CS261 Winter 2018 - 2019
Lecture 15: Sketching (Part 1)

Instructor: Ashish Goel
Scribe: Kaidi Yan
Edited: Geoff Ramseyer

Feburary 25, 2019

1 The Car Clump Problem: A Mathematical Puzzle

Consider a long road with N cars travelling in the same direction. We label the cars using 1 to N where car 1 is in the front and car N is at the back. The road has only one lane, so no car can overtake another. Let σ be a permutation of 1 to N which indicates the relative speed of these N cars — $\sigma(j)$ is the rank of car j in speed (the slowest car has rank 1).

Because faster cars cannot pass slower cars, the cars end up traveling in clumps, where the car in front of each clump is the slowest car of the clump.

Clearly, car j starts a new clump if it is slower than all cars in front of it. We are interested in knowing the expected number of clumps on this road if σ is a uniformly random permutation. To solve this problem, define the indicator X_j for $1 \leq j \leq N$: $X_j = 1$ if j starts a new clump and $X_j = 0$ otherwise. Hence the expected number of clumps on the road is

$$E[\sum_{j=1}^{N} X_j] = \sum_{j=1}^{N} E[X_j]$$

using linearity of expectations. Note that $X_j = 1$ if and only if $\sigma(j)$ is smaller than $\sigma(1), ..., \sigma(j-1)$. Since σ is a uniformly random permutation, the probability of $\sigma(j)$ being the smallest element of $\{\sigma(1), ..., \sigma(j)\}$ is exactly $1/j$. Hence $E[X_j] = 1/j$. The expected number of clumps, therefore, is $\sum_{j=1}^{N} 1/j = H_N$, which is the N-th harmonic number. Using the integral of $1/x$ to approximate this sum, we can see that $H_N \approx \ln N$, and hence the expected number of clumps is roughly $\ln N$.

2 Uniform Random Sampling

Now consider another problem. Suppose you are given a stream of data $a_1, a_2, ..., a_t, ...$ where $a_i \in \mathbb{R}$.At time t, you are asked to draw a uniform random sample from $\{a_{t-k+1}, a_{t-k+2}, ..., a_t\}$

for some $k \geq 1$ (given as part of the query and not known in advance). How can one perform such a sample efficiently?

One naive approach is to simply store all data seen, and when requested, randomly sample from the last k entries. However, this requires storing the entire stream.

As a better solution, generate a random number X_i for each new stream element a_i. Here, the X_is are i.i.d. random variables following a uniform distribution between 0 and 1 — $X_i \sim Unif[0, 1]$. Whenever we need a unform random sample from $\{a_{t-k+1}, a_{t-k+2}, ..., a_t\}$ we simply return a_j with the smallest X_j $(j = t - k + 1, ..., t)$.

Consider the ordering induced on the last k a_is by their X_is. Because the X_is are uniformly random, a symmetry argument shows that the probability that for any j, X_j is the smallest of these X_is is equal, and hence this algorithm in fact gives a uniformly random sample of the last k entries (although repeated queries will not give i.i.d samples).

What is the space complexity of this algorithm? The key observation is that one need not store all a_is and X_i's — in fact this problem is reduced to the car clump problem, where we only need to store entries which have an X_i smaller than all entries after it (in fact, the ordering of elements in the entire stream induced by the X_is is, given the randomness of the X_is, a uniformly random permutation).

More formally we need to store:

$$\{\langle j, a_j, X_j \rangle : X_j < X_{j'} \text{ for all } j' > j\}$$

Any X_i where there exists $j > i$ with $X_j \leq X_i$ will never be returned as a sample - if a_i is in the sample range, then so is a_j, and $X_j > X_i$.

Based on the analysis in Section 1, the expected number of entries to store at time t is H_t, which is roughly $\ln t$ when t is large. This is a great improvement from the naive approach, which requires storing t entries.

In other words, this system enables answering some, but not all, queries about the stream, for a significant reduction in space complexity. We will call such a system a *sketch* of the data stream.

In the rest of these notes, σ will no longer be a permutation, but will refer to one component of a sketch. For this application, in the context of the rest of the notes, $\sigma(\langle a_1, ..., a_t \rangle) = \{\langle j, a_j, X_j \rangle : X_j < X_{j'} \text{ for all } j' > j\}$.

3 Sketch Definition

A *sketch* is a pair of functions (σ, τ) defined over a set, a sequence or a multi-set such that:

1. $\sigma(S)$ is a function of set/sequence/multi-set S
2. $\tau(\sigma(S_1), \sigma(S_2))$ is the same as:
    (a) $\sigma(S_1 \cup S_2)$ if S_1 and S_2 are sets
    (b) $\sigma(S_1 \uplus S_2)$ if S_1 and S_2 are multi-sets

(c) $\sigma(S_1 \circ S_2)$ if S_1 and S_2 are sequences

In natural language, σ is a "summary" of S and τ is a method for combining independent summaries of S_1 and S_2 in order to obtain the summary of S_1 combined with S_2.

