Course: Data mining
Lecture: Mining data streams

Aristides Gionis
Department of Computer Science
Aalto University

visiting in Sapienza University of Rome
fall 2016
reading assignment

- LRU book: chapter 4

- optional reading
  - paper by Alon, Matias, and Szegedy
    [Alon et al., 1999]
  - paper by Charikar, Chen, and Farach-Colton
    [Charikar et al., 2002]
  - paper by Cormode and Muthukrishnan
    [Cormode and Muthukrishnan, 2005]
data streams

- a data stream is a **massive** sequence of data
- too large to store (on disk, memory, cache, etc.)
- **examples:**
  - social media (e.g., twitter feed, foursquare checkins)
  - sensor networks (weather, radars, cameras, etc.)
  - network traffic (trajectories, source/destination pairs)
  - satellite data feed
- how to deal with such data?
- what are the issues?
issues when working with data streams

- **space**
  - data size is very large
  - often not possible to store the whole dataset
  - inspect each data item, make some computations, do not store it, and never get to inspect it again
  - sometimes data is stored, but making one single pass takes a lot of time, especially when the data is stored on disk
  - can afford a small number of passes over the data

- **time**
  - data “flies by” at a high speed
  - computation time per data item needs to be small
data streams

- data items can be of complex types
  - documents (tweets, news articles)
  - images
  - geo-located time-series
  - ...

- to study basic algorithmic ideas we abstract away application-specific details

- consider the data stream as a sequence of numbers
data-stream model

... 23 5 7 12 9 2 34 89 47 8 11 29 63 42 3 15 19 21 5 41 22...

input

memory

algorithm

output
(any time)

31
data-stream model

• **stream**: $m$ elements from universe of size $n$, e.g.,

$$\langle x_1, x_2, \ldots, x_m \rangle = 6, 1, 7, 4, 9, 1, 5, 1, 5, \ldots$$

• **goal**: compute a function over the elements of the stream, e.g., median, number of distinct elements, quantiles, . . .

• **constraints**:

1. limited working memory, sublinear in $n$ and $m$
   e.g., $O(\log n + \log m)$,
2. access data sequentially
3. limited number of passes, in some cases only one
4. process each element quickly, e.g., $O(1)$, $O(\log n)$, etc.
warm up: computing some simple functions

- assume that a number can be stored in $O(\log n)$ space
- max, min can be computed with $O(\log n)$ space
- sum, mean (average) need $O(\log n + \log m)$ space

$$\mu_X = \mathbb{E}[X] = \mathbb{E}[x_1, \ldots, x_m] = \frac{1}{m} \sum_{i=1}^{m} x_i$$

- what about variance?

$$\text{Var}[X] = \text{Var}[x_1, \ldots, x_m] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_X)^2$$

- two passes? one pass?

Data mining — Mining data streams
how to tackle massive data streams?

• a general and powerful technique: \textit{sampling}

• idea:
  ① keep a random sample of the data stream
  ② perform the computation on the sample
  ③ extrapolate

• example: compute the median of a data stream

  (how to extrapolate in this case?)

• but ... how to keep a random sample of a data stream?
reservoir sampling

- **problem**: take a uniform sample \( s \) from a stream of unknown length
- **algorithm**:
  - initially \( s \leftarrow x_1 \)
  - on seeing the \( t \)-th element, \( s \leftarrow x_t \) with probability \( 1/t \)
- **analysis**:
  - what is the probability that \( s = x_i \) at some time \( t \geq i \)?

\[
\Pr[s = x_i] = \frac{1}{i} \cdot \left(1 - \frac{1}{i+1}\right) \cdots \left(1 - \frac{1}{t-1}\right) \cdot \left(1 - \frac{1}{t}\right) \\
= \frac{1}{i} \cdot \frac{i}{i+1} \cdots \frac{t-2}{t-1} \cdot \frac{t-1}{t} = \frac{1}{t}
\]

- how much space? \( \mathcal{O}(\log n) \)
- to get \( k \) samples we need \( \mathcal{O}(k \log n) \) bits
infinite data-stream model

