Due date: March 04, 2025

Problem 1: Cosine Distance & LSH. In this problem we consider a family $\mathcal{F}$ of hash functions built upon the idea of cosine distance. Consider some high-dimensional space $\mathbb{R}^n$. Each hash function f in our collection is constructed from a randomly chosen vector $v_f \in \mathbb{R}^n$. Given two vectors x and y, we say $f(x) = f(y)$ if and only if the dot products $v_f^T x$ and $v_f^T y$ have the same sign.

(1) Describe one way to define f (i.e., given x, how do you define $f(x)$?).

(2) Describe a geometric interpretation of these hash functions. What does it mean if $f(x) = f(y)$? Explain why $\mathcal{F}$ can be viewed as a locality-sensitive family for the cosine distance on $\mathbb{R}^n$.

(3) Given $r \in (0,1)$ and ϵ, find $p_1, p_2 \in (0,1)$ such that $\mathcal{F}$ is $(r, (1+\epsilon)r, p_1, p_2)$-sensitive under cosine distance between v_f.

(4) Suppose now we want to amplify the sensitivity, i.e., aim for larger $p_1' < 1$ and smaller $p_2' > 0$. Describe how you would modify $\mathcal{F}$ or your previous solutions (or both) to achieve this. Try to be as detailed as possible, though the grading will prioritize right high-level ideas over the math.

Solution.

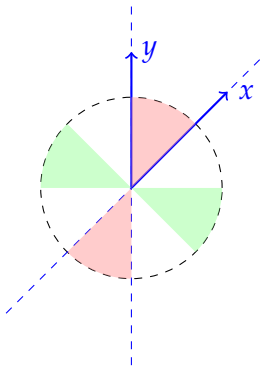
(1) Since given x, y, we want $f(x) = f(y)$ iff $v_f^T x$ and $v_f^T y$ share the same sign, we can simply use the sign function (which is commonly denoted as sgn() or sign())

$$f(x) = \text{sign}(v_f^T x) = \begin{cases} 1 & \text{if } v_f^T x \geqslant 0 \\ -1 \text{ otherwise.} \end{cases}$$

(2) Recall that two vectors have a positive inner product if their angle is acute, and negative otherwise. In this sense, the hyperplane $H_f = \{x \mid x_f^T x = 0\}$ partitions the entire space into two parts, where $\{x \mid x_f^T x > 0\}$ is made up of vectors "close" to v_f so that $f(x) = 1$, and the opposite for $\{x \mid x_f^T x < 0\}$.

Therefore, if $f(x) = f(y)$ then x, y lie on the same side of H_f, which roughly means x is somewhat "close" to y.

In more mathematical details, let $\theta(x, y)$ be the angle between x and y. For a function f to "separate" them (i.e. make $f(x) \neq f(y)$), the corresponding H_f must lie between their intersection. This happens with probability $\theta(x, y)/\pi$ — see the figure below. Given x, y, the red regions are where the hyperplanes H_f can be in order for $f(x) \neq f(y)$. Correspondingly, as v_f is orthogonal to H_f, they can lie in the green regions. The total probability of this happening is $(2\theta(x, y))/(2\pi) = \theta(x, y)/\pi$.



Why does this lead to LSH? If x, y are close, their angle θ is small, and it is unlikely that a randomly chosen hash function separates them; conversely, if x, y are far (the most extreme case being $x = -y$ so the angle is π), then with probability 1 a randomly chosen hash function will separate them.

(3) [Solutions partly due to Nick Maroulis] Let $d(x, y)$ denote the cosine distance between x and y. If $d(x, y) \leqslant r$ then $1 - \cos[\theta(x, y)] \leqslant r$, or $\theta(x, y) \leqslant \arccos(1 - r)$. The previous part implies

$$\mathbb{P}(f(x) = f(y) \mid d(x, y) \leqslant r) \geqslant 1 - \frac{\arccos(1 - r)}{\pi}. \tag{*}$$

Likewise, if $d(x, y) \geqslant (1+\epsilon)r$ then $1 - \cos[\theta(x, y)] \geqslant (1+\epsilon)r$, i.e., $\theta(x, y) \geqslant \arccos(1 - (1+\epsilon)r)$, so

$$\mathbb{P}(f(x) = f(y) \mid d(x, y) \geqslant (1+\epsilon)r) \leqslant 1 - \frac{\arccos(1 - (1+\epsilon)r)}{\pi}. \tag{**}$$

Setting p_1, p_2 to the RHS of (*), (**) respectively completes this part.