For the uniform random sampling problem discussed above, $S = \langle X_1, ..., X_t \rangle$, and $\sigma(S) = \{\langle j, a_j, X_j \rangle : X_j < X_{j'} \text{ for all } j' > j\}$. We are left with defining τ, which is the function for composing $\sigma(S_1)$ and $\sigma(S_2)$ together in order to get $\sigma(S_1 \circ S_2)$.

In order to compute $\sigma(S_1 \circ S_2)$ based on $\sigma(S_1)$ and $\sigma(S_2)$, notice that $\sigma(S)$ ($S = \langle X_1, ..., X_t \rangle$) always includes $\langle t, a_t, X_t \rangle$; hence we can read the size of S easily from $\sigma(S)$. For each $\langle j, a_j, X_j \rangle$ in $\sigma(S_2)$, increment j by the size of S_1. This gives us a new set of tuples $\sigma'(S_2)$. Next, take the union of $\sigma(S_1)$ and $\sigma'(S_2)$ to get the combined set $\sigma''(S_1 \circ S_2)$. Finally, throw out all tuples from $\sigma''(S_1 \circ S_2)$ that do not satisfy the condition that for all $j' > j$, $X_j < X_{j'}$. Ultimately, this process gives $\sigma(S_1 \circ S_2)$. The time complexity of this entire process is linear in terms of the total size of S_1 and S_2.

In general, a sketch is interesting when:

1. the size of $\sigma(S)$ is much smaller than the size of S
2. σ and τ are easy to compute
3. Some useful query on S can be answered efficiently (perhaps approximately) using just $\sigma(S)$

In the next section, we are going to study the *Count-Distinct* problem, which makes use of an important sketch called *Count-Min* sketch.

4 Count-Distinct Problem & Count-Min Sketch[1]

The Count-Distinct problem states the following: given a stream $S = a_1, a_2, ..., a_t, ...$ that may contain duplicate elements, find or estimate the number of distinct elements present in the stream at time t.[2]

To present an efficient sketch, we first define a consistent uniform [0,1] hash function $h(a)$. Function $h(a)$ returns a random variable uniformly distributed in [0,1] and the random variables $h(a)$ and $h(b)$ are i.i.d if and only if $a \neq b$. When $a = b$, the function is consistent, i.e., $h(a) = h(b)$. We can (approximately) implement[3] this function using the following pseudo-code:

[1]In literature, the count-min sketch is often used to refer to a more complex sketching for counting. However, for the purpose of this class this year, we will use our definition of count-min consistently.

[2]If you are interested in more streaming algorithms, you can read the survey by Muthu Muthukrishnan on `https://www.cs.rutgers.edu/~muthu/streams.html`.

[3]Technically this code won't guarantee that $h(a)$ and $h(b)$ are i.i.d when $a \neq b$ or even that $h(a)$ is uniformly distributed in [0,1] since most languages only provide a pseudo-random generator. But in practice the instructor does not know of any instance where this has been a problem for sketching. The formal understanding of the property of pseudo-random number generators is more relevant for adversarial situations such as cryptography or where you are trying to use a small number of random bits to generate more.

```
function h($a$)
    $srandom(a)$
    return $random()$
end function
```

Now let's assume we are given such a hash function $h(a)$. Define $\sigma(S) = \min_{a \in S} h(a)$. Since $h(a_i)$ are i.i.d and follows a uniform distribution over [0,1], we get:

$$E[\sigma(S)] = \frac{1}{|S|+1} \tag{1}$$

where $|S|$ is the number of distinct elements in S. We also define

$$\tau(S_1, S_2) = \arg\min_{x \in \{S_1, S_2\}} h(x)$$

to finish up the definition of this sketch. If we know $\sigma(S)$, then we can estimate $|S|$ using (1).

