ECEN 689
Special Topics in Data Science for Communications Networks

Nick Duffield
Department of Electrical & Computer Engineering
Texas A&M University

Lecture 5
Optimizing Fixed Size Samples
Sampling as a Mediator of Constraints

Data Characteristics
(Heavy Tails, Correlations)

Resource Constraints
(Bandwidth, Storage, CPU)

Sampling

Query Requirements
(Ad Hoc, Accuracy, Aggregates, Speed)
Why Summarize (ISP) Big Data?

• When transmission bandwidth for measurements is limited
  – Not such a big issue in ISPs with in-band collection
• Typically raw accumulation is not feasible (even nation states)
  – High rate streaming data
  – Maintain historical summaries for baselining, time series analysis
• To facilitate fast queries
  – When infeasible to run exploratory queries over full data
• As part of hierarchical query infrastructure:
  – Maintain full data over limited duration window
  – Drill down into full data through one or more layers of summarization
The need to limit sample volume
Fixed Size Sample

- Given $N$ objects, select $k < N$ objects uniformly at random
  - Each subset of $k$ objects should be equally likely
- One way: uniformly sample from $N$, $k$ times w/o replacement

1/$N$
1/(N-2)
1/(N-3)

k objects selected
Fixed Size Stream Sample: Reservoirs

• Stream constraint: see each item once
  – Discard permanently if not selected

• Assume reservoir of capacity k items available
  – Reasonable: $k = \text{final sample size}$

• Can provisionally include items from stream in reservoir
  – Take first k items w.p. 1

• Can discard later to select different items later in stream

$k = 3$
Reservoir Sampling in Practice

How to achieve uniform sampling distribution?
No need to know stream size in advance

• Include first $k$ items w.p. 1
• Include item $n > k$ w.p. $p_n = k/n$, $n > k$, if included: evict one item
  – Pick $j$ uniformly from $\{1,2,\ldots,n\}$
  – If $j \leq k$, swap item $n$ into location $j$ in reservoir, discard replaced item

• Neat proof shows the uniformity of the sampling method:
  – Let $S_n =$ sample set after $n$ arrivals

Previously sampled item: induction
$m \in S_{n-1}$ w.p. $p_{n-1} \Rightarrow m \in S_n$ w.p. $p_{n-1} \times (1 - p_n / k) = p_n$

New item: selection probability
$\text{Prob}[n \in S_n] = p_n := k/n$
Reservoir Sampling: Skip Counting

• Simple approach: check each item in turn
  – $O(1)$ per item:
  – Fine if computation time < interarrival time
  – Otherwise build up computation backlog $O(N)$

• Better: “skip counting”
  – Find random index $m(n)$ of next selection > $n$
  – Distribution: $\text{Prob}[m(n) \leq m] = 1 - (1-p_{n+1})*(1-p_{n+2})*…*(1-p_m)$

• Expected number of selections from stream is
  $$k + \sum_{k<m\leq N} p_m = k + \sum_{k<m\leq N} k/m = O(k \left( 1 + \ln \left( \frac{N}{k} \right) \right))$$

• There is algorithm with this average running time
IPPS Stream Reservoir Sampling

- Each arriving item:
  - Provisionally include item in reservoir
  - If \( m+1 \) items, discard 1 item randomly (same as include \( m \) items randomly)
  - Choose inclusion probabilities to be previous IPPS
    - Calculate threshold \( z \) to include \( m \) items on average: \( z \) solves \( \sum_i p_z(x_i) = m \)
    - Discard item \( i \) with probability \( q_i = 1 - p_z(x_i) \)
  - Adjust \( m \) surviving \( x_i \) with Horvitz-Thompson \( x'_i = x_i / p_i = \max\{x_i, z\} \)

Example: \( m=9 \)

![Diagram showing the process of IPPS Stream Reservoir Sampling](image)
Computation in IPPS Stream Sampling

• “Calculate threshold $z$ to include $m$ average items: $z$ solves $\Sigma_i p_z(x_i) = m$”?

