Mining Data Streams (Part 3)

A. Farzindar
farzinda@usc.edu

Mining of Massive Datasets
Jure Leskovec, Anand Rajaraman, Jeff Ullman Stanford University http://www.mmds.org

Today's Lecture

- **More algorithms for streams:**
  - **Sampling data from a stream**
  - **Filtering a data stream: Bloom filters**
  - **Counting distinct elements: Flajolet-Martin**
  - **Estimating moments: AMS method.**

Counting Distinct Elements

Counting Distinct Elements

- **Problem:**
  - Data stream consists of a **universe 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 so far
  - That is, keep a **hash table** of all the distinct elements seen so far.

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 Web pages does each customer request in a week?**

- **How many distinct products have we sold in the last week?**

Using Small Storage

- **Real problem: What if we do not have space to maintain the set of elements seen so far?**
- **Estimate the count in an unbiased way**
- **Accept that the count may have a little error, but limit the probability that the error is large.**

Flajolet-Martin algorithm

- Estimating the counts

1. Hash every element ***a*** to a sufficiently long bit-string (e.g., h(a) = 1100)

2. Maintain R = length of longest trailing zeros among all bit-strings (e.g., R = 2)

3. Estimate count = 2^R (e.g., need to hash about 4 elements before we see a bit string with 2 trailing 0's).

Example

- Consider 4 distinct elements: a, b, c, d
- Hash value into bit string of length 4

- How likely do we see at least one hash value with a 0 in the last bit?
  - hash(a) = 0010
  - hash(b) = 0111
  - hash(c) = 1010
  - hash(d) = 1111

Example: at least one ends with 0

- E.g.,
  - hash(a) = 0010
  - hash(b) = 0111
  - hash(c) = 1010
  - hash(d) = 1111
- Prob. of none of hash values ending with 0:
  - $(1-½)^4$
- Prob. of at least one ending with 0:
  - $1-(1-½)^4 = .9375$

Example: at least one ends with oo

- E.g.,
  - hash(a) = 0100
  - hash(b) = 0111
  - hash(c) = 1010
  - hash(d) = 1111
- Prob. of none ending with 00:
  - $(1-(½)^2)^4$=.32
- Prob. of at least one ending with 00:
  - 1-.32 = .68

Example: at least one ends with ooo

- E.g.,
  - hash(a) = 0000
  - hash(b) = 0111
  - hash(c) = 1010
  - hash(d) = 1111
- Prob. of none ending with 000:
  - $(1-(½)^3)^4$=.59
- Prob. of at least one ending with 000:
  - 1-.59 = .41

Why It Works: Intuition

- **Very very rough and heuristic intuition why Flajolet-Martin works:**
  - ***h(a)*** hashes ***a*** with **equal prob.** to any of ***N*** values
  - Then ***h(a)*** is a sequence of $\log_2 N$ bits, where 2^{-r} fraction of all *a*s have a tail of *r* zeros
  - **So, it takes to hash about 2^r items before we see one with zero-suffix of length *r*.**

Why It Works: More formally

- **What is the probability that a given *h*(*a*) ends in at least *r* zeros is 2^{-r}**
  - **h(a)** hashes elements uniformly at random
  - Probability that a random number ends in at least ***r*** zeros is $\mathbf{2^{-r}}$
- Then, the probability of **NOT** seeing a tail of length ***r*** among ***m*** elements:

$$\boxed{\left(\boxed{1 - 2^{-r}}\right)^{m}}$$

Prob. all end in fewer than ***r*** zeros.

