Mining Streams

Shantanu Jain

Data Streams

- In many data mining situations, we do not know what data will arrive in advance

- Stream Management is important when the input rate is controlled externally:
  - Google queries
  - Twitter or Facebook status updates

- Can think of streams as infinite and non-stationary (the distribution changes over time)

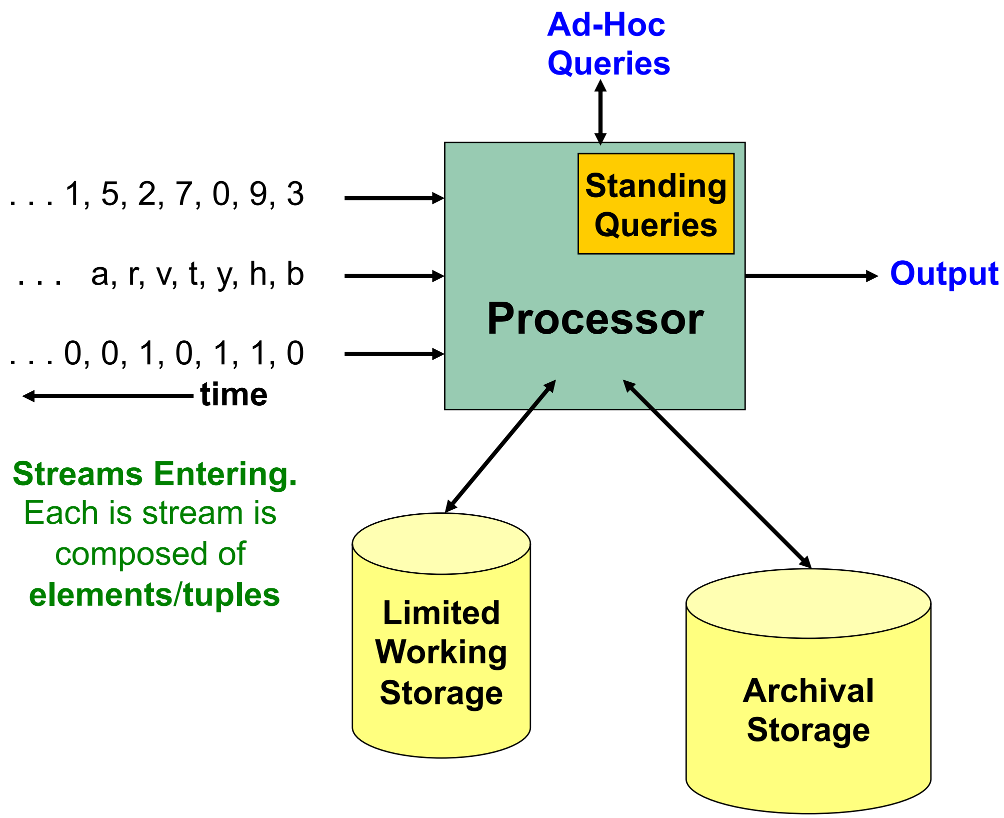
adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

The Stream Model

- Input elements enter at a rapid rate, at one or more input ports (i.e., streams)
  - We often represent elements as tuples
- The system cannot store the entire stream
- Q: How do you make calculations about the stream using a limited amount of (secondary) memory?

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

General Stream Processing Model



- Common Types of Queries:
  - Sampling data from a stream
    - Construct a random sample
  - Queries over sliding windows
    - Number of items of type ***x*** in the last ***k*** elements of the stream
  - Filtering a data stream
    - Select elements with property ***x*** from the stream
  - Counting distinct elements
    - Number of distinct elements in last ***k*** elements of the stream
  - Estimating moments
    - Estimating frequency/surprise

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Applications (1)

- Mining query streams
  - Google wants to know what queries are more frequent today than yesterday
- Mining click streams
  - Wikipedia wants to know which of its pages are getting an unusual number of hits in the past hour
- Mining social network news feeds
  - E.g., look for trending topics on Twitter, Facebook

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Applications (2)

