Data Streams

Lecture Notes for Big Data Analytics

Naimat Ullah

March 2019

Contents

1 The data stream phenomenon

Stream Processing or Stream Analytics or Streaming Algorithms is the application of data analysis and algorithmic methods on a continuous stream of data items and drawing meaningful analytics and knowledge from it. Stream processing is generally performed in the following settings.

- **Single Pass over the stream:** The common requirement is to process each item exactly once. Though in certain cases we do use 2 pass or a small number of passes over the stream.
- **Limited Memory:** The common characteristic of stream processing is the requirement to use poly-logarithmic space (in length of stream or domain of data objects). This immediately implies that we cannot store the whole stream (and the single pass requirement becomes clear)
- **Constant per item processing:** Given an object the requirement is to process it in near real time. Usually the goal is to $O(1)$ time on each stream element.
- **Arbitrary arrival order:** Generally, in streaming algorithm we do not make any assumption on distribution or order of items. For instance we cannot say that objects are streamed in a random order or that objects are coming from a certain distribution. In other words, we essentially assume adversarial (worse-case) order.

1.1 Characteristic of data streams

Some characteristic of data streams or some key aspects in which data stream processing is different than the usual data processing are given as follows. These are based on the textbook Han & Kamber, Data Mining Concepts & Techniques [9].

- Huge volumes of continuous data, possibly infinite
- Fast changing and requires fast, real-time response
- Data stream captures nicely our data processing needs of today
- Random access is expensive
- Single scan algorithm (can only have one look)
- Store only the summary of the data seen thus far
- Most stream data are at pretty low-level or multi-dimensional in nature, needs multi-level and multi-dimensional processing

1.2 Stream Analysis versus Traditional Data Analysis

The concept of stream processing will be more clear if we observe how it differs with the traditional data analysis that we have been studying thus far in this course. The following is a contrast with traditional data analysis and in what aspects stream data is fundamentally different than traditional datasets based on a tutorial by Babcock et.al. in PODS 2002 [2].

Traditional Data (DBMS)	Data Stream
Persistent storage	Transient stream(s)
One-time query	Continuous query
Random access	Sequential access
Unbounded disk storage	Bounded main memory
Only current state matters	Arrival-order is critical
No real time services	Real-time requirements
Low update rate	Possibly multi-GB arrival rate (dynamic & fast)
Mixed granularity	Data at fine granularity

Table 1: Comparison of stream data vs. traditional data (DBMS)

1.3 Data Stream Processing Model

Streams are long (potentially unbounded) exact algorithms with limited memory are possible only for a few simple queries. In most cases it suffice to design approximate algorithms to derive analytics from the stream. Assume we are given an input $\mathcal{S}$ of data items in a streaming fashion and we are required to process $\mathcal{S}$ to compute a function $f(\mathcal{S})$ to output.

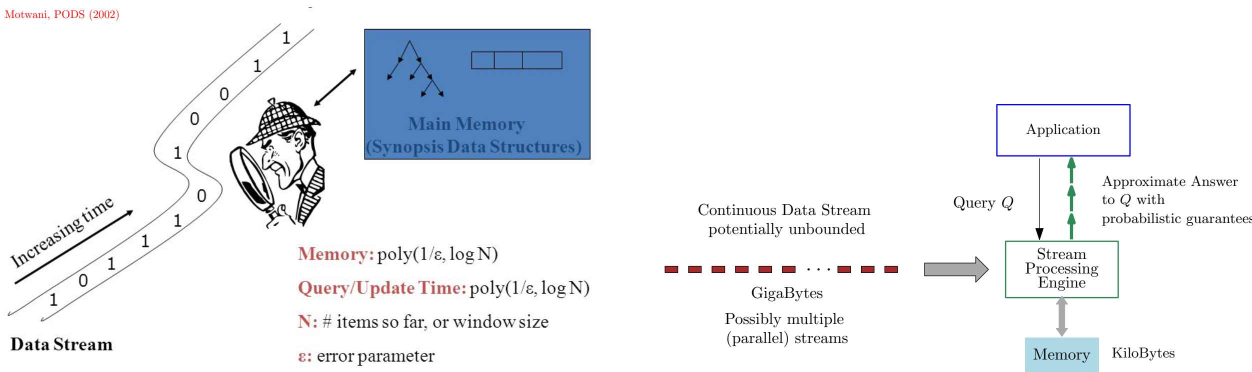


Figure 1: Summary of the stream processing model

1.3.1 Approximation Algorithms

As mentioned above, in most of interesting cases of computing functions over streams, $f(\mathcal{S})$ can provably not be computed given the sublinear space and 1 pass requirements. We, therefore often allow approximation algorithms that make errors with bounded probability.

Definition 1 (Approximation Algorithm with multiplicative guarantee)**.** *Let $\mathcal{A}$ be an algorithm to compute $f(\mathcal{S})$. Denote by $\mathcal{A}(\mathcal{S})$ the output of $\mathcal{A}$ on input $\mathcal{S}$. We say that $\mathcal{A}$ is an (ϵ, δ) approximation algorithm if*

$$\mathbb{P}\left((\mathcal{A}(\mathcal{S}) - f(\mathcal{S})) > \epsilon f(\mathcal{S})\right) \leq \delta.$$

Instead of the above requirement of a multiplicative approximation guarantee, an algorithm could provide an additive approximation guarantee.

Definition 2 (Additive Approximation Algorithm)**.** *Let $\mathcal{A}$ be an algorithm to compute $f(\mathcal{S})$. Denote by $\mathcal{A}(\mathcal{S})$ the output of $\mathcal{A}$ on input $\mathcal{S}$. We say that $\mathcal{A}$ is an (ϵ, δ) additive approximation algorithm if*

$$\mathbb{P}\left((\mathcal{A}(\mathcal{S}) - f(\mathcal{S})) > \epsilon\right) \leq \delta.$$

2 Data Stream Applications

Stream data comes in many domains and has various applications. Here are some application domains from the textbook Han & Kamber, Data Mining Concepts & Techniques [9].

- Telecommunication calling records
- Business: credit card transaction flows
- Network monitoring and traffic engineering
- Financial market: stock exchange

- Engineering & industrial processes: power supply & manufacturing
- Sensor, monitoring & surveillance: video streams, RFIDs
- Security monitoring
- Web logs and Web page click streams
- Massive data sets (even saved but random access is too expensive)

We elaborate on some of these applications.

2.1 Sensor Network Data and IoT

With the advent of IoT and sensor networks, there has been a huge amount of data generated by sensor nodes. The data is practically unlimited but the sensor devices have very limited computation power and memory. The limited computation power and (battery) power requirements to transmit data necessitate processing the stream of data at the sensor node without the need to storing or transmitting the whole data. It is estimated that 1 bit transmission consumes power almost equivalent to executing 800 instructions [11]. Certain key statistical aggregates and drawn analytics can be transmitted to a server for detailed processing. We note that even if there is no storage or communication bandwidth limitation, the near real time requirements for analytics warrants stream processing, while can keep the whole dataset (in a persistent manner) at the back-end or further detailed processing. Results of the basic stream analytics can be used to guide detailed and sophisticated analyses offline in data warehouses.

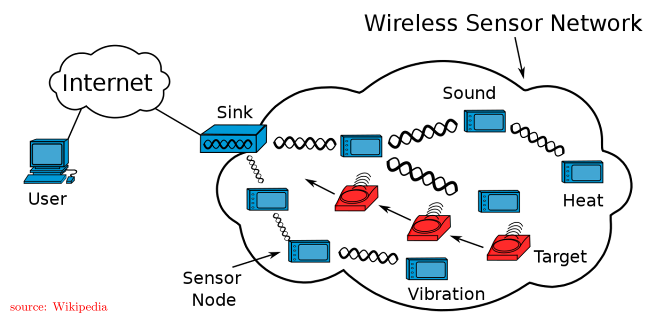


Figure 2: Every sensor in a network (WSN) produced a huge amount of data (streams) with limited capacity and power

2.2 Network Monitoring and Management

NetFlow: A Cisco tool for network administrators is widely used for performance metrics, security analysis, detection and forensics. For each flow it reports among other measurements

- Network Interface
- Source/Destination IP Addresses
- IP Protocol

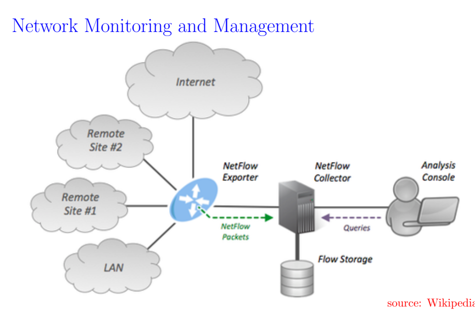


Figure 3: NetFlow architecture

- Source/Destination port
- TCP Flags
- Total packets/bytes in flow

Stream of this NetFlow data is widely used for network traffic engineering and monitoring, Traffic Volume estimation & analysis, Load balancing and Efficient Resource Utilization. This is also used for various network security purposes such as (D)DOS Attack Detection and identifying Service Level Agreement(SLA) Violation in real time.

