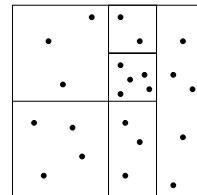


Streaming Computation



For each update:

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



51

Fundamentals of Analyzing and Mining Data Streams



Efficient Implementation

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

$$\sum_i (p(r_i) + 1/(2w)) \log \frac{p(r_i) + 1/(2w)}{q(r_i) + 1/(2w)}$$
 - 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, $O(w)$

52

Fundamentals of Analyzing and Mining Data Streams



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...

53

Fundamentals of Analyzing and Mining Data Streams



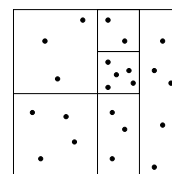
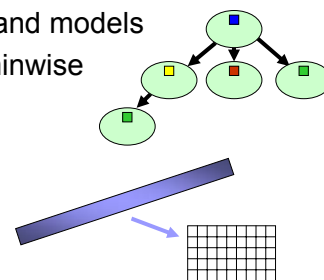
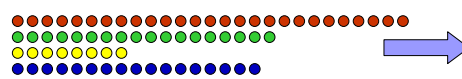
Outline

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**



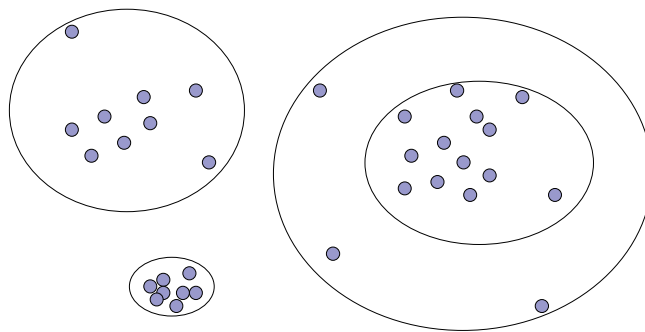
54

Fundamentals of Analyzing and Mining Data Streams



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...

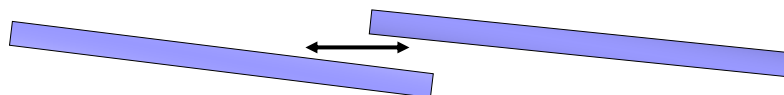


55

Fundamentals of Analyzing and Mining Data Streams



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

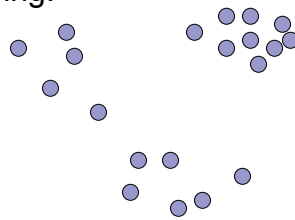
56

Fundamentals of Analyzing and Mining Data Streams



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.



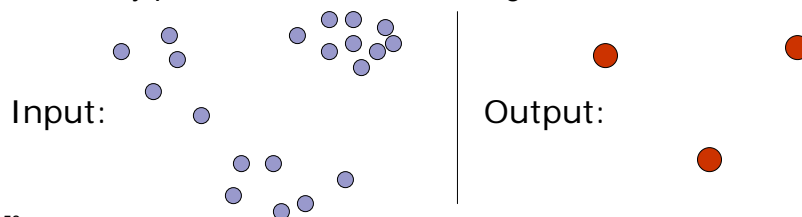
57

Fundamentals of Analyzing and Mining Data Streams



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.



58

Fundamentals of Analyzing and Mining Data Streams



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].

