Locality Sensitive Hashing and Approximate Nearest Neighbors

Kent Quanrud

November 27, 2020

1 Nearest neighbor search

In the previous discussion on dimensionality reduction, we mentioned the many natural settings where data can be represented as numerical vectors in an high-dimensional space. Surprising, by simply randomly projecting by Gaussian vectors, one can embed an n-point data set into $O\big(\log(n)/\epsilon^2\big)$ dimensions while preserving all pairwise distances up to an $(1 \pm \epsilon)$-multiplicative factor.

A natural query in many different domains is *similarity search* where we want to preprocess a large collection of items such that, given a query in the form of an item, we can quickly retrieve the "most similar" item in our collection by some metric. The "most similar" item is sometimes called the **nearest neighbor**.

A simple example might be where we have a collection of n real numbers. Given another number as a query, we want to retrieve the closest number in our collection to the query number. For this problem, we would sort our collection of numbers and store them in an array. Given a query $q \in \mathbb{R}$, we run binary search on our array to find the first number smaller and bigger than q. We return the closer of the two. This approach would take $O(\log(n))$ time. The above approaches extends to low-dimensional data sets, via **quadtrees** and multi-dimensional **range trees**. Both of these approaches scale exponentially in the d. If the data lives very few dimensions - like 3 dimensions, as in many physical situations - this is still very good.

Continuing the previous discussion on dimensionality reduction, here we are interested in nearest neighbor search with *high dimensional data*. We have a collection of n points $P \subset \mathbb{R}^d$ that we can preprocess to build some kind of data structure. Each query is in the form of another point $z \in \mathbb{R}^d$, and the goal is to output the point $x \in P$ closest to z by some metric. Here we will consider two settings.

1. The **Euclidean distance**, $\|x - z\|$.
2. The **angle** between x and z (as a real value between 0 and π).

If all the vectors in P are normalized to have the same length, then the nearest neighbor in Euclidean distance and angle are the same. However, we will consider *approximations* for this problem, in which case there is a difference between the two metrics. We consider angular nearest neighbors in Section 2 and Euclidean nearest neighbors in Section 3.

A natural idea, at least for Euclidean distance, is to use the dimensionality reduction techniques from the previous discussion to reduce to $O(\log(n))$ dimensions, and apply the low-dimensional data structures. But the exponential dependence in the dimension means that even $O(\log(n))$ dimensions – which is fairly small by our standards – will still require $n^{O(1)}$ time and space.

From nearest neighbor to close enough. Fix $\sigma > 1$. Enumerating distances to the nearest neighbor by powers of σ allows us to reduce to the following problem with only logarithmic overhead:

> *Let a target distance $r > 0$ be fixed. Given a query point z, either output a point at distance $\leq \sigma r$, or declare that all points have distance $\geq r$.*

It also suffices, up to logarithmic factors, to succeed with constant probability. We can amplify to high probability with repetition.

When hash collisions are good. The algorithms we develop are based on hashing, although the intuition is the opposite of previous discussions on hashing. Previously, in applications such as heavy hitters and hash tables, hashing was used to randomly *spread out* the elements. Here, we will design hash functions where closer points are *more likely to collide*. This technique is called **locality sensitive hashing** [1].[1] The high level strategy is to build a hash table over all the input set P using locality sensitive hash functions. Given a query point z, we hash z and hope to find the nearest neighbor in the same hash bucket. In the algorithms we discuss, this algorithm will only succeed with limited probability. We amplify by building many such hash tables independently. On a query point z, we hash z into all of the hash tables, and return the first point we find that is close enough.

2 Locality Sensitive Hashing for Angles

We consider the setting of *angular distance*, which is commonly used when the point set P lies on the hypersphere $\mathbb{S}^{d-1} = \left\{x \in \mathbb{R}^d : \|x\| = 1\right\}$. In this case a common measure of distance is the *angle* between points. For two points x and y, let $\angle(x, y) \in [0, \pi]$ denote the angle between the points x and y between 0 and π. We may assume that all the points in our data set P, as well as any query point, is normalized to lie on $\mathbb{S}^{d-1}$. We describe an LSH scheme due to [2].

Let $\theta \in [0, \pi]$ be a fixed angle. In this discussion, given a query point y, our goal is to either output a point $x \in P$ with $\angle(x, y) \leq 4\theta$, or decide that there are no points $x \in P$ with $\angle(x, y) \leq \theta$.

