Generating a random sample

Introduction to Discrete Probability

Ian Chen

Samples

- Why do we want a random sample?
- What is a random sample?
- How do we generate a random sample?
- What is a random sample?
  - Should be *smaller* than the dataset
  - Each *item* should have equal chances of being in the sample
  - Each *pair of items* should have equal chances
  - ...

Streams

- A stream is a sequence of items
  - You don't know how many items you have
  - You can only read each item once, in order
- We want to generate a sample from a stream

Lots

```
DrawLots(array L):
    n ← |L|
    for i from 1 → n
        remove a random lot x from L
        R[i] ← x
    return R[1...n]
```

- Correctness?
  - There are n equally likely choices for $R[1]$
  - There are $n-1$ equally likely choics for $R[2]$
  - ... There are $n!$ equally likely results for R

Fisher-Yates

```
InsertionShuffle(stream S):
    Initialize A as empty array
    while S is not empty
        Append next item in S to A
        swap A[|A|] ↔ A[Random(|A|)]
```

- Correctness?
  - There are 1 equally likely choices for Random(1)
  - There are 2 equally likely choices for Random(2)
  - ... There are $n!$ equally likely results for the algorithm

Permutations to samples

- Interpret the *key* of an item as the position in the permutation
- The sample is created from the k smallest keys
- Normalize keys between $[0, 1]$

```
k-Min-Sample(stream S, int k):
    Initialize R ← ∅ for output
    while S is not done
        x ← next item in S
        x.key ← random number between [0, 1]
        Update R to contain the k smallest keys
    return R
```

- Correctness?
  - Same as discretized version!

Making it faster

- Most of the time we do not update the reservoir
- Could we only generate random numbers when we need to update the reservoir?
- We need to know how many items to skip between random number calls
- What is the *probability* of skipping 0 items? 1? j?
- Geometric! $\mathbf{Pr}(\text{skip } j \text{ items}) = p(1-p)^j$

Reservoir Sampling I

- Generate numbers according to the distribution of skipping j items

```
RESERVOIR-SAMPLE(stream S, int k):
  Initialize R ← ∅, p ← 1
  while S is not done
      x ← next item in S
      x.key ← random key between [0, M]
      Update R to contain the k smallest keys
      Update p to be largest key in R
      Create j with distribution p(1 − p)^j and skip next j items
  return R
```

- What is the *expected* number of random numbers generated?
  - Let n be number of elements in stream
  - Let I_i denote whether we generate a random number for element i

Reservoir Sampling II

- N is the total number of generated numbers, to find $\mathbf{Ex}(N)$,

$$\begin{aligned}\mathbf{Ex}(N) &= \mathbf{Ex}(\sum_{i=1}^{n} I_i) = \sum_{i=1}^{n} \mathbf{Ex}(I_i) \\ &= k + \sum_{i=k+1}^{n} \frac{k}{i} \\ &< k(1 + \ln(n/k)) + 1\end{aligned}$$

- Much better than n

Introduction

Permutations

Samples

What's next?

Conclusions I

- Generating geometric random variables
  - If $X \sim \text{Geom}(p)$, $U \sim \text{Uniform}(0, 1)$, then

$$\mathbf{Pr}(X < x) = \mathbf{Pr}(\left\lfloor \frac{\ln(U)}{\ln(1-p)} \right\rfloor < x)$$

- Weighted sampling
  - If $X_i \overset{indep}{\sim} \text{Exp}(\lambda_i)$, then for $i \neq j$,

$$\mathbf{Pr}(X_i < X_j) = \frac{\lambda_i}{\lambda_i + \lambda_j}$$

  - Generate keys according to exponential distribution
  - The probability of being in the sample is proportional to the weight

Conclusions II

- Getting rid of the heap
  - The position of the largest key is distributed uniformly in the array
  - Pick an element at random as the max!
  - Simulates the heap
- Sketching, or making faster algorithms with random samples
  - Median selection
  - Cardinality estimation
  - Linear queries