... 23 5 7 12 9 2 34 89 47 8 11 29 63 42 3 15 19 21 5 41 22...

input

memory

algorithm

36

output (any time)

time

Data mining — Mining data streams
infinite data-stream model

... 23 5 7 12 9 2 34 89 47 8 11 29 63 42 3 15 19 21 5 41 22...

input

memory

algorithm

36

output

(any time)

time
sliding-window data-stream model
sliding-window data-stream model

... 23 5 7 12 9 2 34 89 47 8 11 29 63 42 3 15 19 21 5 41 22...

time

output
(any time)

input

25

memory

algorithm

Data mining — Mining data streams
sliding-window data-stream model

\[
\begin{array}{cccccccccccccccc}
23 & 5 & 7 & 12 & 9 & 2 & 34 & 89 & 47 & 8 & 11 & 29 & 63 & 42 & 3 & 15 & 19 & 21 & 5 & 41 & 22 & \ldots
\end{array}
\]

input

memory

algorithm

\begin{array}{c}
\text{time}
\end{array}

\begin{array}{c}
\text{output (any time)}
\end{array}

\begin{array}{c}
32
\end{array}

Data mining — Mining data streams
sliding-window data-stream model

• does sliding-window model makes computation easier or harder?
• how to compute sum?
• how to keep a random sample?
• all computations can be done with $O(w)$ space
• can we do better?
priority sampling for sliding window

- maintain a uniform sample from the last \( w \) items
- reservoir sampling does not work in this model
- **algorithm:**
  1. for each \( x_i \) we pick a random value \( v_i \in (0, 1) \)
  2. for window \( \langle x_{j-w+1}, \ldots, x_j \rangle \) return \( x_i \) with smallest \( v_i \)
- to do this, maintain set of all elements in sliding window whose \( v \) value is minimal among all subsequent values
priority sampling for sliding window

... 23 5 7 12 9 2 34 89 47 8 11 29 63 ...

.64 .12 .31 .84 .27 .56 .91
priority sampling for sliding window
priority sampling for sliding window
priority sampling for sliding window

... 23  5  7  12  9  2  34  89  47  8  11  29  63 ...

.64  .12  .31  .84  .27  .56  .91  .42
priority sampling for sliding window
priority sampling for sliding window

Data mining — Mining data streams
priority sampling for sliding window

Data mining — Mining data streams
priority sampling for sliding window
priority sampling for sliding window

Data mining — Mining data streams
priority sampling for sliding window
priority sampling for sliding window

- **Correctness 1**: in any given window each item has equal chance to be selected as a random sample
- **Correctness 2**: each removed minimal element has a smaller element that comes after

- **Space efficiency**: how many minimal elements do we expect at any given point?
  - $O(\log w)$
  - so, expected space requirement is $O(\log w \log n)$

- **Time efficiency**: maintaining list of minimal elements requires $O(\log w)$ time
mining data streams

- what are real-world applications?
- imagine monitoring a social feed stream
  - a stream of hashtags in twitter
  - what are interesting questions to ask?
  - do data stream considerations (space/time) really matter?
how to tackle massive data streams?

- a general and powerful technique: sketching
- general idea:
  - apply a linear projection that takes high-dimensional data to a smaller dimensional space
  - post-process lower dimensional image to estimate the quantities of interest
computing statistics on data streams

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each $x_i$ is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of $i$
- define the $k$-th frequency moment

$$F_k = \sum_{i=1}^{n} m_i^k$$

- $F_0$ is the number of distinct elements
- $F_1$ is the length of the sequence
- $F_2$ is the second moment: index of homogeneity, size of self-join, and other applications
- $F^*_\infty$ frequency of most frequent element
computing statistics on data streams

• how much space I need to compute the frequency moments in a straightforward manner?

• how to compute the frequency moments using less than $O(n \log m)$ space?

