① Shuffling (= uniformly random permutation)

Given a list of n elements, randomly permute

Specifically, want each possible permutation of $[n] = (1, 2, \dots, n)$ to have an equal chance, i.e., $1/n!$ chance.

Naively: draw n iid samples from any absolutely cts. probability distribution (eg. Gaussian) then sort these.

Downside: a sort costs $O(n \cdot \log n)$ flops.

Can we get a linear time algo.?

Yes! "Knuth-shuffle" aka "Fisher Yates shuffle"

Input $x = (x_1, x_2, \dots, x_n)$

For $i = 1, 2, \dots, n-1$

$j \in \{i, i+1, \dots, n\}$ chosen uniformly at random

exchange x_i and x_j

time to compute this is independent of i

<u>Thm</u>: this is a uniform random permutation, i.e., $P(\text{any permutation on } [n]) = \frac{1}{n!}$

<u>Proof</u> via induction, aka "loop invariant" in CS terms

<u>def</u> a "k-permutation" on $[n]$ is a list of k elements, each element from $[n]$ and no repetitions.

i.e., the 1st k elements of any permutation

There are $\frac{n!}{(n-k)!}$ of these ($n!$ total permutations, only count those whose 1st k elements differ)

so... we want a n-permutation.

<u>claim</u>: after iteration k, all k-permutations have an equal chance of equalling $\underbrace{x(1:k)}_{\text{Matlab notation}}$,
i.e., $\frac{(n-k)!}{n!}$ chance, since there are $n!/(n-k)!$ k-perms.

<u>proof-of-claim</u>:
iter 1: there are n 1-permutations. By construction, all are equally likely.
iter $k+1$: (induction step)
Let $(x_{\sigma_1}, x_{\sigma_2}, \ldots, x_{\sigma_k}, x_{\sigma_{k+1}})$ be any $k+1$ permutation.

$$P\left(x(1:k+1) = (x_{\sigma_1}, \ldots, x_{\sigma_k}, x_{\sigma_{k+1}})\right)$$

$$= P\left(x(k+1) = x_{\sigma_{k+1}} \mid x(1:k) = (x_{\sigma_1}, \ldots, x_{\sigma_k})\right) \cdot P\left(x(1:k) = (x_{\sigma_1}, \ldots, x_{\sigma_k})\right) = \frac{(n-k)!}{n!} \text{ via induct.}$$

$x_{\sigma_{k+1}}$ is some entry not already selected

In algo, j is chosen uniformly at random among $\underbrace{\{k+1, \ldots, n\}}^{n-k}$
$n-k$ entries that have not already been selected.
So, this is $\frac{1}{n-k}$

$$\ldots = \frac{1}{n-k} \cdot \frac{(n-k)!}{n!} = \frac{(n-(k+1))!}{n!}$$

So, after $k=n$ iterations, $\frac{0!}{n!} = \frac{1}{n!}$ chance of any given permutation,
i.e., this is a shuffle. $\square$

cf. wikipedia, §4.5.5 Rajaraman + Ullman 2nd ed.
or §2.7 Cormode et al. "Synopses..." 2011

(2) Reservoir Sampling to form a Simple Random Sample (SRS)

"randsample(n, k)" in Matlab

i.e. combination not permutation

not ordered list

Def A "SRS" of k elements (from n possible) is a subset of size k from $[n]$ such that all such subsets have equal chance, namely $1/\binom{n}{k} = \frac{k!(n-k)!}{n!}$

"overkill" since these also shuffle

Example:
- shuffle the data and take 1st k entries (inefficient)
- do the 1st k steps of Knuth / Fisher-Yates
- a SRS followed by a k-element shuffle is a k-perm.

Vitter '85 but also '60's

Reservoir Sampling is for the situation where
1) we want only 1-pass over the data, eg a data **stream**, aka streaming.

2a) n is unknown! ← classical

or 2b) other things unknown! ← applicable to our class

Start here:

2a) First, observe Fisher-Yates looped $i = 1, 2, \ldots, n-1$
$j \in \{i, \ldots, n\}$

but we can do an "inside-out" version

- Input $x \in \mathbb{R}^n$, output $y \in \mathbb{R}^n$ (shuffled), no longer in-place
- For $i = 1, 2, \ldots, n-1, n$ ← different!

  $j \in \{1, 2, \ldots, i\}$ uniformly at random
  (not $\{i, i+1, \ldots, n\}$ any more)

  if $j \neq i$,
  $y_i \leftarrow y_j$
  $y_j \leftarrow x_i$

  if we initialized $y = x$, then this is like swap(y_i, y_j) ... and running outer loops **backward**!

(we won't show it here, but similar inductive proof shows this is correct) i.e., reverse input, then flip "for" loop.

Observe that since $j \in \{1, \dots, i\}$ not $\{i, \dots, n\}$, we don't need to know n.

This is essentially the basic reservoir sampling algo, "Algo R" (Alan Waterman)

Input: $x \in \mathbb{R}^n$, n "unknown" or "we won't know it 'til we see it"

Output: $y \in \mathbb{R}^k$, y is the "reservoir"

initialize $y = x(1:k)$

For $i = k+1, \dots, n$ — only place n appears

$j \in \{1, 2, \dots, i\}$ uniformly at random

if $j \le k$

$y_j = x_i$

} if $j > k$, we'd be updating y past it's last entry. We don't care, so skip it.