• Weight order: $x_1 \leq x_2 \leq \ldots \leq x_n$. Any $z$: small items: $x_i \leq z$; Large $x_i > z$

• If $x(t)$ is a small item, then:
  
  $m = \Sigma_i p_z(x_i) = \Sigma_i \min\{1, x_i / z\}$  
  (implicit definition of $z$, using form of $p_z$)
  
  $\leq \Sigma_{i \leq t} x_i / z + (m+1 - t)$  
  ($p_z(x_i) \leq 1$ for the $m+1 - t$ terms $i > t$)
  
  $\leq \Sigma_{i \leq t} x_i / x(t) + (m+1 - t)$  
  ($x(t)$ small and hence $\leq z$)

• In other words: $\Sigma_{i \leq t} x_i / x(t) \geq t-1$

• Largest possible index $t$ for small item:
  
  $t^* = \max\{t: \Sigma_{i \leq t} x_i / x(t) \geq t-1\}$

• Then find $z$ from $\Sigma_{i \leq t^*} x_i / z = t^*-1$ (why?)

• How to find $t^*$?
How to find $t^*$?

- $t^* = \max\{ t: \sum_{i\leq t} x(i) \cdot x(t) \geq t-1 \}$

- Exercise:
  - show $g(t) = \sum_{i\leq t} x(i) / x(t) - t - 1$ is nonincreasing in $t$.

- Show that this makes $t^*$ easier to find
Monotonic Functions

• Searching for changepoints
  – f is binary function on \{1,2,\ldots,n\}: f(t) is either 0 or 1
  – f is monotonic: for some \( t^* \), \( f(t) = 1 \) if and only if \( t > t^* \)
  – Example:
    \[ f(t) = 0 \text{ if } \sum_{i \leq t} x(i) / x(t) \geq t-1; \]
    \[ f(t) = 1 \text{ if } \sum_{i \leq t} x(i) / x(t) < t-1; \]
    \[ t^* = \max\{t: f(t) = 0\} = \max\{ t: \sum_{i \leq t} x(i) / x(t) \geq t-1\} \]

• Task
  – Find changepoint \( t^* \)

• Simple approach:
  – inspect \( f(t) \) for \( t = 1,2,\ldots, t^* \) = last \( t \) for which \( f(t) = 0 \)
  – computational cost: \( O(n) \): might have to inspect all \( n \)
Binary Search for Changepoint

- **Initialize:**
  - Set t in center

- **Iterate (until can move t no further right):**
  - If \( f(t) = 1 \)
    - we know \( t^* < t \)
    - restrict attention to lower half subinterval, set t to its center
  - If \( f(t) = 0 \)
    - we know \( t^* \geq t \)
    - restrict attention to upper half subinterval, set t to its center

- \( t^* = t \)

- If h is number of halvings then \( 2^h \sim n \)

- #iterations = number of halvings \( h = O(\log n) < O(n) \)
Need to generalize binary search

- Domain may be point subset of real numbers
  - E.g. set of weight $x_i$, not simply $\{1, 2, \ldots, n\}$
- Domain can change with time
  - Database insertions and deletions
  - For a given $f$, this can alter changepoint: need to locate again
- Need general way
  - To store/retrieve data points
  - That enables finding function changepoints
  - Abstracts the idea of binary search from any detailed setting
  - Is efficient
Introducing binary search trees

- Data structure to store and retrieve points in ordered set
  - e.g. numbers with order of “<”
- Points do not need to be added in any particular order
  - No presorting needed
- Tree like structure is very efficient
  - Stores $n=2^h-1$ items if depth $h$
  - Computational cost $O(\log n)$ to retrieve any item
  - We will use to locate changepoint of binary monotonic function
Storing in a Binary Search Tree

• Storing an ordered set, e.g., \{17, 4, 6, 23, 36, 4\} with > order
  – Store first element at root” 17
  – Pass further elements down tree
    • To left child if element ≤ child
    • To right child if element > child

• Tree is depth \(O(\log m)\) for \(m\) items
• Computational complexity to add, delete or retrieve item is \(O(\log m)\)
• Binary search easy: go left/right until found, \(O(\log m)\) steps
• Need to **rebalance**: maintain bounded depth (i.e. approx. symmetry)
Search on a Binary Tree