Let $k \in \mathbb{N}$ be a parameter TBD. We will generate a k-ary hash function $h : \mathbb{S}^{d-1} \to \{-1, 1\}^k$ where each coordinate is generated by splitting $\mathbb{S}^{d-1}$ in half along a random hyperplane. For each coordinate $i \in [k]$, let $g_i \sim \mathcal{N}^d$ be a random Gaussian vector. We define a hash function $h : \mathbb{S}^{d-1} \to \{-1, 1\}^k$ by

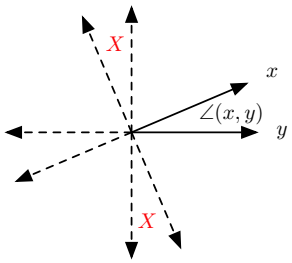
$$h_i(x) = \text{sign}(\langle g_i, x \rangle)$$

for each coordinate i.

We will leverage the following fact that is entirely based on the rotational invariance of $\mathcal{N}^d$.

Lemma 1. *Let $x, y \in P$, and let $g \sim \mathcal{N}$.*

$$\mathrm{P}[\text{sign}(\langle g, x \rangle) \neq \text{sign}(\langle g, y \rangle)] = \frac{\angle(x, y)}{\pi}.$$



Proof. Recall that $\mathcal{N}^d$ is *rotationally symmetric*. Rotating, we may assuming that x and y are spanned by the first two coordinates, and consider the simpler (to visualize) setting where $x, y \in \mathbb{R}^2$ and $g \sim \mathcal{N}^2$, as on the right.

[1] Here the author would like to propose the term *clashing*.

We are particularly interested in the angle of the random vector g. Since $\mathcal{N}^2$ is rotationally symmetric, it is equally likely to take any particular angle with equal probability. Out of a total of 2π, there are two regions of angle $\angle(x,y)$ in which $\text{sign}(\langle g,x\rangle) \neq \text{sign}(\langle g,y\rangle)$, here marked with a red X. ■

Now, let $k = \dfrac{\pi \log(n)}{2\theta}$. If $\angle(x,y) < \theta$, then

$$\mathrm{P}[h(x) = h(y)] = \left(1 - \frac{\angle(x,y)}{\pi}\right)^k \approx e^{-\frac{\angle(x,y)}{\pi}k} = e^{-\log(n)/2} = \frac{1}{\sqrt{n}}.$$

On the other hand, if $\angle(x,y) > 2\theta$, then

$$\mathrm{P}[h(x) = h(y)] = \left(1 - \frac{\angle(x,y)}{\pi}\right)^k \leq e^{-\angle(x,y)k/\pi} \leq e^{-\log(n)} = \frac{1}{n}.$$

Thus, when querying a point y:

1. If there is a point $x \in P$ with $\angle(x,y) \leq \theta$, then it will collide with probability (approximately) $\geq 1/\sqrt{n}$.
2. We expect to collide with at most 1 point $x \in P$ such that $\angle(x,y) \geq 2\theta$.

If we repeat the experiment $O\big(\sqrt{n}\log(n)\big)$ times, then with high probability we will find a close neighbor if one exists. The query time is $O\big(d\sqrt{n}\log(n)\big)$ in expectation, because each time we expect 1 "junk" neighbor on average.

Theorem 2. *With $O\Big(dn^{3/2}\,\text{polylog}(n)\Big)$ preprocessing time and space, one can query for 2-approximate nearest neighbors w/r/t angular distance in $O\big(d\sqrt{n}\,\text{polylog}(n)\big)$ expected randomized time and with high probability.*

3 Locality Sensitive Hashing for Euclidean Distance

We now consider σ-approximate nearest neighbors in Euclidean metrics, for fixed $\sigma > 1$. As mentioned above, it suffices to consider the simpler setting where there is a fixed target distance r. We want to preprocess a set of n points $P \subseteq \mathbb{R}^d$ as to quickly answer the following query with constant probability of success:

> Given a query point $z \in \mathbb{R}^d$, either find a point $x \in P$ with $\|x - z\| \leq \sigma r$, or declare that there are no points $x \in P$ with $\|x - z\| \leq r$.

