Mining Data Streams (Part 2)

CS246: Mining Massive Datasets

Jure Leskovec, Stanford University

http://cs246.stanford.edu

Today's Lecture

- More algorithms for streams:
  - (1) Filtering a data stream: **Bloom filters**
    - Select elements with property x from stream
  - (2) Counting distinct elements: **Flajolet-Martin**
    - Number of distinct elements in the last *k* elements of the stream
  - (3) Estimating moments: **AMS method**
    - Estimate std. dev. of last *k* elements
  - (4) Counting frequent items

(1) Filtering Data Streams

- Each element of data stream is a tuple
- Given a list of keys S
- Determine which elements of stream have keys 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

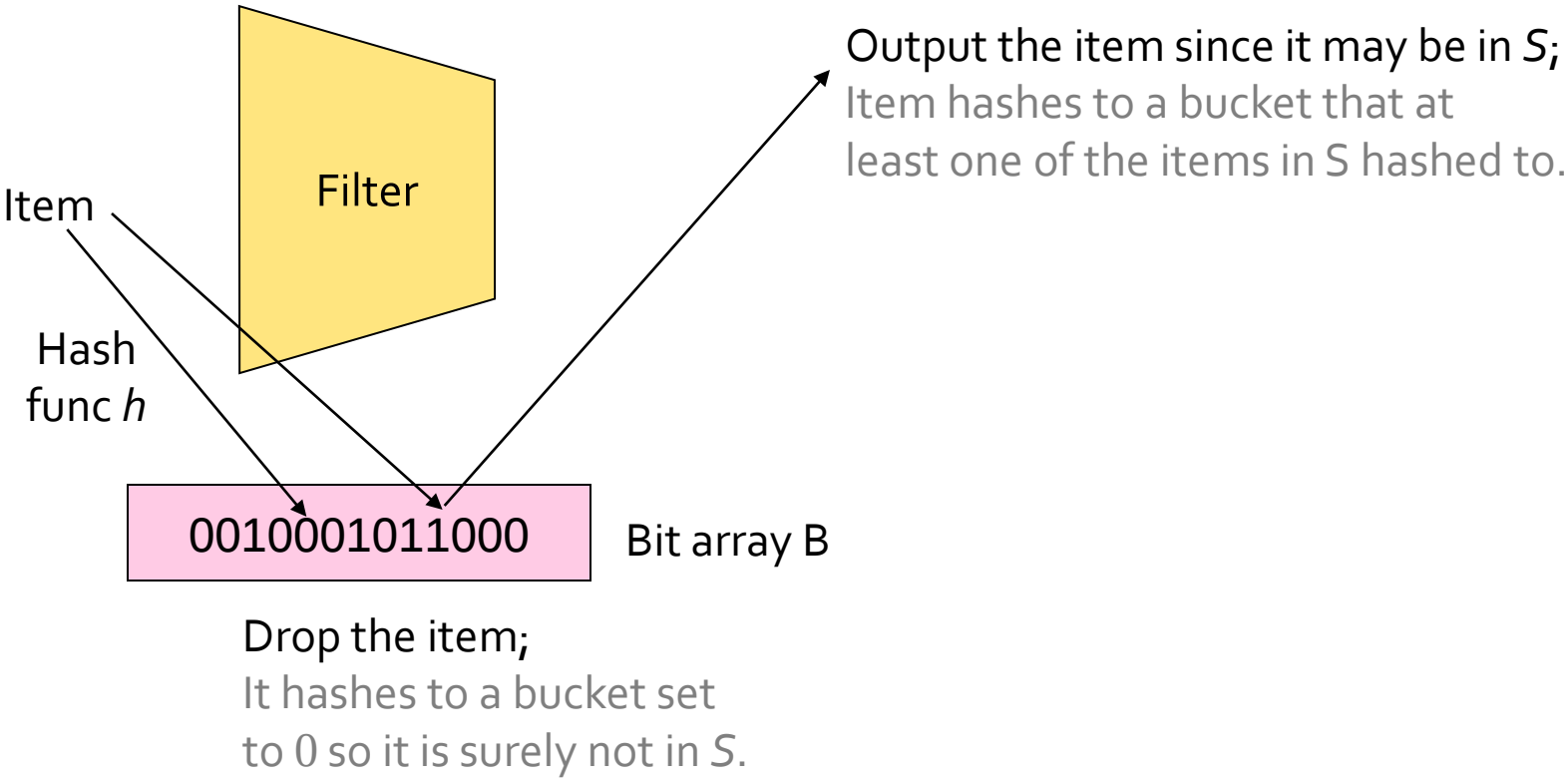
Applications

- 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:
  - People express interest in certain sets of keywords
  - Determine whether each message matches user's interest