• problem studied by Alon, Matias, Szegedy [Alon et al., 1999]

• sketching: create a sketch that takes much less space and gives an estimation of $F_k$
estimating the number of distinct values ($F_0$)

[Flajolet and Martin, 1985]

- consider a bit vector of length $O(\log n)$
- initialize all bits to 0
- upon seen $x_i$, set:
  - the 1-st bit with probability $1/2$
  - the 2-nd bit with probability $1/4$
  - ...  
  - the $i$-th bit with probability $1/2^i$
- important: bits are set deterministically for each $x_i$
- let $R$ be the index of the largest bit set
- return $Y = 2^R$
estimating the number of distinct values \( (F_0) \)

[Flajolet and Martin, 1985]

intuition:

- the \( i \)-th bit is set with probability \( \frac{1}{2^i} \)
- e.g., after seeing roughly 32 distinct elements, we expect to get the 5-th bit set
- if the bit vector is 00000011111 the estimate is 32
estimating number of distinct values \((F_0)\)

**Theorem.** For every \(c > 2\), the algorithm computes a number \(Y\) using \(O(\log n)\) memory bits, such that the probability that the ratio between \(Y\) and \(F_0\) is not between \(1/c\) and \(c\) is at most \(2/c\).
estimating $F_2$

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each $x_i$ is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of $i$
- $F_k = \sum_{i=1}^{n} m_i^k$

**algorithm:**
- hash each $i \in \{1, \ldots, n\}$ to a random $\epsilon_i \in \{-1, +1\}$
- maintain sketch $Z = \sum_i \epsilon_i m_i$
  just need space $O(\log n + \log m)$
- take $X = Z^2$
- return the average $Y$ of $k$ such estimates $X_1, \ldots, X_k$
- $Y = \frac{1}{k} \sum_{j=1}^{k} X_j$ where $k = \frac{16}{\lambda^2}$
The expectation of the estimate is correct.

\[
\mathbb{E} [X] = \mathbb{E} [Z^2] = \mathbb{E} \left[ \left( \sum_{i=1}^{n} \epsilon_i m_i \right)^2 \right] = \sum_{i=1}^{n} m_i^2 \mathbb{E} [\epsilon_i^2] + 2 \sum_{i<j} m_i m_j \mathbb{E} [\epsilon_i] \mathbb{E} [\epsilon_j] = \sum_{i=1}^{n} m_i^2 = F_2
\]
accuracy of the estimate

easy to show

\[ E \left[ X^2 \right] = \sum_{i=1}^{n} m_i^4 + 6 \sum_{i<j} m_i^2 m_j^2 \]

which gives

\[ \text{Var} [X] = E \left[ X^2 \right] - E [X]^2 = 4 \sum_{i<j} m_i^2 m_j^2 \leq 2 F_2^2 \]

and by Chebyshev’s inequality

\[ \Pr[|Y - F_2| \geq \lambda F_2] \leq \frac{\text{Var} [Y]}{\lambda^2 F_2^2} = \frac{\text{Var} [X]/k}{\lambda^2 F_2^2} \leq \frac{2 F_2^2/k}{\lambda^2 F_2^2} = \frac{2}{k \lambda^2} = \frac{1}{8} \]
finding frequent items in a data stream

• optional reading:
  paper by Charikar, Chen, and Farach-Colton
  [Charikar et al., 2002]
finding frequent items in a data stream

- consider again a data stream
- \( X = (x_1, x_2, \ldots, x_m) \) a data stream
- each \( x_i \) is a member of the set \( N = \{1, \ldots, n\} \)
- \( m_i = |\{j : x_j = i\}| \) the number of occurrences of \( i \)
- \( f_i = m_i/m \) the frequency of item \( i \)

- problem : estimate most frequent items in data stream
finding frequent items in a data stream

• problem formalization
• rename items \( \{o_1, \ldots, o_n\} \) so that \( m_1 \geq \ldots \geq m_n \)
• given \( k < n \) want to return top-\( k \) items \( o_1, \ldots, o_k \)
finding frequent items in a data stream

- problem formalization — first attempt

- problem $\text{FindCandidateTop}(X, k, \ell)$
  - given stream $X$ and integers $k$ and $\ell$
  - return list of $\ell$ items, so that top most frequent $k$ items of $X$ occur in the list