Following are some example queries that can be (approximately) answered in the streaming setting (single pass and sub-linear memory). It is not very hard to see how answers to these queries can and will be used for the applications mentioned above.

- How many bytes sent b/w IP-1 and IP-2?
- How many IP addresses are active?
- What are the top 100 IP's by traffic volume (busiest)?
- What is the average duration of IP session and standard deviation?
- What is the median number of bytes in an IP session?
- Find sessions that transmitted more than x bytes
- Find sessions whose duration is more than twice the average session duration,
- List all IP's with a sudden spike in traffic
- List all IP involved in more than 1k sessions

As an example, AT&T Processes over 567 billion flow records per day, which amounts to about $\sim$ 15 PBytes of data (Fred Stinger (AT&T) FloCon (2017) Netflow Collection and Analysis..). AT&T detects and characterizes approximately 500 anomalies per day using this data.

2.3 Web Click Stream Analysis

Clickstream analysis or clickstream analytics is the process of tracking, collecting, processing and reporting analytics about the number, order, and routes of visits to components (such as webpages) on a website. The data is collected through cookies or from the web server logs.

Figure credit: Alex Smola @Yahoo research & ANU

Figure 4: Tracking and analysis of websites visits

Monitoring stream of website activity and collecting aggregates are used to learn trends of user behavior such as routing, stickiness (a user's tendency to remain at the website), number of clicks on various buttons, where users come from and where they go from the site. It is also used for various aggregate measurements such as frequency of users visits, number of visits for a given web component, number of unique visitors, top users and frequency norms of visits. These statistics are used to enhance the user experience on the website, improve GUI and other design aspects related to website usability. It also helps a great deal for technical aspects such as security and resource allocation in website design.

Another very important use of clickstream analytics is to increase revenue from e-commerce related websites. The goal is to improve the effectiveness of the website as a channel-to-market. The information such as number of visits to pages, time spent on pages and navigation between pages is used to enhance the so-called conversion rate (converting page visits to items sale) and digital marketing (recommending items based on identified user interests). Search for keywords up-selling and cross-selling to see more details of application of clickstream analysis.

2.4 Search Queries Stream

Figure credit: Alex Smola @Yahoo research & ANU

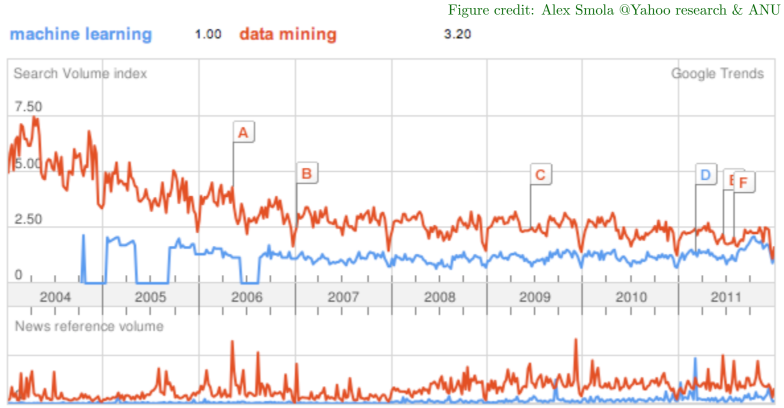
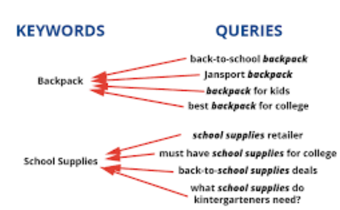


Figure 5: text

There are generally the following three types of search queries on general websearch.

1. Navigational search queries (e.g. facebook or youtube)
2. Informational search queries (e.g. Kruskal's algorithm for MST, Weather in Lahore)



3. Transactional search queries (Shoe stores in Lahore)

Briefly, review how advertisement works on a search engine. Since advertisers bid on keywords, in order to determine the best ad campaign search queries streams are processed to discover trends and patterns in queries, identify relevant keywords for website (ad campaign), to estimate competition scores or difficulty and predict keywords CPC (cost per click).

2.5 AMI Data Streams

Energy consumption Analysis from smart meters transmitting information of load per minute. The stream is of electricity consumption data (or any other utility) from AMI (Automatic Metering Interface) (aka smart meters). Some uses of stream analytics on this data are

- To find average hourly load, load surges, anomaly
- Short term load forecast (total or for individual consumers)
- Identify faults, drops, failures

Figure 6: Electricity consumption data analysis from AMI

2.6 Financial Time Series

Here the data is time stamped real time (multiple) stocks data. A classic application is the so-called algorithmic trading, which requires near real time prediction.

2.7 Optimized Query Execution Plan

In some cases the data doesn't have to be streamed in, but one can deal with massive data that is stored on disk in a streaming fashion (Single Pass, Sequential I/O). Such algorithms are also judged by the per item processing time, storage required and computation time.

Suppose we have a large students database with many fields. We do not have the usual (sub-linear) space restriction, streaming could still help us. We give an example of optimizing query execution plans by *selectivity estimation*. Suppose we want to execute a query like

```
SELECT * FROM Table WHERE 25 <= age <= 35 AND 54 <= weight <= 60
```

Suppose there are $n = 1M$ students in total. **Brute force execution time** is $2n$ comparisons, as we have to compare the two fields of every record.

If we have some statistical information about the distribution of each attribute in particular age and weight, then runtime will be significantly different.

Age	Freq
$0 - 10$	7%
$11 - 20$	8%
$21 - 30$	10%
$31 - 40$	12%
$41 - 50$	13%
$51 - 60$	25%
$61 - 70$	20%
$71+$	5%

Weight	Freq.
$0 - 20$	20%
$21 - 40$	25%
$41 - 60$	10%
$61 - 80$	15%
$81+$	30%

If we first first filter on Age, then on weight i.e. first compare every records age to see if is within the range and then compare the weight of only the filtered records, then the query can be executed in using $1.22n$ comparisons. On the other hand first filtering on weight, then on age would require $1.1n$ comparisons.

These aggregate statistics can be obtained using one scan on the whole database for all attributes or while the data is being stored in the database. These kind of techniques are often used for executing joins though (not just select queries). These techniques can also be used for performing Exploratory data analytics or Database monitoring.

3 Stream Model of Computation

A data stream is a sequence of m items $\mathcal{S} = a_1, a_2, a_3, \ldots, a_m$, where each item is chosen from some universe of size n. Typically we take the universe to be $[n] := \{1, 2, \ldots, n\}$. Some comments about streams:

- n and m are two size parameters.
- We do not assume anything about **distribution of items**
- Typically we work in the model where we see each **item only once** in the order given by $\mathcal{S}$, in general we cannot (or do not want to) save the whole stream. In the literature there are algorithms that take more than one passes over the stream but here we restrict ourselves to one-pass algorithms only.

- Our goal is to use a very **small amount of memory** for computing some function over stream, $f(\mathcal{S})$, i.e. space requirement of algorithm should be $o(\min\{m, n\})$. Ideally we should use $O(\log n + \log m)$ space. Note that this space is needed anyway to store one (or some constant number of) stream items for processing. Sometimes space requirement could be relaxed to polylogarithmic in $\min\{m, n\}$ (like $(\log n)^c$ for some constant c).

- In most of interesting cases of computing functions over streams, $f(\mathcal{S})$ can provably not be computed given the sublinear space and 1 pass requirements. We, therefore often allow **approximation algorithms** that make errors with bounded probability.

