Chapter 4

Hashing with real numbers and their big-data applications

Using only memory equivalent to 5 lines of printed text, you can estimate with a typical accuracy of 5 per cent and in a single pass the total vocabulary of Shakespeare. This wonderfully simple algorithm has applications in data mining, estimating characteristics of huge data flows in routers, etc. It can be implemented by a novice, can be fully parallelized with optimal speed-up and only need minimal hardware requirements. Theres even a bit of math in the middle!

Opening lines of a paper by Durand and Flajolet, 2003.

As we saw in Lecture 1, hashing can be thought of as a way to *rename* an address space. For instance, a router at the internet backbone may wish to have a searchable database of destination IP addresses of packets that are whizzing by. An IP address is 128 bits, so the number of possible IP addresses is 2^{128}, which is too large to let us have a table indexed by IP addresses. Hashing allows us to rename each IP address by fewer bits. In Lecture 1 this hash was a number in a finite field (integers modulo a prime p). In recent years large data algorithms have used hashing in interesting ways where the hash is viewed as a *real number*. For instance, we may hash IP addresses to real numbers in the unit interval $[0, 1]$.

EXAMPLE 2 (DARTTHROWING METHOD OF ESTIMATING AREAS) Suppose gives you a piece of paper of irregular shape and you wish to determine its area. You can do so by pinning it on a piece of graph paper. Say, it lies completely inside the unit square. Then throw a dart n times on the unit square and observe the fraction of times it falls on the irregularly shaped paper. This fraction is an estimator for the area of the paper.

Of course, the digital analog of throwing a dart n times on the unit square is to take a random hash function from $\{1, \ldots, n\}$ to $[0, 1] \times [0, 1]$.

Strictly speaking, one cannot hash to a real number since computers lack infinite precision. Instead, one hashes to *rational* numbers in $[0, 1]$. For instance, hash IP addresses to the set $[p]$ as before, and then think of number "i mod p"as the rational number i/p. This works OK so long as our method doesn't use too many bits of precision in the real-valued hash.

A general note about sampling. As pointed out in Lecture 3 using the random variable "Number of ears," the expectation of a random variable may never be attained at any point in the probability space. But if we draw a random sample, then we know by Chebysev's inequality that the sample has chance at least $1 - 1/k^2$ of taking a value in the interval $[\mu - k\sigma, \mu + k\sigma]$ where μ, σ denote the mean and variance respectively. Thus to get any reasonable idea of μ we need σ to be less than μ. But if we take t independent samples (even pairwise independent will do) then the variance of the mean of these samples is σ^2/t. Hence by increasing t we can get a better estimate of μ.

4.1 Estimating the cardinality of a set that's too large to store

Continuing with the router example, suppose the router wishes to maintain a count of the number of *distinct* IP addresses seen in the past hour. There are many practical applications for this, including but not limited to traffic accounting, quality of service, detecting denial-of-service (DoS) attacks.

The mathematical formalism is that we receive a *stream* of bit strings $x_1, x_2, \ldots, x_n$, among which there are at most N distinct strings. We wish to estimate N, using very little memory. (We're aiming for using $\approx \log N$ memory.)

How might we go about solving the problem? The simplest thing to do would be to store all IP addresses in some data structure as they are coming, and check whenever we try to put the string in the data structure if it's already there. However, this would clearly take $\Theta(N)$ memory. Another simple thing one could try would be to subsample the stream: i.e. keep a string with probability p, and throw it away with probability $1-p$ – then try to estimate the number of distinct elements from the subsampled version. But to distinguish between streams that look like $\underbrace{a_1, a_1, \ldots, a_1}_{\text{n-k times}}, a_2, \ldots, a_k$, where $n \gg k$ and $\underbrace{a_1, a_1, a_1, \ldots, a_1}_{\text{n times}}$, we would basically have to keep all the elements.

