CS265, Fall 2022

Class 8: Agenda and Questions

1 Announcements

- HW3 due tomorrow!
- HW4 out now!

2 Recap and Questions

We'll do a quick recap of the JL lemma and the (approximate) nearest neighbors problem.

3 A better scheme for approximate nearest neighbors, and locality sensitive hashing

[A bit of lecture with setup. Summary below. This is also covered in the lecture notes.]

Recall the setup for c-approximate-nearest neighbors. We have a set S of size n, and **for today $S \subset \mathbb{S}^d$ lives on the surface of the d-dimensional sphere.** That is, $S = \{x_1, \ldots, x_n\}$, so that $x_i \in \mathbb{R}^{d+1}$ and $\|x_i\|_2 = 1$ for all $i \in [n]$.

Our goal is to come up with some data structure to store the x_i's, so that:

- We don't use too much space (ideally, use space poly(n), where the exponent in the polynomial doesn't depend on d).
- Given $y \in \mathbb{S}^d$, we can find $x_i \in S$ so that
$$\|x_i - y\|_2 \leq c \cdot \min_j \|x_j - y\|_2$$
in time sublinear in n.

3.1 Nearest-Neighbors vs. Near Neighbors

[A bit of lecture, summary below and also in the lecture notes and posted slides – so many resources!!]

Consider the following problem, called (r, c)**-near-neighbors.** We have a set $S \subset \mathbb{S}^d$ of size n as before, and our goal is to come up with some data structure (that doesn't use too much space) to store the x_i's, so that the following holds.

> Given $y \in \mathbb{S}^d$ so that $\min_j \|x_j - y\|_2 \leq r$, we can find $x_i \in S$, in sublinear time, so that $\|x_i - y\|_2 \leq cr$.

It turns out that if we can solve (r, c)-near-neighbors (for a decent range of r's) then we can solve c-nearest-neighbors.

3.2 A solution to (r, c)-near-neighbors

[A bit of lecture for setup; summary below and also in the lecture notes.]

Let s, k be parameters, chosen as follows:

$$s = \sqrt{n}, \qquad k = \frac{\pi \log n}{2r}$$

For $i = 1, \ldots, s$, let $A_i \in \mathbb{R}^{k \times d+1}$ have i.i.d. $\mathcal{N}(0, 1)$ entries. For $y \in \mathbb{S}^d$, define

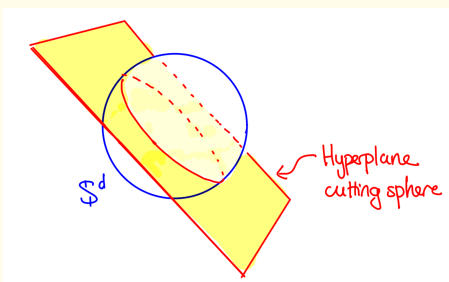
$$h_i(y) = \text{sign}(A_i y),$$

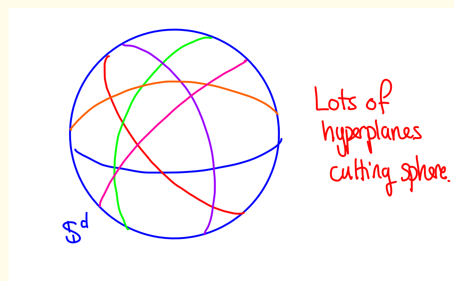
where for a vector $a \in \mathbb{R}^k$, $\text{sign}(a) \in \{\pm 1\}^k$ is just the vector whose i'th entry is $+1$ if $a_i > 0$ and -1 if $a_i \leq 0$.

