CS5234 - Algorithms at Scale

Liew Zhao Wei

Semester 1, 2023-2024

---

1 Probability and Hashing

1.1 Probability

We start by stating some crucial probability results that will be used throughout the course.

Lemma 1.1 (Union Bound)
For a countable set of events $A_1, A_2, \ldots$, we have

$$\Pr\left[\bigcup_{i=1}^{\infty} A_i\right] \leq \sum_{i=1}^{\infty} \Pr[A_i] \tag{1}$$

Lemma 1.2 (Linearity of Expectation)
For any random variables $X_1, X_2, \ldots, X_n$, we have

$$\mathrm{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{E}[X_i] \tag{2}$$

Lemma 1.3 (Markov's Inequality)
For any *non-negative* random variable X and $t > 0$, we have

$$\Pr[X \geq t] \leq \frac{\mathrm{E}[X]}{t} \tag{3}$$

Lemma 1.4 (Chebyshev's Inequality)
For any random variable X with mean μ and variance σ^2, we have

$$\Pr[|X - \mu| \geq t] \leq \frac{\sigma^2}{t^2} \tag{4}$$

In fact, this holds for any moment p instead of 2.

Lemma 1.5 (Chernoff-Hoeffding Bounds)
Suppose $X_1, \ldots, X_n$ are *independent* random variables with $X_i \in [0,1]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$ such that $\mu_L \leq \mu \leq \mu_H$.

(Chernoff) For any $0 \leq \delta \leq 1$,

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left\{-\frac{\delta^2\mu}{3}\right\} \tag{5}$$

$$\Pr[X \leq (1-\delta)\mu] \leq \exp\left\{-\frac{\delta^2\mu}{2}\right\} \tag{6}$$

$$\Pr\left[|X-\mu| \geq \delta\mu\right] \leq 2\exp\left\{-\frac{\delta^2\mu}{3}\right\} \tag{7}$$

$$\tag{8}$$

(Hoeffding) For any $\delta \geq 0$,

$$\Pr[X \geq \mu + \delta] \leq \exp\left\{-\frac{2\delta^2}{n}\right\} \tag{9}$$

$$\Pr[X \leq \mu - \delta] \leq \exp\left\{-\frac{2\delta^2}{n}\right\} \tag{10}$$

$$\Pr\left[|X-\mu| \geq \delta\right] \leq 2\exp\left\{-\frac{2\delta^2}{n}\right\} \tag{11}$$

More generally, if $a_i \leq X_i \leq b_i$, then

$$\Pr[X \geq \mu + \delta] \leq \exp\left\{-\frac{2\delta^2}{\sum_{i=1}^{n}(b_i-a_i)^2}\right\} \tag{12}$$

$$\Pr[X \leq \mu - \delta] \leq \exp\left\{-\frac{2\delta^2}{\sum_{i=1}^{n}(b_i-a_i)^2}\right\} \tag{13}$$

1.2 Hashing

Definition 1.6 (k-Universal Hash)
A hash family $\mathcal{H} = \{h : \mathcal{U} \to S\}$ is *k-universal* if for any distinct k elements $x_1, \ldots, x_k \in \mathcal{U}$ and any k elements $y_1, \ldots, y_k \in S$, we have

$$\Pr_{h \in_R \mathcal{H}}[h(x_1) = y_1 \wedge \ldots \wedge h(x_k) = y_k] = \frac{1}{|S|^k} \tag{14}$$

Such a hash family is also called a *k-wise independent hash family.*

Lemma 1.7 (Construction of k-Universal Hash Family)
There is a construction of a k-universal hash family from $[n]$ to $[n]$ that takes $O(k \log n)$ space.

2 Simple Techniques

2.1 Reservoir Sampling

We can use *reservoir sampling* to uniformly sample an element from a stream without having to store the stream in advance.

Algorithm 1 Reservoir Sampling algorithm

```
1: procedure ReservoirSampling(stream s)
2:     for i = 1, 2, ... do
3:         Flip a coin with probability 1/i
4:         if coin was heads then
5:             x ← s_i
6:         end if
7:     end for
8:     return x
9: end procedure
```

By a simple proof by induction, we can show that the probability of any element x being sampled is $1/n$.

2.2 Mean Trick to Reduce Variance

We can take the average of multiple independent estimators to reduce the variance of the final estimator. This is useful when the variance of the single basic estimator is too large to apply concetration bounds.

Lemma 2.1 (Reduce Variance by Taking Mean of k Samples)
Let $X = \frac{1}{k}\sum_{i=1}^{k} X_i$ where each X_i has mean μ and variance σ^2. Then, $\mathrm{E}[X] = \mu$ and $\mathrm{Var}[X] = \sigma^2/k$.