Our goal is to develop a locality-sensitive hash function[2] $h : \mathbb{R}^d \to \mathbb{Z}^k$ for Euclidean distance. We will design h such that the collision probobilities of two points $x, y \in \mathbb{R}^d$ depend only on the ratio $\|x - y\|$ to r. Therefore, we can also rescale and assume that $r = 1$.

> Given a query point $z \in \mathbb{R}^d$, either find a point $x \in P$ with $\|x - z\| \leq \sigma$, or declare that there are no points $x \in P$ with $\|x - z\| \leq 1$.

[2] a *clash function*?

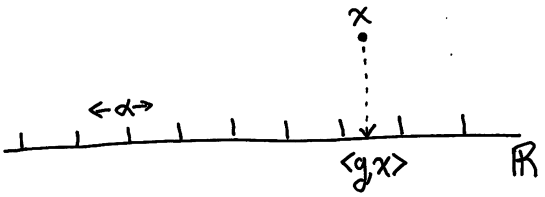
3.1 Random line embeddings and buckets

In this section, we first define a hash function $h : \mathbb{R}^d \to \mathbb{Z}$ that outputs a single hash code. Later we will consider a k-coordinate hash where each coordinate is an independent copy of the single coordinate hash we consider here.

We define $h : \mathbb{R}^d \to \mathbb{Z}$ by the function

$$h(x) = \lfloor \langle g, x \rangle + \alpha \rfloor,$$

where $g \sim \mathcal{N}^d$ and $\alpha \in [0, 1]$ uniformly at random. Geometrically, we can interpret $h(x)$ as randomly projecting x onto a line, and then bucketing the points in intervals of length 1. The random value $\alpha \in [0, 1]$ translates the buckets randomly.



Lemma 3. *Let $x, y \in \mathbb{R}^d$.*

$$\mathrm{P}[h(x) = h(y) \mid g] = \max\{0, 1 - |\langle x - y, g \rangle|\}.$$

Proof. Once the Gaussian is fixed, so are the coordinates $\langle g, x \rangle$ and $\langle g, y \rangle$ on the line. We have $h(x) \neq h(y)$ iff a randomly shifted divider (determined by α) falls between $\langle g, x \rangle$ and $\langle g, y \rangle$. If $|\langle g, x \rangle - \langle g, y \rangle| \geq 1$, this this always happens. Otherwise, it happens with probability $|\langle g, x \rangle - \langle g, y \rangle|$. ■

Lemma 4. *Let $x, y \in \mathbb{R}^d$, and let $f(t)$ be the density function of the standard Gaussian $\mathcal{N}$.*

$$\mathrm{P}[h(x) = h(y)] = 2 \int_0^1 f(\|x - y\| t)(1 - t)$$

Proof. Recall that $\langle x - y, g \rangle \sim \mathcal{N}\big(0, \sigma^2\big)$. In particular, $\langle x - y, g \rangle$ has density function $f(\sigma t)$, and $|\langle x - y, g \rangle|$ has density function $2f(\sigma t)$. We have

$$\begin{aligned} \mathrm{P}[h(x) = h(y)] &= 2 \int_0^1 \mathrm{P}[h(x) = h(y) \mid |\langle g, x - y \rangle| = t] f(\|x - y\| t)\, dt \\ &= 2 \int_0^1 (1 - t) f(\|x - y\| t)\, dt, \end{aligned}$$

as desired. ■

Remark 5. The collision probability obtained in Lemma 4 is a function of σ, which is the ratio between $\|x - y\|$ and the target distance (here normalized to 1). This justifies our normalization. Note also that $\mathrm{P}[h(x) = h(y)]$ is decreasing in $\|x - y\|$.

Lemma 4 derives the exact probability of a collision of two points x and y as a function of the distance between $\|x - y\|$. Let us compare the probabilities of a "close" pair of points, with $\|x - y\| \leq 1$, and a far pair of points, with $\|x - y\| \geq \sigma$. Let

$$p = 2 \int_0^1 (1 - t) f(t)\, dt$$

be a lower bound on the collision probability when $\|x - y\| \leq 1$. Let

$$q = 2\int_0^1 (1-t)f(\sigma t)\,dt$$

be an upper bound on the collision probability when $\|x - y\| \geq \sigma$. We note that p is a fixed constant, about .368746....

