Mining Data Streams

(Part 2)
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
(1) 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
Applications

- **Example: Email spam filtering**
  - We know 1 billion “good” email addresses
    - Or, each user has a list of trusted 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

- **Content filtering:**
  - You want to make sure the user does not see the same ad multiple times

- **Web cache filtering:**
  - Has this piece of content been requested before? Then cache it now.
First Cut Solution (1)

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 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
  $|B| = 1$GB = 8 billion bits

- If the email address is in $S$, then it surely hashes to a bucket that has the bit 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
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?

\[
1 - \left(1 - \frac{1}{n}\right)^n \approx 1 - e^{-m/n}
\]

Probability some target \( X \) not hit by a dart

Equivalent to \( 1/e \)
as \( n \to \infty \)

Probability at least one dart hits target \( X \)

Approximation is especially accurate when \( n \) is large
Analysis: Throwing Darts (3)

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

- **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 $0$s
  - Hash each element $s \in S$ using each hash function $h_i$, set $B[h_i(s)] = 1$ (for each $i = 1, \ldots, k$) *(note: we have a single array $B$!)*
- Run-time:
  - When a stream element with key $x$ arrives
    - If $B[h_i(x)] = 1$ for all $i = 1, \ldots, k$ then declare that $x$ is in $S$
    - That is, $x$ hashes to a bucket set to 1 for every hash function $h_i(x)$
    - 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 $\left(1 - e^{-km/n}\right)$

- 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 $= \left(1 - e^{-km/n}\right)^k$
Bloom Filter – Analysis (2)

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

- What happens as we keep increasing \( k \)?

- Optimal value of \( k \): \( \frac{n}{m} \ln(2) \)
  - In our case: Optimal \( k = 8 \ln(2) = 5.54 \approx 6 \)
    - Error at \( k = 6 \): \( (1 - e^{-3/4})^6 = 0.0216 \)

Optimal \( k \): \( k \) which gives the lowest false positive probability
Bloom filters allow for filtering / set membership

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

Is it better to have 1 big B or k small Bs?
- It is the same: \((1 - e^{-km/n})^k\) vs. \((1 - e^{-m/(n/k)})^k\)
- But keeping 1 big B is simpler
(2) 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?**
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

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

$$
(1 - 2^{-r})^m
$$

- Prob. all $m$ elements 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 \(m \ll 2^r\), 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 \(m \gg 2^r\), 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 \(m!\)
Why It Doesn’t Work

- **E[2^R]** is actually infinite
  - Observing R has some probability
  - Probability halves when \( R \rightarrow R+1 \), but value doubles
  - Each possible large R contributes to exp. value

- **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
(3) Computing 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 $k^{\text{th}}$ (frequency) moment is

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

This is the same way as moments are defined in statistics. But there one typically “centers” the moment by subtracting the mean.
Special Cases

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

- **0\textsuperscript{th} moment** = number of distinct elements
  - The problem just considered
- **1\textsuperscript{st} moment** = count of the numbers of elements = length of the stream
  - Easy to compute, so not particularly useful
- **2\textsuperscript{nd} moment** = *surprise number* \( S \) = a measure of how uneven the distribution is
  - Very useful
Moments

- Third Moment is Skew:

- Fourth moment: Kurtosis
  - peakedness (width of peak), tail weight, and lack of shoulders (distribution primarily peak and tails, not in between).
Example: Surprise Number

- **Measure of how uneven the distribution is**

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

- **Item counts** \( m_i : 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 \)  
  \[ \text{Surprise } S = 910 \]

- **Item counts** \( m_i : 90, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 \)  
  \[ \text{Surprise } S = 8,110 \]
AMS Method

- AMS method works for all moments
- Gives an unbiased estimate
- We will just concentrate on the 2\textsuperscript{nd} moment
  - Will generalize later
- 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 $m_i$ 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 2\textsuperscript{nd} moment ($\sum \binom{m_i^2}{2}$) is:
  $$S = f(X) = n \cdot (2 \cdot c - 1)$$
  - Note, we will keep track of multiple $X$s, ($X_1, X_2, \ldots X_k$)
  and our final estimate will be $S = \frac{1}{k} \sum_{j=1}^{k} f(X_j)$
**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 \( (c_1=m_a, \ c_2=m_a-1, \ c_3=m_b) \)
- \( E[f(X)] = \frac{1}{n} \sum_{t=1}^{n} n(2c_t - 1) \)
  \[ = \frac{1}{n} \sum_i n \left(1 + 3 + 5 + \cdots + 2m_i - 1\right) \]

- 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: 
\[
\sum_{i=1}^{m_i}(2i - 1) = 2 \frac{m_i(m_i+1)}{2} - m_i = (m_i)^2
\]

Then 
\[ E[f(X)] = \frac{1}{n} \sum_i n (m_i)^2 \]

So, 
\[ E[f(X)] = \sum_i (m_i)^2 = S \]

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 $f(X)$:
  - For $k=2$ we used $n \ (2 \cdot c - 1)$
  - For $k=3$ we use: $n \ (3 \cdot c^2 - 3c + 1)$ (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: $2c - 1 = c^2 - (c - 1)^2$
  - For $k=3$: $c^3 - (c-1)^3 = 3c^2 - 3c + 1$

- Generally: Estimate $f(X) = n \ (c^k - (c - 1)^k)$
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 $X$ 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 / reservoir sampling!)
    - 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
Problems on Data Streams

- Filtering a data stream
  - Select elements with property $x$ from the stream

- Counting distinct elements
  - Number of distinct elements in the stream

- Estimating moments
  - Estimate avg./std. dev. of elements in stream