Counting Distinct Elements

Applications
Flajolet-Martin Approximation Technique
Generalization to Moments

Counting Distinct Elements

- Problem: a data stream consists of elements chosen from a set of size *n*. Maintain a count of the number of distinct elements seen so far.
- Obvious approach: maintain the set of elements seen.

Applications

- How many different words are found among the Web pages being crawled at a site?
  - Unusually low or high numbers could indicate artificial pages (spam?).
- How many unique users visited Facebook this month?
- How many different pages link to each of the pages we have crawled.

Estimating Counts

- Real Problem: what if we do not have space to store the complete set?
- Estimate the count in an unbiased way.
- Accept that the count may be in error, but limit the probability that the error is large.

Flajolet-Martin Approach

- Pick a hash function *h* that maps each of the *n* elements to at least $\log_2 n$ bits.
- For each stream element *a*, let $r(a)$ be the number of trailing 0's in $h(a)$.
- Record R = the maximum $r(a)$ seen.
- Estimate = 2^R.

Why It Works

- The probability that a given $h(a)$ ends in at least i 0's is 2^{-i}.
- If there are m different elements, the probability that $R \geq i$ is $1 - \boxed{(\boxed{1 - 2^{-i}})^m}$.

Prob. all h(a)'s end in fewer than i 0's.

Prob. a given h(a) ends in fewer than i 0's.

Why It Works – (2)

- Since 2^{-i} is small, $1 - (1-2^{-i})^m \approx 1 - e^{-m2^{-i}}$.
- If $2^i >> m$, $1 - e^{-m2^{-i}} \approx 1 - (1 - m2^{-i}) \approx m/2^i \approx 0$.
- If $2^i << m$, $1 - e^{-m2^{-i}} \approx 1$.
- Thus, 2^R will almost always be around m.

First 2 terms of the
Taylor expansion of e^x

Why It Doesn't Work

- E(2^R) is, in principle, infinite.
  - Probability halves when R -> R+1, but value doubles.
- Workaround involves using many hash functions and getting many samples.
- How are samples combined?
  - Average? What if one very large value?
  - Median? All values are a power of 2.

Solution

- Partition your samples into small groups.
  - Log n, where n = size of universal set, suffices.
- Take the average of groups.
- Then take the median of the averages.

Generalization: Moments

- Suppose a stream has elements chosen from a set of n values.
- Let m_i be the number of times value i occurs.
- The k^{th} *moment* is the sum of $(m_i)^k$ over all i.

Special Cases

- 0^{th} moment = number of different elements in the stream.
  - The problem just considered.
- 1^{st} moment = count of the numbers of elements = length of the stream.
  - Easy to compute.
- 2^{nd} moment = *surprise number* = a measure of how uneven the distribution is.

Example: Surprise Number

- Stream of length 100; 11 values appear.
- Unsurprising: 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9. Surprise # = 910.
- Surprising: 90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1. Surprise # = 8,110.

AMS Method

- Works for all moments; gives an unbiased estimate.
- We'll just concentrate on 2nd moment.
- Based on calculation of many random variables *X*.
  - Each requires a count in main memory, so number is limited.

One Random Variable

- Assume stream has length *n*.
- Pick a random time to start, so that any time is equally likely.
- Let the chosen time have element *a* in the stream.
- $X = n$ * ((twice the number of *a*'s in the stream starting at the chosen time) – 1).
  - Note: store *n* once, count of *a*'s for each *X*.

Expected Value of X

- 2^{nd} moment is $\Sigma_a (m_a)^2$.
- $E(X) = (1/n)\left(\Sigma_{\text{all times } t}\ n * \text{(twice the number of times the stream element at time } t \text{ appears from that time on)} - 1\right)$.
- $= \Sigma_a\ (1/n)(n)(1+3+5+\ldots+2m_a-1)$.
- $= \Sigma_a\ (m_a)^2$.

Group times by the value seen

Time when the last a is seen

Time when penultimate a is seen

Time when the first a is seen

Problem: Streams Never End

- We assumed there was a number *n*, the number of positions in the stream.
- But real streams go on forever, so *n* changes; it is the number of inputs seen so far.

Fixups

1. The variables *X* have *n* as a factor – keep *n* separately; just hold the count in *X*.
2. Suppose we can only store *k* counts. We must throw some *X*'s out as time goes on.
   - Objective: each starting time *t* is selected with probability *k*/*n*.

Solution to (2)

- Choose the first *k* times for *k* variables.
- When the n^{th} element arrives ($n > k$), choose it with probability k/n.
- If you choose it, throw one of the previously stored variables out, with equal probability.
- Probability of each of the first n-1 positions being chosen:

$$(n-k)/n * k/(n-1) + k/n * k/(n-1) * (k-1)/k = k/n$$

n-th position not chosen

Previously chosen

n-th position chosen

Previously chosen

Survives