Sampling in data streams

RA-MIRI QT Curs 2020-2021

1 Data stream models

2 Sampling

The data

The data

- Data arrives as sequence of items.
- Sometimes continuously and at high speed.

The data

- Data arrives as sequence of items.
- Sometimes continuously and at high speed.
- Can't store them all in main memory.
- Can't read again; or reading again has a cost.

The data

- Data arrives as sequence of items.
- Sometimes continuously and at high speed.
- Can't store them all in main memory.
- Can't read again; or reading again has a cost.

- We abstract the data to a particular feature, the data field of interest the label.

The data

The data

- We have a set of n labels Σ and our input is a stream $s = x_1, x_2, x_3, \ldots x_m$, where each $x_i \in \Sigma$.

The data

- We have a set of n labels Σ and our input is a stream $s = x_1, x_2, x_3, \dots x_m$, where each $x_i \in \Sigma$.
- Take into account that some times we do not know in advance the length of the stream.

The data

- We have a set of n labels Σ and our input is a stream $s = x_1, x_2, x_3, \ldots x_m$, where each $x_i \in \Sigma$.
- Take into account that some times we do not know in advance the length of the stream.
- Goal Compute a function of stream, e.g., median, number of distinct elements, longest increasing sequence.

Why has it become popular?

- Practical appeal:

Why has it become popular?

- Practical appeal:
    - Faster networks, cheaper data storage, ubiquitous data-logging results in massive amount of data to be processed.
    - Applications to network monitoring, query planning, I/O efficiency for massive data, sensor networks aggregation, ...

Why has it become popular?

- Practical appeal:
  - Faster networks, cheaper data storage, ubiquitous data-logging results in massive amount of data to be processed.
  - Applications to network monitoring, query planning, I/O efficiency for massive data, sensor networks aggregation, ...
- Theoretical Appeal:
  - Easy to state problems but hard to solve.
  - Links to communication complexity, compressed sensing, embeddings, pseudo-random generators, approximation, parallel computation, ...

Why has it become popular?

- Practical appeal:
  - Faster networks, cheaper data storage, ubiquitous data-logging results in massive amount of data to be processed.
  - Applications to network monitoring, query planning, I/O efficiency for massive data, sensor networks aggregation, ...
- Theoretical Appeal:
  - Easy to state problems but hard to solve.
  - Links to communication complexity, compressed sensing, embeddings, pseudo-random generators, approximation, parallel computation, ...
- Origins in 70's but has become popular in this century because of growing theory and very applicable.

The computational data stream models

- Classical streaming model

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.
  - Measures of complexity: number of passes over the data, the size of the working memory, the per-item processing time.

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.
  - Measures of complexity: number of passes over the data, the size of the working memory, the per-item processing time.
- Semi-streaming model

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.
  - Measures of complexity: number of passes over the data, the size of the working memory, the per-item processing time.
- Semi-streaming model
  - Usual for graph problems.
  - Working memory is $O(n \text{ polylog } n)$, for a graph with n vertices.
  - Enough space to store vertices but not for storing all the edges.

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.
  - Measures of complexity: number of passes over the data, the size of the working memory, the per-item processing time.
- Semi-streaming model
  - Usual for graph problems.
  - Working memory is $O(n \text{ polylog } n)$, for a graph with n vertices.
  - Enough space to store vertices but not for storing all the edges.
- Streaming with sorting

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.
  - Measures of complexity: number of passes over the data, the size of the working memory, the per-item processing time.
- Semi-streaming model
  - Usual for graph problems.
  - Working memory is $O(n \text{ polylog } n)$, for a graph with n vertices.
  - Enough space to store vertices but not for storing all the edges.
- Streaming with sorting
  - Allows the creation of intermediate streams.
  - Streams can be sorted at no cost.
  - Algorithms run in phases reading and creating a stream

Algorithmic goals

- Data streams are potentially of unbounded size.
- As the amount of computation and memory is limited it might be impossible to provide exact answers.

Algorithmic goals

- Data streams are potentially of unbounded size.
- As the amount of computation and memory is limited it might be impossible to provide exact answers.
- Algorithms use randomization and seek for an approximate answer.

Algorithmic goals

- Data streams are potentially of unbounded size.
- As the amount of computation and memory is limited it might be impossible to provide exact answers.
- Algorithms use randomization and seek for an approximate answer.
- Typical approach:
  - Build up a synopsis data structure
  - It should be enough to compute answers with a high confidence level.

1 Data stream models

2 Sampling

Sampling

Sampling

- Sampling is a general technique for tackling massive amounts of data.

Sampling

- Sampling is a general technique for tackling massive amounts of data.
- Example: To compute the median packet size of some IP packets, we could just sample some and use the median of the sample as an estimate for the true median. Statistical arguments relate the size of the sample to the accuracy of the estimate.