Since $f(\sigma t)$ is decreasing in σ, $p > q$. That is, close points are more likely to collide then far points. To what extent? It turns out that the gap is not enough to simply use h as a locality sensitive hash function. But we can remedy this simply by amplification, as follows.

3.2 Amplifying the gap

Let $k \in \mathbb{N}$ be a parameter TBD. We define a hash function

$$h : \mathbb{R}^d \to \mathbb{R}^k$$

by defining each coordinate $h_i(x)$ according to the single-coordinate hash function in Section 3.1; namely, as

$$h_i(x) = \lfloor \langle g_i, x \rangle + \alpha_i \rfloor$$

where $g_i \sim \mathcal{N}$ and $\alpha_i \in [0, 1]$ uniformly at random.

For any two points x and y, by Lemma 4, we have

$$\mathrm{P}[h(x) = h(y)] = \left(2\int_0^1 (1-t)f(\|x-y\|t)\,dt\right)^k.$$

In particular, recalling the values of p and q as above, we have

$$\mathrm{P}[h(x) = h(y)] \geq p^k \text{ when } \|x - y\| \leq 1$$

and

$$\mathrm{P}[h(x) = h(y)] \leq q^k \text{ when } \|x - y\| \geq \sigma.$$

Now, k decreases both p^k and q^k decreases which is both good and bad. As p^k decreases, so do the odds of finding a near neighbor when we hash. We will need to rebuild the data structure $\ell = 1/p^k$ times to be able to find a good neighbor with constant probability, which is expensive.[3] On the other hand, as q^k decreases, the number of hash collisions with bad points also reduces. Ultimately, the algorithm pays a running time proportional to the total number of bad collisions as it scans the lists in the hash bucket as it searches for a good quantity. Overall, the ratio $(q/p)^k$ decreases, so to at least some extent k is useful.

Ultimately, the quantity we want to minimize is

$$k\left(\frac{1}{p}\right)^k + \left(\frac{q}{p}\right)^k n.$$

[3]Indeed, the probability of failing to find the good neighbor in each of ℓ constructions, each of which has k hash coordinates, is

$$\left(1 - p^k\right)^\ell \approx e^{-p^k \ell}.$$

The $(1/p)^k$ term represents having the compute $\ell = (1/p)^k$ hash codes. The $\left(\frac{q}{p}\right)^k n$ represents the expected number of hash collisions across all ℓ instances with bad elements. The above quantity models the running time up to an additional factor of d, incurred from either hashing the query point or comparing the distance between the query point and a point in the same hash bucket. We can rewrite the above as

$$\left(\frac{1}{p}\right)^k \Big(k + q^k n\Big).$$

Choose $k = \log(1/n)/\log(q) = \log(n)/\log(1/q)$. Then $q^k n = 1$, and

$$k\left(\frac{1}{p}\right)^k = \frac{\log(n)}{\log(1/q)} n^{\log(1/p)/\log(1/q)}.$$

Consider the exponent $\log(1/p)/\log(1/q)$: since $p > q$, this quantity is less then 1. In fact, one can show that it is about $1/1+\epsilon$.

Theorem 6. *One can compute an $(1+\epsilon)$-approximate nearest neighbor w/r/t Euclidean distance with high probability in $\tilde{O}\Big(n^{\rho(\sigma)}\Big)$ randomized time, where*

$$\rho(\sigma) = \frac{\log(1/p)}{\log(1/q)}, \qquad p = 2\int_0^1 (1-t) f(t)\, dt \approx .368746, \qquad q = 2\int_0^1 (1-t) f(\sigma t)\, dt,$$

and $f(t) = e^{-t^2/2}/\sqrt{2\pi}$ is the density function of the standard Gaussian.

Below, we show that for $\sigma^2 \leq 1.6$, we have

$$\rho(\sigma) \leq 1/\sigma^2.$$

3.3 Analyzing $\rho(\sigma)$

For any fixed value of $\sigma > 1$, one can run a numerical computation to get an accurate estimate of $\rho(\sigma)$. See for example [3]. Below, we analyze $\rho(\sigma)$ and obtain some upper bounds on ρ. While these bounds are of course not as tight as a computer simulation, they reveal some intuition for how the value of $\rho(\sigma)$ varies with σ.