Prob. that given **h(a)** ends in fewer than ***r*** zeros

where m is the **number of distinct elements** seen so far in the stream

Why It Works: More formally

- Note= $(1-2^{-r})^m = (1-2^{-r})^{2^r(m2^{-r})} \approx e^{-m2^{-r}}$

- Prob. that h(a) for some element a has at least r trailing 0's
  - $p = 1-(1-2^{-r})^m = 1- \ e^{-\frac{m}{2^r}}$

1) If $2^r \gg \mathrm{m}, p = 1- \ e^{-\frac{m}{2^r}} \approx 1-(1-\frac{m}{2^r}) \approx \frac{m}{2^r} \approx 0$
2) If $2^r \ll \mathrm{m}, p = 1-1/e^{\frac{m}{2^r}} \to 1$

First 2 terms

$$e^{ix} = \underline{1+x}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}+\cdots$$

where m is the **number of distinct elements** seen so far in the stream

- **Thus, 2^R will almost always be around *m*.**

Why It Doesn't Work

- **E[2^R] is actually infinite**
  - Probability halves when $R \rightarrow R+1$, but value doubles
- Workaround involves using **many hash functions** h_i and getting **many samples of** R_i
- **How are samples** R_i **combined?**
  - **Average?** What if one very large value 2^{R_i}?
  - **Median?** All estimates 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.

Computing Moments

Generalization: Moments

- Suppose a stream has elements chosen from a set *A* of *N* values
- Let m_i be the number of times value *i* occurs in the stream
- The k^{th} ***moment*** is

$$\sum_{i \in A} (m_i)^k$$

Special Cases

$$\sum_{i \in A} (m_i)^k$$

- **0thmoment =** number of distinct elements
  - The problem just considered
- **1st moment =** count of the numbers of elements = length of the stream
  - Easy to compute
- **2nd moment =** ***surprise number S*** **=** a measure of how uneven the distribution is.

Example: Surprise Number

- **Stream of length 100**
- **11 distinct values**

- **Unsurprising**:
- Item counts: **10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9**
  Surprise *S* = 910
  - Surprise number = $10^2 + 10 * 9^2 = 910$
- **Surprising** :
- Item counts: **90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1**
- **Surprise *S* = ?**
- Surprise number = $90^2 + 10 * 1^2 = 8{,}110$

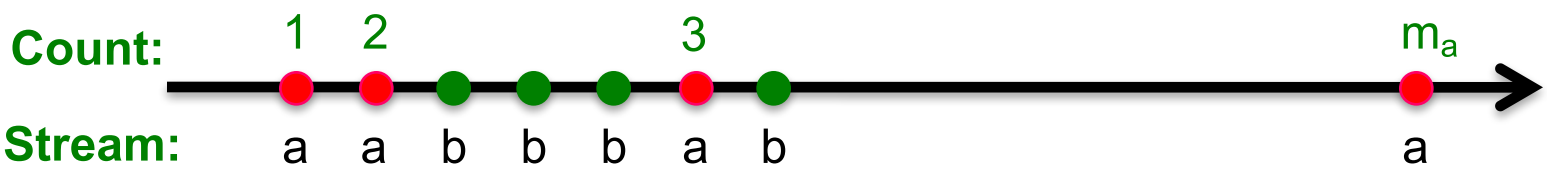
AMS (Alon-Matias-Szegedy) Method

- **AMS method works for all moments**
- **Gives an unbiased estimate**
- **We will just concentrate on the 2nd moment *S***
- **We pick and keep track of many variables *X*:**
  - For each variable ***X***
  - X.element: element in X
  - X.value: # of occurrences of X from time **t to n**
    - Note this requires a count in main memory, so number of ***X***s is limited
- **Our goal is to compute** $S = \sum_i m_i^2$

One Random Variable (X)

- 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
- $X = n * ((\textbf{twice the number of } \boldsymbol{a}\textbf{'s}$ in the stream starting at the chosen **time**$) - 1)$
  - **Note:** store n once, count of a's for each X.

