Mining Data Streams-Approximate Heavy Hitters

Barna Saha

February 9, 2016

Finding Majority

- Input. An array A of length m with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- Problem. Find the Majority element.

Finding Majority

- Input. An array A of length m with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- Problem. Find the Majority element in linear time.

Finding Majority

- Input. An array A of length m with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- Problem. Find the Majority element in linear time.
- Compute median of A.

Finding Majority

- Input. An array A of length m with the promise that it has a majority element–a value that is repeated strictly more than $\frac{m}{2}$ times.
- Problem. Find the Majority element in linear time in a single left to right pass in "constant" space.

Finding Majority

- Problem. Find the Majority element in linear time in a single left to right pass in "constant" space.
- Algorithm.
  1. Set $count = 1$, $current = A(1)$.
  2. For $i = 2, 3, \ldots$
     2.1 If $count == 0$, set $current = A(i)$, $count = 1$,
     2.2 If $A(i) == current$, set $count = count + 1$
     2.3 Else set $count = count - 1$
  3. Return current

Finding Majority

- Problem. Find the Majority element in linear time in a single left to right pass in "constant" space.
- Algorithm.
  1. Set $count = 1$, $current = A(1)$.
  2. For $i = 2, 3, \ldots$
     2.1 If $count == 0$, set $current = A(i)$, $count = 1$,
     2.2 If $A(i) == current$, set $count = count + 1$
     2.3 Else set $count = count - 1$
  3. Return current
- Exercise. Given there exists a majority element, show that the above algorithm correctly returns the majority.

Heavy Hitter Problem

- Problem. Given an array A of length m, and a parameter k, find those values that occur at least $\frac{m}{k}$ times.

Heavy Hitter Problem

- Problem. Given an array A of length m, and a parameter k, find those values that occur at least $\frac{m}{k}$ times.
- Applications:
  1. **Computing popular products.** A could be all of the page views of products on *amazon.com* yesterday. The heavy hitters correspond to frequently viewed items.
  2. **Computing frequent search queries.** For example, A could be all of the searches on Google yesterday. The heavy hitters are then searches made most often.
  3. **Identifying heavy TCP flows.** Here, A is a list of data packets passing through a network switch, each annotated with a source-destination pair of IP addresses. The heavy hitters are then the flows that are sending the most traffic. This is useful for, among other things, to identify denial-of-service attacks.
  4. **Identifying volatile stocks.** Here, A is a list of stock trades.

Heavy Hitter Problem

- Can we solve Heavy Hitter Problem in small space? Ideally in $\tilde{O}(k)$ space.

Heavy Hitter Problem

- Can we solve Heavy Hitter Problem in small space? Ideally in $\tilde{O}(k)$ space.
- There is no algorithm that solves the Heavy Hitters problems in one pass while using a sublinear amount of auxiliary space.

ϵ-Approximate Heavy Hitter Problem

- Input is an array A of length m with two parameters ϵ and k.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in A is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in A

ϵ-Approximate Heavy Hitter Problem

- Input is an array A of length m with two parameters ϵ and k.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in A is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in A
- Why not set $\epsilon = 0$?

ϵ-Approximate Heavy Hitter Problem

- Input is an array A of length m with two parameters ϵ and k.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in A is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in A
- Why not set $\epsilon = 0$?
- Space usage grows proportionately with $\frac{1}{\epsilon}$.

ϵ-Approximate Heavy Hitter Problem

- Input is an array A of length m with two parameters ϵ and k.
- Output
  1. Every value that occurs at least $\frac{m}{k}$ times in A is in the list.
  2. Every value in the list occurs at least $\frac{m}{k} - \epsilon m$ times in A
- Why not set $\epsilon = 0$?
- Space usage grows proportionately with $\frac{1}{\epsilon}$.
- If we take $\epsilon = \frac{1}{2k}$, space usage is $\tilde{O}(k)$, all elements with frequency $\frac{m}{k}$ is in the list and the elements in the list have frequency at least $\frac{m}{2k}$.