- should return all most frequent items
finding frequent items in a data stream

- \textbf{FindCandidateTop}(X, k, \ell) can be too hard to solve
- consider the case \( m_k = m_{\ell+1} + 1 \)
  - i.e., number of occurrences of \( k \)-th frequent item exceeds only by 1 the number of occurrences of the \((\ell + 1)\)-th frequent item
- almost impossible to find a list that contains the \( k \) most frequent items
finding frequent items in a data stream

• problem formalization — second attempt

• problem \text{FindApproxTop}(X, k, \epsilon)
  
  – given stream \(X\), integer \(k\), and real \(\epsilon \leq 1\)
  
  – return list of \(k\) items, so that for each item \(i\) in the list it is \(m_i \geq (1 - \epsilon) m_k\)

• no guarantee to return all most frequent items, but if return an item it should be frequent enough
finding frequent items in a data stream

• problem: \texttt{FindCandidateTop}(X, k, \ell)

• algorithm: \texttt{Sampling}

• modification of \textit{reservoir sampling}
  – keep a list of sampled items, plus a counter for each item
  – if an item is sampled again, increment its counter
analysis of **Sampling** algorithm

- let $x$ the number of items need to keep in the sample
- probability to be included in the sample is $x/m$
- want to ensure that $o_k$ appears in the sample
- need to set $x/m$ at least $\mathcal{O}((\log m)/m_k)$
- so $x$ should be at least $\mathcal{O}((\log m)/f_k)$
- so we have solution for $\text{FindCandidateTop}(X, k, \mathcal{O}((\log m)/f_k))$
- **limitation**: it requires knowing $m$ and $f_k$
finding frequent items in a data stream

• problem: \textsc{FindApproxTop}(X, k, \epsilon)

• algorithm: \textsc{CountSketch}
  – based on sketching techniques

• intuition
  – use a hash function \( s \) and a counter \( c \)
  – function \( s \) hashes objects to \( \{-1, +1\} \)
  – for each item \( o_i \) seen in the stream, set \( c \leftarrow c + s[o_i] \)
  – then \( \mathbb{E}[c \cdot s[o_i]] = m_i \) (prove it!)
  – so, estimate \( m_i \) by \( c \cdot s[o_i] \)
the **CountSketch** algorithm

- problem with using one hash function and one counter
  - very high variance

- remedy 1
  - use $t$ hash functions $s_1, \ldots, s_t$ and $t$ counters $c_1, \ldots, c_t$
  - for each item $o_i$ seen in the stream,
    set $c_j \leftarrow c_j + s_j[o_i]$, for all $j = 1, \ldots, t$
  - to estimate $m_i$ take median of $\{c_1 \cdot s_1[o_i], \ldots, c_t \cdot s_t[o_i]\}$
    (as before $\mathbb{E}[c_j \cdot s_j[o_i]] = m_i$ for all $j = 1, \ldots, t$)
the **CountSketch** algorithm

- problem with previous idea
  - high-frequency items (e.g., \( o_1 \)) may spoil estimates of lower-frequency items (e.g., \( o_k \))

- remedy 2
  - do not update all counters with all items
  - replace each counter with a hash table of \( b \) counters
  - items update different subsets of counters, one per hash table
  - each item gets enough high-confidence estimates (those avoiding collisions with high-frequency elements)
the **CountSketch** algorithm

- use parameters $t$ and $b$
- let $h_1, \ldots, h_t$ be hash functions from items to $1, \ldots, b$
- let $s_1, \ldots, s_t$ be hash functions from items to $\{-1, +1\}$
- consider $t \times b$ table of counters

- for each item $o_i$ seen in the stream, set $h_j[o_i] \leftarrow h_j[o_i] + s_j[o_i]$, for all $j = 1, \ldots, t$

- to estimate $m_i$ take median of
  \[
  \{h_1[o_i] \cdot s_1[o_i], \ldots, h_t[o_i] \cdot s_t[o_i]\}
  \]
an improved data stream summary

