CS5112: Algorithms and Data Structures for Applications

Lecture 13: Streaming algorithms

Ramin Zabih

Some figures from: Wikipedia/Google image search;
Administrivia

- Reminder: HW comments and minor corrections on Slack
  - Important announcements via email (best efforts)
- Q6 out tonight
- HW3 (and HW2!) delayed due to grading software issues
- Anonymous survey coming re: speed of course, etc.
Today

• Comments about the prelim!
• Six approximate algorithms for histograms and NN
Prelim comments

- NOTHING TODAY IS AUTHORITATIVE (YET)
- Closed book, multiple choice and short answer
- I will give example questions throughout today’s lecture
Online histogram approximations

• Many AI/ML applications hinge on understanding the distribution of your input data
  – Classification is just one example
• Classically we assume that all the input data is available
  – You can run an offline algorithm over it
  – Then do NN classification, density estimation, etc.
• This assumption is often wrong: data comes streaming in
  – And you cannot afford to store the entire data set
Some natural histogram queries

Top-k most frequent elements

What is the frequency of element 3?

What is the total frequency of elements between 8 and 14?

Find all elements with frequency > 0.1%

How many elements have non-zero frequency?
Why approximation?

• Suppose you want an (exact) histogram of your data

• This requires space linear in the number of data points
  – Which is intractable for many internet applications!
    • Examples: IP addresses for DDOS detection; most popular page/item to buy

• So instead we will use a small amount of space but solve the problem approximately
  – But, not precisely the same problem

• Instead of computing the histogram we will look at several key properties of a histogram we can efficiently approximate
  – Typically with constant or logarithmic space and time
Relevant quantities to compute

- **Majority**: if there is a single item comprising more than half the input stream, find it

- **Frequent items**: find all items that comprise more than a given percent $\phi$ of the input stream
  - Approximate version: find all items that comprise between $\phi - \epsilon$ and $\phi$ percent of the input stream (exact when $\epsilon = 0$)
  - Recent version: find and update the most recent popular items

- **Distinct items**: how many different items are there?
1. Boyer-Moore majority algorithm
2. Misra-Gries

- Generalization of Boyer-Moore majority algorithm
- Store $k - 1$ counters, for a parameter $k$
  - Larger $k$ means more space and accuracy
- Any item that appears more than $\frac{n}{k}$ times in the input stream of size $n$ will be present when the algorithm terminates
- If $k = 1/\epsilon$ then each count is at most $\epsilon n$ below its true value
Misra-Gries algorithm in action

**Figure 2.** Counter-based data structure: the blue (top) item is already stored, so its count is incremented when it is seen. The green (middle) item takes up an unused counter, then a second occurrence increments it.

**Algorithm 1:** \texttt{Frequent}(k)

\begin{align*}
n &\leftarrow 0; \\
T &\leftarrow \emptyset; \\
\textbf{foreach} \ i \ \textbf{do} & \\
& n \leftarrow n + 1; \\
& \textbf{if} \ i \in T \textbf{ then} \\
& \quad c_i \leftarrow c_i + 1; \\
& \textbf{else if} \ |T| < k-1 \textbf{ then} \\
& \quad T \leftarrow T \cup \{i\}; \\
& \quad c_i \leftarrow 1; \\
& \textbf{else for all} \ j \in T \textbf{ do} \\
& \quad c_j \leftarrow c_j - 1; \\
& \quad \textbf{if} \ c_j = 0 \textbf{ then} \ T \leftarrow T \setminus \{j\};
\end{align*}
3. Find popular recent items

• Want to be able to naturally update this over time
  – Think of popular: movies, shopping items, web pages, etc.

• We could run, e.g., Misra-Gries on a sliding window
  – This is both impractical and wrong

• Wrong because the importance of an item should not “fall off a cliff” when it moves outside of our window
Weighted average in a sliding window

• Computing the average of the last $k$ inputs can be viewed as a dot product with a constant vector $\mathbf{v} = \left[ \frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k} \right]$

• Sometimes called a box filter
  – Easy to visualize

• This is also a natural way to smooth, e.g., a histogram
  – To average together adjacent bins, $\mathbf{v} = \left[ \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right]$

• This kind of weighted average has a famous name
Decaying windows

- Let our input at time $t$ be $\{a_1, a_2, \ldots, a_t\}$
- With a box filter over all of these elements we computed
  \[
  \frac{1}{t} \sum_{i=0}^{t-1} a_{t-i}
  \]
- Instead let us pick a small constant $c$ and compute
  \[
  \hat{a}_t = \sum_{i=0}^{t-1} a_{t-i} (1 - c)^i
  \]
Easy to update this

- Update rule is simple, let the current dot product be $\hat{a}_t$
  \[
  \hat{a}_{t+1} = (1 - c)\hat{a}_t + a_{t+1}
  \]
- This downscales the previous elements correctly, and the new element is scaled by $(1 - c)^0 = 1$
- This avoids falling off the edge
- Gives us an easy way to find popular items
4. Popular items with decaying windows

• We keep a small number of weighted sum counters
• When a new item arrives for which we already have a counter, we update it using decaying windows, and update all counters
• How do we avoid getting an unbounded number of counters?
• We set a threshold, say $\frac{1}{2}$, and if any counter goes below that value we throw it away
• The number of counters is bounded by $\frac{2}{c}$
How many distinct items are there?

• This tells you the size of the histogram, among other things
• To solve this problem exactly requires space that is linear in the size of the input stream
  – Impractical for many applications
• Instead we will compute an efficient estimate via hashing