Sampling

- Sampling is a general technique for tackling massive amounts of data.
- Example: To compute the median packet size of some IP packets, we could just sample some and use the median of the sample as an estimate for the true median. Statistical arguments relate the size of the sample to the accuracy of the estimate.
- Challenge: But how do you take a sample from a stream of unknown length or from a sliding window?

Reservoir Sampling (Vitter 1985)

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.
- Algorithm:

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.
- Algorithm:
  - Initially $x = x_1$
  - On seeing the t-th element, $x = x_t$ with probability $1/t$

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.
- Algorithm:
  - Initially $x = x_1$
  - On seeing the t-th element, $x = x_t$ with probability $1/t$
- Analysis:

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.
- Algorithm:
  - Initially $x = x_1$
  - On seeing the t-th element, $x = x_t$ with probability $1/t$
- Analysis:
  - 1 pass, $O(\log n)$ memory (in bits), and $O(1)$ time (in operations) per item.

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.
- Algorithm:
  - Initially $x = x_1$
  - On seeing the t-th element, $x = x_t$ with probability $1/t$
- Analysis:
  - 1 pass, $O(\log n)$ memory (in bits), and $O(1)$ time (in operations) per item.
  - Quality?
    What is the probability that $x = x_i$ at some time $t \geq i$?

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream of unknown length.
- Algorithm:
  - Initially $x = x_1$
  - On seeing the t-th element, $x = x_t$ with probability $1/t$
- Analysis:
  - 1 pass, $O(\log n)$ memory (in bits), and $O(1)$ time (in operations) per item.
  - Quality?
    What is the probability that $x = x_i$ at some time $t \geq i$?
    At any time step t, for $i \leq t$, $Pr[x = x_i] = 1/t$

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - On seeing the t-th element, $t > k$, add x_t to X with probability k/t.
  - If x_t is selected to be added, before adding it, evict a element from X, selected uniformly at random.

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - On seeing the t-th element, $t > k$, add x_t to X with probability k/t.
  - If x_t is selected to be added, before adding it, evict a element from X, selected uniformly at random.
- Analysis:

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - On seeing the t-th element, $t > k$, add x_t to X with probability k/t.
  - If x_t is selected to be added, before adding it, evict a element from X, selected uniformly at random.
- Analysis:
  - 1 pass, $O(k \log n)$ memory, and $O(1)$ time per item.

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - On seeing the t-th element, $t > k$, add x_t to X with probability k/t.
  - If x_t is selected to be added, before adding it, evict a element from X, selected uniformly at random.
- Analysis:
  - 1 pass, $O(k \log n)$ memory, and $O(1)$ time per item.
  - Quality?
    What is the probability that $x_i \in X$ at some time $t \geq i$?

Reservoir Sampling II: Quality

- At any time step t, for $i \leq t$, $Pr[x_i \in X] = k/t$

Reservoir Sampling II: Quality

- At any time step t, for $i \leq t$, $Pr[x_i \in X] = k/t$
- The proof is by induction on t.

Reservoir Sampling II: Quality

- At any time step t, for $i \leq t$, $Pr[x_i \in X] = k/t$
- The proof is by induction on t.
  - Base $t = k$: $Pr[x_i \in X] = 1$, for $i = 1, \dots, k$.
  - Induction hypothesis: true for time steps up to $t - 1$
    - $Pr[x_t \in X] = k/t$
    - For $i < t$, $x_i \in X$ when x_t is not selected and x_i was in the sample at step $t - 1$, or when x_t is selected, x_i was in the sample at step $t - 1$ and x_i is not evicted.

$$\begin{aligned} Pr[x_i \in X] &= \left(1 - \frac{k}{t}\right)\frac{k}{t-1} + \frac{k}{t}\frac{k}{t-1}\left(1 - \frac{1}{k}\right) \\ &= \frac{k}{t-1} - \frac{k}{t}\frac{k}{t-1}\frac{1}{k} = \frac{k}{t-1} - \frac{1}{t}\frac{k}{t-1} = \frac{k}{t} \end{aligned}$$

Reservoir Sampling for Sliding Windows

- Problem: Maintain a uniform sample of k items from the last w items.

Reservoir Sampling for Sliding Windows

- Problem: Maintain a uniform sample of k items from the last w items.
- Why reservoir sampling does not work?

Reservoir Sampling for Sliding Windows

- Problem: Maintain a uniform sample of k items from the last w items.
- Why reservoir sampling does not work?
  - Suppose an element in the reservoir expires
  - Need to replace it with a randomly-chosen element from the current window
  - However, in the data stream model we have no access to past data
  - Could store the entire window but this would require $O(w)$ memory.

Sliding Windows: Replace-Sampling algorithm

- Algorithm:
  - Maintain a reservoir sample for the first w items in s.
  - When the arrival of an item causes an element in the sample to expire, replace it with the new arrival.