First Cut Solution – (1)

- Create a bit array *B* of *n* bits, initially all 0s
- Choose a hash function *h* with range [0,m)
- Hash each member of $s \in S$ to one of *m* buckets, and set that bit to 1, i.e., *B[h(s)]=1*
- Hash each element *a* of the stream and output only those that hash to bit that was set to 1
  - Output *a* if B[h(a)] == 1

First Cut Solution – (2)



- Creates false positives but no false negatives
  - If the item is in S we surely output it, if not we may still output it

First Cut Solution – (3)

- |S| = 1 billion email addresses
  |B|= 1GB = 8 billion bits
- If the email address is in *S*, then it surely hashes to a bucket that has the big 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 to the output (*false positives*)
  - Actually, less than $1/8^{th}$, because more than one address might hash to the same bit

Analysis: Throwing Darts

- 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 items

Analysis: Throwing Darts – (2)

- We have *m* darts, *n* targets
- What is the probability that a target gets at least one dart?

Equals 1/e as n $\to\infty$

Equivalent

$$1 - (1 - 1/n)^{n(m/n)}$$

$$1 - e^{-m/n}$$

Probability target not hit by one dart

Probability at least one dart hits target

Analysis: Throwing Darts – (3)

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

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

Bloom Filter

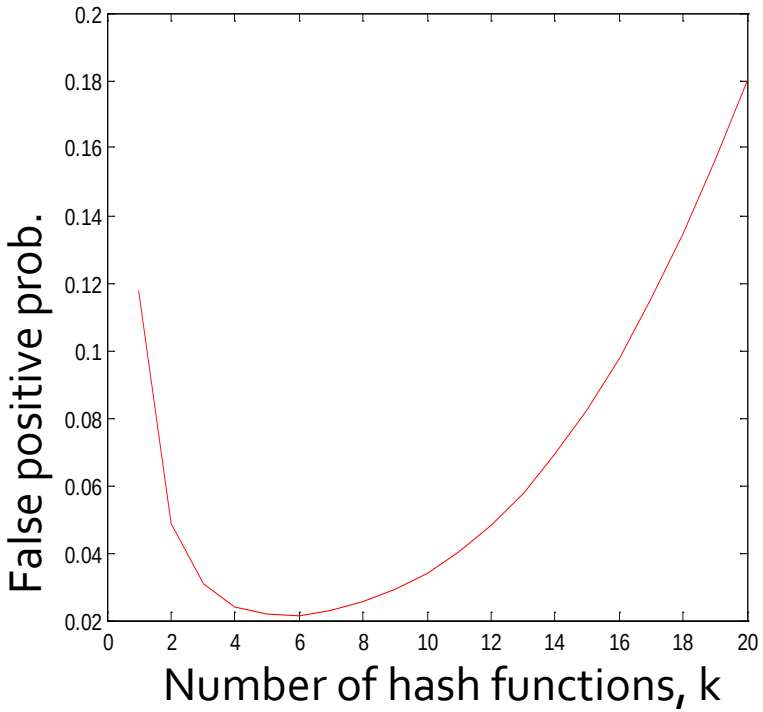
- Consider: $|S| = m$, $|B| = n$
- Use k independent hash functions $h_1, \ldots, h_k$
- Initialization:
  - Set B to all 0s
  - Hash each element $s \in S$ using each hash function h_i, set $B[h_i(s)] = 1$ (for each $i = 1, .., k$)
- Run-time:
  - When a stream element with key x arrives
    - If $B[h_i(x)] = 1$ for <u>all</u> $i = 1, \ldots, k$, then declare that x is in S
      - i.e., x hashes to a bucket set to 1 for every hash function $h_i()$
    - Otherwise discard the element x

Bloom Filter -- Analysis

- What fraction of the bit vector B are 1s?
  - Throwing $k \cdot m$ darts at n targets
  - So fraction of 1s is $(1 - e^{-km/n})$
- But we have k independent hash functions
- So, false positive probability = $(1 - e^{-km/n})^k$

Bloom Filter – Analysis (2)

- 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)
  - E.g.: 8 ln(2) = 5.54

Bloom Filter: Wrap-up

- Bloom filters guarantee no false negatives, and use limited memory
  - Great for pre-processing before more expensive checks
  - E.g., Google's BigTable, Squid web proxy
- Suitable for hardware implementation
  - Hash function computations can be parallelized

(2) Counting Distinct Elements

- New topic: 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

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?

