6.897: CSCI8980 Algorithmic Techniques for Big Data September 26, 2013

Lecture 7

Dr. Barna Saha *Scribe: Ali Mahdavi*

1 Developing a Good Hash Function for LSH

In the previous lecture we saw how to design locality sensitive hashing (LSH) for hamming and l_1 distance, as a solution to the (c, R)-Near Neighbor problem. Whenever we use LSH for the nearest neighbor search, using some distance measure, the main task is to come up with a good elementary hashing function. This elementary hash function (H) is then used to create a composite hash function (G) for LSH exactly as described in the previous lecture.

Here we discuss how to develop a good elementary hash function for the l_2 (euclidean) distance. Consider a point in a D-dimensional space $x = (x_1, x_2, ..., x_D)$;D coordinates. The hash family consists of hash functions of the form

$$H = \{h_{(\hat{a},b)} = \lfloor \frac{(\hat{a}, x)}{w} + b \rfloor\}$$

where

$\hat{a} = [a_1, a_2, ..., a_D]$; $a_i \sim \mathcal{N}(0, 1)$ *and* $b \in [0, w]$ is a random number chosen uniformly from $[0, w]$

- $\hat{a}$ is a random vector of length D whose entries are random i.i.d picks from standard normal distributions (mean=0, standard deviation=1)
- b is a random value between 0 and w; i.e. a random pick from $\mathcal{U}(0, w)$.
- w is a constant (e.g. $w = 4$).

We project each vector x on R_1 using a random vector $\hat{a}$, producing $r(x) = \sum_1^D a_i x_i$. In other terms we convert x from the D-dimensional space to a one dimension value, $r(x)$, and shift it by value b. The value of w determines the number of buckets on R_1. We mentioned in lecture 2 that the basic idea behind LSH is that two points that are close to each other should hash to different buckets. So, now we calculate the probability that two points, x and y, from the D-dimensional space do not produce the same hash value; $Pr(h_{\hat{a},b}(x) \neq h_{\hat{a},b}(y))$

for the resulting $r(x)$ and $r(y)$ there can be two situations based on where $r(x)$ and $r(y)$ are located on R_1:

$|r(x) - r(y)| > w$; the two points will fall into different buckets for sure, irrespective of the shift b

$|r(x) - r(y)| < w$; here the shift value, b, determines whether the resulting hash values will fall into different buckets. we may have

- $r(x)$ and $r(y)$ fall initially(before shift by b) into the same bucket: in this case they will have different values $if\ b \in \lfloor d, d + |r(x) - r(y)| \rfloor \Longrightarrow Pr(h_{\hat{a},b}(x) \neq h_{\hat{a},b}(y)) = \frac{|r(x)-r(y)|}{w}$ where d= min distance of $r(x)$ or $r(y)$ from the begin of the bucket

- $r(x)$ and $r(y)$ fall initially into different buckets: here we again derive the same result

  $Pr(h_{\hat{a},b}(x) \neq h_{\hat{a},b}(y)) = \frac{|r(x)-r(y)|}{w}$

Hence we have $Pr(h_{\hat{a},b}(x) = h_{\hat{a},b}(y)) = 1 - \frac{|r(x)-r(y)|}{w}$

We would like to show that $1 - \frac{|r(x)-r(y)|}{w} \backsimeq 1 - \frac{||x-y||_2}{w}$, because then we have LSH property and we are done.

Recall Johnson-Lindenstrauss theorem (refer to lecture 5, JL theorem part for details). In the JL lemma, we were dealing with a N by D matrix multiplied by a column vector (x), of length D. We can consider the current case as a simplification of it where we only have one row/sketch (one vector with length D) and only two points, x and y, instead of N points (could be any small constant number of points instead of N). In fact, using the same proof as we did in lecture 5, one can show if $z = r(x) - r(y)$, then $\mathsf{E}[z^2] = ||x-y||_2^2$ and $\sigma(z^2) = O(||x-y||_2^2)$. Therefore, $|r(x)-r(y)| = ||r(x)-r(y)||_2 = \Theta(||x-y||_2)$. The above is a rough proof, for detailed calculations see [1]. Note that here the randomness is in choosing the hash functions. Once the hash function is selected, the mapping is fixed.

some final points: Recall from the previous lecture that the parameter $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$ determines the goodness of the LSH family. The smaller the value of ρ, better that family we have in terms of space and query time.

- For the hamming distance the best LSH scheme guarantees $\rho = 1/C$.
- For the l_2 distance the best LSH scheme gives $\rho = 1/C^2$, however the above scheme only guarantees ρ which is slightly less than $\frac{1}{C}$.
- For implementations of the LSH scheme, see http://www.mit.edu/ andoni/LSH/

There are LSH schemes for other distances. For example, for Jackard distance, popularly used in document classification, the min-wise independent permutations serve as LSH family.