4 Synopsis of the stream

The Fundamental Methodology in stream processing is to keep a synopsis of the stream and answer query based on it. The synopsis is updated after examining each item in $O(1)$ time. The synopsis is a succinct summary of the stream (observed so far) with total size poly-log bits

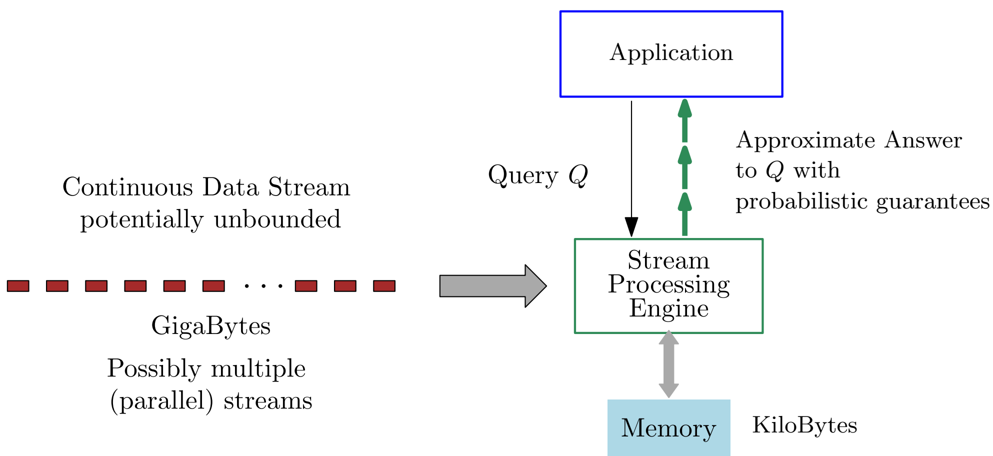


Figure 7: Stream processing: answer the query Q based on the synopsis of the stream

4.1 Synopsis Based Exact Stream Computation

Given a stream $\mathcal{S} = a_1, a_2, \ldots, a_m$, such that each $a_i \in [n]$. Using a small synposis, we can compute quite a few functions of the stream $\mathcal{S}$. These examples are included as a motivation and to make the concepts of stream computation and synopsis clear.

- Length of $\mathcal{S}$ (m): This can clearly be computed by storing a running counter. The size of synopsis is one integer.

- Sum of $\mathcal{S}$: This can also be computed by storing a running sum. I.e. an integer variable initialized to 0 is added to the next stream item a_i. At the end of stream or at query time, the variable contain the sum of all elements of the $\mathcal{S}$.

- Mean of $\mathcal{S}$: The mean value can be computed from the sum and length of $\mathcal{S}$ (computed above). Note the size of synposis in this case is 2 integers.
- Variance of $\mathcal{S}$: This can be computed from keeping a synopsis of 3 integers to get the sum of elements of $\mathcal{S}$, sum of squares of elements of $\mathcal{S}$, and length of $\mathcal{S}$.

$$Var(X) = E(X^2) - (E(X))^2$$

  Recall that computing variance using the definition of $Var(X) = E((X - \mu)^2)$ will not work in the streaming model.

All these use $O(\log n)$ bits memory and constant time per element. This is just to emphasize the fact, that though the streaming model is restrictive, we can still achieve quite a lot.

4.2 Finding Missing Integer(s): A motivating example

Suppose you are given $n-1$ distinct positive integer in $[n]$ and you want to find the one integer that is missing. You can solve this problem easily by maintaining a bit vector $B[1 \dots n]$ and marking $B[i] = 1$ if the input integer is i. Then at the end of input, one scan through B and identifying the index of 0 reveals the missing integer. Runtime of this algorithm is $2n - 1$ and space is n bits. Note that this is 2-pass algorithm.

We can solve the problem in a streaming model (low memory) by realizing that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ and maintaining only one integer S that is the running sum of all input integers. Then the missing integer is $\frac{n(n+1)}{2} - S$. Runtime of this algorithm is n additions and space complexity is $2n \log n$

What if two integers are missing and input is $n - 2$ distinct integers in $[n]$. You can do this by maintaining the sum S and product P of input integers and in the end solving a system of two equations and two unknowns. i.e.

$$\frac{n(n+1)}{2} - S \quad \text{and} \quad n! - P$$

Though this will solve the problem, but observe that the product P of $O(n)$ integers in $[n]$ is $n! \in \Omega(n^n)$ (by Sterling's approximation), hence storing P will take $n \log n$ bits, which is about the same number of bits to store the whole input, (each integer taking $\log n$ bits and there are n of them). Note that the original brute-force solution requires only n bits but that was two pass algorithm.

Another method, (**to find any constant k missing integers**) which is more space efficient is to maintain S_i, where $S_i = \sum_{j \in \mathcal{S}} j^i$, i.e. the sum of i^{th} powers of input integers. Since we know $\sum_{=1}^{n} j^i$, in the end just as above we solve k equations with k unknowns to finding the k missing integers.

Notice that the above puzzle and the algorithms for mean, variance, length etc. are data stream algorithms that are exact and deterministic. Such algorithms are uncommon in the streaming model, generally randomized and approximate algorithms are known for problems in the streaming model. In the following we discuss some representative and widely applied problems.

4.3 Common Families of Synopses

In these subsections, we are just going to mention the common families of synopses. You are encouraged to read and explore more about these topics for your research (c.f. [5]).

4.3.1 Synopsis: Sliding Window

Sliding window is the simplest synopsis of the stream. In this we keep the last w elements as synopsis and perform all processing on this subset. w is called the length of the window. The synopsis is updated as follows. On input a_i ($i \geq w$), a_{i-w} expires (is removed from the synopsis) and a_i added to the window. The sliding window can be used for queries like mean, sum, variance, count of pre-specified element(s) (e.g. non-zero, even) in the stream. This can also be extended to compute approximate median and k-median. A closely related synopsis is the so-called moving window, where next window is made up of the next w elements in the stream.

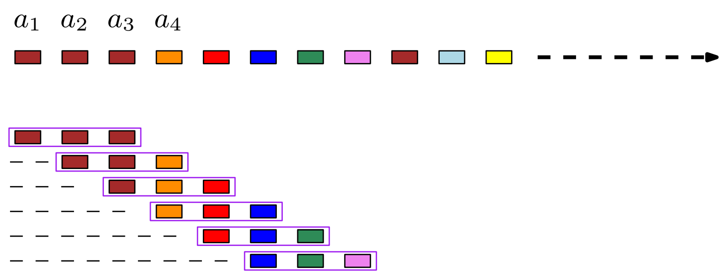


Figure 8: Data stream: sliding window example

4.3.2 Synopsis: Random Sample

Random sample is the most versatile general purpose synopsis with deep statistical foundations. Recall from your undergraduate statistics courses some relevant concepts like population, sample, confidence interval, size of sample, bias, weighted sampling, etc. We keep a "representative" subset of the stream as a synopsis and compute answers to the given query based on the sample (with appropriate scaling etc.) Probabilistic guarantees on the approximation quality of the query answer are derived using the fact that the sample is a (random) representative sample.

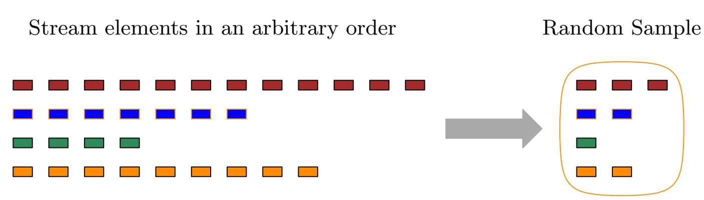


Figure 9: The random subset is a representative sample of the stream

4.3.3 Sampling Algorithms – Reservoir Sampling

In the following we briefly discuss algorithms to get a random sample from a dataset. We first discuss when the whole data is provided (in this case we assume the data is one-dimensional and given as an array or list). This is followed by sampling one or more elements from a stream. Sampling algorithm is a fascinating area, interested readers are highly encouraged to read P. Haas, *Data-Stream Sampling: Basic Techniques and Results* [8].

Problem 1. *Sample a random element from array A of length n. Generally, we require a uniform sample, i.e. each data point is chosen equally likely, (pick $A[i]$ with probability $1/n$).*

This problem can be solved as follows: Generate a random number $r \in [\,0, n\,]$. Many programming languages provide a pseudo random number generator function. Most of them generate real numbers in $[0, 1]$. Thus, to get a number in $[0, n]$ we can do $r \leftarrow \textsc{rand}() \times n$. The number r is then rounded and we return $A\,[\,\lceil r \rceil\,]$.

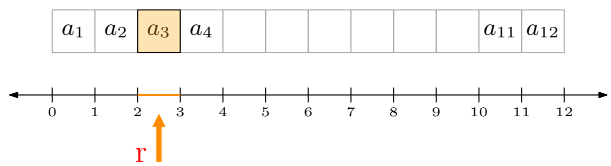


Figure 10: Sample an element from A (length 12)

Problem 2. *Given an array A of n elements, where the elements $A[i]$ has an associated weight w_i. Sample a random element (by weight) from A, i.e. choose $A[i]$ with probability w_i/W, where W is the total (sum of) weights of elements in A.*

Let $W_i = \sum_{j=1}^{n} w_i$ (this implies $W = W_n$), we generate a random number r in $[\,0, W\,]$. As discussed above this can be done for instance as $r \leftarrow \textsc{rand}() \times W_n$. Then we return the element $A\,[\,i\,]$ such that $W_{i-1} \leq r < W_i$. This returns $A[i]$ with probability w_i/W because the probability that r is such that $W_{i-1} \leq r < W_i$ is $W_i - W_{i-1}/W = w_i/W$.