2.3 Median Trick to Amplify Success Probability

We can use the median of multiple independent estimators to reduce the probability of the final estimator being far from the true value (typically by Chernoff-Hoeffding bounds).

Algorithm 2 Median Trick for Amplifying Sucess Probability

```
1: procedure AmplifyByMedian(algorithm A, integer ℓ)
2:     for i = 1, ..., ℓ do
3:         Let Z_i be the output of A with new independent random values
4:     end for
5:     return Median(Z_1, ..., Z_ℓ)
6: end procedure
```

This trick can only be used when the success probability of the original estimator is more than 50%. We commonly combine both the Mean and the Median tricks to form the Median of Means trick. The typical space blow-up of such a technique is $O(\epsilon^{-2}\log(1/\delta))$.

A slightly different version of the Mean Trick is the Min Trick, where we take the minimum instead of the median. This is useful when there is only one-sided error – think of the median as a two-sided error version of the min trick. An example of this is the *Count-Min-Sketch* data structure.

3 Sketches

Definition 3.1 (Sketches and Linear Sketches)
A data structure S is a *sketch* if there is a space-efficient combining algorithm COMB such that for any two streams s_1 and s_2, we have $S(s_1 \cdot s_2) = \text{COMB}(S(s_1), S(s_2))$. A sketch is *linear* if $S(s_1 \cdot s_2) = S(s_1) + S(s_2)$.

3.1 Misra-Gries

The *Misra-Gries* algorithm solves the k-heavy hitters problem:

- If x is a k-heavy hitter, then x must be in the output.
- If x is not a k-heavy hitter, then x may or may not be in the output.

Algorithm 3 Misra-Gries heavy hitters algorithm

```
1:  procedure MisraGries(stream s, integer k)
2:      Let C be an empty hash table
3:      for i = 1, ..., m do
4:          if a_i ∈ C then
5:              C[a_i] ← C[a_i] + 1
6:          else if |C| < k − 1 then
7:              C[a_i] ← 1
8:          else
9:              for j ∈ C do
10:                 C[j] ← C[j] − 1
11:                 if C[j] = 0 then
12:                     Delete C[j]
13:                 end if
14:             end for
15:         end if
16:     end for
17:     return Keys(C)
18: end procedure
```

Lemma 3.2 (Misra-Gries Analysis)
The Misra-Gries algorithm with parameter k uses one pass and $O(k(\log m + \log n))$ bits of space and provides, for any token j, an estimate $\hat{f}_j$ satisfying

$$f_j - \frac{m - \hat{m}}{k} \leq \hat{f}_j \leq f_j \tag{15}$$

As expected of a sketch, two Misra-Gries sketches can be combined by adding the counts of the same keys, sorting the counts in descending order to form a new stream of counts, and re-running the algorithm on the combined sketch.

3.2 Count-Min-Sketch

The *Count-Min-Sketch* data structure solves the frequency estimation problem. It takes $O(\epsilon^{-1} \log(1/\delta) \cdot (\log m + \log n))$ space and provides, for any token j, an estimate $\hat{f}_j$ satisfying $f_a \leq \hat{f}_a \leq \epsilon ||f_{-a}||$ with probability at least $1 - \delta$.

Algorithm 4 Count-Min-Sketch frequency estimation algorithm

Require: $C[1 \dots t][1 \dots k] \leftarrow \vec{0}$, where $k = 2/\epsilon$ and $t = \lceil \log_2(1/\delta) \rceil$
Require: Choose t independent hash functions $h_1, \dots, h_t \colon [n] \to [k]$, each from a 2-universal family

```
procedure PROCESS(token (j, c))
    for each token (j, c) in s do
        for i = 1, ..., t do
            C[i][h_i(j)] ← C[i][h_i(j)] + c
        end for
    end for
end procedure
procedure OUTPUT(query a)
    return f̂_a = min_{i=1,...,t} C[i][h_i(a)]
end procedure
```

4 Dimensions and Distances

Lemma 4.1 (Johnson-Lindenstrauss Lemma)
For any set $S \subseteq \mathbb{R}^d$ of n-points, there is an embedding $f \colon \mathbb{R}^d \to \mathbb{R}^m$ for $m = O(\epsilon^{-2} \log n)$ such that

$$\forall u, v \in S \quad (1-\epsilon)\|u-v\|_2^2 \leq \|f(u)-f(v)\|_2^2 \leq (1+\epsilon)\|u-v\|_2^2 \tag{16}$$

In other words, we can embed S into a lower-dimensional space while approximately preserving ℓ_2 norms.
Some observations:

- The embedding has only a logarithmic dependence on n and *no* dependence on d.
- The embedding is can be generated using a Gaussian distribution.
- The embedding can be represented as a linear transformation, or in other words, a matrix.