What We Covered so Far

- Some hashing schemes;

  universal hashing

  LSH

- Some sketching techniques;

  CM-sketch (sketching by hashing)

  computing the 2nd frequency moment using the linear sketch

  dimensionality reduction via sketching, JL lemma.

2 Sampling

Another way of designing efficient algorithms is by sampling. In a data stream setting, there are two characteristics of the stream we have to consider:

- the stream length, **m**.
- the length of the domain (**n**) from which each element of the stream (a_i) is coming, $a_i \in [0, n]$

based on whether **m** and/or **n** are known (or unknown) there will be different techniques for sampling.

2.1 Estimating F_k

In lecture 4 we learned about the 2nd frequency moment, $F_2 = \sum_{i=1}^{n} f_i^2$. We showed that there is a randomized algorithm for estimating F_2 within error $(1 \pm \epsilon)$ with probability at least $(1 - \delta)$ that takes space $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$.

Here we are interested in F_k (the kth frequency moment), as it provides useful statistics on a sequence. Let $A = (a_1, a_2, ..., a_m)$ be a sequence of elements (known stream length m), where each a_i is a member of $N = \{1, 2, ..., n\}$. Let $m_i = |\{j : A - j = i\}|$ denote the number of occurrences of i in the sequence A, we define F_k for each $k \geq 0$ as

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

the following algorithm gives us a space complexity of $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} k n^{(1-\frac{1}{k})})$, which is sub-linear in n and not logarithmic as for F_2.

The lower bound for estimating F_k is $\Omega(n^{1-\frac{2}{k}})$ (there are other recent algorithms that almost match this lower bound).

Algorithm to estimate F_k:

- Draw a uniform random index p with probability $\frac{1}{m}$. Suppose $a_p = l$.
- Starting from p, count how many times l reoccurs;i.e. how many time will we see $a_i = l$ again. Compute $r = |\{q : q \geq p, a_q = l\}|$
- Return $X = m(r^k - (r-1)^k)$
- We have $E(X) = F_k$; X is an unbiased estimator
- $Var(X) \leq n^{1-\frac{1}{k}} (F_k)^2$; the variance is large compared to $Var(Y) = constant * F_k$ which we had for F_2 in lecture 4)

To reduce the variance of this estimator we proceed as following (same approach we used for F_2 in lecture 4):

- Maintain $s_1 = O(\frac{kn^{1-\frac{1}{k}}}{\epsilon^2})$ such estimates $X_1, X_2, ..., X_{s_1}$. Take the average, $Y = \frac{1}{s_1}\sum_{i=1}^{s_1} X_i$, apply the Chebyshev bound.
- Maintain $s_2 = O(\log \frac{1}{\delta})$ of these average estimates, $Y_1, Y_2, ..., Y_{s_2}$ and take the median. Apply Chernoff's bound.
- Follows $(1 \pm \epsilon)$ approximation with probability $\geq (1-\delta)$.

Lemma 1. $E[X] = F_k$

$$\begin{aligned}
E[Y] &= \sum_{i=1}^{n}\sum_{j=1}^{f_i} E[X \mid i \textit{ is sampled on } j\textit{th occurrence}]\frac{1}{m} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{f_i} m((f_i - j + 1)^k - (f_i - j)^k)\frac{1}{m} \\
&= \sum_{i=1}^{n}\left[1^k + (2^k - 1^k) + (3^k - 2^k) + ... + (f_i^k - (f_i - 1)^k)\right] \\
&= F_k
\end{aligned}$$

Lemma 2. $Var[X] \leq kn^{1-\frac{1}{k}}(F_k)^2$

To estimate $Var(X) = E(X^2) - (E(X))^2$, we bound $E(X^2)$;

$$E[X^2] = \sum_{i=1}^{n}\sum_{j=1}^{f_i} E[X^2 \mid i \text{ is sampled on } j\text{th occurrence}]\frac{1}{m}$$

$$\begin{aligned}
&= \sum_{i=1}^{n}\sum_{j=1}^{f_i} m^2((f_i - j + 1)^k - (f_i - j)^k)^2\frac{1}{m} \\
&= m\sum_{i=1}^{n}\left[1^{2k} + (2^k - 1^k)^2 + (3^k - 2^k)^2 + ... + (f_i^k - (f_i - 1)^k)^2\right] \\
&\leq m\sum_{i=1}^{n} k1^{2k-1} + k2^{k-1}(2^k - 1^k) + ... + kf_i^{k-1}(f_i^k - (f_i - 1)^k)
\end{aligned}$$