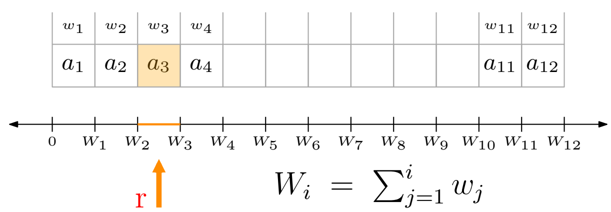


Figure 11: Sample random element (by weight) from array A of length 12

Problem 3. *Given a stream $\mathcal{S} = a_1, a_2, \ldots,$. Sample a random element from S, i.e. choose the element a_i with probability $1/m$. Here m is the length of the stream.*

If m is known, we can use the algorithm for uniform sampling an element from an array. In other words, we pick a random index r between 1 and m and then choose a_r (when it arrives). When m is unknown (think of it as follows: elements are streamed one at a time and at any time you may be asked to return a random sample. So m is known at the query time, but since we cannot store the whole stream, we have to be prepared with a random sample from the currently observed elements of the stream. The algorithm for this is called reservoir sampling.

Algorithm : Reservoir Sampling ($\mathcal{S}$)

```
R ← a_1                                      ▷ R (reservoir) maintains the sample
for i ≥ 2 do
    Pick a_i with probability 1/i
    Replace with current element in R        ▷ If a_i is picked
```

Let us analyze the probability that a given element a_i is chosen. The probability that a_i is in the reservoir R (at query time m or end of the stream) is given by

$$= \underbrace{\text{Pr that } a_i \text{ was selected at time } i}_{\frac{1}{i}} \times \underbrace{\text{Pr that } a_i \text{ survived in } R \text{ until time } m}_{\prod_{j=i+1}^{m}\left(1-\frac{1}{j}\right)}$$

$$= \frac{1}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \times \frac{\cancel{i+2}}{\cancel{i+3}} \times \ldots \times \frac{\cancel{m-2}}{\cancel{m-1}} \times \frac{\cancel{m-1}}{m} = \frac{1}{m}$$

Problem 4. *Given a stream $\mathcal{S} = a_1, a_2, \ldots,$. Sample k random element from S, i.e. the element a_i should be in the sample with probability k/m. Again, m is the length of the stream.*

Reservoir sampling is easily extended to solve this problem.

Algorithm : Reservoir Sampling $(\mathcal{S}, k)$

$R \leftarrow a_1, a_2, \ldots, a_k$ ▷ R (reservoir) maintains the sample
for $i \geq k+1$ **do**
  Pick a_i with probability k/i
  If a_i is picked, replace with it a randomly chosen element in R

The probability that a_i is in the reservoir R (at query time m or end of the stream) is given by

$$= \underbrace{\text{Pr that } a_i \text{ was selected at time } i}_{\frac{k}{i}} \times \underbrace{\text{Pr that } a_i \text{ survived in } R \text{ untill time } m}_{\prod_{j=i+1}^{m}\left(1-\left(\frac{k}{j}\times\frac{1}{k}\right)\right)}$$

$$= \frac{k}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \times \frac{\cancel{i+2}}{\cancel{i+3}} \times \ldots \times \frac{\cancel{m-2}}{\cancel{m-1}} \times \frac{\cancel{m-1}}{m} = \frac{k}{m}$$

For more discussion on efficiently sampling k elements in streaming fashion with probability proportional to weights but with dynamic weights see [3].

4.3.4 Synopsis: Histogram

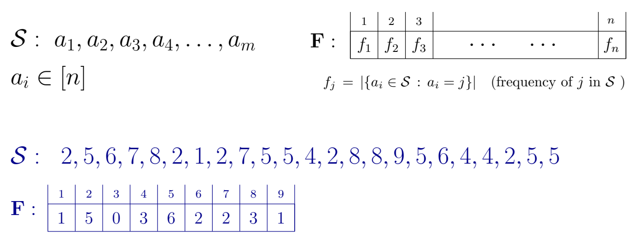
As we discussed earlier, a histogram is great tool to summarize and describe data. In the streaming setting, the histogram synopsis is some summary statistics (e.g. frequencies, means) of groups (subsets, buckets) in streams values. What statistic is recorded and computed and how buckets are formed depend on the application at hand. Quite a few different types of histograms are used in the literature as stream synopsis. Some of these are listed below. Equi-width histogram, Equi-depth histogram, V-optimal histogram, Multi-dimensional histogram.

4.3.5 Synopsis: Wavelets

Wavelets are essentially histograms of features (coefficients) in the frequency domain representation of the stream. Interested readers are referred to [5].

4.3.6 Synopsis: Linear Sketch

Among the above synopses, sample is a general purpose synopsis and can be used to answer any query about the whole stream. However, in sampling we only process the sampled elements and do not take any advantage from observing the whole stream. Sketches, histograms and wavelets take advantage from the fact the processor see the whole stream (though can't remember the whole stream). We discuss linear sketch in some detail and give some example of uses of them. Sketches are generally specific to a particular purpose (meaning sketches are designed to answer specific queries).



First, we briefly discuss how a sketch based streaming algorithm view a stream. A data stream is a sequence of m items $\mathcal{S} = a_1, a_2, a_3, \ldots, a_m$, where each item is chosen from some universe of size n. Typically we take the universe to be $[n] := \{1, 2, \ldots, n\}$. We are interested in computing some statistical property of the multiset of items in $\mathcal{S}$. The stream describes some one-dimensional function (a signal), this is represented as an array of length n. Quite often the signal is the frequency vector $F = \boxed{f_1 \mid f_2 \mid \ldots \quad \ldots \quad \ldots \mid f_n}$ where f_i is the frequency of element i, i.e. $f_j = |\{i : a_i = j\}|$.

A linear sketch interprets the stream as defining the frequency vector. Often we are interested in functions of the frequency vector from a stream.

$$\mathcal{S} \ = < a_1, a_2, a_3, \ldots, a_m > \qquad a_i \in [n]$$

$$f_i : \text{frequency of } i \text{ in } \mathcal{S} \qquad \mathbf{F} = \{f_1, f_2, \ldots, f_n\}$$

$$F_0 := \sum_{i=1}^{n} f_i^0 \qquad \triangleright \text{ number of distinct elements}$$

$$F_1 := \sum_{i=1}^{n} f_i \qquad \triangleright \text{ length of stream, } m$$

$$F_2 := \sum_{i=1}^{n} f_i^2 \qquad \triangleright \text{ second frequency moment}$$

Definition 3 (Linear Sketch)**.** *Given a stream $\mathcal{S}$, a data structure $sk(\mathcal{S})$ is called a sketch if*

- *$sk(\mathcal{S})$ is computed in streaming fashion (by processing each item in the steam)*
- *We can easily combine the sketches of two streams $\mathcal{S}_1$ and $\mathcal{S}_2$. More precisely, there is a space efficient streaming algorithm to compute $sk(\mathcal{S}_1 \circ \mathcal{S}_2)$, where $\mathcal{S}_1 \circ \mathcal{S}_2$ is the concatenation of these two streams.*

Basically (and quite often) we want to be able to efficiently update the sketch while processing the next item in the stream (think of $\mathcal{S}_1 = a_1, a_2, \ldots, a_{i-1}$ and $\mathcal{S}_2 = a_i$. But technically since we can have multiple streams the definition takes care of it. Linear sketch can be computed as a linear transform of $\mathbf{F}$. They are best suited for data streams and are highly parallelizable. Sketches are very useful the problems of computing norms of $\mathbf{F}$ that we focus on . Furthermore, sketches can be readily extended to variations of the basic stream model. We discuss some basic stream input models below.

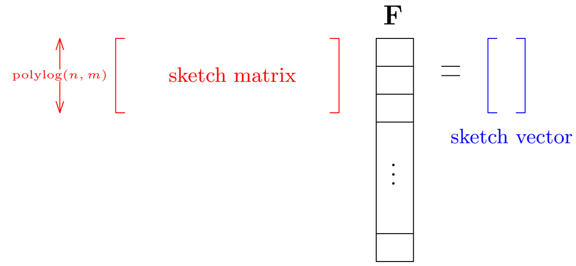


Figure 12: Linear sketches

5 Models of data streams

Various stream models differ how the items in the stream describe the signal. There are no standard definitions and a lot of variations. We consider the following three models and give examples of streams and the underlying signal.

1. **Time Series Model:** In this case if A is the underlying signal, then each a_i represents $A[a_i]$. For example if we want to record traffic on a given router every 5 minutes. A is an array of real number such that $A[i]$ is the traffic at the ith 5-min internal and the the values of traffic reported by the router every 5 minutes are stream items (a_i). Other example include the number of emails every day processed by a mail server, hourly trade volume in a certain stock exchange etc.