Lemma 7. *Suppose $\sigma^2 = 1+\epsilon$ for $\epsilon \in (0,1)$. Then*

$$q \geq (1-\epsilon)^2 p.$$

In particular,

$$\rho(\sigma) \geq \frac{\log(1/p)}{\log(1/p) - 2\log(1-\epsilon)}.$$

Proof. We have

$$\sqrt{2\pi}(p-q) = \int_0^1 (1-t)\Big(e^{-t^2/2} - e^{-\sigma^2 t^2/2}\Big)$$

Say $\sigma^2 = (1+\epsilon)$ with ϵ sufficiently small. Then for all $t \in [0,1]$, we have

$$e^{-t^2/2} - e^{-(1-\epsilon)t^2/2} = e^{-t^2/2}\Big(1 - e^{-\epsilon t^2/2}\Big) = e^{-t^2/2}\Big(\epsilon t^2/2 - \big(\epsilon t^2/2\big)^2/2\Big) \leq \epsilon(1-\epsilon)e^{-t^2/2}.$$

Thus

$$\begin{aligned}\int_0^1 (1-t)\Big(e^{-t^2/2} - e^{-t^2/2\sigma^2}\Big)\,dt &\leq \epsilon(1-\epsilon)\int_0^1 (1-t)e^{-t^2/2}\,dt \\ &= \epsilon(1-\epsilon)\sqrt{2\pi}p.\end{aligned}$$

Thus

$$\sqrt{2\pi}(p-q) \leq \epsilon(1-\epsilon)\sqrt{2\pi}p;$$

in turn;

$$(1-\epsilon)^2 p \leq q.$$

■

Lemma 8. *Let $\sigma^2 \leq 1.6$. Then*

$$\rho(\sigma) \leq \frac{1}{\sigma^2}.$$

Proof. We continue from the conclusion of Lemma 7 with $\epsilon = \sigma^2 - 1$. One can directly compute that $\log(1/p)$ is a constant, about (and slightly greater than) .77. On the other hand, $\log(1-\epsilon) \approx -\epsilon$ for small ϵ; in particular, $-2\log(1-\epsilon) \geq -\log(1/p)\epsilon$ for $\epsilon \leq .6$. Thus

$$\rho(\sigma) \leq \frac{\log(1/p)}{\log(1/p) - 2\log(1-\epsilon)} \leq \frac{\log(1/p)}{(1+\epsilon)\log(1/p)} = \frac{1}{1+\epsilon} = \frac{1}{\sigma^2}.$$

■

4 Exercises

Exercise 1. In this exercise, we will develop a 2-approximation LSH scheme for bit strings $s \in \{0,1\}^d$ of a fixed length d w/r/t *Hamming distance*. The Hamming distance between two strings $s, t \in \{0,1\}^d$ this fraction of coordinates in which they differ:

$$(\text{Hamming})(s,t) = \frac{|\{i \in [d] : s_i \neq t_i\}|}{d}.$$

Of course one can treat bit strings as vectors in $\mathbb{R}^d$ where the Hamming distance coincides with the Euclidean distance squared. Here we explore an alternative approach.

1. Consider the randomly constructed hash functinon $h : \{0,1\}^d \to \{0,1\}$ defined by

   $$h(x) = x_i,$$

   where $i \in [d]$ is sampled uniformly at random. For two points $s, t \in \{0,1\}^d$, what is $\mathrm{P}[h(s) = h(t)]$, as a function of the Hamming distance between s and t?

2. Fix a target distance $r \in [0, 1]$. Construct a data structure that over a set P of n strings $\{0,1\}^d$ to answer the following query with high probability.

   > *Given a query point $s \in \{0,1\}^d$, either return a point $x \in P$ with Hamming distance $\leq 2r$ from s, or declare that there are no points within Hamming distance r from s.*

   In addition to describing the algorithm, one should analyze the preprocessing time and space, the query time, and the probability of correctness.

3. Briefly describe how to use the above data structure to efficiently find 2-approximate nearest neighbors w/r/t Hamming distance.

References

[1] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. Ed. by Jeffrey Scott Vitter. ACM, 1998, pp. 604–613. DOI: 10.1145/276698.276876. URL: https://doi.org/10.1145/276698.276876.

[2] Moses Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. Ed. by John H. Reif. ACM, 2002, pp. 380–388.

[3] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*. Ed. by Jack Snoeyink and Jean-Daniel Boissonnat. ACM, 2004, pp. 253–262.