Definition 4.2 (Locality Sensitive Hash)
A hash family $\mathcal{H} = \{h \colon \mathcal{U} \to S\}$ is a (r_1, r_2, p_1, p_2)-locally sensitive if for all points $p, p' \in \mathcal{U}$,

1. if $d(p, p') \leq r_1$, then $\Pr_{h \in \mathcal{H}}[h(p) = h(p')] \geq p_1$,
2. if $d(p, p') > r_2$, then $\Pr_{h \in \mathcal{H}}[h(p) = h(p')] \leq p_2$.

In other words, a *locality sensitive hash* (LSH) is a hash family where similar items are more likely to collide. Note that the definition makes sense only if $r_1 < r_2$ and $p_1 > p_2$.

4.1 ϵ-Approximate Nearest Neighbour Problem

Definition 4.3 (ϵ-Approximate Nearest Neighbour Problem)
Given a set P of points in $\mathbb{R}^d$, construct a data structure, that given any point $q \in \mathbb{R}^d$, returns a point $p \in P$ such that $d(p, q) \leq c \min_{p' \in P} d(p', q)$.

We can reduce this problem to a simpler one called the ϵ-Point Location in Equal Balls problem (ϵ-PLEB).

Definition 4.4 (ϵ-Point Location in Equal Balls)
Given a set P of points in $\mathbb{R}^d$ and radii r_1, r_2, construct a data structure, that given any point $q \in \mathbb{R}^d$:

1. If there is a point $p \in P$ such that $d(p, q) \leq r_1$, returns **YES** and any point $p' \in P$ with $d(p', q) \leq r_2$.
2. If there is no point $p \in P$ such that $d(p, q) \leq r_2$, returns **NO**.

Lemma 4.5 (ANN to PLEB)
Suppose there is a data structure for $(r, (1+\epsilon)r) - PLEB$ with space S and query time T for any radius r. Then, there is an algorithm for $(1+\epsilon)^2$-ANN with space $O(S \log_{1+\epsilon} \frac{D_{max}}{D_{min}})$ and query time $O(T \log\log_{1+\epsilon} \frac{D_{max}}{D_{min}})$, where D_{max} and D_{min} denote the maximum and minimum distance between points respectively.

Proof. Search over the radii using $(r, (1+\epsilon)r)$-PLEB:

$$\frac{D_{min}}{2}, (1+\epsilon)\frac{D_{min}}{2}, (1+\epsilon)^2\frac{D_{min}}{2}, \ldots, \approx D_{max} \tag{17}$$

and binary search for the minimum radius r so that the algorithm returns **YES** for a single query point p. Convince yourself that the point returned is a $(1+\epsilon)^2$-approximate nearest neighbour to p. □

In fact, there is a more efficient reduction to reduce $(1+\epsilon)$-ANN to $(r, (1+\epsilon)r)$-PLEB.
Finally, we can reduce $(r, (1+\epsilon)r)$-PLEB to $(r, (1+\epsilon)r)$-LSH.

Lemma 4.6 (PLEB to LSB)
Suppose there is a (r_1, r_2, p_1, p_2)-LSH $\mathcal{H} = \{h \colon U \to S\}$. Then, there is an algorithm for (r_1, r_2)-PLEB which uses

- $O(dn + n^{1+\rho})$ space, and
- $O(n^\rho)$ query time, measured in hash evaluations,

where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$. This algorithm succeeds with constant probability.

Proof. The algorithm is as follows: Let k and ℓ be parameters (which we will set later). Define a new hash family $\mathcal{G} = \{g : U \to S^k\}$. Each hash function $g \in \mathcal{G}$ takes the form $g(p) = (h_1(p), h_2(p), \ldots, h_k(p))$, where each hash function h_i is in $\mathcal{H}$.

First, preprocessing:

1. Choose hash functions $g_1, \ldots, g_\ell$ from $\mathcal{G}$.
2. For each p in the given set of points P, store p in each of the buckets specified by $g_1(p), \ldots, g_\ell(p)$.
3. Discard all empty buckets.

Now, when queried with the point q, we search through the buckets $g_1(q), \ldots, g_\ell(q)$, and stop after the first 2ℓ points. If any of these points p has $d(p, q) \le r_2$, return p with **YES**; otherwise, return **NO**.

Convince yourself that this algorithm achieves the desired guarantees. □