- Sensor Networks
  - Many sensors feeding into a central controller
- Telephone call records
  - Data feeds into customer bills as well as settlements between telephone companies
- IP packets monitored at a switch
  - Gather information for optimal routing
  - Detect denial-of-service attacks

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Mining Streams

Shantanu Jain

Sampling from Streams

Sampling from a Data Stream

- Since **we can not store the entire stream**, one obvious approach is to store a sample (i.e. a random subset of elements)

- Two different approaches:
  1. Sample a fixed fraction of elements (say 1 in 10)
  2. Maintain a random sample of fixed size over a potentially infinite stream
     - At "any time" k we would like a random sample of s elements
       - For all time steps k, each of k elements so far should have an equal probability of being included in the s elements

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Approach 1: Sampling a Fixed Proportion

- Scenario: Search engine query stream
  - Stream of tuples: (user, query, time)
  - Answer questions such as: How many queries from a typical user in past 30 days are repeat queries.
  - Have space to store **1/10th** of query stream

- Naïve solution: Random subsampling
  - Generate a random number **u ~ Uniform([0, 1))**
  - Store the query if **u < 0.1**, discard if **u ≥ 0.1**

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Problem with Naïve Approach

- Suppose each user issues ***x*** number of queries once and ***d*** number of queries twice (total of ***x*+2*d*** queries)
  - True Fraction of Duplicates (unknown): *d* / (*x*+*d*)

- Naive Estimate: Keep 10% of the queries
  - Sample will contain ***x* / 10** of the singleton queries and **2*d* / 10** of the duplicate queries at least once
  - But only ***d* / 100** pairs of duplicates

    d/100 = 1/10 · 1/10 · d
  - Of ***d*** "duplicates" ***18d / 100*** appear exactly once

    18d / 100 = ((1/10 · 9/10) + (9/10 · 1/10)) · d

Fraction in Sample $\dfrac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}} = \dfrac{\boldsymbol{d}}{\boldsymbol{10x+19d}}$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Problem with Naïve Approach

- Suppose each user issues n_x number of queries once and n_d number of queries twice (total of $n_x + 2n_d$ queries)
  - True Fraction of Duplicates (unknown): $n_d/(n_x + n_d)$

- Naive Estimate: Keep 10% of the queries
  - Sample will contain $n_x/10$ of the singleton queries and $2n_d/10$ of the duplicate queries at least once
  - But only $n_d/100$ pairs of duplicates

    $n_d/100 = 1/10 \cdot 1/10 \cdot n_d$
  - Of *d* "duplicates" *18*n_d */ 100* appear exactly once

    $18n_d / 100 = ((1/10 \cdot 9/10) + (9/10 \cdot 1/10)) \cdot n_d$

Fraction in Sample $\dfrac{\frac{n_d}{100}}{\frac{n_x}{10} + \frac{n_d}{100} + \frac{18n_d}{100}} = \dfrac{n_d}{10n_x + 19n_d}$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Solution: Sample Users

Alternative Solution:

- Pick 1/10th of users and take all their searches in the sample
- Use a hash function to uniformly assign user names to10 buckets

Example hash function

$$h(u) = (u \text{ mod } 10) + 1$$

maps user id to a value in $\{1,2,\ldots,10\}$

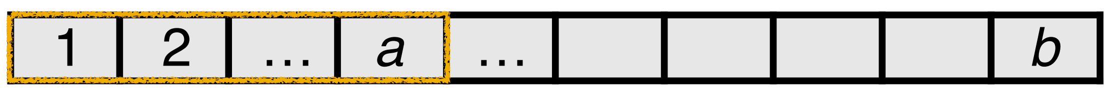
each user is assigned a randomly generated id

If $h(u) \leq 1$ store the tuple, else discard it.