Group Work

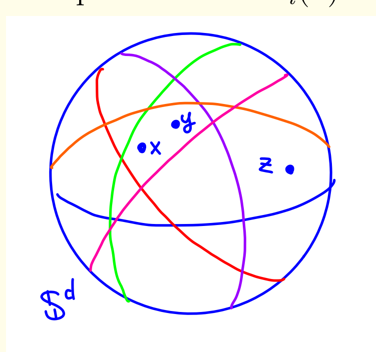
1. Consider a hash function $h_i : \mathbb{S}^d \to \{\pm 1\}^k$ as defined above. Explain why "$h_i(x) = h_i(y)$" has the following geometric meaning:

   Imagine choosing k uniformly random hyperplanes in $\mathbb{R}^d$, and using them to slice up the sphere $\mathbb{S}^d$ like this:





   Then $h_i(x) = h_i(y)$ if and only if x and y are in the same "cell" of this slicing. For example, in the picture below $h_i(x) = h_i(y) \neq h_i(z)$.



   Hint: *Use the spherical symmetry of the Gaussian distribution.*

2. Explain why, for $x, y \in \mathbb{S}^d$, and for any $i = 1, \ldots, s$,

$$\Pr[h_i(x) = h_i(y)] = \left(1 - \frac{\text{angle}(x,y)}{\pi}\right)^k,$$

where $\text{angle}(x,y) = \arccos(x^T y)$ is the arc-cosine of the dot product of x and y, aka, the angle between x and y.

Hint: *Think about the geometric intuition in the plane spanned by x and y.*

3. Suppose that $x, y \in \mathbb{S}^d$. Fill in the blank, using the previous part:

$$\Pr[\forall i \in \{1, \ldots, s\}, h_i(x) \neq h_i(y)] = \_\_\_\_\_$$

(Don't worry about simplifying, you'll do that in the next part).

4. Let $x, y \in \mathbb{S}^d$ and suppose that the angle between x and y is pretty small. Using our choices of s and k above, along with extremely liberal use of the approximation that $1 - x \approx e^{-x}$ for small x, convince yourself that

$$\Pr[\forall i \in \{1, \ldots, s\}, h_i(x) \neq h_i(y)] \approx \exp\left(-n^{1/2 - \text{angle}(x,y)/(2r)}\right).$$

5. Fill in the blanks (assuming that your approximation from the previous step is valid):

   (a) If $\text{angle}(x,y) \leq r$, then

   $$\Pr[\exists i \in \{1, \ldots, s\} \text{ so that } h_i(x) = h_i(y)] \geq \_\_\_\_\_\_$$

   (b) If $\text{angle}(x,y) \geq 5r$, then

   $$\Pr[\exists i \in \{1, \ldots, s\} \text{ so that } h_i(x) = h_i(y)] \leq \_\_\_\_\_\_.$$

Group Work: Solutions

1. Fix i. For the j'th row a_j of A_i, consider the hyperplane $P_j = \{x \in \mathbb{S}^d : a_j^T x = 0\}$. If the j'th coordinate of $A_i x$ is negative, this means that $h_i(x)$ lies on one side of P_j, and if that coordinate is positive, it lies on the other. Thus, if $h_i(x) = h_i(y)$, then x and y lie on the same side of each of the hyperplanes P_j defined by the rows of A_i.

   By the spherical symmetry of Gaussians, each one of these hyperplanes is uniformly random. Thus, we get the geometric intuition from the exercies.

2. Because of the geometric intuition from above, the probability over h_i that $h_i(x) = h_i(y)$ is the probability that none of the k hyperplanes go separate x and y. The

probability that a single hyperplane separates x and y is the angle between x and y divided by π. (This is clear if you think about the projection onto the plane spanned by x and y; our random hyperplane is just a random line through the origin in this projection).

Thus, the probability that none of the k hyperplanes separate x and y is $(1 - angle(x,y)/\pi)^k$, using the independence of the k hyperplanes.

3. The probability that $h_i(x) \neq h_i(y)$ for all $i \in [s]$ is, by the previous part,

$$(1 - (1 - \text{angle}(x,y)/\pi)^k)^s.$$

4. Recall that we chose $s = \sqrt{n}$ and $k = \pi \log n/(2r)$. If we assume that the angle between x and y is pretty small compared to π, then we can approximate

$$(1 - angle(x,y)/\pi)^k \approx e^{-\text{angle}(x,y)k/\pi} = e^{-\text{angle}(x,y)\log n/(2r)} = n^{-\text{angle}(x,y)/(2r)}$$

with the choice of k. Then, the whole probability of collision from the previous part is approximately

$$e^{-sn^{-\text{angle}(x,y)/(2r)}} = \exp(-n^{1/2-\text{angle}(x,y)/(2r)}).$$

5. (a) If $\text{angle}(x,y) \leq r$, then

$$\Pr[\exists i \in [s], h_i(x) \neq h_i(y)] \lesssim \exp(-n^{1/2-1/2}) = 1/e,$$

so

$$\Pr[\exists i \in [s], h_i(x) = h_i(y)] \geq 1 - 1/e$$

or so.