- the **CountMinSketch** data stream summary
- optional reading
  paper by Cormode and Muthukrishnan
  [Cormode and Muthukrishnan, 2005]
the **CountMinSketch** data stream summary

- **limitations** of existing sketches
  - model limitations (a sequence of items / numbers)
  - space required is $O\left(\frac{1}{\epsilon^2}\right)$
    recall that guarantees are quantified by $\epsilon$, $\delta$ parameters
    $\epsilon$ : accuracy
    $\delta$ : probability of failure
  - update time proportional to the whole sketch
  - different sketch for each summary

- **CountMinSketch** addresses all those limitations
incremental data-stream model

- consider a vector $\mathbf{x}(t) = \{x_1(t), \ldots, x_n(t)\}$
- number of coordinates $n$ potentially very large
- $\mathbf{x}(t)$ the values of vector at time $t$
- at each time $t$ a vector coordinate is updated
- data stream : updates $(i_t, c_t)$ for $t = 1, \ldots$
- then
  $$x_{i_t}(t) \leftarrow x_{i_t}(t-1) + c_t$$
  and
  $$x_j(t) \leftarrow x_j(t-1), \text{ for } j \neq i_t$$
incremental data-stream model

• generalization of previous model
  previous model was $c_t = 1$

• special cases
  – cash register model: $c_t \geq 0$
  – turnstile model: $c_t$ can be negative
    – non-negative turnstile model: $x_i(t) \geq 0$
    – general turnstile model: $x_i(t)$ can be negative
the **CountMinSketch** data stream summary

- interesting queries that we would like to handle
  - point query $Q(i)$: approximate $x_i$
  - range query $Q(\ell, r)$: approximate $\sum_{i=\ell}^{r} x_i$
  - inner product $Q(x, y)$: approximate $x \cdot y = \sum_{i=1}^{n} x_i y_i$
  - $\phi$-quantiles
  - heavy-hitters: most frequent items
    - given frequency threshold $\phi$, find items $i$ for which $x_i \geq (\phi - \epsilon)\|x\|_1$ for some $\epsilon < \phi$
the **CountMinSketch** data structure

- similar to **CountSketch**
- a table of counters $C$ of dimension $d \times w$
- $d$ hash functions $h_1, \ldots, h_d$ from $\{1, \ldots, n\}$ to $\{1, \ldots, w\}$ chosen from a pairwise-independent family

\[
C = \begin{pmatrix}
C[1, 1] & \cdots & C[1, w] \\
\vdots & \ddots & \vdots \\
C[d, 1] & \cdots & C[d, w]
\end{pmatrix}
\]

- parameters $d$ and $w$ specify the space requirements depend on error bounds we want to achieve
**CountMinSketch** : update summary

• given \((i_t, c_t)\) update one counter in each row of \(C\), in particular

\[
C[j, h_j(i_t)] \leftarrow C[j, h_j(i_t)] + c_t
\]

for all \(j = 1, \ldots, d\)
CountMinSketch: point query

- the answer to $Q(i)$ is $\hat{x}_i = \min_j C[j, h_j(i)]$

- theorem: the estimate $\hat{x}_i$ satisfies
  
  (i) $x_i \leq \hat{x}_i$

  (ii) $\hat{x}_i \leq x_i + \epsilon \|x\|_1$ with prob at least $1 - \delta$
CountMinSketch

• similar type of estimates for other queries
  – range, inner product, etc.

• parameters are set to

\[ d = \left\lfloor \log \frac{1}{\delta} \right\rfloor \quad \text{and} \quad w = \left\lfloor \frac{1}{\epsilon} \right\rfloor \]

– improved space; instead of usual \( O\left(\frac{1}{\epsilon^2}\right) \)
– improved update time: access only \( d \) counters
references

The space complexity of approximating the frequency moments.

Finding frequent items in data streams.
In *International Colloquium on Automata, Languages, and Programming*, pages 693–703.

An improved data stream summary: the count-min sketch and its applications.

Probabilistic counting algorithms for data base applications.