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Better Solution: Sample Keys

- Assume tuples with keys:
  - Key is some subset of each tuple's components
    - e.g., tuple is **(user, search, time)**; key is user
  - Choice of key depends on application
- To get a sample of a / b fraction of the stream:
  - Hash each tuple's key uniformly into *b* buckets
  - Pick the tuple if its hash value is at most *a*



Hash table with **b** buckets, pick the tuple if its hash value is at most **a.**
How to generate a 30% sample?
Hash into b=10 buckets, take the tuple if it hashes to one of the first 3 buckets

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Approach 2: Fixed-size Sample

- Suppose we want to maintain a random sample S of size exactly s tuples
  - E.g., main memory size constraint
  - Why? May not know length of stream in advance
- Goal: Ensure equal probability of inclusion
  - Suppose at time n we have seen n items
  - Each item should occur in the sample S with probability s / n

Approach 2: Fixed-size Sample

- Algorithm (a.k.a. Reservoir Sampling)
  - Store all the first ***s*** elements of the stream to ***S***
  - Suppose we have seen ***n-1*** elements, and now the n^{th} element arrives ($n > s$)
    - With probability s / n, keep the n^{th} element, else discard it
    - If we picked the n^{th} element, then it replaces one of the s elements in the sample S, picked uniformly at random

- Claim: This algorithm maintains a sample ***S*** with the desired property:
  - After n elements, the sample contains each element seen so far with probability s / n

Proof: By Induction

- We prove this by induction:
  - Assume that after *n* elements, the sample contains each element seen so far with probability *s/n*
  - We need to show that after seeing element *n+1* the sample maintains the property
    - Sample contains each element so far with prob *s/(n+1)*
- Base case:
  - After we see *n=s* elements the sample *S* has the desired property
    - Each out of *n=s* elements is in the sample with probability s/s = 1

Proof: By Induction

- **Inductive hypothesis:** After *n* elements, the sample ***S*** contains each element seen so far with prob. *s / n*
- **Inductive step:** When element *n+1* arrives, the probability for retention of each of the first *n* elements given it is already included in S is

$$\underbrace{\left(1-\frac{s}{n+1}\right)}_{\text{Element } \mathbf{n+1} \text{ discarded}}+\underbrace{\left(\frac{s}{n+1}\right)}_{\text{Element } \mathbf{n+1}\ \textit{not} \text{ discarded}}\underbrace{\left(\frac{s-1}{s}\right)}_{\text{Element in } \boldsymbol{S}\ \textit{not} \text{ replaced}}=\frac{n}{n+1}$$

The unconditional probability for retention after *n*+1 steps is

First ***n*** Elements $\underbrace{\left(\frac{s}{n}\right)}_{\text{Retained after } \boldsymbol{n} \text{ steps}}\underbrace{\left(\frac{n}{n+1}\right)}_{\text{Retained in step } \boldsymbol{n+1}}=\frac{s}{n+1}$ New Element $\underbrace{\frac{s}{n+1}}_{\text{Element } \mathbf{n+1}\ \textit{not} \text{ discarded}}$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Mining Streams

Shantanu Jain

Counting with Exponetially Decaying Windows

Sliding Windows

- One model for stream processing is to apply queries to a *window* of ***N*** most recent elements

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

Past Future

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Sliding Windows

- One model for stream processing is to apply queries to a *window* of ***N*** most recent elements

Stream of sales

10, 200, 100, 1000, 125, 500, 23, 72, 1250 …

Stream of bits

1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Sliding Windows

- Difficult case: Window size ***N*** is so large that the data cannot be stored in memory, or even on disk
    - Or, there are so many streams that windows for all cannot be stored
- Amazon example:
    - For every product ***X*** we keep **0/1** stream of whether that product was sold in the **n-th** transaction
    - We want answer queries, such as finding frequent items that were sold more than ***s*** times

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Sliding Windows

- Difficult case: Window size ***N*** is so large that the data cannot be stored in memory, or even on disk
  - Or, there are so many streams that windows for all cannot be stored
- Amazon example:
  - For every product ***X*** we keep **0/1** stream of whether that product was sold in the **n-th** transaction
  - We want answer queries, such as finding frequent items that were sold more than ***s*** times

Exponentially Decaying Windows

- **Sliding window:** Count occurrences in last ***N*** elements

$$\sigma_t^{\mathrm{SW}}(x) = \sum_{i=t-N}^{t} I[\, x \in A_i \,] \qquad I[a_i \in x] = \begin{cases} 1 & x \in A_i \\ 0 & x \notin A_i \end{cases} .$$