Sliding Windows: Replace-Sampling algorithm

- Algorithm:
  - Maintain a reservoir sample for the first w items in s.
  - When the arrival of an item causes an element in the sample to expire, replace it with the new arrival.
- Analysis
  - The algorithm solves the problem
  - 1 pass, $O(k \log n)$ space and $O(1)$ time per item.
- Trouble: The sample is highly periodic, this might look as unfair in many applications.

Sliding Windows: Backing-Sample algorithm

- Algorithm:
  - Maintain a backing sample B:

Sliding Windows: Backing-Sample algorithm

- Algorithm:
  - Maintain a backing sample B:
    - Add x_t to B with probability $2ck \log w/w$
    - Remove from B all the elements that expire at time t.
  - The sample X of size k is obtained by an uniform sampling of k items from B.

Sliding Windows: Backing-Sample algorithm

- Algorithm:
  - Maintain a backing sample B:
    - Add x_t to B with probability $2ck \log w/w$
    - Remove from B all the elements that expire at time t.
  - The sample X of size k is obtained by an uniform sampling of k items from B.
- Analysis
  - 1 pass, $O((k + |B|) \log n)$ space and $O(k)$ time per item.
  - $|B|$? Should be small compared to w.
  - Quality? The algorithm might fail if $|B| < k$ at some step.

Sliding Windows: Backing-Sample size

- Exercise Using Chernoff bounds, the size of the backing sample is between k and $4ck \log w$ with probability $c' w^{-c}$.
- Selecting the adequate c, with high probability the algorithm succeeds in keeping a large enough backing sample.

Sliding Windows: Backing-Sample size

- Exercise Using Chernoff bounds, the size of the backing sample is between k and $4ck \log w$ with probability $c' w^{-c}$.
- Selecting the adequate c, with high probability the algorithm succeeds in keeping a large enough backing sample.
- The bound on the space is $O(k \log w)$ with high probability.

Chain-Sampling for Sliding Windows

Chain-Sampling for Sliding Windows

- Algorithm: $(k = 1)$

Chain-Sampling for Sliding Windows

- Algorithm: $(k = 1)$
  - Maintain a reservoir sample for the first w items in s, but
  - whenever an element x_i is selected, choose and index $j \in [w]$ uniformly at random, x_{i+j} will be the replacement for x_i.
  - For $t > w$, when $t = i + j$, set $x = x_{i+j}$ (and choose the next replacement).

Chain-Sampling for Sliding Windows

- Algorithm: $(k = 1)$
  - Maintain a reservoir sample for the first w items in s, but
  - whenever an element x_i is selected, choose and index $j \in [w]$ uniformly at random, x_{i+j} will be the replacement for x_i.
  - For $t > w$, when $t = i + j$, set $x = x_{i+j}$ (and choose the next replacement).
- Analysis
  - 1 pass, $O(\log n + \log w)$ space and $O(1)$ time per item (some better bound?).
  - Provides a uniform sample.
- For higher values of k run k parallel chain samples.
  With high probability, for large enough w, such chains will not intersect.

Chain-Sampling: total time

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along the chain of selected items.

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along the chain of selected items.
- the number of possible chains of elements with more than a data elements is bounded by the number of partitions of m into a ordered integer parts, which is bounded by $\binom{m}{a}$.
- Each such chain has probability at most m^{-a}.

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along the chain of selected items.
- the number of possible chains of elements with more than a data elements is bounded by the number of partitions of m into a ordered integer parts, which is bounded by $\binom{m}{a}$.
- Each such chain has probability at most m^{-a}.
- The probability of updating x steps is therefore at most $\binom{m}{a} m^{-a}$.
- Using Stirling's approximation we get the bound $\left(\frac{e}{a}\right)^{a}$.

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along the chain of selected items.
- the number of possible chains of elements with more than a data elements is bounded by the number of partitions of m into a ordered integer parts, which is bounded by $\binom{m}{a}$.
- Each such chain has probability at most m^{-a}.
- The probability of updating x steps is therefore at most $\binom{m}{a} m^{-a}$.
- Using Stirling's approximation we get the bound $\left(\frac{e}{a}\right)^a$.
- For $a = O(\log m)$ this is less than m^{-c}, for constant c

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along the chain of selected items.
- the number of possible chains of elements with more than a data elements is bounded by the number of partitions of m into a ordered integer parts, which is bounded by $\binom{m}{a}$.
- Each such chain has probability at most m^{-a}.
- The probability of updating x steps is therefore at most $\binom{m}{a} m^{-a}$.
- Using Stirling's approximation we get the bound $\left(\frac{e}{a}\right)^a$.
- For $a = O(\log m)$ this is less than m^{-c}, for constant c
- With high probability the number of updates is $O(\log m)$.