CS 261: Data Structures

Week 4: Streaming and sketching

Lecture 4a: Medians and sampling

David Eppstein
University of California, Irvine

Spring Quarter, 2023

Streaming and sketching

The main idea

Sometimes data is too big to fit into memory...

Sketching

Represent data by a structure of sublinear size

Preferably $O(1)$; at most $n^{1-\varepsilon}$ for some $\varepsilon > 0$

Keep enough information to solve what you want either exactly or approximately

Streaming

Read through your data once, in some given order

Use a sketch to represent information about it

Produce a result from the sketch

Example: Sketch for the average

Sketch for the average of a collection of numbers: pair (n, t)

n is the number of elements in the collection
t is the sum of the elements

Operations:

- Incorporate new element x into collection:
  Add one to n and add x to t
- Remove element from collection:
  Subtract one from n and subtract x from t
- Compute average of collection:
  Return t/n

Total space is $O(1)$ regardless of how big n is

(Time/operation is also $O(1)$ but our main concern is space)

Example: Streaming average

To compute the average of a sequence S:

- Initialize an empty sketch for the average
- For each x in the sequence, incorporate x into the sketch
- Return the average of the the sketch

Time for an n-element sequence is $O(n)$
Working space (not counting the space for the input) is $O(1)$

This works even when the sequence is generated somehow in a way that does not involve local storage (e.g. internet traffic)

Cash register versus turnstile

Two different variations of streaming algorithms:

Cash register: Money goes in but doesn't come back out

Meaning for streaming:

Input sequence has only insertions, no removals

Turnstile: Counts people passing in either direction

Meaning for streaming:

Input sequence can have both insertions and removals

Streaming average was presented in cash register model but the same sketch also works for turnstile model

Images: File:Credit card terminal in Laos.jpg and File:Warszawa - Metro - Świętokrzyska (17009062241).jpg from Wikimedia commons

Beyond streaming

Sketches can sometimes be combined to produce information about multiple data sets

Example: Can produce an average-sketch for the concatenation of two sequences by adding the sketches of the two sequences

If we have n sequences, of total length N, we can compute all pairwise averages in time $O(n^2 + N)$, faster than computing each pairwise average separately

Medians and reservoir sampling

Medians

Median: Sort given list of n items; which one goes to position $n/2$?

Like average, an important measure of the central tendency of a sample, more robust against arbitrarily-corrupted outliers

Applications in facility location: Given n points on a line, place a center minimizing the average distance to the given points

Impossibility of streaming median

No streaming algorithm can compute the median!

Consider sketch after seeing the first $n/2$ items in a stream

Depending on the remaining items, any one of these first $n/2$ could become the median of the whole stream

So we have to remember all their values

Space is $\Omega(n)$, not $O(n^{1-\varepsilon})$ for some $\varepsilon > 0$

Approximate medians

We want the item at position $\frac{1}{2}n$ in sorted sequence

Relaxation: δ-approximate median

Position should be between $(\frac{1}{2} - \delta)n$ and $(\frac{1}{2} + \delta)n$

Easy randomized method: Choose a sample of $\Theta(\delta^{-2})$ sequence members and return median of sample

Chernoff $\Rightarrow$ constant probability of being in desired position

Bigger constant in sample size $\Rightarrow$ better probability

(There also exist non-random solutions but more complicated and with non-constant space complexity)

Reservoir sampling

How to maintain a sample of k random elements (without replacement) from a longer sequence?

Store an array A of size k, and the number n
For each element x:
    If $n < k$:
        Store x in $A[n]$
    Else:
        Choose a random integer i from 0 to $n - 1$
        If $i < k$, store x in $A[i]$
    Increment n

MinHash

Hashing for reservoir sampling

We saw how to maintain a sample of k elements, using a random choice for each element

Instead, make a single random choice, of a hash function $h(x)$

Sample = the k elements with the smallest values of h

Two advantages:

- Less use of (possibly slow) random number generation
- Repeatable: if different agents use the same hash function to sample the same sets, they will get the same samples

We can take advantage of repeatability to use this for more applications!

Broder, "On the resemblance and containment of documents", 1997

Combining sets with MinHash

To compute the union:

$$\text{MinHash}(S \cup T) = \text{MinHash}\big(\text{MinHash}(S) \cup \text{MinHash}(T)\big)$$

Because any element that is one of the k smallest hashes in the union must also be one of the k smallest in either of the smaller sets that contain it

We can get useful information about sets S and T from this computation!

Key question: how many elements from $\text{MinHash}(S) \cup \text{MinHash}(T)$ survive into $\text{MinHash}(S \cup T)$?

Original motivation for MinHash

Before the success of Google, a different company (DEC) ran a different popular search engine (AltaVista)

They had a problem: Many web pages can be found in multiple similar (but not always identical) copies

Search engines should return only one copy of the same page, so user can see other results without getting swamped by duplicates

$\Rightarrow$ Need to quickly test similarity of web pages

From documents to sets

"Bag of words" model: represent any web page by the set of words used in it

Similar documents (web pages) should have similar words

So after converting web pages to bags of words, the search engine problem becomes: quickly test similarity of pairs of sets

Jaccard similarity

A standard measure of how similar two sets are:

$$J(S, T) = \frac{|S \cap T|}{|S \cup T|}$$

Scale-invariant (normalized for how big the two sets are)

Ranges from 0 to 1

- 0 when sets are disjoint
- close to 0 when sets are not very similar
- close to 1 when sets are similar
- 1 when sets are identical

Estimating Jaccard similarity

Theorem:

$$J(S,T) = \frac{1}{k} E\Big[\big|\text{MinHash}(S \cup T) \cap \text{MinHash}(S) \cap \text{MinHash}(T)\big|\Big]$$

The intersection consists of the elements of $S \cap T$ that also belong to $\text{MinHash}(S \cup T)$

Linearity of expectation $\Rightarrow$

$$\begin{aligned}
E[\ldots] &= \sum_{x \in S \cup T} \Pr[x \in \text{intersection of MinHashes}] \\
&= \sum_{x \in S \cup T} \Pr[x \in S \cap T] \times \Pr[x \in \text{MinHash}(S \cup T)] \\
&= \sum_{x \in S \cup T} \frac{|S \cap T|}{|S \cup T|} \times \frac{k}{|S \cup T|} \\
&= |S \cup T| \times J(S,T) \times \frac{k}{|S \cup T|} = k\, J(S,T)
\end{aligned}$$

How accurate is it?

Suppose we use

$$\frac{1}{k}\left|\mathrm{MinHash}(S \cup T) \cap \mathrm{MinHash}(S) \cap \mathrm{MinHash}(T)\right|$$

as an estimate for $J(S, T)$

Previous analysis shows that this has the correct expected value: it is an unbiased estimator

Our analysis decomposed the estimate into a sum of independent 0-1 random variables (is each member of $S \cup T$ in the intersection)

$\Rightarrow$ Can use Chernoff bounds for being within $1 \pm \varepsilon$ of expectation

To get expected error $|J(S, T) - \text{estimate}| \leq \varepsilon$, set $k = \Theta(1/\varepsilon^2)$

Summary of MinHash

Main idea: sketch any set by choosing the k elements that have the smallest hash values

Can reduce randomness in reservoir sampling

Sketches of unions can be computed from unions of sketches

Provides accurate estimates of Jaccard similarity

All-pairs similarity of n sets with total length N: time $O(n^2 + N)$