(4) [Solutions partly due to Nick Maroulis] TL;DR: concatenating individual hash functions help reduce p_2 but in doing also also inadvertently reduces p_1. To fix this, we use multiple hash tables to boost p_1.

To amplify sensitivity, we concatenate multiple hash functions into one. In (2), a hash function partitions the space into two parts, and two points collide iff they belong to the same part. Here, we randomly choose k hash functions $f_1, \ldots, f_k$. Correspondingly, the space is now divided into many more fine-grained regions. Defining $g = (f_1, \ldots, f_k)$ such that $g(x) = g(y)$ iff $f_i(x) = f_i(y)$ for *all* i gives us a much stronger function such that two points collide iff they belong to the same (smaller) region. Hence, g helps us reduce the probability of collision between dissimilar points into p_2^k.

However, p_1 now also becomes p_1^k. To fix this, we define multiple hash tables, say m copies of g_i (each consisting of k individual f_j's), and say $h(x) = h(y)$ iff $g_i(x) = g_i(y)$ for *some* g_i.

Problem 2: ANN Query. Let S be a set of n points in $\mathbb{R}^2$, let q_i be the charge of point p_i in S, and let ϵ be a parameter. Assume that the spread of S is $n^{O(1)}$. Define

$$F(S) = \sum_i \sum_{j \neq i} \frac{q_i q_j}{\|p_i - p_j\|^2}.$$

Describe an $O(n\varepsilon^{-2} \log n)$ algorithm to compute an estimate $\tilde{F}(S)$ of $F(S)$ that has a multiplicative error of $\leqslant \epsilon$, i.e., $|\tilde{F}(S) - F(S)| \leq \varepsilon F(S)$. **Hint**: *Use the quad tree query process covered in lecture.*

Solution. We construct a quad tree T on S. For each node $v \in T$, we associate it with a total charge $q(v) = \sum_{p_j \in S_v} q_j$. We also let c_v be the square associated with v, and write its side length/diameter as ℓ_v. For a fixed point $x \in \mathbb{R}^2$, $d(x, c_v) = \min_{y \in c_v} \|x - y\|$.

The total time spent in preprocessing is $\mathcal{O}(n \log n)$. For a point p_i, we visit T in a top-down manner and maintain a queue Q and a count F.

Below, we present a function that, for a given point $p_i \in S$, approximately computes

$$F(p_i) = q_i \sum_{j \neq i} \frac{q_j}{\|p_i - p_j\|^2}$$

within a multiplicative error of ϵ, i.e., it outputs a value $\tilde{F}(p_i)$ such that $|F(p_i) - \tilde{F}(p_i)| \leqslant \epsilon F(p_i)$.

Algorithm 1 Compute-Charge(p_i)

$Q \leftarrow \{\text{root}\}$, $F \leftarrow 0$
while $Q \neq \emptyset$ **do**
  $v \leftarrow \text{dequeue}(Q)$
  if v is a leaf **then**
    Let p_j be the unique point in S_v
    if $p_i \neq p_j$ **then**
      $F \leftarrow F + \dfrac{q_i \cdot q_j}{\|p_i - p_j\|^2}$
    end if
  else if $\ell_v \leq \frac{\epsilon}{3} \cdot d(p_i, c_v)$ **then** ▷ $(*)$ x_v: center of c_v
    Let x_v be the center of c_v
    $F \leftarrow F + \dfrac{q_i \cdot q(v)}{\|p_i - x_v\|^2}$
  else
    Insert children of v into Q ▷ v is internal and $\ell_v \geq \frac{\epsilon}{3} \cdot d(p_i, c_v)$
  end if
end while
return F

For a node v in line (*), all points of S_v lie inside the distance range $[d(p_i, c_v), (1 + \epsilon/3)d(p_i, c_v)]$. The value F returned by the algorithm is an ϵ-approximate value of $F(p_i)$,

namely $|F - F(p_i)| \leqslant \epsilon/2F(p_i)$. Querying over all points of S and adding all the returned values, we obtain the desired $\tilde{F}(S)$. Following the same analysis as for ϵ-ANN, the above procedure visits $O(\log n + \epsilon^{-2})$ nodes. Hence the overall runtime is $O(n(\log n + \epsilon^{-2}))$.