CSE5401 Lec. 22

1. Locality-sensitive Hashing
2. Approximate Nearest Neighbor

Hashing is a versatile technique that, among other things, enables $O(1)$-time dictionary structures. In many applications, however, what we really need is a variant of a dictionary structure that finds "near-matches." For example, we might be trying to identify an image or sound, and we can't hope that the input will precisely match what we've seen previously. We could exhaustively search our database to find the nearest match, but can we get a data structure that is fast like hash functions? We want a "locality-sensitive hash function":

Defn: Let $d: \mathcal{U} \times \mathcal{U} \to \mathbb{R}^{\geq 0}$ be a metric on the "universe" $\mathcal{U}$.
$\mathcal{H} = \{h: \mathcal{U} \to \mathcal{S}\}$ is a $(r, \varepsilon, P_\ell, P_u)$-locality sensitive hash family w.r.t. d (for $r \in \mathbb{R}^{>0}$, $\varepsilon \in \mathbb{R}^{\geq 0}$, $P_\ell > P_u \in (0,1)$) if for all $x, x' \in \mathcal{U}$,
if $d(x,x') \leq r$, then $\Pr_H[H(x) = H(x')] \geq P_\ell$ and
if $d(x,x') \geq (1+\varepsilon) r$, then $\Pr_H[H(x) = H(x')] \leq P_u$

Let's see an example of a LSH family for strings: $\mathcal{U} = \{0,1\}^m$ and $\mathcal{S} = \{0,1\}$, with $d(\vec{x}, \vec{x}') = \sum_{i=1}^{m} I[x_i \neq x_i']$ (the "Hamming distance.")

Theorem $\mathcal{H} = \{h^{(i)}(\vec{x}) = x_i : i = 1, \ldots, m\}$ is a $(r, \varepsilon, 1 - \frac{r}{m}, 1 - (1+\varepsilon)\frac{r}{m})$-LSH family for Hamming distance on $\{0,1\}^m$.

Proof: If $d(\vec{x}, \vec{x}') = \sum_{i=1}^{m} I[x_i \neq x_i'] \leq r$, $\Pr_H[H(\vec{x}) \neq H(\vec{x}')] = \frac{\sum_{i=1}^{m} I[x_i \neq x_i']}{m} \leq \frac{r}{m}$
Similarly, if $d(\vec{x}, \vec{x}') \geq (1+\varepsilon) r$, $\Pr_H[H(\vec{x}) \neq H(\vec{x}')] \geq (1+\varepsilon)\frac{r}{m}$ ▨
This gap of $\varepsilon \frac{r}{m}$ is pretty weak, but we can amplify it easily.

Suppose that instead of choosing just one $h^{(i)} \in \mathcal{H}$, we chose ℓ independent members, and used this ℓ-bit value as the hash value. Then $Pr[\vec{H}(\vec{x}) = \vec{H}(\vec{x}')] = \prod_{j=1}^{\ell} Pr[H_j(\vec{x}) = H_j(\vec{x}')]$, so a $(r, \varepsilon, P_\ell, P_u)$-LSH becomes a $(r, \varepsilon, P_\ell^\ell, P_u^\ell)$-LSH. Concretely, our Hamming LSH then has $p_\ell = (1 - \frac{r}{m})^\ell$ and $p_u = (1 - (1+\varepsilon)\frac{r}{m})^\ell$, where the gap is approximately $(1 - \ell\frac{r}{m}) - (1 - \ell(1+\varepsilon)\frac{r}{m}) = \ell \cdot \varepsilon \frac{r}{m}$. The downside is that the hash function is less tolerant: close strings are much less likely to match. We can compensate for this by a similar construction, where we again draw k independent members, but now consider a pair to "collide" if <u>any</u> of the hashes collide. (If you prefer, we insert the item into the buckets for k different tables.) Then $Pr[\vec{H}(\vec{x}) \neq \vec{H}(\vec{x}')] = \prod_{j=1}^{k} Pr[H_j(\vec{x}) \neq H_j(\vec{x}')]$ so a $(r, \varepsilon, P_\ell, P_u)$-LSH becomes a $(r, \varepsilon, 1-(1-P_\ell)^k, 1-(1-P_u)^k)$-LSH. Indeed, we will combine the two, to obtain a $(r, \varepsilon, 1-(1-P_\ell^\ell)^k, 1-(1-P_u^\ell)^k)$-LSH. It may not be obvious that we can choose ℓ & k in a way that keeps P_ℓ high while bounding the number of unwanted collisions (by pushing P_u towards 0). But, for a given number of elements n, here is how we can do so:

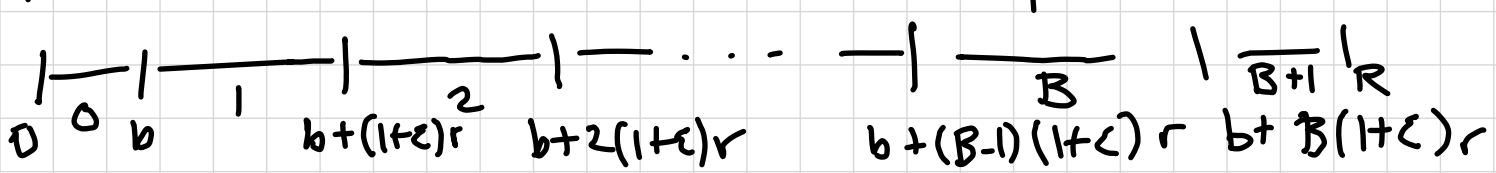
<u>Lemma</u> Let $\mathcal{X} \subseteq \mathcal{U}$ be a set of n points, $\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}$, and let $\mathcal{H}$ be a $(r, \varepsilon, P_\ell, P_u)$-LSH family. For $\rho = \frac{\ln 1/P_\ell}{\ln 1/P_u}$, suppose we use $k = n^\rho$ tables indexed by $\ell = \ln_{1/P_u} n$-tuples drawn independently from $\mathcal{H}$. Then for any $\vec{q} \in \mathcal{U}$, the expected number of $\vec{x} \in \mathcal{X}$ with $d(\vec{q}, \vec{x}) \geq (1+\varepsilon) r$ that collide with $\vec{q}$ is $\leq k$. Also, moreover, <u>for any</u> $\vec{x}^* \in \mathcal{X}$ s.t. $d(\vec{q}, \vec{x}^*) \leq r$, $\vec{q}$ collides with $\vec{x}^*$ w.p. $\geq 1 - \frac{1}{e}$.

<u>Proof</u>: First, note that the expected number of collisions of $\vec{q}$ with $\vec{x}$ s.t. $d(\vec{q}, \vec{x}) \geq (1+\varepsilon) r$ is at most $k \cdot n \cdot Pr_H[H(\vec{x}) = H(\vec{q})]^\ell \leq k \cdot n \cdot P_u^{\ln_{1/P_u} n}$

$= k \cdot n \cdot \frac{1}{n} \leq k$ as promised (one per table). Meanwhile, in each of the k tables, $\Pr_{\vec{H}}[\vec{H}(\vec{x}^*) = \vec{H}(\vec{q})] \geq p_\ell^\ell = p_u^{\frac{\ln p_\ell}{\ln p_u}\ell} = p_u^{\ln_{1/p_u} n \cdot \rho} = \frac{1}{n^\rho}$.
Since the k tables are independent, the probability $\vec{q}$ does not collide with $\vec{x}^*$ in any of them is $\leq (1-\frac{1}{n^\rho})^k \leq e^{-\frac{1}{n^\rho}n^\rho} = \frac{1}{e}$ ▨
In particular, by Markov's Inequality, the number of collisions for $\vec{q}$ is $\leq 4k$ w.p. $\geq \frac{3}{4}$. Then w.p. $\geq \frac{3}{4} - \frac{1}{e} > \frac{1}{2}$, by a union bound, we find $\vec{x}^*$ among the $4k = O(n^\rho)$ collisions in time $O(n^\rho)$ (assuming evaluating h & $d(\vec{q}, \vec{x})$ each take $O(1)$-time).

Again, concretely, for our Hamming LSH family, $\ln p_\ell = \ln 1 - \frac{r}{m} \approx -\frac{r}{m}$ and $\ln p_u = \ln(1-(1+\varepsilon)\frac{r}{m}) \approx -(1+\varepsilon)\frac{r}{m}$, so $\rho = \frac{\ln p_\ell}{\ln p_u} \approx \frac{1}{1+\varepsilon} = 1 - \frac{\varepsilon}{1+\varepsilon}$. Thus, we are saving asymptotically over exhaustive search, though it is not as fast as a standard hash dictionary. (It turns out, this is essentially optimal)

What about for Euclidean distance? In 1-d, we can use a simple scheme: supposing the points lie in $[0, R]$, we can create $\frac{R}{(1+\varepsilon)r} + 1$ buckets: choose $b \in [0, (1+\varepsilon)r]$ uniformly at random, and for $i = 1, \ldots, B-1$ we let the i^{th} bucket contain the points between $b+(i-1)(1+\varepsilon)r$ and $b+i(1+\varepsilon)r$



Then any pair x, x' of distance $> (1+\varepsilon)r$ must hash to distinct buckets, while a pair with $|x - x'| \leq r$ hash to different buckets w.p. $\frac{|x-x'|}{(1+\varepsilon)r} \leq \frac{1}{1+\varepsilon}$
To extend this to higher dimensions, we can use essentially the same functions as for dimensionality reduction: let $\vec{\sigma}$ be a vector of Rademacher random variables (1 w.p. $1/2$, -1 w.p. $1/2$ each).

Then we can apply (essentially) the 1-d scheme to $\langle \vec{\sigma}, \vec{x} \rangle$.

To see that this is still good, recall we (essentially) proved

Theorem Let R be a $m \times \ell$ matrix of Rademacher RVs. Then $\forall \vec{x} \in \mathbb{R}^m$

$\Pr[\|R\vec{x}\|_1 > \ell(1+\varepsilon)\|\vec{x}\|_2] < e^{-\ell\varepsilon^2/9}$ and $\Pr[\|R\vec{x}\|_1 < \ell(1-\varepsilon^2)\|x\|_2] < e^{-\ell\varepsilon^2/9}$.