Estimating Frequency of Elements

- Input Stream of m elements from a universe $[1, n]$: $A(1), A(2), ..., A(m)$.
- Frequency of an element $i \in [1, n]$ in the stream is $f_i = |t \mid A(t) = i|$.
- Problem
  - For $i \in [n]$, estimate f_i (Point Query)
  - For $\phi \in [0, 1]$, find all i with $f_i \geq \phi m$. (Heavy Hitter)

Count-Min Sketch

- Select an $\epsilon > 0$ and $\delta > 0$: ϵ denotes the error-parameter, and δ denotes our confidence.
- Select $d = \ln \frac{1}{\delta}$ hash functions $h_1, h_2, ..., h_d$ independently and randomly from a pair-wise independent hash family. Each $h_i : \{1, 2, ..., n\} \rightarrow \{1, 2, ..., w\}$ where $w = \frac{e}{\epsilon}$.
- Initialize a table T of dimension $d \times w$ all with 0.
- Update: At time t, when $A(t)$ arrives, do the following.
  - $T(1, h_1(A(t))) = T(1, h_1(A(t))) + 1$
  - $T(2, h_2(A(t))) = T(2, h_2(A(t))) + 1$
  - .
  - .
  - $T(d, h_d(A(t))) = T(d, h_d(A(t))) + 1$

http://research.neustar.biz/tag/count-min-sketch/

Count-Min Sketch:Point Query

- Problem For $i \in [n]$, estimate f_i
- Output An estimate $\hat{f}_i$ such that $f_i \leq \hat{f}_i \leq f_i + \epsilon ||\mathbf{f}||_1$
- Algorithm Construct Count-Min sketch. Return

$$\min_{l=1}^{d} T(l, h_l(i))$$

.

Count-Min Sketch:Point Query

- Algorithm Construct Count-Min sketch. Return

$$\min_{l=1}^{d} T(l, h_l(i))$$

.

- Each $T(l, h_l(i)) \geq f_i$. Hence $\min_{l=1}^{d} T(l, h_l(i)) \geq f_i$.
- Define an indicator random variable X_j^l, $j = 1, 2, ..n$ and $l = 1, 2, .., d$.

$$X_j^l = 1 \text{ if } h_l(j) = h_l(i), \text{ else } X_j^l = 0$$

- Define $Y = \sum_{j=1}^{j=n} f_j X_j^l$. Then $T(l, h_l(i)) = Y$.

Count-Min Sketch:Point Query

$$
\begin{aligned}
E[T(l, h_l(i))] = E[Y] &= \sum_{j=1}^{j=n} E[f_j X_j^l] = \sum_{j=1}^{j=n} f_j E[X_j^l] \\
&= \sum_{j=1}^{j=n} f_j Prob(h_l(j) = h_l(i)) \\
&= \sum_{j=1}^{j=n} \frac{f_j}{w} \text{ (h is picked from a pair-wise family)} \\
&= \frac{||\mathbf{f}||_1}{w}
\end{aligned}
$$

Count-Min Sketch:Point Query

$$
\begin{aligned}
Prob\,(T(l, h_l(i))] > \epsilon ||\mathbf{f}||_1) &= Prob\,(T(l, h_l(i))] > w\epsilon E[T(l, h_l(i))]) \\
&\leq \frac{1}{w\epsilon} \quad \text{(By Markov Inequality)} \\
&= \frac{1}{e} \quad \text{(since } w = \frac{e}{\epsilon})
\end{aligned}
$$

Count-Min Sketch:Point Query

$$
\begin{aligned}
& Prob\left(\min_{l=1}^{d} T(l, h_l(i))] > \epsilon||\mathbf{f}||_{\mathbf{1}}\right) \\
& = Prob\left(\bigcap_{l=1}^{d} T(l, h_l(i))] > \epsilon||\mathbf{f}||_{\mathbf{1}}\right) \\
& = \prod_{l=1}^{d} Prob\left(T(l, h_l(i))] > \epsilon||\mathbf{f}||_{\mathbf{1}}\right) \leq \left(\frac{1}{e}\right)^{\ln\frac{1}{\delta}} = \delta
\end{aligned}
$$