"Indicator" function: returns 1 when query matches, 0 when not

- **Exponentially decaying window:** Give lower "weight" to occurences that are farther back in time

$$\sigma_t^{\mathrm{SDW}}(x) = \sum_{i=1}^{t} I[\, x \in A_i \,]\,(1-c)^{t-i}$$

c is a small constant (e.g. **0.001**) such that **(1-*c*)** is close to **1**, but **$(1-c)^{t-i}$** decays to **0** when **$t \gg i$**

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Exponentially Decaying Windows

- **Convenient Property:** Can compute sum at time ***t*** from sum at time ***t-1***

$$\sigma_t^{\text{EDW}}(x) = \sum_{i=1}^{t} I[x \in A_i](1-c)^{t-i}$$

$$= \underbrace{I[x \in A_i]}_{\text{Term for i = t}} + \underbrace{\sum_{i=1}^{t-1} I[x \in A_i](1-c)^{t-i}}_{\text{Terms for i < t}}$$

$$= I[x \in A_i] + (1-c)\,\sigma_{t-1}^{\text{EDW}}(x)$$

Don't need to keep transactions $A_1, \ldots, A_t$ in memory, just need to keep track of running weights **σ(x)**

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Counting Items with Decaying Windows

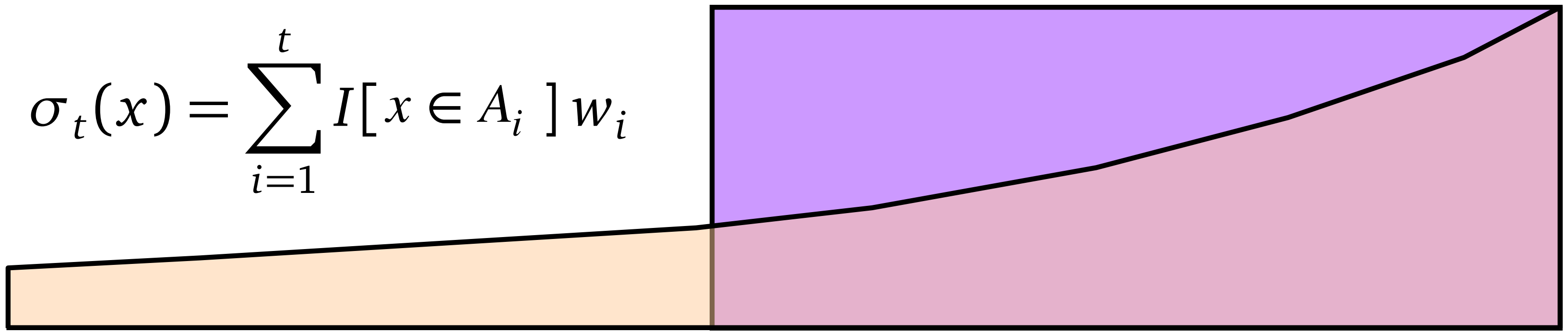
$$\sigma_t^{\mathrm{EDW}}(x) = I[\, x \in A_i \,] + (1-c)\, \sigma_{t-1}^{\mathrm{EDW}}(x)$$

- Initialization: Set **σ(x) = 0** for all items x in some set **X**
- For each time step
  - Apply decay factor to all item weights **σ(x) = (1-c) σ(x)**
  - Increment weight for items in the current transaction A_i:
    $a \in A_i$, **σ[a] = σ[a]+1**