   For stream $\mathcal{S} : \langle 7, 3\rangle, \langle 3, 3\rangle, \langle 2, 9\rangle, \langle 7, 2\rangle, \langle 9, 1\rangle, \langle 3, 1\rangle$

   The final frequency vector will be $\mathbf{F}$ =

1	2	3	4	5	6	7	8	9
0	9	1	0	0	0	2	0	1

2. **Cash Register Model:** In this model each stream item is an increment to $A[i]$, so stream item a_i is like a pair (j, c), $(c > 0)$ and it means that $A_i[j] = A_{i-1}[j] + c$. This is the most common model and is very applicable to the case of frequency, (so far we considered $c = 1$ only). Applications: Suppose we want to monitor bandwidth consumed by different IP's. Each time an IP x uses a link it sends a certain number of packets, that could be treated as increments to x's previous usage, CDR (call data record) is exactly this thing.

   For stream $\mathcal{S} : \langle 7, 3\rangle, \langle 3, 3\rangle, \langle 2, 9\rangle, \langle 7, 2\rangle, \langle 9, 1\rangle, \langle 3, 1\rangle$

   The final frequency vector will be $\mathbf{F}$ =

1	2	3	4	5	6	7	8	9
0	9	4	0	0	0	5	0	1

3. **Turnstile Model:** In this case a_i's are updates to $A[a_i]$ (instead of just increments we allow decrements also). Again think of a_i as a pair (j, c), where c could be positive or

negative. Usually it is very hard to get tight results in this model, that is one reason (certainly for us) not to study this. In some cases we restrict $A[j]$ to always stay non-negative.

For stream $\mathcal{S}$: $\langle 7,3\rangle, \langle 3,3\rangle, \langle 2,9\rangle, \langle 7,-2\rangle, \langle 9,1\rangle, \langle 3,-1\rangle$

The final frequency vector will be $\mathbf{F}$ =

1	2	3	4	5	6	7	8	9
0	9	2	0	0	0	1	0	1

6 Finding frequency of an element: The Count-Min sketch

we are given a data stream $\mathcal{S} = a_1, a_2, a_3, \ldots, a_m$, where each $a_i \in [n]$. This implicitly defines a frequency vector $F =$ $\boxed{f_1 \mid f_2 \mid \ldots \quad \ldots \quad \ldots \mid f_n}$ where f_i is the frequency of i in the stream, i.e. $f_j = |\{i : a_i = j\}|$. Our goal is to estimate the frequencies f_i for all elements.

6.1 Applications

One can easily imagine applications for such queries on a data stream in the cash-register model but with c fixed to be 1. It is an easy exercise to extend this to any increments values. We can trivially solve this by keeping a counter array of length n, but we want streaming solution.

For a quick application of estimates of frequency vectors we have used it to compute kernel values between two objects. Which as we discussed earlier is defined to be the dot-product between the corresponding feature vectors. For more details see [7, 10]

6.2 The Count-Min sketch

The Count-Min sketch is a simple sketch and has found many applications. It was introduced by Cormode & Muthukrishnan[6]. The algorithm takes an error bound ϵ and an error probability bound δ and provides an additive approximation guarantee. We begin with a simpler version.

Let $h : [n] \to [1,k]$ be a function chosen uniformly from a 2-universal family of hash functions. A 2-universal family $\mathcal{H}$ of hash functions have that property,

$$Pr_{h \in_R \mathcal{H}}[h(x) = h(y)] = {}^1\!/\!_k$$

We keep an array $Count[1 \ldots k]$ of k integers. Consider the following simple algorithm.

Algorithm : Count-Min Sketch (k, ϵ, δ)

COUNT $\leftarrow$ ZEROS(k) ▷ sketch consists of k integers
Pick a random $h : [n] \mapsto [k]$ from a 2-universal family $\mathcal{H}$
On input a_i
  COUNT$[h(a_i)] \leftarrow$ COUNT$[h(a_i)] + 1$ ▷ increment count at index $h(a_i)$

On query j ▷ query: $\mathbf{F}[j] =$?
  return COUNT$[h(j)]$

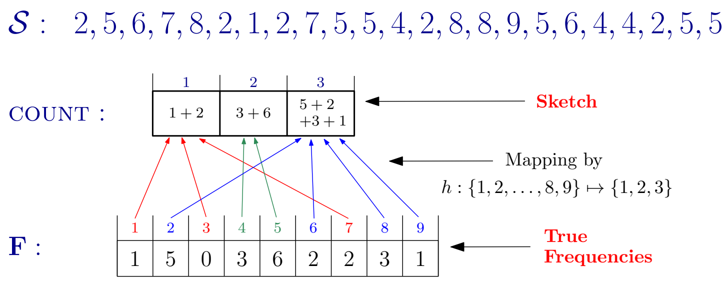
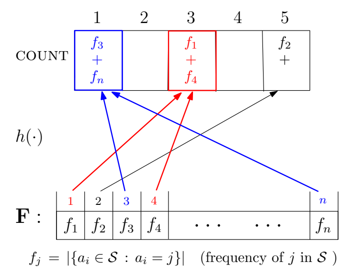


Figure 13: Count-Min Sketch

Note that when $k < n$ (which is typically the case, actually we require that $k \in o(n)$ i.e. $k \ll n$), the algorithm provides an upper bound on actual frequency (since the algorithm returns $Count[h(j)]$, other elements that hash to the same value, i.e. elements i, such that $h(i) = h(j)$ also contribute to the returned value $Count[h(j)]$).

Let $\tilde{f}_j = Count[h(j)]$ be the estimate provided by the algorithm for query j then from the above reasoning we get that



$$\tilde{f}_j \geq f_j.$$

For $j \in [n]$, we estimate the excess (error), X_j in $\tilde{f}_j$. Clearly, $X_j = \tilde{f}_j - f_j$. Let $1_{h(i)=h(j)}$ be the indicator random variable for the event $h(i) = h(j)$.

$$1_{h(i)=h(j)} = \begin{cases} 1 & \text{if } h(i) = h(j) \\ 0 & \text{otherwise} \end{cases}$$

Note that i makes contribution to $\tilde{f}_j$ iff $h(i) = h(j)$ ($1_{h(i)=h(j)} = 1$) and when it does contribute, its contribution is exactly f_i. We therefore get that

$$X_j = \sum_{i \in [n] \setminus j} f_i \cdot 1_{h(i)=h(j)}.$$

By the *goodness* of h, (h is a 2-universal hash function), we have that $\forall\ i \neq j\ \mathbb{P}[h(i) = h(j)] = \frac{1}{k}$. This gives us

$$\mathbb{E}[1_{h(i)=h(j)}] = \mathbb{P}[h(i) = h(j)] = {}^1\!/\!_k.$$

We find the expectation of X_j.

$$\begin{aligned} \mathbb{E}(X_j) &= \mathbb{E}(\sum_{i \in [n] \setminus j} f_i \cdot 1_{h(i)=h(j)}) = \sum_{i \in [n] \setminus j} \mathbb{E}(f_i \cdot 1_{h(i)=h(j)}) && \text{Linearity of expectation} \\ &= \sum_{i \in [n] \setminus j} f_i \cdot \mathbb{E}(1_{h(i)=h(j)}) = \sum_{i \in [n] \setminus j} f_i \cdot \frac{1}{k} \leq \sum_{i \in [n] \setminus j} \|F\|_1 \cdot \frac{1}{k} \end{aligned}$$

Since all frequencies are non-negative it is actually the L_1 norm of the frequency vector, that is why we denoted it by the L_1 norm of F.

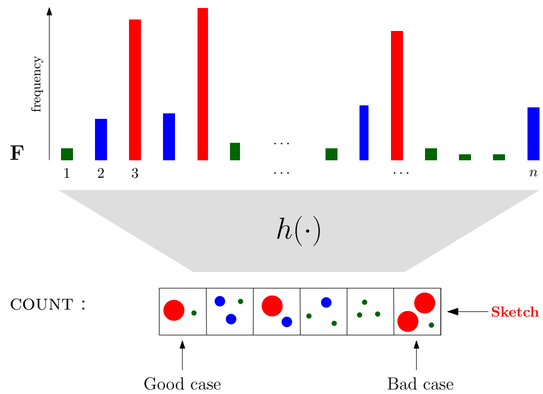


Figure 14: Comparison of good vs. bad case

Theorem 4 (Markov's Inequality)**.** *If Z be a non-negative random variable, then*

$$\mathbb{P}[Z \geq t \cdot \mathbb{E}(Z)] \leq \frac{1}{t}$$

Substitute $k = {}^2/_\epsilon$ (since k, the length of hash table is in our control) and using Markov's inequality we get that

$$\mathbb{P}[X_j \geq \epsilon \|F\|_1] = \mathbb{P}[X_j \geq 2\,\mathbb{E}[X_j]] \leq {}^1/_2$$

Summarizing the analysis of this algorithm we get

- $\tilde{f}_j \geq f_j$
- $\tilde{f}_j \leq f_j + \epsilon\|F\|_1$ with probability at least ${}^1/_2$

Hence Algorithm 1 is an $(\epsilon\|F\|_1, \frac{1}{2})$-additive approximation algorithm. Space required by the algorithm is k integers (plus some more for processing etc.) and $k = {}^2/_\epsilon$ is a constant.