So what's the small memory solution? We will draw inspiration from the quote at the start of the lecture. We will take a hash function h that maps an IP address to a random real number in $[0, 1]$. (For now let's suppose that this is actually a random function.) Imagine also having a register, such that whenever a packet x_i whizzes by, we compute $h(x_i)$. If $h(x_i)$ is less than the number currently stored in the register, then we rewrite the register with x_i.

Let Y be the random variable denoting the contents of the register at the end. (It is a random variable because the hash function was chosen randomly. The packet addresses are not random.) Realize that Y is nothing but the *lowest value of $h(x_i)$ among all IP addresses x_i seen so far.*

Suppose the number of distinct IP addresses seen is N. This is what we are trying to estimate. We have the following lemma:

Lemma 5
$\mathbf{E}[Y] = \frac{1}{N+1}$ *and the variance of Y satisfies* $\mathbf{Var}[Y] \leq \frac{1}{(N+1)^2}$.

The expectation looks intuitively about right: the minimum of N random elements in $[0, 1]$ should be around $1/N$.

Let's do the expectation calculation. To do this, we need to calculate the PDF of the distribution of the minimum of N random elements. The CDF is quite easy: $\Pr[Y \leq r] = 1 - \Pr[Y \geq r] = 1 - (1-r)^N$, where the last line follows since $\min_{x_i} h(x_i) \geq r$ if all elements are mapped to numbers greater than r. To get the PDF, we just need to take the derivative thereof – so $\Pr[Y = r] = N(1-r)^{N-1}$. Then the expectation is just the integral

$$\mathbf{E}[Y] = \int_{r=0}^{1} rN(1-r)^{N-1}dr = \int_{t=0}^{1} (1-t)Nt^{N-1}dt = \int_{t=0}^{1} Nt^{N-1}dt - \int_{t=0}^{1} Nt^{N} = \frac{1}{N+1}$$

The variance calculation is very similar: we just use the fact that $\mathbf{Var}[Y] = \mathbf{E}[Y^2] - (\mathbf{E}[Y])^2$, and both terms amount to integrals similar to the one above.

(Note there's a slicker alternative proof for the expectation of Y. Imagine picking $N+1$ random numbers in $[0,1]$ and consider the chance that the $(N+1)$-st element is the smallest. By symmetry this chance is $1/(N+1)$. But this chance is exactly the expected value of the minimum of the first N numbers.)

Now we use our previous observation about sampling and variance reduction: suppose we repeat the procedure above with k independent random hash functions $h_1, h_2, \ldots h_k$, and the random variable corresponding to the register of the i-th hash function is Y_i. Let $\overline{Y}$ is be their mean. Then the variance of $\overline{Y}$ is $1/k(N+1)^2$, in other words, k times lower than the variance of each individual Y_i. Thus if $1/k$ is less than $\epsilon^2/3$ the standard deviation is less than $\epsilon/3(N+1)$, whereas the mean is $1/(N+1)$. Thus, by Chebyshev's inequality

$$\Pr\left[|\overline{Y} - \mathbf{E}[\overline{Y}]| \geq 3 \times \frac{\epsilon}{3(N+1)}\right] \leq \frac{1}{9}$$

This means that with probability at least 8/9, the estimate $1/\overline{Y} - 1$ is within $(1+\epsilon)$ factor of N.

Because we only need to store the value of the register corresponding to each hash function, and we can implement the real-valued hash by a hash of the form i/N^2 (say), each hash value takes $O(logN)$ bits to store. So the total memory usage is $O(\frac{1}{\epsilon^2}\log N)$, which is what we wanted!

4.1.1 Pairwise independent hash functions

All this assumed that the hash functions are random functions from 128-bit numbers to $[0,1]$. Let's now show that it suffices to pick hash functions from a pairwise independent family, albeit now yielding an estimate that is only correct up to some constant factor. Specifically, we'll modify the algorithm slightly to take k pairwise independent hashes and consider the *median* of the registers as an estimate for $\frac{1}{N}$. We will show that this estimate lies in the interval $[\frac{1}{5N}, \frac{5}{N}]$ with high probability. (This of course, trivially gives a constant factor estimate for N.)