To reduce the variance of this estimator, one can make many independent copies of a sketch and "average" the estimates. Define a family of consistent uniform [0,1] hash functions $h_j(a)$, where $1 \leq j \leq M$ for some constant $M > 1$. In particular, now $h_i(a)$ and $h_j(b)$ are i.i.d if and only if either $a \neq b$ or $i \neq j$. Again, we can write (appoximately) $h_j(a)$ using the following pseudo-code:

```
function h($j, a$)
    $srandom(j, a)$
    return $random()$
end function
```

Now define $\sigma_j(S) = \arg\min_{a \in S} h_j(a)$ for $1 \leq j \leq M$, and we have:

$$\sigma(S) = \langle \sigma_1(S), ..., \sigma_M(S) \rangle$$
$$\tau(S_1, S_2) = \langle \arg\min_{x \in \{S_1, S_2\}} h_1(x), ..., \arg\min_{x \in \{S_1, S_2\}} h_M(x) \rangle$$

We will call this the Count-Min sketch. This sketch gives an estimator $|\hat{S}|$ for $|S|$ defined as:

$$|\hat{S}| = \text{Average}(\frac{1}{\sigma_1(S)}, ..., \frac{1}{\sigma_M(S)}) - 1$$

This estimator, however, still has large variance. To solve this problem, we can compute the median instead of the mean:

$$\text{Median}(\frac{1}{\sigma_1(S)} - 1, ..., \frac{1}{\sigma_M(S)} - 1)$$

Intuitively, finding the median is more robust against outliers and hence solves the problem of large variance. In the next section we introduce the Median Lemma, which gives us a quantitative idea of how large M should be in order for the result to be precise.

5 The Median Lemma

Given a distribution D, let G_D be its culmulative density function.

The Median Lemma states the following:

Theorem 1 (The Median Lemma) *There exists a constant $c > 0$ such that for all distributions D where G_D is continuous, for all $\delta \in (0, 1/2)$ and for all $\epsilon > 0$, if $X_1, X_2, ..., X_M$ are i.i.d random variables with distribution D and $M > \frac{c}{\delta^2}\ln(\frac{1}{\epsilon})$, then Median($X_1, ..., X_M$) lies within $[G_D^{-1}(1/2 - \delta), G_D^{-1}(1/2 + \delta)]$ with probability at least $1 - \epsilon$.*

We skip the proof of this lemma. Notice that dependence on $1/\epsilon$ is logarithmic. The key component for determining the size of M is thus δ, which is also related to the confidence interval of our estimate. The smaller δ is, the larger M becomes and the median we find is also more precise.

To clarify G_D^{-1}, if $x = G_D^{-1}(p)$, then, given some value y drawn from distribution D, $y \leq x$ with probability p. $G_D^{-1}(1/2)$, then, is (informally speaking) a generalization of the median of a distribution.

Suppose you have some set of numbers $x_0 < ... < x_n$ (say n is even). Then consider the probability distribution induced by choosing a random value independently at random. Then G_D increases stepwise by $1/n$ at each of the x_is from 0 to 1. Of course, this distribution is not continuous (and G_D^{-1} is technically not well-defined as a function) as required by the Median Lemma. However, we can approximate it arbitrarily closely by a continuous function, and get that $G_D^{-1} = x_{n/2}$ (and indeed, $G_D^{-1}(x) = x_k$ for $x \in [k/n, (k+1)/n)$.)

6 Min-Hash for Jaccard Similarity

Jaccard Similarity is a measure for set similarity, used commonly in statistics and sometimes in natural language analysis. Given two sets S_1 and S_2, the Jaccard Similarity between them, denoted $JS(S_1, S_2)$, is:

$$JS(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

In order to estimate $JS(S_1, S_2)$, we define $\sigma_1(S)$ using the same definition as for the Count-Min sketch. Now consider the probability that $\sigma_1(S_1) = \sigma_1(S_2)$. If we apply h_1 to elements of S_1 and S_2, then we have $\sigma_1(S_1) = \sigma_1(S_2)$ exactly when the element of $S_1 \cup S_2$ with the smallest h_1 hash value appears in the intersection $S_1 \cap S_2$. Hence:

$$\Pr[\sigma_1(S_1) = \sigma_1(S_2)] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = JS(S_1, S_2)$$

Hence if we can estimate $\Pr[\sigma_1(S_1) = \sigma_1(S_2)]$ then we can estimate the Jaccard Similarity between S_1 and S_2. Inspired by what we did for Count-Min sketch, we again define a family of hash functions $h_1, ..., h_M$ and define $\sigma_j(S)$ $(1 \leq j \leq M)$. Then we get an estimate for $JS(S_1, S_2)$:

$$\widehat{JS(S_1, S_2)} = \frac{|\{j : 1 \leq j \leq M, \sigma_j(S_1) = \sigma_j(S_2)\}|}{M}$$

This sketch is called the Min-Hash sketch. Although we are computing the mean for the estimator $\widehat{JS(S_1, S_2)}$, the guarantee given by the median lemma still holds here and hence we can use that to compute a desirable M[4].

[4]The median lemma is particularly useful for analyzing distributions that are not well behaved. For simple distribution like Bernoulli, which is the case here, or squares of Gaussians, as we will see later, we can use the mean as opposed to the median with essentially the same guarantee.