6.3 Amplifying the probability

We can amplify the probability of success by selecting t independent hash functions $h_1, h_2, \ldots, h_t$ each from a 2-universal family of hash functions and proceed as follows. Each $h_i : [n] \to [1 \ldots k]$

Algorithm : Count-Min Sketch (k, ϵ, δ)

COUNT $\leftarrow$ ZEROS$(t \times k)$ ▷ sketch consists of t rows of k integers
Pick t random functions $h_1, \ldots, h_t : [n] \mapsto [k]$ from a 2-universal family
On input a_i
for $r = 1$ to t **do**
  COUNT$[r][h_r(a_i)] \leftarrow$ COUNT$[r][h_r(a_i)] + 1$
  ▷ increment COUNT$[r]$ at index $h_r(a_i)$
On query j ▷ query: $\mathbf{F}[j]$ =?
  return $\underset{1 \leq r \leq t}{\text{MIN}}$ COUNT$[r][h_r(j)]$

So we keep t estimates instead of 1, and since every estimate is an upper bound, it is clear that we should return the minimum of all estimates (the one which has minimum contribution from other elements).

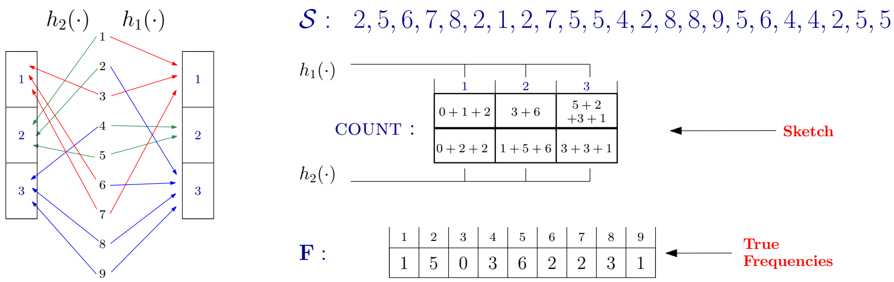


Figure 15: An example of the count-min sketch

Define X_{jr} to be the contribution of other elements to $Count[r][h(j)]$. We know that if the length of hash table is $k = 2/\epsilon$

$$\mathbb{P}[X_{jr} \geq \epsilon \|F\|_1] \leq 1/2$$

Now if $\tilde{f}_j \geq f_j + \epsilon \|F\|_1$, then for all $1 \leq r \leq t$, we must have that $X_{jr} \geq \epsilon \|F\|_1$, and the probability of this event is (since h_r's are independent) is bounded as

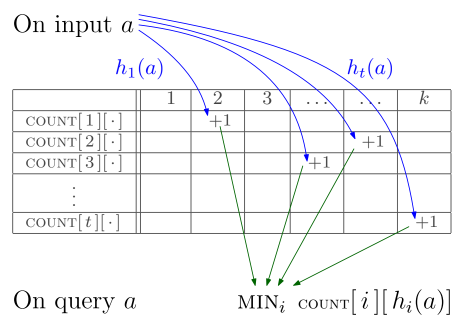


Figure 16: The Count-min sketch

$$\mathbb{P}[\ \forall\ r\ X_{jr} \geq \epsilon \|F\|_1] \leq (1/2)^t$$

Substitute $t = \log(\frac{1}{\delta})$ (the number of hash functions is in our control) and we get

$$\mathbb{P}[\ \forall\ r\ X_{jr} \geq \epsilon \|F\|_1] \leq (1/2)^{\log 1/\delta} = \delta$$

Summarizing the analysis of this algorithm 2 we get

- $\tilde{f}_j \geq f_j$
- $\tilde{f}_j \leq f_j + \epsilon \|F\|_1$ with probability at least $1 - \delta$

Hence Algorithm 2 is an $(\epsilon \|F\|_1, \delta)$ additive approximation algorithm. Space required by the algorithm is $k \cdot t$ integers (plus some more for processing etc), and $k = \frac{2}{\epsilon}$ and $t = \log(1/\delta)$.

7 Finding frequency of an element: The Count sketch

Since the Count-Min sketch always provides an upper bound, errors always accumulate, the Count sketch attempts to have the errors cancel each other. It was introduced [4]. Again we begin with a simpler version and later extend it.

7.1 The Count sketch

Let $h : [n] \to [1, k]$ and $g : [n] \to \{+1, -1\}$ be two hash functions independent of each other. We keep an array $Count[1 \ldots k]$ of k integers. Consider the following simple algorithm.

Algorithm : Count Sketch (k, ϵ, δ)

Pick a random $h : [n] \mapsto [k]$ from a 2-universal family $\mathcal{H}$
Pick a random $g : [n] \mapsto \{-1, 1\}$ from a 2-universal family
COUNT $\leftarrow$ ZEROS(k) ▷ sketch consists of k integers
On input a_i
    COUNT$[h(a_i)] \leftarrow$ COUNT$[h(a_i)] + g(a_i)$
    ▷ increment or decrement, depending on value of $g(a_i)$ COUNT at index $h(a_i)$
On query j ▷ query: $\mathbf{F}[j] =?$
    return $g(j) \times$ COUNT$[h(j)]$

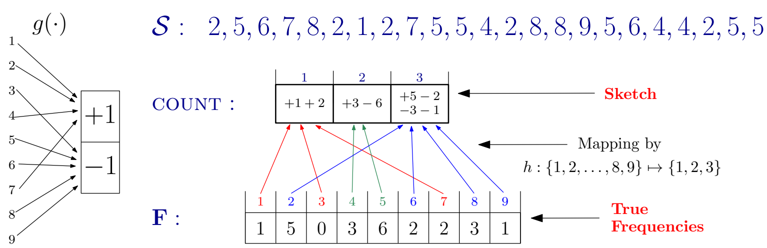


Figure 17: An example of the Count Sketch

Let $\tilde{f}_j$ be the estimate provided by the algorithm for query j, (i.e. $\tilde{f}_j = g(j) \cdot Count[h(j)]$), we will first show that in expectation the algorithm provides the correct answer

$$\mathbb{E}(\tilde{f}_j) = f_j.$$

For $j \in [n]$, we will estimate the error in $\tilde{f}_j$. Examining the algorithm it is easy to see that the $i \in [n]$ contributes to $Count[h(j)]$ iff $h(i) = h(j)$ and when it does contribute, its contribution is either f_i or $-f_i$, depending on $g(i)$.

From the above reasoning we get $Count[h(j)] = \sum_{i \in [n]} f_i \cdot g(i) \cdot 1_{h(i)=h(j)}$. On query j the output of the algorithm, $\tilde{f}_j$

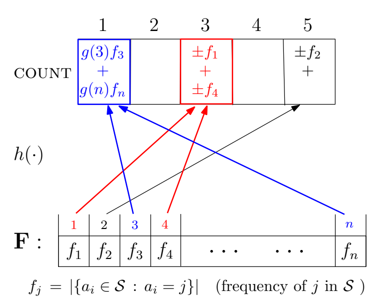


Figure 18: Count Sketch

$$\tilde{f}_j = g(j) Count[h(j)] = g(j) \cdot \sum_{i \in [n]} f_i \cdot g(i) \cdot 1_{h(i)=h(j)} = g(j) \left(f(j) g(j) + \sum_{i \in [n] \setminus j} f_i g(i) \cdot 1_{h(i)=h(j)} \right)$$

$$= f(j) \cdot (g(j))^2 + \sum_{i \in [n] \setminus j} f_i \cdot g(i) g(j) \cdot 1_{h(i)=h(j)} = f(j) + \sum_{i \in [n] \setminus j} f_i \cdot g(i) g(j) \cdot 1_{h(i)=h(j)}$$

The expected value of $\tilde{f}_j$ is

$$\begin{aligned}
\mathbb{E}(\tilde{f}_j) &= \mathbb{E}\left(f(j) + \sum_{i\in[n]\setminus j} f_i \cdot g(i)g(j) \cdot 1_{h(i)=h(j)}\right) = f(j) + \mathbb{E}\left(\sum_{i\in[n]\setminus j} f_i \cdot g(i)g(j) \cdot 1_{h(i)=h(j)}\right) \\
&= f(j) + \sum_{i\in[n]\setminus j} \mathbb{E}\left(f_i \cdot g(i)g(j) \cdot 1_{h(i)=h(j)}\right) \qquad \text{linearity of expectation} \\
&= f(j) + \sum_{i\in[n]\setminus j} f_i \cdot \mathbb{E}\left(g(i)g(j)\right) \cdot \mathbb{E}\left(1_{h(i)=h(j)}\right) \qquad h \text{ and } g \text{ are independent}
\end{aligned}$$