For a particular hash function h, we'll bound the probability that we hash N different IP addresses, and the smallest hash is not in $[\frac{1}{5N}, \frac{5}{N}]$. We will do this by individually bounding the probability that it is less than $\frac{1}{5N}$ and bigger than $\frac{5}{N}$, and apply union bound over these two events.

First, what's the probability that the smallest hash is *less* than $\frac{1}{5N}$? For each IP address x_i, $\Pr[h(x_i) < \frac{1}{5N}]$ is at most $\frac{1}{5N}$, so by a union bound, the probability in question is at most $N \times \frac{1}{5N} = 1/5$.

To bound the probability that the smallest hash is is bigger than $5/N$, we have to do something a little more complicated. Let $x'_1, x'_2, \ldots, x'_N$ be the N distinct IP addresses. Let Z_i be a random variable which is 1, if $h(x'_i) \leq 5/N$, and 0 otherwise. Let $Z = \sum_{i=1}^{N} Z_i$ be the random variable counting the number of IP addresses that hash to a value below $5/N$. Thus if $\min_{x_i} h(x_i) > 5/N$ then $Z = 0$. So we just need to upper bound the probability that $Z = 0$.

Let's inspect the random variable Z. First, $\mathbf{E}[Z] = \sum_{i=1}^{N} \mathbf{E}[Z_i] = N \times \frac{5}{N} = 5$. Second, by pairwise independence, $\mathbf{Var}[Z] = \sum_{i=1}^{N} \mathbf{Var}[Z_i]$. However, $\mathbf{Var}[Z_i] = \mathbf{E}[Z_i^2] - (\mathbf{E}[Z_i])^2 \leq \mathbf{E}[Z_i^2] = \mathbf{E}[Z_i]$, where the last line follows since Z_i is a 0-1 variable. Hence, $\mathbf{Var}[Z] \leq 5$ as well. But then, we can use Chebyshev to conclude that $\Pr[Z = 0] \leq \Pr[|Z - \mathbf{E}[Z]| \geq 5] \leq \frac{1}{5}$

By union bound then, the probability that the smallest hash is not in $[\frac{1}{5N}, \frac{5}{N}]$ is at most $\frac{2}{5}$.

Now, let's see what happens when we take the median of the registers from multiple hash functions. If the median is larger than $5/N$, this means that at least half of the registers Y_i exceeded $5/N$. But since the probability that any Y_i exceeds $5/N$ is at most $\frac{2}{5}$, by Chernoff bounds, this event happens with probability at most $\exp(-\left((\frac{3}{5})^2 \frac{2}{5} k\right)/3)$. A similar calculation holds for the probability that the median is smaller than $\frac{1}{5N}$. By making $k = \Omega(1/\log\delta)$, we can make this probability less than δ for any $\delta > 0$.

4.2 Estimating document similarity

One of the aspects of the data deluge on the web is that often one finds duplicate copies of the same thing. Sometimes the copies may not be exactly identical: for example mirrored copies of the same page but some are out of date. The same news article or blog post may be reposted many times, sometimes with editorial comments. By detecting duplicates and near-duplicates internet companies can often save on storage by an order of magnitude.

We want to do significantly better than the trivial method of looking at all pairs of documents and comparing them. (Doing computations which take quadratic time in the total number of documents is completely infeasible, as it's not unusual for the number of documents in consideration to be on the order of billions.) Note also that the need to detect "near-duplicates"instead of duplicates makes the problem significantly harder: if we just want to detect duplicates, we could simply hash the documents to a much smaller set, and tag as duplicates only the ones that have collisions.

We present a technique called *locality sensitive hashing* such that the hash preserves some information about the "content" of a document. Two documents' similarity can be estimated by comparing their hashes. This is an example of a burgeoning research area of hashing while preserving some *semantic* information. In general finding similar items in databases is a big part of data mining (find customers with similar purchasing habits, similar tastes, etc.) and also "big data "research in biology and other sciences. Today's simple hash is merely a way to dip our toes in these waters.

