Data Streams

- A stream is a sequence of data that is too large to be stored in available memory

Examples

- Internet search logs
- Network Traffic
- Sensor networks
- Scientific data streams (astronomical, genomics, physical simulations)...

Streaming Model

- Stream of elements $a_1, \ldots, a_i, \ldots$ each from an alphabet $\Sigma$ and taking $b$ bits to represent
- Single or small number of passes over the data
- Almost all algorithms are randomized and approximate
  - Usually necessary to achieve efficiency
  - Randomness is in the algorithm, not the input
- Goals: minimize space complexity (in bits), processing time

Example Streaming Problems

- Let $a_{[1:t]} = \langle a_1, \ldots, a_t \rangle$ be the first $t$ elements of the stream
- Suppose $a_1, \ldots, a_t$ are integers in $\{-2^b + 1, -2^b + 2, \ldots, -1, 0, 1, 2, \ldots, 2^b - 1\}$
  - Example stream: $3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32$
  - How many bits do we need to maintain $f(a_{[1:t]}) = \sum_{i=1}^{t} a_i$?
    - Outputs on example: $3, 4, 21, 25, 16, 48, 149, 152, -570, -567, 333, 337, 379, \ldots$
    - $O(b + \log t)$
- How many bits do we need to maintain $f(a_{[1:t]}) = \max_{i=1}^{t} a_i$?
  - Outputs on example: $3, 3, 17, 17, 17, 32, 101, 101, 101, 101, 900, 900, 900, \ldots$
  - $O(b)$ bits
Example Streaming Problems

- The median of all the numbers we’ve stored so far
  - Example stream: 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32
  - Median: 3, 1, 3, 3, 3, 4, 3, …
  - This seems harder...

- The number of distinct elements we’ve seen so far?
  - Outputs on example: 1, 2, 3, 4, 5, 6, 7, 7, 8, 8, 9, 9, 9, …

- The elements that have appeared at least an $\epsilon$-fraction of the time? These are the $\epsilon$-heavy hitters
  - Cover today

Many Applications

- Internet router may want to figure out which IP connections are heavy hitters, e.g., the ones that use more than .01% of your bandwidth
  - Or maybe the router wants to know the median (or 90-th percentile) of the file sizes being transferred
  - Hashing is a key technique

Finding $\epsilon$-Heavy Hitters

- $S_t$ is the multiset of items at time $t$, so $S_0 = \emptyset$, $S_1 = \{a_1\}$, …, $S_t = \{a_1, …, a_t\}$, $\text{count}_t(e) = |\{i \in \{1, 2, …, t\} \text{ such that } a_i = e\}|$

- $e \in \Sigma$ is an $\epsilon$-heavy hitter at time $t$ if $\text{count}_t(e) > \epsilon \cdot t$

- Given $\epsilon > 0$, can we output the $\epsilon$-heavy hitters?
  - Let’s output a set of size $\frac{1}{\epsilon}$ containing all the $\epsilon$-heavy hitters

- Note: can output “false positives” but not allowed to output “false negatives”, i.e., not allowed to miss any heavy hitter, but could output non-heavy hitters

Finding $\epsilon$-Heavy Hitters

- Example: E, D, B, D, D, D, B, A, C, B, 10, B, E, E, E, E_E, E_E
  - (the subscripts are just to help you count)

- At time 5, the element D is the only 1/3-heavy hitter
- At time 11, both B and D are 1/3-heavy hitters
- At time 15, there is no 1/3-heavy hitter
- At time 16, only E is a 1/3-heavy hitter

Can’t afford to keep counts of all items, so how to maintain a short summary to output the $\epsilon$-heavy hitters?
Finding a Majority Element

• First find a .5-heavy hitter, that is, a majority element:
  memory ← empty and counter ← 0
  when element $a_t$ arrives
  if (counter == 0)
    memory ← $a_t$ and counter ← 1
  else
    if $a_t$ = memory
      counter ++
    else
      counter --
      (discard $a_t$)
  • At end of the stream, return the element in memory

Analysis of Finding a Majority Element

• If there is no majority element, we output a false positive, which is OK
• If there is a majority element, we will output it. Why?
  • When we discard an element $a_t$, we throw away a different element
  • Every time we throw away a copy of a majority element, we throw away another element, but majority element is more than half the total number of elements, so can’t throw away all of them

Extending to $\epsilon$-Heavy Hitters

Set $k = \lceil \frac{n}{2} \rceil - 1$
Array $T[1, ..., k]$, where each location can hold one element from $\Sigma$
Array $C[1, ..., k]$, where each location can hold a non-negative integer
$C[i] \leftarrow 0$ and $T[i] \leftarrow \perp$ for all $i$

If there is $j \in \{1, 2, ..., k\}$ such that $a_t = T[j]$, then $C[j] +$
Else if some counter $C[j]$ = 0 then $T[j] \leftarrow a_t$ and $C[j] \leftarrow 1$
Else decrement all counters by 1 (and discard element $a_t$)

$est_\epsilon(e) = C[j]$ if $e = T[j]$ for some $j$, and $est_\epsilon(e) = 0$ otherwise
Analyzing Counts

• **Lemma**: \(0 \leq \text{count}_t(e) - \text{est}_t(e) \leq \frac{1}{k+1} \leq \epsilon \cdot t\)

• **Proof**: \(\text{count}_t(e) \geq \text{est}_t(e)\) since we never increase a counter for \(e\) unless we see \(e\).

If we don’t increase \(\text{est}_t(e)\) by 1 when we see an update to \(e\), we decrement \(k\) counters and discard the current update to \(e\).