59

Fundamentals of Analyzing and Mining Data Streams



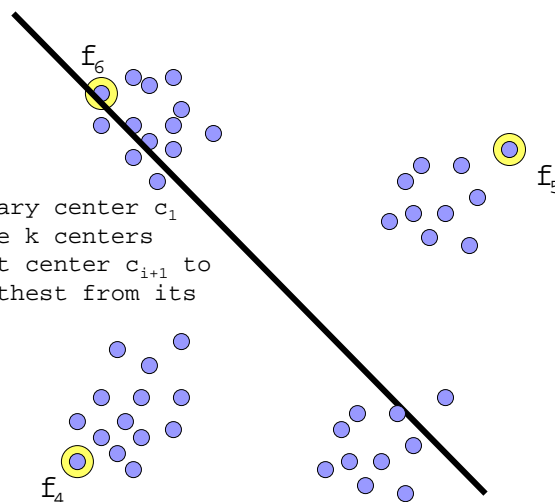
Gonzalez Clustering k=4

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

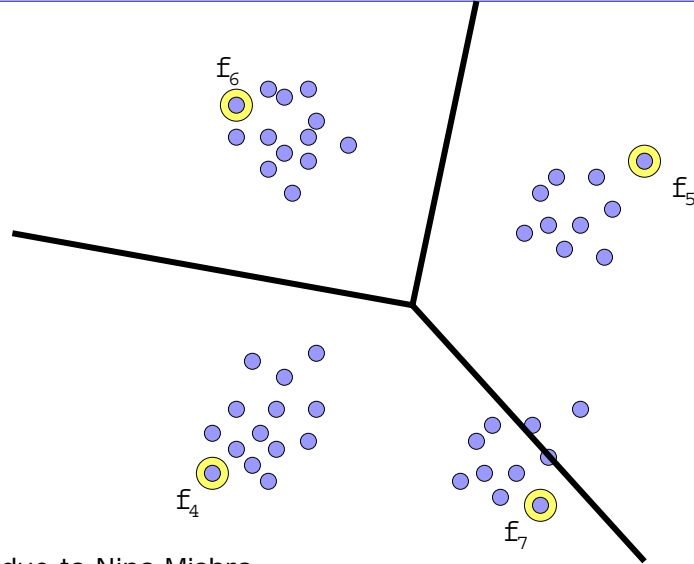


Slide due to Nina Mishra

Fundamentals of Analyzing and Mining Data Streams



Gonzalez Clustering k=4



Slide due to Nina Mishra

Fundamentals of Analyzing and Mining Data Streams



Gonzalez Clustering k=4

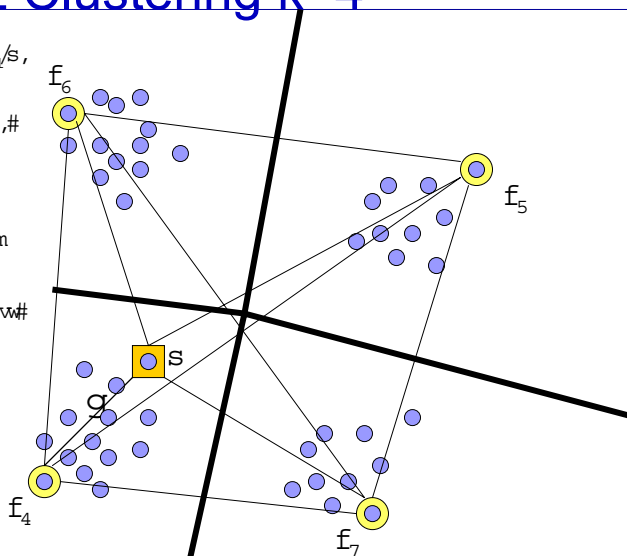
On the top left of the slide, there is a small, illegible text fragment.

For the number of clusters, $k=4$, the algorithm chooses the four initial centroids f_1, f_2, f_3, f_4 and iteratively updates them until convergence.

On the bottom left, there is a small diagram showing a point s and a cluster C with a centroid f_i .

Wkxv = 5R SW

Vgghpxhwr Q hdp kud



Fundamentals of Analyzing and Mining Data Streams



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

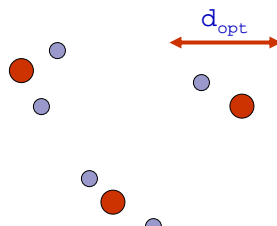
63

Fundamentals of Analyzing and Mining Data Streams



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



64

Fundamentals of Analyzing and Mining Data Streams



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

65

Fundamentals of Analyzing and Mining Data Streams



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 $d_{opt} = 1, 2, 4, 8, \dots$
 - We reject everything $< d_{opt}$, so best guess is $< 2d_{opt}$: our output will be $< 2 \cdot 2d_{opt} / d_{opt} = 4$ 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

66

Fundamentals of Analyzing and Mining Data Streams



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)

67

Fundamentals of Analyzing and Mining Data Streams



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]

68

Fundamentals of Analyzing and Mining Data Streams



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 Proceedings 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 aggregation 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.