• Search for some node (and whatever information attached)
  – say node 9
• Start at root
• Iterate until found:
  – Branch left if $9 \leq \text{node}$
  – Branch right if $9 > \text{node}$

```
4  
 |  
 |  
4
```

```
6
 |
|
 |
4
```

```
17
 |
|  
|  
9
```

```
23
 |
|  
|  
36
```

```
25
 |
|  
|  
9
```

```
9
 |
|  
|  
Found node 9!
```

Found node 9!
Monotone Functions and Binary Search Tree

- Monotone function \( f \) on nodes:
  - \( f(\text{node}_1) \leq f(\text{node}_2) \) if and only if \( \text{node}_1 < \text{node}_2 \)
  - Example: \( f(\text{node}) = \) weight stored at node

- Monotone binary function
  - Takes values 0 or 1
  - Seek \( \text{node}^* \) = largest node with \( f(\text{node}) = 0 \)
  - Example: largest node with weight \( \leq 10 \)
  - Happens to be 9: not known at start
  - \( f(\text{node}) = 0 \) if weight \( \leq 10 \), 1 otherwise

- Start at root and iterate until can't move further right:
  - if( \( f(\text{node}) = 1 \) ) {
    node > \( \text{node}^* \): so branch left
  }
  - if( \( f(\text{node}) = 0 \) ) {
    node \( \leq \) \( \text{node}^* \): so branch right
  }
  - \( \text{node} = \text{node}^* \)

\[ 
\begin{align*}
\text{node} & = 17 \\
\text{node}^* & = 9
\end{align*}
\]

- \( f(17) = 1 \)
- \( f(4) = 0 \)
- \( f(6) = 0 \)
- \( f(9) = 0 \)
- \( f(25) = 0 \)
- \( f(36) = 0 \)
Back to IPPS: maintaining partial sums $\sum_{i \leq t} x_{(i)}$

- **Insert:** each node $x_{(i)}$ maintains partial sums $S_{(i)}$ of weights $x_{(j)} \leq x_{(i)}$ that traverse it
- **Recovery:** from $x_{(i)}$ add up partial sums $S_{(j)}$ from all nodes $x_{(j)} \leq x_{(i)}$ on path back to root
- **Weight of items $\leq 25$:**
  $25+23+17+4+6+4+9$
- **Weight of items $\leq 9$:**
  $9+6+4+4$
The Iteration and its Computational Cost

• Current reservoir of unbiased estimates, threshold $z$
  – Reservoir maintained in two BSTs;
    • One for small items ($x_i \leq z$) and large ($x_i > z$)
    • Maintains partial sums for number and total weight of items $\leq x_i$

• Insert new weight
  – Binary tree search to find $t^* = \max \{ t : \Sigma_{i=t} x(i) / x(t) \geq t-1 \}$
    • Find $\Sigma_{i=t} x(i)$ and $t$ by adding up appropriate partial sums for weights and counts
  – Compute new threshold $z$ from $\Sigma_{i=t^*} x(i) / z = t^*-1$
  – Transfer items between small and large BSTs as needed

• Discard one item at random using discard probabilities
  – Generate random $r$ uniformly on $[0,1]$
  – Find smallest $d \leq t^*$ such at $\Sigma_{i=d} (1 - x(i) / z) \geq r$
    • Binary search on small items: sum is function of counts and weight sums

• Basic computational takeaway
  – Computing in BST gives $O(\log m)$ complexity per arriving item
  – Actually $O(\log \log m)$ averaged over $m$ arrivals
Summary: IPPS Sampling on Data Streams

• Motivation
  – Computer networking: storing sampled flow records for later analysis

• Ingredients:
  – Probability/Statistics, Algorithms, Data structures

• Statistical properties
  – Statistically optimal trade-off between sampling size and variance

• IPPS stream reservoir sampling
  – Iterative algorithm that maintains only current unbiased estimates in fixed size reservoir of size $m$

• Implementation
  – Used binary search trees to get $O(\log m)$ computational cost per arrival
Additional references

• Binary trees in general:
  – Goodrich: Data Structures and Algorithms in Python. Ch. 8
    • TAMU Library, online
  – These notes describe insertion and search; BST also provide for
deletion, rebalance and other operations.

• IPPPS Stream Sampling:
  – Cohen et. Al., Efficient Stream Sampling for Variance-Optimal