As above $\mathbb{E}(1_{h(i)=h(j)}) = \mathbb{P}[h(i) = h(j)] = \frac{1}{k}$ and $\mathbb{E}(g(i)g(j)) = -1(\frac{1}{2}) + 1(\frac{1}{2}) = 0$. We get that

$$\mathbb{E}(\tilde{f}_j) = f_j.$$

Next we calculate the variance of $\tilde{f}_j$.

$$\begin{aligned}
Var(\tilde{f}_j) &= Var\big(f(j) + \sum_{i\in[n]\setminus j} f_i \cdot g(i)g(j) \cdot 1_{h(i)=h(j)}\big) = Var\big(\sum_{i\in[n]\setminus j} f_i \cdot g(j)g(i) \cdot 1_{h(i)=h(j)}\big) \\
&= (g(j))^2 \cdot Var\big(\sum_{i\in[n]\setminus j} f_i \cdot g(i) \cdot 1_{h(i)=h(j)}\big) = Var\big(\sum_{i\in[n]\setminus j} f_i \cdot g(i) \cdot 1_{h(i)=h(j)}\big) \\
&\quad \text{the above equalities use } Var(a+X) = Var(X),\ Var(aX) = a^2 Var(X) \text{ and } (g(j))^2 = 1 \\
&= \sum_{i\in[n]\setminus j} Var\left(f_i \cdot g(i) \cdot 1_{h(i)=h(j)}\right) = \sum_{i\in[n]\setminus j} \mathbb{E}\left(f_i \cdot g(i) \cdot 1_{h(i)=h(j)}\right)^2 - \left(\mathbb{E}(f_i \cdot g(i) \cdot 1_{h(i)=h(j)})\right)^2 \\
&\quad \text{the above two equalities use independence of } g \text{ and } h \text{ and } Var(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 \\
&= \sum_{i\in[n]\setminus j} \mathbb{E}\left(f_i \cdot g(i) \cdot 1_{h(i)=h(j)}\right)^2 - \sum_{i\in[n]\setminus j} \left(\mathbb{E}(f_i \cdot g(i) \cdot 1_{h(i)=h(j)})\right)^2 \qquad \text{linearity of expectation} \\
&= \sum_{i\in[n]\setminus j} \mathbb{E}\left(f_i \cdot g(i) \cdot 1_{h(i)=h(j)}\right)^2 - 0 = \sum_{i\in[n]\setminus j} f_i^2\, \mathbb{E}\left(g(i) \cdot 1_{h(i)=h(j)}\right)^2 - 0 \qquad \text{using } \mathbb{E}(aX) = a\,\mathbb{E}(X) \\
&= \sum_{i\in[n]\setminus j} f_i^2\, \mathbb{E}\left(g(i)^2 \cdot 1^2_{h(i)=h(j)}\right) = \sum_{i\in[n]\setminus j} f_i^2 \cdot {}^1\!/_k \le \sum_{i\in[n]} f_i^2 \cdot \frac{1}{k} \le \sum_{i\in[n]} (\|F\|_2)^2 \cdot {}^1\!/_k
\end{aligned}$$

Theorem 5 (Chebyshev's Inequality)**.** *If Z be a random variable, then*

$$\mathbb{P}[|Z - \mathbb{E}(Z)| \ge t\sqrt{Var(X)}] \le \frac{1}{t^2}$$

Apply the Chebyshev's inequality with $Var(\tilde{f}_j) \le \frac{(\|F\|_2)^2}{k}$, we get that

$$\mathbb{P}\left[|\tilde{f}_j - f_j| \ge \epsilon\sqrt{k}\sqrt{\frac{(\|F\|_2)^2}{k}}\right] \le \frac{1}{k\epsilon^2} \implies \mathbb{P}\left[|\tilde{f}_j - f_j| \ge \epsilon\|F\|_2\right] \le \frac{1}{k\epsilon^2}$$

Substitute $k = {}^{3}/_{\epsilon^2}$ (since k is length of the hash table which in our control) and using Chebyshev's inequality we get that

$$\mathbb{P}\left[|\tilde{f}_j - f_j| \geq \epsilon \|F\|_2\right] \leq {}^{1}/_{3}$$

Summarizing the analysis of this algorithm we get

- $\mathbb{E}(\tilde{f}_j) = f_j$
- $\tilde{f}_j \leq f_j + \epsilon\|F\|_2$ with probability at least $\frac{1}{3}$
- $\tilde{f}_j \geq f_j - \epsilon\|F\|_2$ with probability at least $\frac{1}{3}$ OR
- $f_j - \epsilon\|F\|_2 \leq \tilde{f}_j \leq f_j + \epsilon\|F\|_2$ with probability at least $\frac{1}{3}$

Hence Algorithm 3 is an $(\epsilon\|F\|_2, \frac{1}{3})$-additive approximation algorithm. Space required by the algorithm is k integers (plus some more for processing etc), and $k = \frac{3}{\epsilon^2}$ is a constant.

7.2 Amplifying the probability

We can amplify the probability of success by selecting t independent hash functions $h_1, h_2, \dots, h_t$ and t independent hash functions (signs) $g_1, g_2, \dots, g_t$ each from a 2-universal family of hash functions and proceed as follows. Each $h_i : [n] \to [1 \dots k]$ and $g_i : [n] \to \{+1, -1\}$

Algorithm : Count Sketch (k, ϵ, δ)

COUNT $\leftarrow$ ZEROS$(t \times k)$ ▷ sketch consists of t rows of k integers
Pick t random functions $h_1, \dots, h_t : [n] \mapsto [k]$ from a 2-universal family
Pick t random functions $g_1, \dots, g_t : [n] \mapsto \{-1, 1\}$ from a 2-uni. family
On input a_i
for $r = 1$ to t **do**
  COUNT$[r][h_r(a_i)] \leftarrow$ COUNT$[r][h_r(a_i)] + g_r(a_i)$
  ▷ inc/dec COUNT$[r]$ at index $h_r(a_i)$
On query j ▷ query: $\mathbf{F}[j] = ?$
  return $\underset{1 \leq r \leq t}{\text{MEDIAN}}\ g_r(j) \times$ COUNT$[r][h_r(j)]$

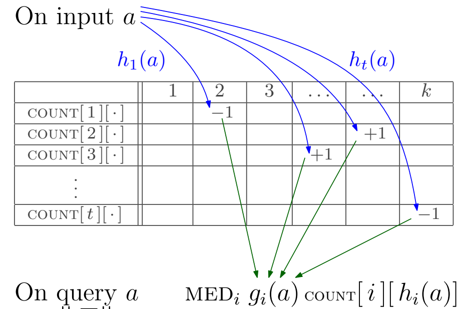
It is clear that why should we return median of the t estimates, as each one of them can be an over or under estimate. For the analysis, let $\tilde{f}_{jr}$, $(1 \leq r \leq t)$, be the r-th estimate, i.e.

$$\tilde{f}_{jr} = g_r(j) \cdot Count[r][h(j)]$$

From above analysis we know that when $k = {}^{3}/_{\epsilon^2}$,

$$\mathbb{P}\left[|\tilde{f}_{jr} - f_j| \geq \epsilon \|F\|_2\right] \leq {}^{1}/_{3}$$

Note that since we returned the median of t estimates, if the median is more than x units away from the actual value, then at least half of all the estimates are more than x units away from the actual value. The probability of many independent estimates being far away is much smaller. More precisely, Let



$$X_r = \begin{cases} 1 & \text{if } |\tilde{f}_{jr} - f_j| \geq \epsilon \|F\|_2 \\ 0 & \text{otherwise} \end{cases}$$

Basically X_r is the indicator random variable, indicating whether $\tilde{f}_{jr}$ has too much error. We count how many of the estimates have too much error, i.e. let

$$X = \sum_{r=1}^{t} X_r.$$

Now since we said that if our answer is wrong (has error beyond the $\epsilon\|F\|_2$ margin), then at least half of the estimates would be wrong. In other words X would be more than $t/2$. But we will show that only with very small probability X could be so large. By the way we have

$$\mathbb{E}(X) = \sum_{r=1}^{t} \mathbb{E}(X_r) = \sum_{r=1}^{t} \frac{1}{3} = \frac{t}{3}$$

Theorem 6 (Chernoff Bound)**.** *Let $Z_1, Z_2, \ldots, Z_n$ be independent random variables, $0 \leq Z_i \leq 1$. Let $Z = Z_1 + Z_2 + \ldots + Z_n$. Suppose $\mu = \mathbb{E}(Z) = \mathbb{E}(Z_1) + \mathbb{E}(Z_2) + \ldots + \mathbb{E}(Z_n)$. Then for any $\gamma \geq 0$*