- one-pass ✓
- independent of n ✓
- Running time: $O(n)$ OK, though "Algo L" is $O(k + k\log(n/k))$ w/ tricks.
- k-perm? No, not shuffled (easy to see if $n=k$)
- SRS? <u>yes</u>.

2b) Let's keep k of n items (order still unimportant) such that x_i is kept with probability proportional to its weight, $w_i = w(x_i) \geq 0$,

i.e., $w_i / W = \sum_{j=1}^{n} w_j$.

New complication: if n is unknown, so is W.

Ex: sample a row of a matrix $X(i,:)$ proportional to it's ℓ_2-norm squared. Then $W = \|X\|_F^2$. If we stream rows, W keeps increasing.

Note: not all weights realizable,

ex. $k=2$, $n=3$, $\underbrace{\text{wts} = [1,0,0]}_{\text{incompatible w/ } k=2}$

or $k=n$, must have wts = uniform

Algo A-Chao (cf. Wikipedia, or M. Chao '82) → link on wikipedia
or P. Efraimidis '15

Input: $x \in \mathbb{R}^n$

Output: $y \in \mathbb{R}^k$

initialize $y = x(1:k)$

$W = \sum_{i=1}^{k} w(y_i)$

For $i = k+1, k+2, \ldots, n$

$W \mathrel{+}= w(x_i)$

$p = \dfrac{k \cdot w(x_i)}{W}$

with prob. p, keep this sample x_i by assigning it to y_j with $j \in [k]$ chosen $\underbrace{\text{uniformly at random}}_{\text{counter-intuitive.}}$

else, do nothing.

Not obvious!

(3) Different types of sampling (about) k elements from $[n]$

(1) SRS (w/o replacement): choose subset $\Omega \subseteq [n]$ of size k such that all subsets equally likely: "uniform"

let Ω_k be such a set

pros: nice conceptually, get right size, no duplicates

cons: no longer independent (but exchangeable)

uniform in diff't ways

(2) SRS w/ replacement:
choose a list $y \in \mathbb{R}^k$ s.t. each $y_i \sim \text{Unif}([n])$ i.i.d.

pros: independent, get right size

cons: may contain duplicates which seems like a waste

(3) "Bernoulli": keep each x_i w/ probability $k/n =: p$
Expect to keep k entries total

pros: independent, no duplicates

cons: the size of our sample is only k on average, not deterministically

Relations

We'll be looking at $P(\text{Failure}(\Omega))$ of some randomized algo. that uses samples Ω

Formalize:

Prop (3.1 in Recht "A Simpler Approach...")

$P(\text{failure via (1)}) \leq P(\text{failure via (2)})$ if $P(\text{failure}(\Omega_k)) \geq P(\text{failure}(\Omega_{k'}))$ whenever $k \leq k'$ (more is better)

Proof Let Ω' be sampled via (2) w/ k entries.

i.e., $\Omega' = \text{unique}(y)$

Ω': set of length $\leq k$; y: list of length k

then $P(\text{failure}(\Omega')) = \sum_{i=0}^{k} \underbrace{P(\text{fail}(\Omega') \mid |\Omega'| = i)}_{\underbrace{P(\text{fail}(\Omega_i))}_{\geq P(\text{fail}(\Omega_k)) \text{ since } i \leq k}} \cdot P(|\Omega'| = i)$

is main observation

$$\geq P(\text{fail}(\Omega_k)) \cdot \underbrace{\sum_{i=0}^{k} P(|\Omega'| = i)}_{=1}$$

$$\geq P(\text{fail}(\Omega_k)). \quad \square$$

<u>Prop</u> (p.15 "Robust Uncertainty Principles..." Candès et al '05)

Let Ω_k be sampled via ① and let

Ω' be sampled via ③ w/ parameter $p = k/n$ so $\mathbb{E}\,|\Omega'| = k$

then if $P(\text{failure}(\Omega_k))$ is non-increasing in k (same as in prev. prop.),

$$P(\text{failure}(\Omega_k)) \leq 2 \cdot P(\text{failure}(\Omega')).$$

[see Appendix of "Robust Principal Component Analysis" Candès et al. '09 for more refined 2-way result]

<u>proof</u>

$$P(\text{failure}(\Omega')) = \sum_{i=0}^{n} P(\text{fail}(\Omega') \mid |\Omega'| = i) \cdot P(|\Omega'| = i)$$

new!

(same as before, but sum goes to n, not k)

$$\geq \sum_{i=0}^{k} P(\ldots\ldots \mid \ldots) \cdot P(\ldots)$$

since non-negative

$$\geq \underbrace{P(\text{fail}(\Omega_k))}_{\text{as before}} \cdot \underbrace{\sum_{i=0}^{k} P(|\Omega'| = i)}_{\leq 1 \text{ now.}}$$

$p \cdot n = k \in \mathbb{Z}$ is the <u>median</u> of $|\Omega'|$ so $P(|\Omega'| \leq k-1) < \frac{1}{2} < P(|\Omega'| \leq k)$ cf. Jogdeo + Samuels '68

... so $\sum_{i=0}^{k} P(|\Omega'| = i) = P(|\Omega'| \leq k) > \frac{1}{2}$

so ... $> \frac{1}{2} P(\text{fail}(\Omega_k))$. $\square$