So we drop \(k+1\) distinct stream updates, but there are \(t\) total updates, so we won’t increase \(\text{est}_t(e)\) by 1, when we should, at most \(\frac{1}{k+1} \leq \epsilon \cdot t\) times.

Heavy Hitters Guarantee

• At any time \(t\), all \(\epsilon\)-heavy hitters \(e\) are in the array \(T\). **Why?**

  • For an \(\epsilon\)-heavy hitter \(e\), we have \(\text{count}_t(e) > \epsilon \cdot t\)
  
  • But \(\text{est}_t(e) \geq \text{count}_t(e) - \epsilon \cdot t\)

  • So \(\text{est}_t(e) > 0\), so \(e\) is in array \(T\)

  • Space is \(O(k \log(\Sigma) + \log t)) = O(1/\epsilon) \log(\Sigma) + \log t\) bits

Heavy Hitters with Deletions

• Suppose we can delete elements \(e\) that have already appeared

• Example: \((\text{add}, A), (\text{add}, B), (\text{add}, A), (\text{del}, B), (\text{del}, A), (\text{add}, C)\)

• Multisets at different times

  \(S_0 = \emptyset, S_1 = \{A\}, S_2 = \{A, B\}, S_3 = \{A, A, B\}, S_4 = \{A, A\}, S_5 = \{A\}, S_6 = \{A, C\}, \ldots\)

• “Active” set \(S_t\) has size \(|S_t| = \sum_{e \in \Sigma} \text{count}_t(e)\) and can grow and shrink

Data Structure for Approximate Counts

• Query “What is \(\text{count}_t(e)\)?”, should output \(\text{est}_t(e)\) with:

  \[\Pr[|\text{est}_t(e) - \text{count}_t(e)| \leq \epsilon |S_t|] \geq 1 - \delta\]

• Want space close to our previous \(O(1/\epsilon) \log(\Sigma) + \log t\) bits

• Let \(h: \Sigma \rightarrow \{0, 1, 2, \ldots, k-1\}\) be a hash function (will specify later)

• Maintain an array \(A[0, 1, \ldots, k-1]\) to store non-negative integers

  when update \(a_t\) arrives:

  - if \(a_t = (\text{add}, e)\) then \(A[h(e)]++\)
  - else \(a_t = (\text{del}, e)\), and \(A[h(e)]--\)

• \(\text{est}_t(e) = A[h(e)]\)
Data Structure for Approximate Counts

- \( A[h(e)] = \sum_{e' \in \Sigma} \text{count}_t(e') \cdot 1(h(e') = h(e)) \), where \( 1(\text{condition}) \) evaluates to 1 if the condition is true, and evaluates to 0 otherwise

- \( A[h(e)] = \text{count}_t(e) + \sum_{e' \neq e} \text{count}_t(e') \cdot 1(h(e') = h(e)) \)

- Since we have a small array \( A \) with \( k \) locations, there are likely many \( e' \neq e \) with \( h(e') = h(e) \), but can we bound the expected error?

High Probability Bounds for CountMin

- Have \( 0 \leq \text{est}_t(e) - \text{count}_t(e) \leq |S_t|/k \) in expectation from CountMin
  - With probability \( 1/2 \), \( \text{est}_t(e) - \text{count}_t(e) \leq 2|S_t|/k \) Why?

  - Can we make the success probability \( 1-\delta \)?
    - Independent repetition: pick \( m \) hash functions \( h_1, ..., h_m \) with \( h_i: \Sigma \to \{0, 1, 2, ..., k-1\} \) independently from \( H \). Create array \( A_i \) for \( h_i \) when update \( a_i \) arrives:
      
      for each \( i \) from 1 to \( m \)
      if \( a_i = \text{add, } e \) then \( A_i[h_i(e)] + + \)
      else \( A_i = \text{del, } e \) and \( A_i[h_i(e)] - - \)

High Probability Bounds and Overall Space

What is our new estimate of \( \text{count}_t(e) \)?

- \( \text{best}_t(e) := \min_{i=1}^m A_i[h_i(e)] \).
- Each \( A_i[h_i(e)] \) is an overestimate to \( \text{count}_t(e) \)
- By independence, \( \Pr[\text{for all } i, A_i[h_i(e)] \geq 2|S_t|/k] \leq \left(\frac{1}{2}\right)^m \)
- For \( k = \frac{2}{\epsilon} \) and \( m = \log_2 \left(\frac{1}{\delta}\right) \), the error is at most \( \epsilon|S_t| \) with probability \( 1-\delta \)
- Space: \( m \cdot k = O\left(\frac{\log(\frac{1}{\delta})}{\epsilon}\right) \) counters each of \( O(\log t) \) bits
  
  \( m \cdot O(\log |\Sigma|) = O\left(\frac{\log(\frac{1}{\delta})}{\epsilon} \log |\Sigma|\right) \) bits to store hash functions
**ε-Heavy Hitters**

- Our new estimate $\text{best}_t(e)$ satisfies
  \[ \Pr[|\text{best}_t(e) - \text{count}_t(e)| \leq \epsilon |S_t|] \geq 1 - \delta \]

  and uses $O\left(\frac{\log(t) \log t}{\epsilon} + \log \left(\frac{1}{\delta}\right) \log |\Sigma|\right)$ bits of space.

- What if we want with probability 9/10, simultaneously for all $e$, $|\text{best}_t(e) - \text{count}_t(e)| \leq \epsilon |S_t|$?

- Set $\delta = \frac{1}{10^{|E|}}$ and apply a union bound over all $e \in \Sigma$. 