Suppose we use the 1-d construction on each of the ℓ coordinates of $R\vec{x}$. Then certainly, the probability that the sum of the coordinates for $\vec{x}$ and $\vec{x}'$ differ by more than $\ell(1+$ is at most the probability

$\Pr[\sum_{i=1}^{\ell} |\langle \vec{\sigma}^{(i)}, \vec{x} \rangle - \langle \vec{\sigma}^{(i)}, \vec{x}' \rangle| > \ell(1+\frac{\varepsilon}{2}) r] = \Pr[\|R(\vec{x}-\vec{x}')\|_1 > \ell(1+\frac{\varepsilon}{2})\|\vec{x}-\vec{x}'\|_2] < e^{-\ell\varepsilon^2/36}$

So e.g., taking buckets of width $(1+\varepsilon) r\ell$ gives $\vec{x}$ & $\vec{x}'$ colliding w.p. $\geq$ $1 - \frac{\varepsilon}{2(1+\varepsilon)} - e^{-\ell\varepsilon^2/36}$ On the other hand, if $\|\vec{x}-\vec{x}'\|_2 > (1+\varepsilon) r$,

$\Pr[\sum_{i=1}^{\ell} |\langle \vec{\sigma}^{(i)}, \vec{x} \rangle - \langle \vec{\sigma}^{(i)}, \vec{x}' \rangle| < \ell(1-\frac{\varepsilon^2}{4})(1+\varepsilon) r] < e^{-\ell\varepsilon^2/36}$ also, where critically $(1-\frac{\varepsilon^2}{4})(1+\varepsilon) < 1+\frac{\varepsilon}{2}$ for $\varepsilon < 1$, so there is a gap.

Approximate Nearest Neighbor

LSH is useful in the design of a data structure to solve an approximate version of the "nearest neighbor" problem. In the (exact) nearest neighbor problem for a metric d, there is a set of points $\mathcal{X} \subseteq \mathcal{U}$ and a query $\vec{q} \in \mathcal{U}$. We must return $\vec{x}^* = \arg\min_{\vec{x} \in \mathcal{X}} d(\vec{x}, \vec{q})$

A naive approach (again) computes $d(\vec{x}, \vec{q})$ for each $\vec{x} \in \mathcal{X}$, and takes the minimum. For exact nearest neighbor, we can't solve the problem much faster in general. But, using LSH families, we can solve the following approximate version:

Defn: For a set of points $\mathcal{X} \subseteq \mathcal{U}$ and a metric d, the $(1+\varepsilon)$-approximate nearest neighbor problem is given $\vec{q} \in \mathcal{U}$, return $\vec{x}' \in \mathcal{X}$ s.t. $d(\vec{q}, \vec{x}') \leq (1+\varepsilon) \min_{\vec{x} \in \mathcal{X}} d(\vec{q}, \vec{x})$.

Supposing that distinct points in $\mathcal{U}$ have distance $\geq D_{min}$ and $\leq D_{max}$,

and we can compute $[r, \varepsilon, P_\ell, P_u]$-LSH for all $r \in [D_{min}, D_{max}]$ using T time and S space, then we solve $(1+\varepsilon)$-Approx. Nearest Neighbor by using our LSH family for $r = \frac{D_{min}}{2}, (1+\varepsilon)\frac{D_{min}}{2}, (1+\varepsilon)^2\frac{D_{min}}{2}, \ldots$ up to D_{max}, i.e., $\log_{1+\varepsilon}\frac{D_{max}}{D_{min}}$ tables. By our lemma, we can guarantee that at $r^* = (1+\varepsilon)^{i^*}\frac{D_{min}}{2} \le (1+\varepsilon)\min_{\vec{x}\in X} d(\vec{q},\vec{x})$, with high probability $\vec{q}$ collides with some $\vec{x}^*$ with $d(\vec{q},\vec{x}^*) \le r^*$. We can use binary search to find i^*, querying $\le \log_2\log_{1+\varepsilon}\frac{D_{max}}{D_{min}}$ of the hash tables each of which (recall) involved $O(n^\rho)$ collisions. Thus, the overall query time is $O(T \cdot n^\rho \cdot \log_2\log_{1+\varepsilon}\frac{D_{max}}{D_{min}})$, and we use a table of size $n^{1+\rho}\log_{1+\varepsilon}\frac{D_{max}}{D_{min}}$ with $\log_{1/P_u} n \cdot \log_{1+\varepsilon}\frac{D_{max}}{D_{min}}$ functions from the basic LSH family. Again, for Hamming distance, $\rho = \frac{1}{1+\varepsilon}$ and $\frac{D_{max}}{D_{min}} = m$ so this solves $(1+\varepsilon)$-Approx. Nearest Neighbor in time $O(n^{1-\frac{\varepsilon}{1+\varepsilon}} m \log\log m)$ (since we can evaluate the Hamming distance in time m). We get a similar result for Euclidean distance using our LSH for that distance as well. LSH methods have been developed for other distances as well, for example including edit distance, ℓ_1 ("statistical distance"), and so on.