Sliding vs Exponential Windows

- Sliding and Exponential windows compute a weighted sum

$$\sigma_t(x) = \sum_{i=1}^{t} I[x \in A_i]\, w_i$$



- What differs is the definition of the weights

Sliding $w_i = \begin{cases} 1 & i > t-N, \\ 0 & i \le t-N. \end{cases}$

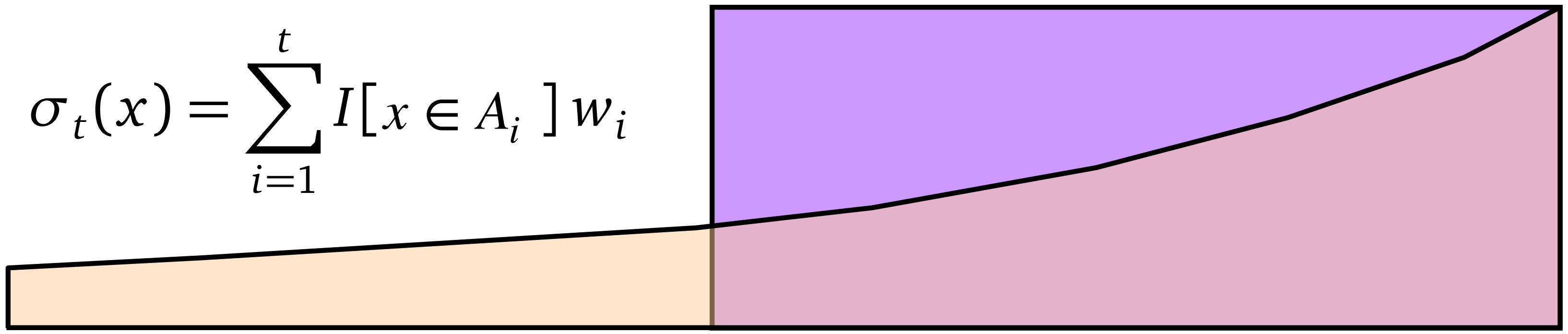
Exponential Decaying $w_i = (1-c)^{t-i}$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Sliding vs Exponential Windows

- Sliding and Exponential windows compute a weighted sum

$$\sigma_t(x) = \sum_{i=1}^{t} I[x \in A_i\,]\, w_i$$



- In a sliding window, the sum of the weights is ***N***
- In a exponentially window, the sum is a *geometric series*

$$\lim_{t\to\infty} \sum_{i=1}^{t} w_i = \lim_{t\to\infty} \sum_{i=1}^{t} (1-c)^{t-i} = \frac{1}{1-(1-c)} = \frac{1}{c}$$

We can think of **1/c** as the "effective window size"

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Extension to Itemsets

- Count (some) itemsets in an E.D.W.
  - What are currently "hot" itemsets?
    - **Problem:** Too many itemsets to keep counts of all of them in memory
- When a basket A_i comes in:
  - Multiply all counts by **(1-c)**
  - For uncounted items in A_i, create new count
  - Add **1** to count of any item in A_i and to any itemset contained in A_i that is already being counted
  - Drop counts **< ½**
  - Initiate new counts (*next slide*)

Initiation of New Counts

- Start a count for an itemset ***S*** $\subseteq A_i$ if every proper subset of ***S*** had a count prior to arrival of basket ***B***
  - Intuitively: If all subsets of ***S*** are being counted this means they are "frequent/hot" and thus ***S*** has a potential to be "hot"
- Example:
  - Start counting ***S***={**i**, **j**} iff both **i** and **j** were counted prior to seeing ***B***
  - Start counting **S={i, j, k}** iff **{i, j}**, **{i, k}**, and **{j, k}** were all counted prior to seeing ***B***

Mining Streams

Shantanu Jain

Filtering Data Streams

Filtering Data Streams

- Each element of data stream is a tuple
- Given a list of keys ***S***
- Determine which tuples of stream are in ***S***

- Obvious solution: Hash table
  - But suppose we do not have enough memory to store all of ***S*** in a hash table
    - E.g., we might be processing millions of filters on the same stream