The formal definition of a *locality sensitive hash* is the following: for a given similarity measure $\mathrm{sim}(A, B)$ defined on the set of documents, a locality-sensitive hash family $\mathcal{H}$ is hash family satisfying $\Pr_{\mathrm{hash}\in\mathcal{H}}[\mathrm{hash}(A) = \mathrm{hash}(B)] = \mathrm{sim}(A, B)$.

In many settings, it is popular to use the *bag of words* viewpoint according to which the semantic content of a document is defined as the multiset of words in it (that is, the list of words in it, together with the number of times each occurs in the document). For simplicity in this lecture we simplify it further, and represent documents using just the *set* of words in it. Then the *Jaccard similarity* of documents/sets A, B is defined to be $|A \cap B| / |A \cup B|$. (This is 1 iff $A = B$ and 0 iff the sets are disjoint.)

The basic idea behind the hash is as follows: pick a random hash function mapping the underlying universe of elements to $[0, 1]$. Define the hash of a set A to be the *minimum* of $h(x)$ over all $x \in A$. Then by symmetry, $\Pr[\mathrm{hash}(A) = \mathrm{hash}(B)]$ is exactly the Jaccard similarity. (Note that if two elements x, y are different then $\Pr[h(x) = h(y)]$ is 0 when the hash is real-valued. Thus the only possibility of a collision arises from elements in the intersection of A, B.) Thus one could pick k random hash functions and take the fraction of instances of $\mathrm{hash}(A) = \mathrm{hash}(B)$ as an estimate of the Jaccard similarity. This has the right expectation but we need to repeat with k different hash functions to get a better estimate.

The analysis goes as follows. Suppose we are interested in flagging pairs of documents whose Jaccard-similarity is at least 0.9. Then we compute k hashes and flag the pair if at least $0.9 - \epsilon$ fraction of the hashes collide. Chernoff bounds imply that if $k = \Omega(1/\epsilon^2)$ this flags all document pairs that have similarity at least 0.9 and with good probability does not flag any pair with similarity less than $0.9 - 3\epsilon$.

To make this method more realistic we need to replace the idealized random hash function with a real one and analyse it. That is beyond the scope of this lecture. Indyk showed that it suffices to use a k-wise independent hash function for $k = \Omega(\log(1/\epsilon))$ to let us estimate Jaccard-similarity up to error ϵ. Thorup recently showed how to do the estimation with pairwise independent functions. This analysis seems rather sophisticated; let me know if you happen to figure it out.

We remark though that there's a dose of sublety as to which similarity measure admit a locality sensitive hash. We picked a nice measure here, but tweaking it slightly results in one which does not have a locality sensitive hash family attached to it. For instance, the so called Dice's coefficient $\mathrm{sim}_{\mathrm{Dice}}(A, B) = \frac{|A\cap B|}{\frac{1}{2}(|A|+|B|)}$ doesn't have one. Trying to prove this yourself is a good exercise, but the relevant reference is the paper by Charikar listed below.

Bibliography

1. Broder, Andrei Z. (1997), *On the resemblance and containment of documents*, Compression and Complexity of Sequences: Proceedings, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997.

2. Broder, Andrei Z.; Charikar, Moses; Frieze, Alan M.; Mitzenmacher, Michael (1998), *Min-wise independent permutations*, Proc. 30th ACM Symposium on Theory of Computing (STOC '98).

3. Charikar, Moses S. *Similarity estimation techniques from rounding algorithms*, In Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 380-388. ACM, 2002.

4. Gurmeet Singh, Manku; Das Sarma, Anish (2007), *Detecting near-duplicates for web crawling*, Proceedings of the 16th international conference on World Wide Web, ACM.

5. Indyk, P (1999). A small approximately min-wise independent family of hash functions. Proc. ACM SIAM SODA.

6. Thorup, M. (2013). `http://arxiv.org/abs/1303.5479`.