6 Locality Sensitive Hashing

In the last few lectures we saw how to convert from a document full of words or characters to a set, and then to a matrix, and then to a k-dimensional vector. And from the final vector we could approximate the Jaccard distance between two documents.

However, now we face a new challenge. We have many many documents (say n documents) and we want to find the ones that are close. But we don't want to calculate $\binom{n}{2} \approx n^2/2$ distances to determine which ones are really similar.

In particular, consider we have $n = 1{,}000{,}000$ items and we want to ask two questions:

(Q1): Which items are similar?

(Q2): Given a query item, which others are similar to the query?

For **(Q1)** we don't want to check all roughly n^2 distance (no matter how fast each computation is), and for **(Q2)** we don't want to check all n items. In both cases we somehow want to figure out which ones might be close and the check those.

Consider n points in the plane $\mathbb{R}^2$. How can we quickly answer these questions:

- Hierarchical models (range trees, kd-trees, B-trees) don't work well in high dimensions. We will return to these in **L8**.
- Lay down a Grid: Close points should be in same grid cell. But some can always lay across the boundary (no matter how close). Some may be further than 1 grid cell, but still close. And in high dimensions, the number of neighboring grid cells grows exponentially. One option is to randomly shift (and rotate) and try again.

The second idea is close to a technique called *Locality Sensitive Hashing* (or LSH) which we will explore.

6.1 Properties of Locality Sensitive Hashing

We start with the goal of constructing a *locality-preserving* hash function h with the following properties (think of a random grid). Do not confuse this with a (random) hash function discussed in **L2**. The locality needs to be with respect to a distance function $\mathtt{d}(\cdot, \cdot)$. In particular, if h is $(\gamma, \phi, \alpha, \beta)$-*sensitive* with respect to $\mathtt{d}$ then it has the following properties:

- $\mathbf{Pr}[h(a) = h(b)] > \alpha$ if $\mathtt{d}(a, b) < \gamma$
- $\mathbf{Pr}[h(a) = h(b)] < \beta$ if $\mathtt{d}(a, b) > \phi$

For this to make sense we need $\alpha > \beta$ for $\gamma < \phi$. Ideally we want $\alpha - \beta$ to be large and $\phi - \gamma$ to be small. Then we can repeat this with more random hash functions to *amplify* the effect, according to a Chernoff-Hoeffding bound. This will effectively make $\alpha - \beta$ larger for any fixed $\phi - \gamma$ gap, and will work for all such $\phi - \gamma$ simultaneously.

6.1.1 Minhashing as LSH

Minhashing is an instance of LSH (and perhaps the first such realized instance). Recall we had t hash functions $\{h_1, h_2, \ldots, h_t\}$ where each $h_i : [n] \to [n]$ (at random).

Documents:	D_1	D_2	D_3	D_4	$\ldots$	D_n
h_1	1	2	4	0	$\ldots$	1
h_2	2	0	1	3	$\ldots$	2
h_3	5	3	3	0	$\ldots$	1
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
h_t	1	2	3	0	$\ldots$	1

Where

$$\mathsf{JS}_t(D_1, D_2) = \mathbf{E}[(1/t) \cdot (\text{ \# rows } h_i(D_1) = h_i(D_2))]$$

and in general $\mathsf{JS}(a, b) = \mathbf{Pr}[h(a) = h(b)]$.

Then for some similarity threshold $1 - \mathsf{JS}(a, b) = \mathbf{d}(a, b) = \tau$ we can set $\tau = \gamma = \phi$ and the have $\alpha = 1 - \tau$ and $\beta = \tau$.

This scheme will work for any similarity such that $s(a, b) = \mathbf{Pr}[h(a) = h(b)]$.

6.2 Banding for LSH

But how do we use LSH to answer questions **Q1** and **Q2**?

- If we only check the items that *always* fall into the same bin, then with more hash functions eventually almost nothing will be in the same bin (acts like a t-dimensional grid).
- If we check items that *for any hash* fall into the same bin, then with more hash functions, eventually almost all items will be checked.

If the first approach is "Papa" bear's LSH, and the second approach is "Mama" bear's, then we will need a "Baby" bear approach. Baby bear likes *banding*.