the above inequality is obtained using $a^k - b^k = (a-b)(a^{k-1} + ba^{k-2} + .. + b^{k-1}) \leq (a-b)ka^{k-1}$ which holds for any numbers $a > b > 0$

$$\begin{aligned}
& m\sum_{i=1}^{n} k1^{2k-1} + k2^{k-1}(2^k - 1^k) + ... + kf_i^{k-1}(f_i^k - (f_i - 1)^k) \\
\leq\ & mk\sum_{i=1}^{n} f_i^{2k-1} = mkF_{2k-1} = kF_1F_{2k-1} \\
=\ & kF_1F_{2k-1} \leq kn^{1-\frac{1}{k}}\left(\sum_{i=1}^{n} f_i^k\right)^2 = kn^{1-\frac{1}{k}}(F_k)^2
\end{aligned}$$

For the last inequality we need to show the fact that for every n positive reals m_1,m_2,...,m_n

$$(\sum_{i=1}^{n} m_i)(\sum_{i=1}^{n} m_i^{2k-1}) \le n^{1-1/k}(\sum_{i=1}^{n} m_i)^2$$

(note that the sequence $m_1 = n^{1/k}$, $m_2 = ... = m_n = 1$ shows that this is tight, up to a constant factor)

Proof of above fact: Put $M = max_{1 \le i \le n} m_i$, then $M^k \le \sum_{i=1}^{n} m_i^k$ and hence

$$\begin{aligned}
(\sum_{i=1}^{n} m_i)(\sum_{i=1}^{n} m_i^{2k-1}) &\le (\sum_{i=1}^{n} m_i)(M^{k-1}\sum_{i=1}^{n} m_i^k) \\
&\le (\sum_{i=1}^{n} m_i)(\sum_{i=1}^{n} m_i^k)^{(k-1)/k}(\sum_{i=1}^{n} m_i^k) \\
&= (\sum_{i=1}^{n} m_i)(\sum_{i=1}^{n} m_i^k)^{(2k-1)/k} \\
&\le n^{1-1/k}(\sum_{i=1}^{n} m_i^k)^{1/k}(\sum_{i=1}^{n} m_i^k)^{(2k-1)/k} \\
&\le n^{1-1/k}(\sum_{i=1}^{n} m_i^k)^2
\end{aligned}$$

where the last inequality uses the fact that $(\sum_{i=1}^{n} m_i)/n \le (\sum_{i=1}^{n} m_i^k/n)^{1/k}$.

By the above fact, the definition of the random variables Y_i and the computation above we have,

$$Var(Y) \le E(X^2) \le kF_1F_{2k-1} \le kn^{1-1/k}F_k^2$$

for further details on approximating F_k see [2].

2.2 Reservoir sampling

Now consider the setting where we don't know the stream length m. The goal is to derive a uniform sample s from the stream.

- Initially $s = a_1$; store the first element as s
- On seeing the t-th element set $s = a_t$ with probability $\frac{1}{t}$; then we will have

  $Prob[s = a_i]$ =Probability of including the i-th element in the sample, which implies we do haven't included any of the other i-th to t-th elements.

  $Prob[s = a_i] = \frac{1}{i}\left(1 - \frac{1}{i+1}\right)\left(1 - \frac{1}{i+2}\right) ... \left(1 - \frac{1}{t}\right) = \frac{1}{t}$
- s gives us a uniform sample where each element appears with probability $1/t$ at time t

We apply a similar algorithm to get a uniform sample s of size k (i.e. suppose we have a reservoir of size k).

- Initially $s = \{a_1, a_2, ..., a_k\}$; initially empty
- On seeing the t-th element, pick a number $r \in [1, t]$ uniformly and randomly
- If $r \leq k$, replace the rth element by a_t
  otherwise (i.e. If $k < r \leq t$), do nothing
  we have $Prob[a_i \in s] = \frac{k}{i}\left(1 - \frac{1}{i+1}\right)\left(1 - \frac{1}{i+2}\right)...\left(1 - \frac{1}{t}\right) = \frac{k}{t}$
- s gives us a uniform sample (of size k) where each element appears with probability k/t at time t.

2.3 Priority sampling

Now we consider the case where each element i has weight w_i. We want to keep a sample s of size k such that can be used to estimate the total weight of arbitrary subsets (**subset sum query**). The approaches we discussed so far won't work in this case:

- Uniform Sampling: Will miss few heavy hitters, i.e. elements that are few but have a very high weight (unless we have a very high sampling probability)
- Weighted Sampling with Replacements: heavy hitters will be sampled several times (duplicated) and elements with a small weight won't be represented properly in our sample.
- Weighted Sampling Without Replacement: Very complicated expression-does not work for subset sum

Priority sampling is a weight-sensitive sampling scheme without replacement that works in a streaming context and is suitable for estimating subset sums [3].