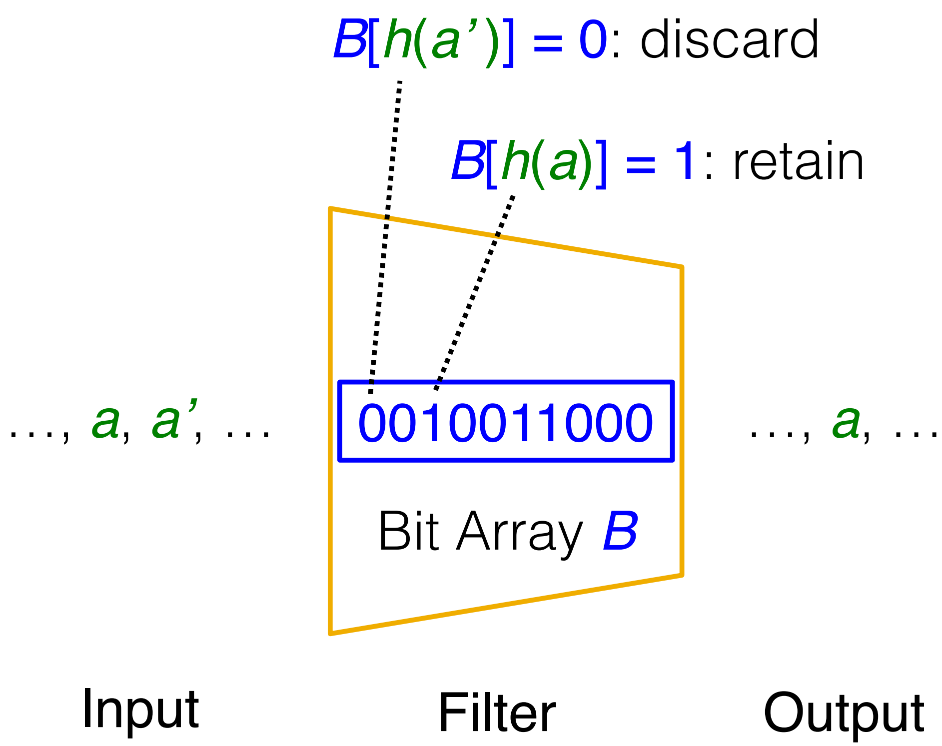
adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Applications

- Example: Email spam filtering
  - We know 1 billion “good” email addresses
  - If an email comes from one of these, it is NOT spam

- Publish-subscribe systems
  - You are collecting lots of messages (news articles)
  - People express interest in certain sets of keywords
  - Determine whether each message matches user’s interest

Idea: Hash-based Filtering



- Given a set of keys S that we want to filter
  - Create a bit array B of n bits, initially all 0s
  - Choose a hash function h with range $[0,n)$
  - Hash each member of $s \in S$ to one of n buckets and set $B[h(s)]=1$
  - For each element a, output a if $B[h(a)] == 1$
- No false negatives
- Can have false positives (hash collision)

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Idea: Hash-based Filtering

Example:

S = 1 billion email addresses
B = array of 1 billion bytes (1GB)

- If the email address is in ***S***, then it must hash to a bucket that has is set to 1, so it always gets through (*no false negatives*)

- Approximately **1/8** of the bits are set to **1**, so about **1/8**th of the addresses not in ***S*** get through (*false positives*)
  - Actually, *less* than **1/8**th, because more than one address might hash to the same bit

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Analysis: False Positive Rate

- More accurate analysis for the number of false positives
- Consider: If we throw ***m*** darts into ***n*** equally likely targets, what is the probability that a target gets at least one dart?
- In our case:
  - Targets = bits / buckets
  - Darts = hash values of query keys

Analysis: False Positive Rate

- We have ***m*** "darts" (hash values of items in ***S***), ***n*** "targets" (bits in array)
- Assuming darts hit targets uniformly at random, what is the probability that a target is hit?

Probability that *all darts miss*

$$\underbrace{1-\overbrace{\Big(\underbrace{1-\frac{1}{n}}_{\text{Probability that a single dart misses}}\Big)^{m}}^{\text{Probability that all darts miss}}} = 1-\left(\left(1-\frac{1}{n}\right)^{n}\right)^{m/n} \simeq 1-\left(\frac{1}{e}\right)^{m/n} = 1-e^{-m/n}$$

Probability that a *single dart misses*

*When **n** is large*

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Analysis: False Positive Rate

- Fraction of **1s** in the array ***B***
  = probability of false positive = $\mathbf{1} - e^{-m/n}$

- Example: $\mathbf{10^9}$ darts, $\mathbf{8 \cdot 10^9}$ targets
  - Fraction of **1s** in **B** = $\mathbf{1 - e^{-1/8} = 0.1175}$
    - Compare with our earlier estimate: **1/8 = 0.125**