Suppose we have a budget of t hash functions. We are going to use $b < t$ in Papa bear's approach and $r = t/b$ in Mama bear's approach (Baby bear takes after both of her parents after all $\ddot\smile$). Here b is the number of hashes in a band, and r is the number of rows of bands.

Recall that $s = \mathsf{JS}(D_1, D_2)$ is the probability that D_1 and D_2 have a hash collision. Then we have

$$\begin{aligned}
s^b &= \text{ probability all hashes collide in 1 band} \\
(1 - s^b) &= \text{ probability not all collide in 1 band} \\
(1 - s^b)^r &= \text{ probability that in no bands do all hashes collide} \\
f(s) = 1 - (1 - s^b)^r &= \text{ probability all hashes collide in at least 1 band}
\end{aligned}$$

We can plot f (as seen in Figure 6.2) as an S-curve where on the x-axis $s = \mathsf{JS}(D_1, D_2)$ and the y-axis represents the probability that the pair D_1, D_2 is a candidate to check the true distance. In this example we

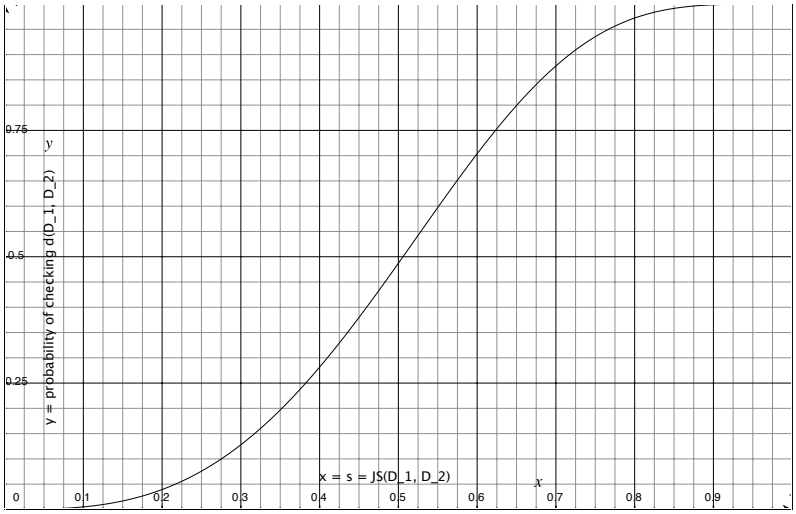

Figure 6.1: Probability that the distance $d(D_1, D_2)$ is checked in a LSH scheme with $r = 3$ bands with $b = 5$ hashes each, as a function of $s = \mathsf{JS}(D_1, D_2)$.

have $t = 15$ and $r = 3$ and $b = 5$.

$$
\begin{aligned}
f(0.1) &= 0.005 \\
f(0.2) &= 0.04 \\
f(0.3) &= 0.13 \\
f(0.4) &= 0.28 \\
f(0.5) &= 0.48 \\
f(0.6) &= 0.70 \\
f(0.7) &= 0.88 \\
f(0.8) &= 0.97 \\
f(0.9) &= 0.998
\end{aligned}
$$

Choice of r and b. Usually there is a budget of t hash function one is willing to use. Perhaps it is $t = (1/2\varepsilon^2)\ln(2/\delta)$. Then how does one divvy them up among r and b?

The threshold τ where f has the steepest slope is about $\tau \approx (1/r)^{1/b}$. So given a similarity s that we want to use as a cut-off (e.g. $\tau = s = \alpha = 1 - \beta$) we can solve for $b = t/r$ in $\tau = (b/t)^{1/b}$ to (very roughly) yield $b \approx -\log_\tau(t)$.

If there is no budget on r and b, as they increase, the S curve gets sharper.

6.3 LSH for Euclidean Distance

We so far operated on a specific distance and similarity for min hashing (more on other distances in **L7**). This extends to many other distances including, most famously the Euclidean distance between two vectors $v, u \in \mathbb{R}^k$ so $d_{\mathsf{E}}(u, v) = \|u - v\| = \sqrt{\sum_{i=1}^k (v_i - u_i)^2}$. This is most useful when k is quite large.