How it works:

- For each item $i = 0, 1, .., n-1$ generate a random number $\alpha_i \in [0, 1]$ uniformly and randomly.
- Assign priority $q_i = \frac{w_i}{\alpha_i}$ to the ith element.
- Select the k highest priority items in the sample S.

Priority sampling gives unbiased weight estimates $E[\hat{w}_i] = w_i$

- Let τ be the priority of the $(k+1)$th highest priority.
- Each sampled item $i \in S$ gets a weight $\hat{w}_i = \max(w_i, \tau)$ if i is in the sample and 0 otherwise.
- $E[\hat{w}_i] = w_i$; see [3] for proof
- By linearity of expectation, the total weight of any arbitrary subset $I \subseteq \{0, 1, ..., n-1\}$
  $E[\sum_{i \in I} \hat{w}_i] = \sum_{i \in I} w_i$

Priority sampling yields unbiased estimates of subset sums

- $A(\tau')$:Event τ' is the kth highest priority among all $j \neq i$.
  Conditioned on $A(\tau')$, item i is picked with probability $\min(1, \frac{w_i}{\tau'})$, and if picked, $\tau = \tau'$
- For any value of τ',
  $E[\hat{w}_i \mid A(\tau')] = Pr[i \in S \mid A(\tau')] \max(w_i, \tau')$
- $Pr[i \in S \mid A(\tau')] = Pr[\frac{w_i}{\alpha_i} > \tau'] = Pr[\alpha_i < \frac{w_i}{\tau'}] = \min(1, \frac{w_i}{\tau'})$
- $E[\hat{w}_i \mid A(\tau')] = \max(w_i, \tau') \min(1, \frac{w_i}{\tau'}) = w_i$

$$E[\hat{w}_i \mid A(\tau')] = w_i$$

- This holds for all τ', hence holds unconditionally.

Near optimality Variance of the weight estimator is minimal among all $k+1$-sparse unbiased estimators.

2.3.1 Applications of priority sampling

Internet Traffic Analysis Internet routers export information about flows of packets passing through them (consider them items of a stream). A flow can be transmissions of a specific type of packets, e.g. FTP transfers for a file. Each flow record contains properties such as the source and destination IP addresses, port numbers and protocol number, and a summary of the packets in the flow, e.g. the total number of packets and bytes (think of byte size as the weight).
We want to sample flow records in such a way that we can answer questions like how many bytes of traffic came from a given customer or how much traffic was generated by a certain application. Both of these questions ask what is the total weight of a certain selection of flows. If we knew in advance of measurement which selections were of interest, we could have a counter for each selection and increment these as flows passed by. But there could be a variety of selections not known in advance of measurement, priority sampling would allow a change in selections/questions after the measurement.
An example where the selection is not known in advance was the tracing of the Internet Slammer Worm. Identifying a simple signature in the flow record (UDP traffic to port 1434 with a packet size of 404 bytes) enabled studying this worm by selecting records of flows matching this signature from the sampled flow records. The samples taken in the past thus allowed to trace the history of the attack even though the worm was unknown at the time of sampling.

External Information in the Selection Selection can be based on external information not even imagined relevant at the time when samples are made. Such scenarios preclude any kind of streaming algorithm based on selections of limited complexity, and shows the inherent relevance of sampling preserving full records for the purpose of arbitrary selections.
For example suppose a large chain store saved samples of all their sales where each record contained information such as item, location, time, and price where the weight of a record is the price. Based on sampled records, we might want to ask questions like how many days of rain does it take before

we get a boom in the sale of rain gear. Knowing this would allow us to tell how long we would need to order and disperse the gear if the weather report promised a long period of rain.
Now, the weather information was not part of the sales records, but if they had a database with historical weather information, they could look up each sampled sales record with rain gear, and check how many days it had rained at that location before the sale.

To put this example in a reservoir sampling context (previously discussed), imagine a central reservoir maintaining a priority sample over all sales done so far. The reservoir is small enough that it can be shared easily over the Internet with an analyst at a different location.

References

[1] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the twentieth annual symposium on Computational geometry (SCG '04). ACM, New York, NY, USA, 253-262. DOI=10.1145/997817.997857 http://doi.acm.org/10.1145/997817.997857

[2] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC '96). ACM, New York, NY, USA, 20-29. DOI=10.1145/237814.237823 http://doi.acm.org/10.1145/237814.237823

[3] Nick Duffield, Carsten Lund, and Mikkel Thorup. 2007. Priority sampling for estimation of arbitrary subset sums. J. ACM 54, 6, Article 32 (December 2007). DOI=10.1145/1314690.1314696 http://doi.acm.org/10.1145/1314690.1314696