Bloom Filter

- Consider: $|S| = m$, $|B| = n$
- Use k independent hash functions $h_1, \ldots, h_k$
- Initialization:
  - Set **B** to all **0s**
  - For each $s \in S$ set $B[h_i(s)] = 1$
    (for all $i = 1, \ldots, k$)
- Run-time:
  - For each stream element with key x
    - If $B[h_i(x)] = 1$ for all $i = 1, \ldots, k$
      then retain x, since $x \in S$
    - Otherwise discard the element x

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

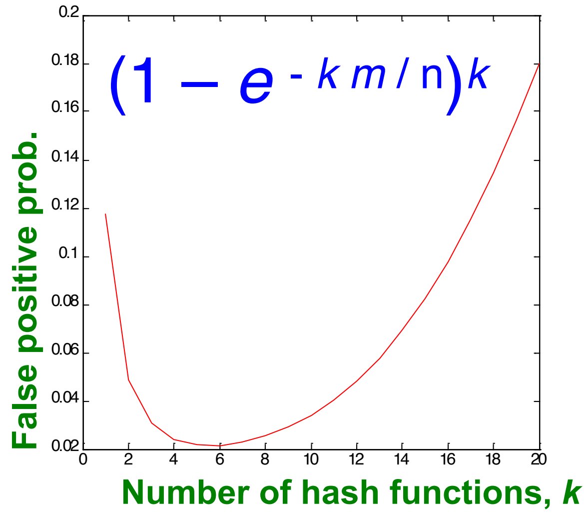
Bloom Filter: False Positive Rate

- What fraction of the bit vector B are 1s?
  - Throwing $k \cdot m$ "darts" at n "targets"
  - So fraction of **1s** is $(1 - e^{-k\,m\,/\,n})$
- But we have *k* independent hash functions and we only let the element *x* through if all *k* hash element *x* to a bucket of value **1**
- So, false positive probability $= (1 - e^{-k\,m\,/\,n})^k$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Bloom Filter: False Positive Rate

- m = 1 billion, n = 8 billion
  - **k = 1**: $(1 - e^{-1/8}) = 0.1175$
  - **k = 2**: $(1 - e^{-1/4})^2 = 0.0493$



- What happens as we keep increasing k ?

- "Optimal" value of k: $(n / m) \ln(2)$
  - In our case: Optimal k = 8 ln(2) = 5.54 ≈ 6
    - Error at k = 6: $(1 - e^{-1/6})^2 = 0.0235$

Bloom Filter: Wrap-up

- Bloom filters guarantee no false negatives, and use limited memory
  - Great for pre-processing before more expensive checks
- Suitable for hardware implementation
  - Hash function computations can be parallelized

Mining Streams

Shantanu Jain

Counting Distinct Elements

Counting Distinct Elements

- Problem:
  - A stream consists of a distribution over elements chosen from a set of size ***N***
  - We would like to count the number of *distinct* elements seen so far
- Obvious approach:
  - Maintain set of elements seen so far
  - That is, keep a hash table of distinct elements

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

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?
- **Real problem:** 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

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Flajolet-Martin Algorithm

- 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 binary representation of $h(a)$
  - $r(a)$ = 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

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

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, it takes to hash about 2^r items before we see one with zero-suffix of length *r*

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Why It Doesn't Work

- Expect value $E[2^R]$ 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
  - Pragmatic Solution:
    - Partition your samples into small groups
    - Take the median of groups
    - Then take the average of the medians

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Mining Streams

Shantanu Jain

Computing Moments

Moments of a Stream

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

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Special Cases

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

- 0^{th} moment = number of distinct elements
  - 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 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 $S = 8{,}110$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

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 approximate $S = \sum_{i \in A} (m_i)^2$

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*** < ***n*** to start (*any time equally likely*)
  - Set *X.el* = *i*, where *i* is the item at time ***t.***
  - We maintain count ***c*** (*X.val* = *c*) of the number of ***is*** in the stream starting from the chosen time ***t***