The hash h is defined as follows.

1. First take a random *unit vector* $u \in \mathbb{R}^d$. A *unit vector* u satisfies that $\|u\| = 1$, that is $d_{\mathsf{E}}(u, 0) = 1$. We will see how to generate a random unit vector below.

2. Project $a, b \in P \subset \mathbb{R}^k$ onto u:

$$a_u = \langle a, u \rangle = \sum_{i=1}^{k} a_i \cdot u_i$$

   This is *contractive* so $\|a_u - b_u\| \leq \|a - b\|$.

3. Create bins of size γ on u (now in $\mathbb{R}^1$). The index of the bin a falls into is $h(a)$.

If $\|a - b\| < \gamma/2$ then $\mathbf{Pr}[h(a) = h(b)] \geq 1/2$. If $\|a - b\| > 2\gamma = \phi$ then $\mathbf{Pr}[h(a) = h(b)] < 2/3$. So this is $(\gamma/2, 2\gamma, 1/2, 1/3)$-sensitive.

To see the second claim (with 2γ) we see that for a collision we need $\cos(a - b, u) < \pi/3$ (out of $[0, \pi]$). Otherwise $\|a - b\| > 2\|a_u - b_u\|$ and thus they must be in different bins.

We can also take $h(a) = \langle a, u \rangle \mod (t\gamma)$. For large enough t, the probability of collision in different bins is low enough that this can be effective.

6.3.1 Random Unit Vector

The easiest way to generate a random unit vector is through Gaussian random variables. A d-dimensional uniform Gaussian distribution is defined:

$$G(x) = \frac{1}{(2\pi)^{d/2}} e^{-\|x\|_2^2/2}.$$

If we have two uniform random numbers $u_1, u_2 \in [0, 1]$ then we can generate two independent 1-dimensional Gaussian random variables as (using the Box-Muller transform):

$$y_1 = \sqrt{-2\ln(u_1)}\cos(2\pi u_2)$$
$$y_2 = \sqrt{-2\ln(u_1)}\sin(2\pi u_2).$$

A uniform Gaussian has the (*amazing!*) property that all coordinates (in any orthogonal basis) are independent of each other. Thus to generated a point $x \in \mathbb{R}^d$ from a d-dimensional Gaussian, for each coordinate i we assign it the value of an independent 1-dimensional Gaussian random variable.

6.3.2 p-Stable Random Variables and LSH for ℓ_p-Norms

A distribution μ over $\mathbb{R}$ is p-stable (for $p \geq 0$) if the following holds. Let $d + 1$ random variables X_0 and $\{X_1, \ldots, X_d\}$ all have the distribution μ. Then considering any d real values $\{v_1, \ldots, v_d\}$, the random variable $\sum_i v_i X_i$ has the same distribution as $(\sum_i |v_i|^p)^{1/p} X_0$.

Intuitively, this allows us to replace a sum of random variables with a single random variables by adjusting coefficients carefully. But actually, it is the composition of the coefficients that is interesting.

p-stable distributions are know for $p \in (0, 2]$. Special cases are

- For $p = 2$, then the Gaussian distribution $g(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$ is 2-stable.
- For $p = 1$, then the Cauchy distribution $c(x) = \frac{1}{\pi} \frac{1}{1+x^2}$ is 1-stable.

In particular, for $p = 2$ and a vector $v = (v_1, v_2, \ldots, v_d)$ where we want to estimate $\|v\|_2 = (\sum_i v_i^2)^{1/2}$, we can consider each coordinate v_i individually as an estimate by using a p-stable random variable. That is we can estimate $\|v\|_2$ by choosing $d + 1$ random Gaussian vectors g_0 and $g_1, \ldots, g_d$, and calculating $(1/g_0) \sum_i g_i v_i = (1/g_0)\langle g, v \rangle$, where $g = (g_1, \ldots, g_d)$ ... a normalized version of our projection operator.

Using the Cauchy random variables c_0 and $c_1, \ldots, c_d$ in place of the Gaussian ones allows us to estimate $\|v\|_1 = \sum_i |v_i|$ as $(1/c_0)\langle c, v \rangle$ with $c = (c_1, \ldots c_d)$.