Expectation Analysis



- **2nd moment is** $S = \sum_i m_i^2$
  - $E[f(X)]$ = (1/n) $\sum_{t=1}^{n} n$ * ((**twice the number of *a*'s** in the stream starting at the chosen **time**) – 1)
- c_t … number of times item at time t appears from time **t** onwards ($c_1 = m_a$, $c_2 = m_a - 1$, $c_3 = m_b$)
- $E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1)$

$$= \frac{1}{n}\sum_i n\,(1 + 3 + 5 + \cdots + 2m_i - 1)$$

m_i … total count of item i in the stream (we are assuming stream has length **n**)

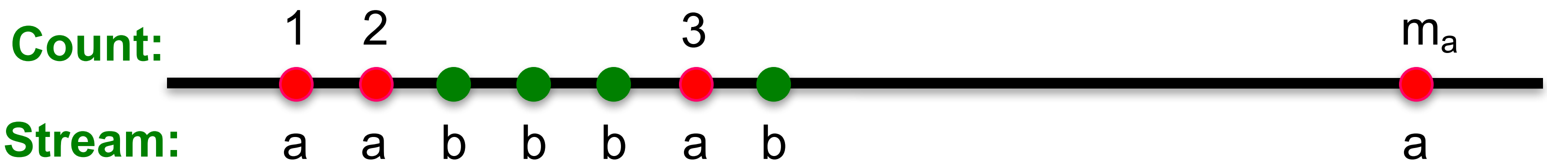
Group times by the value seen

Time t when the last i is seen (c_t=1)

Time t when the penultimate i is seen (c_t=2)

Time t when the first i is seen ($c_t = m_i$)

Expectation Analysis



- $E[f(X)] = \frac{1}{n} \sum_i n\,(1 + 3 + 5 + \cdots + 2m_i - 1)$
  - Little side calculation: $(1 + 3 + 5 + \cdots + 2m_i - 1) = \sum_{i=1}^{m_i} (2i - 1) = 2\frac{m_i(m_i+1)}{2} - m_i = (m_i)^2$
  - Note: 1+3+… + (2n-1) = n^2
- **Then $E[f(X)] = \frac{1}{n} \sum_i n\,(m_i)^2$**
- **So, $\mathbf{E[f(X)]} = \sum_i (m_i)^2 = S$**
- **We have the second moment (in expectation)!**

Higher-Order Moments

- **For estimating k^th^ moment we essentially use the same algorithm but change the estimate:**
  - For **k=2** we used ***n* (2·c – 1)**
  - For **k=3** we use: ***n* (3·c² – 3c + 1)** (where **c=X.val**)
- **Why?**
  - **For k=2:** Remember we had $(1+3+5+\cdots+2m_i-1)$ and we showed terms ***2c-1*** (for **c=1,…,m**) sum to ***m²***
    - $\sum_{c=1}^{m} 2c-1=\sum_{c=1}^{m} c^2-\sum_{c=1}^{m}(c-1)^2=m^2$
    - **So:** $\boldsymbol{2c-1=c^2-(c-1)^2}$
  - **For k=3:** **c^3 - $(c-1)^3$ = $3c^2$ - 3c + 1**
- **Generally:** Estimate $= n\,(c^k-(c-1)^k)$.

Combining Samples

- **In practice:**
  - Compute $\boldsymbol{f(X) = n(2\,c - 1)}$ for as many variables ***X*** as you can fit in memory
  - **Average** them in groups
  - Take **median of averages,**
- **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*** is a variable – the number of inputs seen so far.

Streams Never End: 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: (fixed-size sampling!)**
    - Choose the first ***k*** times for ***k*** variables
    - When the ***n*** $^{\text{th}}$ element arrives (***n*** > ***k***), choose it with probability ***k/n***
    - If you choose it, throw one of the previously stored variables **X** out, with equal probability.

Summary

- **Sampling a fixed proportion of a stream**
  - Sample size grows as the stream grows
- **Sampling a fixed-size sample**
  - Reservoir sampling
- **Filtering a data stream: Bloom filters**
  - Select elements with property **x** from stream
- **Counting distinct elements: Flajolet-Martin**
  - Number of distinct elements in the last ***k*** elements of the stream
- **Estimating moments: AMS method**
  - Estimate std. dev. of last ***k*** elements.