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

- The probability that a given $h(a)$ ends in at least r 0s is 2^{-r}
- Probability of NOT seeing a tail of length r among m elements: $\boxed{(1 - \boxed{2^{-r}})^m}$

Prob. all end in fewer than r 0s.

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

Why It Works – (2)

- 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 $m/2^r \to 0$
    - 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 $m/2^r \to \infty$
    - So, the probability of finding a tail of length *r* tends to 1
- Thus, 2^R will almost always be around *m*

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

Why It Works – (3)

- One can also think of Flajolet-Martin the following way (roughly):
  - *h(a)* hashes item *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 a's have a tail *r* zeros
    - 50% hashes end with ***0, 25% hashes end with **00
    - So, if we saw the longest tail of *r=2* (i.e., item hash ending *100) then we have probably seen *4* distinct items so far
  - So, in expectation it takes 2^r items before we see one with zero-suffix of length *r*

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 and getting many samples
- How are samples combined?
  - Average? What if one very large value?
  - Median? All estimates are a power of 2
  - Solution:
    - Partition your samples into small groups
    - Take the average of groups
    - Then take the median of the averages

(3) Generalization: Moments

- Suppose a stream has elements chosen from a set of *N* values
- Let m_a be the number of times value *a* occurs
- The kth *moment* is $\sum_a (m_a)^k$

Special Cases

- 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* = 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 # = 910
- Item counts: 90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1
  Surprise # = 8,110

[Alon, Matias, and Szegedy]

AMS Method

- Works for all moments
- Gives an unbiased estimate

- We will just concentrate on the 2nd moment

- Based on calculation of many random variables *X:*
  - For each rnd. var. *X* we store *X.el* and *X.val*
  - Note this requires a count in main memory, so number of *X*s is limited

One Random Variable (X)

- How to set *X.val* and *X.el*?
  - Assume stream has length *n*
  - Pick a random time *t* to start, so that any time is equally likely
  - Let at time *t* the stream have element *a* (i.e., *X.el* = *a*)
  - Maintain count *c* (*X.val* = *c*) of the number *a's* in the stream starting from the chosen time *t*
- Then the estimate of the 2nd moment is $n\,(2\,c - 1)$
  - Store *n* once, count *a*'s for each *X*

Expectation Analysis

- 2nd moment is $\Sigma_a (m_a)^2$
- c_t … the number of times the stream element at time t appears from that time on
- $E[X.val] = (1/n) \Sigma_{\text{all times } t}\ n\ (2\ c_t - 1)$

$$= \Sigma_a (1/n)\ (n)\ (1 + 3 + 5 + \ldots + 2m_a - 1)$$

$$= \Sigma_a (m_a)^2$$

Group times by the value seen

Time when the last a is seen

Time when the penultimate a is seen

Time when the first a is seen

Combining Samples

- In practice:
  - Compute n (2 c – 1) for as many variables X as you can fit in memory
  - Average them in groups
  - Take median of averages

  - Proper balance of group sizes and number of groups assures not only correct expected value, but expected error goes to 0 as number of samples gets large

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 Xs out as time goes on:
   - Objective: Each starting time t is selected with probability k/n
   - Solution:
     - Choose the first k times for k variables
     - When the nth 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.

(3) Counting Itemsets

- New Problem: Given a stream, which items appear more than *s* times in the window?
- Possible solution: Think of the stream of baskets as one binary stream per item
  - 1 = item present; 0 = not present
  - Use DGIM to estimate counts of 1's for all items

Extensions

- In principle, you could count frequent pairs or even larger sets the same way
  - One stream per itemset
- Drawbacks:
  - Only approximate
  - Number of itemsets is way too big

