Mining Data Streams (Part 2)

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

(1) 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 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 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 **0s** in ***h*(*a*)**
  - **r(a)** = position of first 1 counting from the right
    - E.g., say ***h(a) = 12***, then ***12*** is ***1100*** in binary, so ***r(a) = 2***
- Record ***R* = the maximum *r*(*a*) seen**
  - $R = \max_a r(a)$, over all the items ***a*** seen so far
- **Estimated number of distinct elements =** 2^R

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
    - About 50% of ***a***s hash to ******0**
    - About 25% of ***a***s hash to *****00**
    - So, if we saw the longest tail of ***r=2*** (i.e., item hash ending ****100**) then we have probably seen **about *4*** distinct items so far
  - **So, it takes to hash about 2^r items before we see one with zero-suffix of length *r***

Why It Works: More formally

- **Now we show why Flajolet-Martin works**
- **Formally, we will show that probability of finding a tail of *r* zeros:**
  - **Goes to 1 if $m \gg 2^r$**
  - **Goes to 0 if $m \ll 2^r$**

  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 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

Why It Works: More formally

- **Note:** $(1-2^{-r})^m = (1-2^{-r})^{2^r (m2^{-r})} \approx e^{-m2^{-r}}$
- **Prob. of NOT finding a tail of length *r* is:**
  - If ***m << 2^r***, then prob. tends to **1**
    - $(1-2^{-r})^m \approx e^{-m2^{-r}} = 1$ as **$m/2^r \rightarrow 0$**
    - So, the probability of finding a tail of length ***r*** tends to **0**
  - If ***m >> 2^r***, then prob. tends to **0**
    - $(1-2^{-r})^m \approx e^{-m2^{-r}} = 0$ as **$m/2^r \rightarrow \infty$**
    - So, the probability of finding a tail of length ***r*** tends to **1**
- **Thus, 2^R will almost always be around *m*!**

Why It Doesn't Work

- **E[2^R] is actually infinite**
  - Probability halves when $R \to R+1$, 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
    - Take the median of groups
    - Then take the average of the medians

(2) 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

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

- **0th moment =** 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**

- Item counts: **10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9**
  Surprise *S* = 910

- Item counts: **90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1**
  Surprise *S* = 8,110

[Alon, Matias, and Szegedy]

AMS 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*** we store ***X.el*** and ***X.val***
    - ***X.el*** corresponds to the item ***i***
    - ***X.val*** corresponds to the **count** of item ***i***
  - Note this requires a count in main memory, so number of ***X***s is limited
- **Our goal is to compute** $S = \sum_i m_i^2$

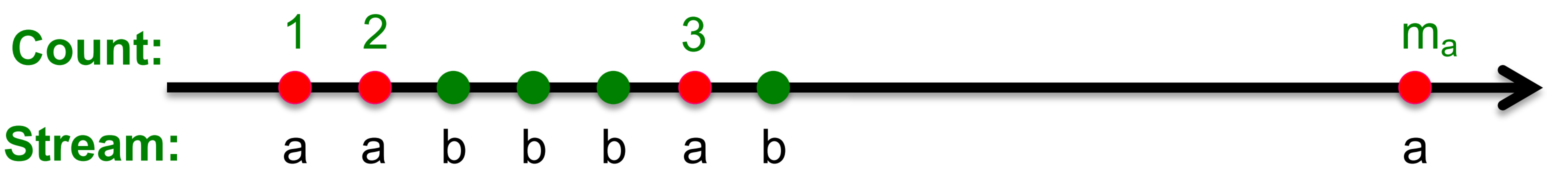
One Random Variable (X)

- **How to set *X.val* and *X.el*?**
  - Assume stream has length ***n*** (we relax this later)
  - Pick some random time ***t*** (***t<n***) to start, so that any time is equally likely
  - Let at time ***t*** the stream have item ***i***. ***We set X.el = i***
  - Then we maintain count ***c*** (***X.val = c***) of the number of ***i*s** in the stream starting from the chosen time ***t***
- **Then the estimate of the 2nd moment ($\sum_i m_i^2$) is:**

$$S = f(X) = n\,(2 \cdot c - 1)$$

  - Note, we will keep track of multiple **X**s, (**X_1, X_2,… X_k**) and our final estimate will be $S = 1/k \sum_j^k f(X_j)$

Expectation Analysis



- **2nd moment is $S = \sum_i m_i^2$**
- 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**)

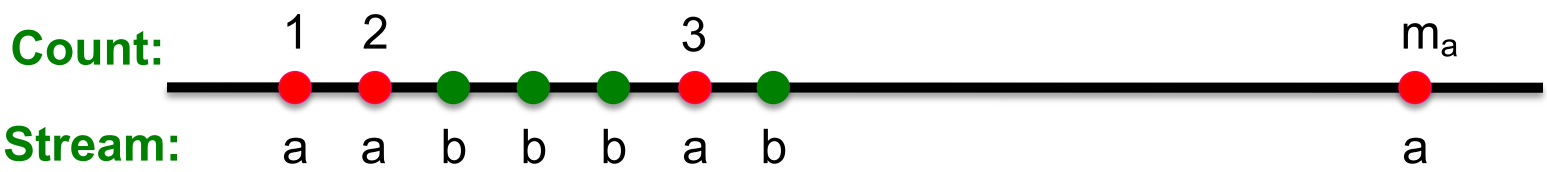
Group times by the value seen

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: $(1 + 3 + 5 + \cdots + 2m_i - 1) =$ $\sum_{i=1}^{m_i}(2i - 1) = 2\frac{m_i(m_i+1)}{2} - m_i = (m_i\ )^2$
- **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·c^2 – 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 ***m*^{2}**
    - $\sum_{c=1}^{m} 2c - 1 = \sum_{c=1}^{m} c^2 - \sum_{c=1}^{m} (c-1)^2 = m^2$
    - **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 $f(X) = n(2c - 1)$ 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 $\boldsymbol{X}$ have $\boldsymbol{n}$ as a factor – keep $\boldsymbol{n}$ separately; just hold the count in $\boldsymbol{X}$
- **(2)** Suppose we can only store $\boldsymbol{k}$ counts. We must throw some $\boldsymbol{X}$s out as time goes on:
  - **Objective:** Each starting time $\boldsymbol{t}$ is selected with probability $\boldsymbol{k/n}$
  - **Solution: (fixed-size sampling!)**
    - Choose the first $\boldsymbol{k}$ times for $\boldsymbol{k}$ variables
    - When the $\boldsymbol{n}^{th}$ element arrives ($\boldsymbol{n > k}$), choose it with probability $\boldsymbol{k/n}$
    - If you choose it, throw one of the previously stored variables **X** out, with equal probability