- Hence $Prob\left(\min_{l=1}^{d} T(l, h_l(i))] \leq \epsilon||\mathbf{f}||_{\mathbf{1}}\right) \geq 1 - \delta$.
- Therefore $f_i \leq \hat{f}_i \leq f_i + \epsilon||\mathbf{f}||_{\mathbf{1}}$ with probability $\geq 1 - \delta$.
- Space$= O(wd) = O(\frac{1}{\epsilon}\ln\frac{1}{\delta})$.

Count-Min Sketch:Heavy Hitter

- Set $\delta' = \frac{\delta}{n}$, using space $O(\frac{1}{\epsilon} \ln \frac{n}{\delta})$ obtain estimates such that "For All is $f_i \leq \hat{f}_i \leq f_i + \epsilon m$.

- Use a min-heap to store the heavy-hitters.
  1. Keep a count on the total number of elements m arrived so far.
  2. When item $A(i)$ arrives, compute its estimated frequency from the count-min sketch data structure.
  3. If the count is above $\frac{m}{k}$, insert it in the heap with key $Count(A(i))$, and delete any previous occurrence of $A(i)$ from the heap.
  4. If any element in the heap has count less than $\frac{m}{k}$ delete it through operations such as *Find-Min* and *Extract*-Min.
  5. Assuming no large error happens in the Count-Min sketch, the heap size is bounded by $2k$. Why? Therefore extra work per item to process the heap is $O(\log k)$.
  6. At the end, scan the heap, and for every item whose estimated frequency is $\geq \frac{m}{k}$ return it as a heavy hitter.

Count-Min Sketch:Heavy Hitter

- Set $\delta' = \frac{\delta}{n}$, using space $O(\frac{1}{\epsilon} \ln \frac{n}{\delta})$ obtain estimates such that "For All is $f_i \leq \hat{f}_i \leq f_i + \epsilon m$.
- Set $\delta' = \frac{\delta}{m}$, using space $O(\frac{1}{\epsilon} \ln \frac{m}{\delta})$ obtain estimates such that "For All $t = 1, 2, .., m$s the estimated frequency is within the error-range.
- Use a min-heap to store the heavy-hitters.
  1. Keep a count on the total number of elements m arrived so far.
  2. When item $A(i)$ arrives, compute its estimated frequency from the count-min sketch data structure.
  3. If the count is above $\frac{m}{k}$, insert it in the heap with key $Count(A(i))$, and delete any previous occurrence of $A(i)$ from the heap.
  4. If any element in the heap has count less than $\frac{m}{k}$ delete it through operations such as *Find-Min* and *Extract*-Min.
  5. Assuming no large error happens in the Count-Min sketch, the heap size is bounded by $2k$. Why? Therefore extra work per item to process the heap is $O(\log k)$.
  6. At the end, scan the heap, and for every item whose estimated frequency is $\geq \frac{m}{k}$ return it as a heavy hitter.

Miscelleneous

- Implementation: http://www.cs.rutgers.edu/~muthu/massdal-code-index.html
- Twitter's algebird and ClearSpring's stream-lib offer implementations of Count-Min sketch and various other data structures applicable for stream mining applications.
- Application: Mostly a list of papers that use CM-sketch
  - http://sites.google.com/site/countminsketch/cm-eclectics
  - http://sites.google.com/site/countminsketch/compressed-sensing
  - http://sites.google.com/site/countminsketch/databases

Boosting by Median

- Suppose there is an Algorithm that returns an estimate $\hat{F}$ of a true estimate F such that $|\hat{F} - F|$ is small with probability $\frac{7}{8}$.
- How can we design an algorithm that will return an estimate G of F such that $|G - F|$ is small with probability $99/100$? (In general $1 - \delta$)

Boosting by Median

- Suppose there is an Algorithm that returns an estimate $\hat{F}$ of a true estimate F such that $|\hat{F} - F|$ is small with probability $\frac{7}{8}$.
- How can we design an algorithm that will return an estimate G of F such that $|G - F|$ is small with probability $99/100$? (In general $1 - \delta$)
- Run $s = 6 \log \frac{1}{\delta}$ independent copies of the Algorithm to obtain estimates $\hat{F}_1, \hat{F}_2, ..., \hat{F}_s$. Set $G = median_{i=1}^{s} \hat{F}_i$.