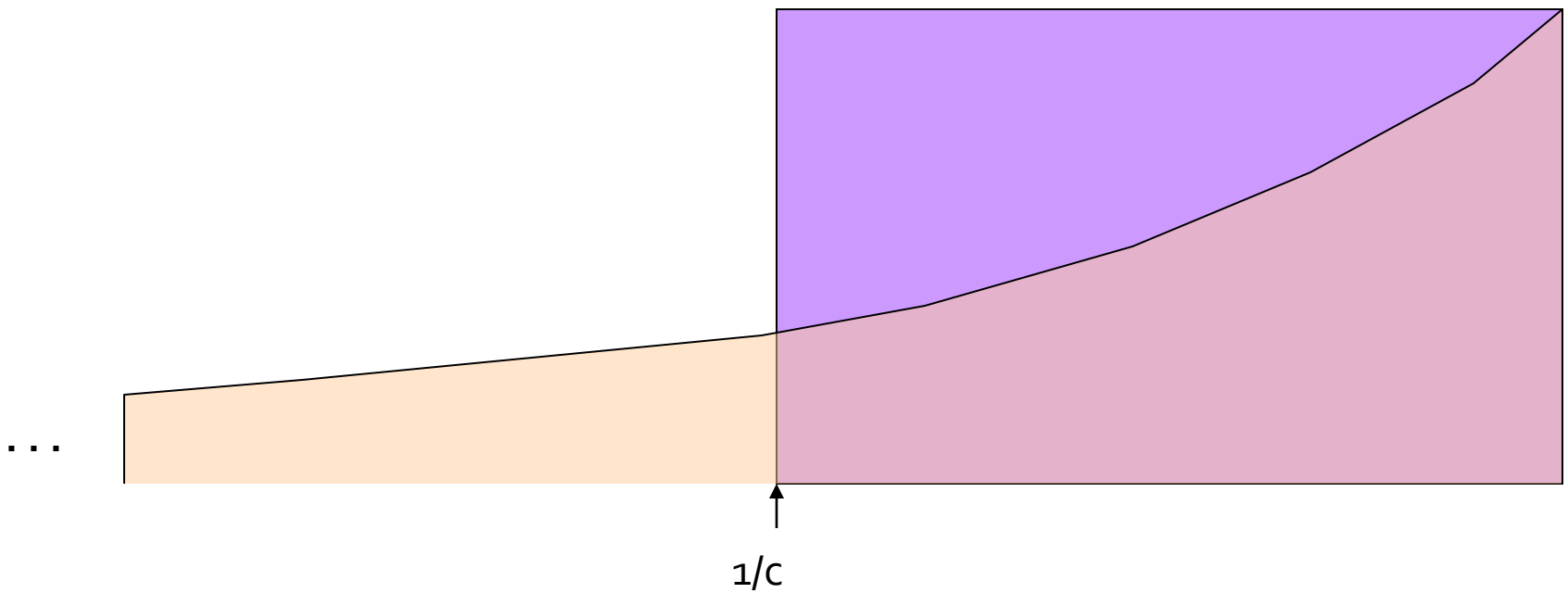
Exponentially Decaying Windows

- Exponentially decaying windows: A heuristic for selecting likely frequent itemsets
  - What are "currently" most popular movies?
    - Instead of computing the raw count in last N elements
    - Compute a smooth aggregation over the whole stream
- If stream is $a_1, a_2, \ldots$ and we are taking the sum of the stream, take the answer at time t to be: $= \Sigma_{i = 1,2,\ldots,t} a_i e^{-c(t-i)}$ (or, $\Sigma_{i = 1,\ldots,t} a_i (1-c)^{t-i}$)
  - c is a constant, presumably tiny, like 10^{-6} or 10^{-9}

Example: Counting Items

- If each a_i is an "item" we can compute the characteristic function of each possible item x as an E.D.W.
- That is: $\Sigma_{i = 1,2,\ldots,t} \delta_i e^{-c(t-i)}$
  - where $\delta_i = 1$ if $a_i = x$, and 0 otherwise
- Call this sum the "weight" item x

Sliding Versus Decaying Windows



Counting Items – (2)

- Suppose we want to find those items of weight at least ½
- Important property: Sum over all weights is $1/(1 - e^{-c})$ or very close to $1/[1 - (1 - c)] = 1/c$
- Thus:
  At most $2/c$ items have weight at least ½.

Extension to Larger Itemsets*

- Count (some) itemsets in an E.D.W.
- When a basket *B* comes in:
  1. Multiply all counts by (1-*c*);
  2. For uncounted items in *B*, create new count.
  3. Add 1 to count of any item in *B* and to any counted itemset contained in *B*.
  4. Drop counts < ½.
  5. Initiate new counts (next slide).

* Informal proposal of Art Owen

Initiation of New Counts

- Start a count for an itemset $S \subseteq B$ if every proper subset of S had a count prior to arrival of basket B
- Example: Start counting {i, j} iff both i and j were counted prior to seeing B
- Example: Start counting {i, j, k} iff {i, j}, {i, k}, and {j, k} were all counted prior to seeing B

How Many Counts?

- Counts for single items < (2/c) times the average number of items in a basket
- Counts for larger itemsets = ??. But we are conservative about starting counts of large sets.
  - If we counted every set we saw, one basket of 20 items would initiate 1M counts.