(b) If $\text{angle}(x,y) \geq 5r$, then

$$\Pr[\exists i \in [s], h_i(x) \neq h_i(y)] \gtrsim \exp(-n^{1/2-5/2}) = \exp(-n^{-2}) \approx 1 - 1/n^2,$$

so

$$\Pr[\exists i \in [s], h_i(x) = h_i(y)] = O(1/n^2).$$

Suppose that $\mathcal{H}$ is a family of hash functions $h : \mathbb{S}^d \to \mathcal{D}$. We say that $\mathcal{H}$ is a *locality sensitive hash (LSH) family* (for the Euclidean metric, with some parameters R, C, p_1, p_2) if:

- If $\|x - y\|_2 \leq R$, then $h(x) = h(y)$ with probability at *least* p_1.
- If $\|x - y\|_2 \geq CR$, then $h(x) = h(y)$ with probability at *most* p_2.

Thus, if we pretend that "$\text{angle}(x,y)$" was "$\|x - y\|_2$", we have just shown that the family of random hash functions from which we chose the h_i is a locality-sensitive hash family.

(Actually, formally we showed something a bit different, since we looked at the probability of *any* collision over s functions drawn from the family).

In the next two problems, you'll see how to use this LSH family to solve the approximate near-neighbors problem.

Group Work

6. Pretend that "angle(x, y)" was "$\|x - y\|_2$" everywhere.

   Come up with a data structure that uses your result from problem 5b and show that it gives a (c, r)-near-neighbors scheme for some constant c. (It's okay if each query succeeds with probability 1/2 or something like that).

   Hint: *As your data structure, consider storing s hash tables, one for each h_i. Hash each item $x \in S$ into these tables. Given a query y, in what bucket(s) should you look for a close-by $x \in S$?*

7. Explain why it's okay to pretend that "angle(x, y)" is "$\|x - y\|_2$," perhaps at the cost changing the constants around.

   Hint: *You can use the fact that*

$$\frac{2}{\pi}\text{angle}(x, y) \le \|x - y\|_2 \le \text{angle}(x, y)$$

   for any $x, y \in \mathbb{S}^d$.

8. **(If you have time)** What is the amount of space that your data structure uses? How much time does a query take?

Group Work: Solutions

1. Following the hint, our data structure will store s hash tables, one for each h_i. (It will also store the h_i). Then we will hash each $x \in S$ in each of the s tables (so, each table will include a pointer to x in some cell).

   To query on y, we do:

   - For $i = 1, 2, \ldots, s$:
     - Compute $h_i(y)$.
     - If there is some $x \in S$ so that $h_i(x) = h_i(y)$, return x.

   To see why this works, notice that if $\|x - y\|_2 \le r$, then with probability at least $1 - 1/e$ (from the above) there will be some i so that $h_i(x) = h_i(y)$. So we'll definitely return something. To make sure that we don't return something incorrect, suppose that $z \in S$ has $\|x - z\|_2 \ge 4r$. Then by (b) above, the probability that there exists some $i \in [s]$ so that $h_i(x) = h_i(z)$ is at most $1/n^2$. By a union bound, the probability that there is any such z is at most $O(1/n)$. Thus, with probability at least $1-1/e-O(1/n)$, we will return something, and that something will be within $5r$ of y.

2. In 5(a), because of the hint, we can replace $\text{angle}(x,y) \le r$ with $\|x-y\|_2 \le \pi r/2$. In 5(b), we can just replace $\text{angle}(x,y) \ge 5r$ with $\|x-y\|_2 \ge 5r$. Thus, if we set $r' \leftarrow \pi r/2$, we can do the whole thing with r', and instead of 5 our constant c becomes $5\pi/2$.