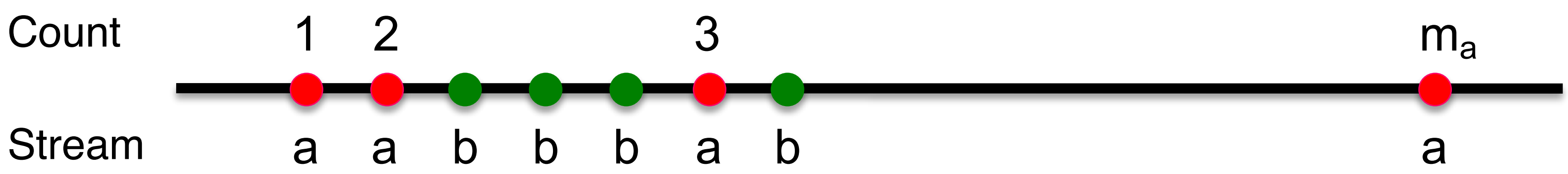
- Then the AMC estimate of the 2nd moment is:

$$S = \sum_i (m_i)^2 \simeq f(X) = n(2 \cdot c - 1)$$

  - Note, we can track multiple variables (X_1, X_2,… X_k) to compute an average $S = \frac{1}{k} \sum_{j=1}^{k} f(X_j)$

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

AMC Estimate: Derivation



- Define c_t = number of future appearances of the item at time t
  - $c_1 = m_a, \;\; c_2 = m_a - 1, \;\; c_3 = m_b, \;\ldots$
- Then the expected value of **f(X)=n(2c-1)** is a sum over t

$$\mathbb{E}[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1)$$

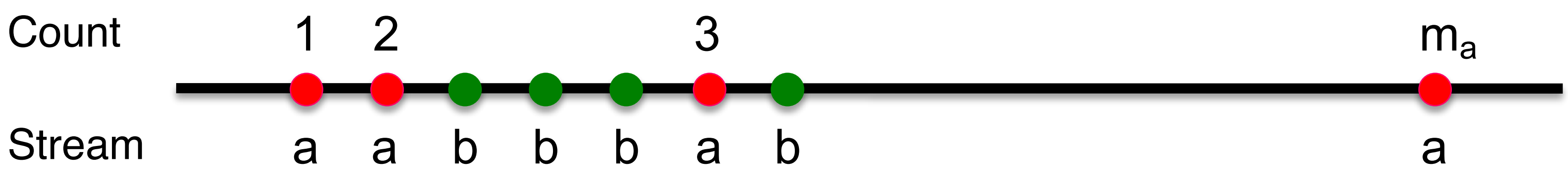
- We can re-arrange to obtain a sum counts c_t for each item i

$$\mathbb{E}[f(X)] = \frac{1}{n}\sum_{i} n\Big(\big(2m_i - 1\big) + \big(2(m_i - 1) - 1\big) + \ldots 5 + 3 + 1\Big)$$

First c_t for item i — Second c_t for item i — Final c_t for item i

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

AMC Estimate: Derivation



- Let's rewrite the result from previous slide:

$$\mathbb{E}[f(X)] = \frac{1}{n}\sum_i n\Big(\big(2m_i - 1\big) + \big(2(m_i-1)-1\big) + \ldots 5 + 3 + 1\Big)$$

$$= \frac{1}{n}\sum_i n \sum_{j=1}^{m_i}(2j-1) = \sum_i \sum_{j=1}^{m_i}(2j-1)$$

- Now use *triangle numbers:* $\sum_{j=1}^{m_i} j = \frac{1}{2}m_i(m_i+1)$

$$\mathbb{E}[f(X)] = \sum_i \big(m_i(m_i+1) - m_i\big) = \sum_i (m_i)^2 = S$$

We have now shown that f(X) is an unbiased estimate of S!

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Combining Samples

- In practice:
  - Compute **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

Moments of Infinite Streams

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 Xs out as time goes on:
   - Objective: Each starting time t is selected with probability k / n
   - Solution: reservoir sampling! (*from previous video*)
     - 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 X out, with equal probability

adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org