- $\mathbb{P}\left[Z \geq (1+\gamma)\mu\right] \leq \exp\left(-\frac{\gamma^2}{2+\gamma}\mu\right)$
- $\mathbb{P}\left[Z \leq (1-\gamma)\mu\right] \leq \exp\left(-\frac{\gamma^2}{2}\mu\right)$

The computer science version of this says that $\mathbb{P}[|Z-\mu| \geq \gamma\mu] \leq 2^{-\Omega(\mu\gamma^2)}$

Since the expected number of erroneous estimates is $t/3$, if the median of estimates is wrong, then the number of erroneous estimates must be at least $t/2 = t/3 + t/6$, i.e. X must deviate from its expected value be at least $t/3$ and by the Chernoff bound the probability of that happening is given as follows:

$\mathbb{P}[X \geq (1+1/2)\frac{t}{3}] \leq \exp\left(-\frac{(1/2)^2}{2+(1/2)} \cdot \frac{t}{3}\right) = \exp\left(-\frac{t}{30}\right)$. We want this probability to be at most δ (the given bound on error probability), substitute $t = 30\ln(\frac{1}{\delta})$. Note that t is in our control, the number of times we want to run Algorithm 3 in parallel. Overall we get,

- $\mathbb{E}(\tilde{f}_j) = f_j$

- $|\tilde{f}_j - f_j| \leq \epsilon \|F\|_2$ with probability at least $1 - \delta$

Hence Algorithm 4 is an $(\epsilon\|F\|_2, \delta)$ additive approximation algorithm. Space required by the algorithm is $k \cdot t$ integers (plus some more for processing etc), and $k = \frac{3}{\epsilon^2}$ and $t = 30 \ln(1/\delta)$ (assuming my calculations are right).

8 Computing higher frequency moments: The AMS Sketch

Notes on Alon-Mathias-Szegedy sketches [1] to estimate higher frequency moments of a stream will be included later. Comprehensive slides for the AMS sketch to estimate F_2 are already shared.

9 Comparison of Count and Count-Min Sketch

Note that the Count sketch uses more space than the Count-Min sketch, but does it give better estimates. Recall our discussion on the L_k norms.

- The deviation for the Count-Min sketch is bounded by $\epsilon\|F\|_1$ and that of the Count sketch is bounded by $\|F\|_2$.
- For all vectors $z \in \mathbb{R}^n$, $\|z\|_1 \geq \|z\|_2$, hence Count-Min sketch will always give looser bound.
- The two give exactly the same answer when only one element has non-zero frequency, since for $F = (f_1, 0, 0, 0, 0, 0)$, $\|F\|_1 = \|F\|_2 = 1$.
- Count outperforms the Count-Min sketch by the most when all frequencies are equal. Say $F = (1, 1, 1, \ldots, 1)$, in this case $\|F\|_1 = n$ and $\|F\|_2 = \sqrt{n}$. So for say $\epsilon = 1/10$, the Count-Min estimate is within $n/10$ of the actual answer (with probability $1 - \delta$), which the Count estimate is within $\sqrt{n}/10$ of the actual answer.
- As the distribution of frequencies spreads out, Count outperforms the Count-Min.

10 Lower bound on Sketches for Frequency

In this section, I want to show that the Count-Min sketch is asymptotically the best with respect to space requirements. This is to give you a general idea of how lower bounds work and what do they mean. Please read it on your own and if you encounter something like this in your research, you would be familiar with the typical structure and terminology. We use a reduction from the well known Index problem from communication complexity.

10.1 Index problem

In this problem there are two players, Alice and Bob. Alice holds a binary array A of length n (bit string). Bob has an index $j \in [n]$. Bob wants to find out the value of $A[j]$. All communication has to be one-way, i.e. only Alice can send information to Bob. Bob knows the value of n but doesn't know the contents of A. We denote an instance of index problem as the pair (A, j). The question is how many bits Alice needs to send to Bob so as he determines the value of $A[j]$. We want to reduce the communication complexity of the protocol (algorithm). We only require that Bob determines $A[j]$ with high probability (one-way probabilistic communication). Please note a few things about the Index problem.

- It is very easy if Alice is allowed to send all n bits, since then Bob will check $A[j]$. This requires n bits communication.
- With some XoR tricks, may be we can reduce to $n-1$ or so bits communication.
- Bob can randomly guess the value of $A[j]$, and his guess will be correct with probability $1/2$. But we want it to be correct with probability $1-\delta$ for a small input parameter δ.
- The Index problem has many applications in Private Information Retrieval etc.

We use the following theorem from information theory.

Theorem 7. *(Lower bound on index problem) For Bob to determine the $A[j]$ correctly with probability at least $1/2 - \delta$, Alice needs to send $\Omega(\delta^2 n)$ bits.*

Basically it says that to improve upon the random guess we need $\Omega(n)$ bits communication.

10.2 Reduction from the Frequency estimation to Index problem

Theorem 8. *To estimate f_j within error $\epsilon\|F\|_1$ with constant probability, one needs at least $\Omega(1/\epsilon)$ space.*

Proof. Assume that there is a solution that estimates f_j within error $\epsilon\|F\|_1$ with probability at least $1-\delta$ and it uses $O(1/\epsilon)$ space. This means there is an algorithm G that outputs estimate $\tilde{f}_i$ such that $\mathbb{P}\left[|\tilde{f}_j - f_j| \leq \epsilon\|F\|_1\right] \leq 1-\delta$ and G uses space at most $1/\epsilon$.

We will show that if there is such an algorithm G, then using G we can efficiently solve the Index problem. To see that suppose we have an instance of the index problem (A, j). We make a stream $\mathcal{S}$ with each element from the set $\{0, 1, \ldots, n\}$. The stream will realize the following frequency vector.

$f_0 = 2 \cdot |\{i : A[i] = 0\}|$, i.e. the frequency of 0 is twice the number of 0's in A and $f_i = 2$, if $A[i] = 1$. In other words, for each 0 in A we put two 0's in $\mathcal{S}$ and for each 1 at location i, we put two i's in $\mathcal{S}$. As an example, for $A = \begin{array}{|c|c|c|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} & \multicolumn{1}{c}{6} \\ \hline 1 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array}$, the stream $\mathcal{S}$ will be $1, 1, 0, 0, 3, 3, 0, 0, 5, 5, 6, 6$

This $\mathcal{S}$ defines the frequency vector $F = \begin{array}{|c|c|c|c|c|c|c|} \multicolumn{1}{c}{0} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} & \multicolumn{1}{c}{6} \\ \hline 4 & 2 & 0 & 2 & 0 & 2 & 2 \\ \hline \end{array}$

Let $\epsilon = 1/2n$, clearly $\|F\|_1 = 2n = 1/\epsilon$ and $\epsilon\|F\|_1 = 1$. Lets apply the algorithm G on $\mathcal{S}$ and estimate the frequency of j (within error $\epsilon\|F\|_1$).

If for $\tilde{f}_j$, G outputs anything above 1, then $A[j] = 1$ and otherwise $A[j] = 0$. Because if $A[j] = 0$, then $f_j = 0$, and the algorithm can overestimate it by at most $0+1 = \epsilon\|F\|_1$ with high probability G, and if $A[j] = 1$, then $f_j = 1$ and the algorithm will return something between 1 and 3 with high probability.

The contradiction comes from the fact that now Alice can make a stream $\mathcal{S}$ given the array A with her. And run algorithm G on $\mathcal{S}$ and send to Bob whatever data structure G is using (the sketch G computes). Since it was claimed that G uses space $O(1/\epsilon)$, hence the communication from Alice to Bob is at most $O(1/\epsilon) = O(n)$, which contradicts the theorem above.

□

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16. ACM, 2002.

[3] M. Beg, M. Ahmad, A. Zaman, and I. Khan. Scalable approximation algorithm for graph summarization. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 502–514, 2018.

[4] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, page 693–703. Springer-Verlag, 2002.

[5] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends Databases*, 4(1–3):1–294, 2012.

[6] Graham 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] M. Farhan, J. Tariq, A. Zaman, M. Shabbir, and I. Khan. Efficient approximation algorithms for strings kernel based sequence classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6935–6945, 2017.

[8] Peter J. Haas. *Data-Stream Sampling: Basic Techniques and Results*, pages 13–44. Springer Berlin Heidelberg, 2016.

[9] Jiawei Han and Micheline Kamber. *Data Mining. Concepts and Techniques.* Morgan Kaufmann, 2nd ed. edition, 2006.

[10] P. Kuksa, I. Khan, and V. Pavlovic. Generalized similarity kernels for efficient sequence classification. In *SIAM Intern. Conf. on Data Mining (SDM)*, pages 873–882, 2012.

[11] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2003.