Boosting by Median

- What is the probability that the median is a bad estimate?

Boosting by Median

- What is the probability that the median is a bad estimate?
- Either all $\lfloor \frac{s}{2} \rfloor$ copies with estimate below G are bad or, $\lfloor \frac{s}{2} \rfloor$ copies with estimate above G are bad. That is there are $3 \log \frac{1}{\delta}$ copies that are at least bad for G to be a bad estimate.

Boosting by Median

- What is the probability that the median is a bad estimate?
- Either all $\lfloor \frac{s}{2} \rfloor$ copies with estimate below G are bad or, $\lfloor \frac{s}{2} \rfloor$ copies with estimate above G are bad. That is there are $3 \log \frac{1}{\delta}$ copies that are at least bad for G to be a bad estimate.
- Define an indicator random variable X_i which is 1 if the ith estimate $\hat{F}_i$ is bad. Then $E[X_i] = \frac{1}{8}$.

Boosting by Median

- What is the probability that the median is a bad estimate?
- Either all $\lfloor \frac{s}{2} \rfloor$ copies with estimate below G are bad or, $\lfloor \frac{s}{2} \rfloor$ copies with estimate above G are bad. That is there are $3 \log \frac{1}{\delta}$ copies that are at least bad for G to be a bad estimate.
- Define an indicator random variable X_i which is 1 if the ith estimate $\hat{F}_i$ is bad. Then $E[X_i] = \frac{1}{8}$.
- Then the number of bad estimates is $Y = \sum_i X_i$. and $E[Y] = \frac{6 \log \frac{1}{\delta}}{8} = \frac{3}{4} \log \frac{1}{\delta}$

Boosting by Median

- What is the probability that the median is a bad estimate?
- Either all $\lfloor \frac{s}{2} \rfloor$ copies with estimate below G are bad or, $\lfloor \frac{s}{2} \rfloor$ copies with estimate above G are bad. That is there are $3 \log \frac{1}{\delta}$ copies that are at least bad for G to be a bad estimate.
- Define an indicator random variable X_i which is 1 if the ith estimate $\hat{F}_i$ is bad. Then $E[X_i] = \frac{1}{8}$.
- Then the number of bad estimates is $Y = \sum_i X_i$. and $E[Y] = \frac{6 \log \frac{1}{\delta}}{8} = \frac{3}{4} \log \frac{1}{\delta}$
- Bound

$$Prob(Y > 3 \log \frac{1}{\delta})$$

using Chernoff's bound.

Boosting by Median

- Upper Tail version of Chernoff Bound. For $\epsilon > 1$

$$Prob(Y > E[Y](1+\epsilon)) \leq e^{-\frac{E[Y]\epsilon^2}{2+\epsilon}}$$

.

Boosting by Median

- Upper Tail version of Chernoff Bound. For $\epsilon > 1$

$$Prob(Y > E[Y](1+\epsilon)) \leq e^{-\frac{E[Y]\epsilon^2}{2+\epsilon}}$$

.

- $$Prob\left(Y > 3\log\frac{1}{\delta}\right) = Prob\left(Y > \frac{3}{4}\log\frac{1}{\delta}(1+3)\right)$$

$$\leq e^{-\frac{3}{4}\left(\log\frac{1}{\delta}\right)9\frac{1}{5}} < \delta$$

Versions of Chernoff Bound

Reference:
https://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/probabilityandcomputing.pdf