3. The amount of space our data structure takes up is:
   - s different $k \times d$ matrices A_i (space $skd = O(\sqrt{n} \cdot (\log n/r)d) = O(d\sqrt{n}\log n)$)
   - s hash tables, each with 2^k buckets: $O(\sqrt{n} \cdot 2^{\pi \log n/(2r)}) = \text{poly}(n)$ if r is constant.
   - The elements of S themselves: $O(nd)$.

   So the total space is $\text{poly}(n,d)$ as desired.

   The query time is:
   - s different $k \times d$ matrix vector multiplies: $O(skd) = O(d\sqrt{n}\log n)$ time.
   - Going through all s hash tables and look in bucket $h_i(y)$: time $O(s) = O(\sqrt{n})$.

   So the total query time is $O(d\sqrt{n}\log n)$, which is $o(n)$ as desired if d isn't too big. Hooray!

Group Work

If you finish the rest, here's some bonus stuff to think about!

1. Why does a solution to (r,c)-near-neighbors give a solution to c-approximate-nearest-neighbors?
2. What happens if our data don't live on the surface of $\mathbb{S}^d$? Explain how to still use the analysis above.
3. Can you think of a way to come up with a better LSH family?
4. Can you think of a way to solve approximate near(est) neighbors *without* going through LSH? Is LSH necessary?

Group Work: Solutions

1. Pick a very small $\varepsilon > 0$. Imagine running the (r,c)-near-neighbors algorithm for $r = \varepsilon, r = 2\varepsilon, r = 4\varepsilon, \ldots, r = 2^i\varepsilon, \ldots, r = 2$, and stop when your (r,c)-near-neighbors algorithm first returns a legit answer. Then return that answer.

   First, suppose that $\min_j \|x_j - y\|_2 \le \varepsilon$. In that case, our very first call to near-neighbors will return some i so that $\|x_i - y\|_2 \le c\varepsilon$.

On the other hand, suppose that $\varepsilon < \min_j \|x_j - y\|_2 \le 2$. (Notice that since everything lives on $\mathbb{S}^1$, nothing is further than 2 from anything else). Say that $\min_j \|x_j - y\|_2 \in (\varepsilon 2^t, \varepsilon 2^{t+1}]$. Then the call to near-nbrs when $r = \varepsilon 2^{t+1}$ will return x_i so that $\|x_i - y\|_2 \le c\varepsilon 2^{t+1} \le 2c\varepsilon \min_j \|x_j - y\|_2$.

Thus, if this procedure returns x_i, the distance $\|y - x_i\|_2$ is bounded by the maximum of these two cases, which is bounded by the sum of these two cases, and that's $2c \min_j \|x_j - y\|_2 + c\varepsilon \le 2c(\min_j \|x_j - y\|_2 + \varepsilon)$. So we solve the $2c$-nearest-nbrs problem, assuming that ε is sufficiently small.

Note that this doesn't exactly solve the ANN problem (e.g., if you query some $x \in S$, the guarantees are not the same), but it can be made to work. See "Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality" by Har-Peled, Indyk and Motwani (ToC 2012).

2. If the data don't live on $\mathbb{S}^d$, then it turns out you can project them onto $\mathbb{S}^{d+1}$ without too much distortion.

3. There are indeed ways! Check out "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions" by Andoni and Indyk. (CACM 2008). https://www.fi.muni.cz/~xkohout7/Research/clanky_cizi/lsh/indyk.pdf One goal is to improve the exponent on the query time. That is, if the query time is $O(n^\rho)$ for $\rho < 1$, we'd like to make ρ as small as possible. For LSH-based schemes, the paper linked above shows how to make $\rho = 1/c^2 + o(1)$, where c is the parameters in the c-approximate-NN problem.

4. LSH is not necessary, and in fact you can (provably) do better without it! See https://www.cs.columbia.edu/~andoni/papers/subLSH.pdf for more details.