Data Science Through a Geometric Lens	Fall 2019

Lecture 09 — October 30, 2019

Prof. Cyrus Rashtchian *Topics: LSH for Euclidean Distance*

Overview. Today we present two LSH families: for Cosine similarity and Euclidean distance.

Review. We recap the definition of approximation near neighbor search (ANNS) and locality sensitive hashing (LSH).

Definition 1. *Let $S \subseteq \mathcal{X}$ be a dataset in a metric space $(\mathcal{X}, d_{\mathcal{X}})$. The (c, r)-ANN problem for $c > 1$ and $r \in \mathbb{R}_{\geq 0}$ is to efficiently pre-process S to quickly answer the following query. Given $q \in \mathcal{X}$, either return $x \in S$ with $d_{\mathcal{X}}(q, x) \leq cr$, or report that no such point exists.*

The algorithm knows the approximation c and the threshold r ahead of time. Therefore, we focus on data structures parameterized by c, r. A useful primitive is an LSH family.

Definition 2. *Let r and $c \geq 1$ and $p_1, p_2 \in (0, 1]$ with $p_1 > p_2$ be parameters. A hash family $\mathcal{H}$ in a metric space $(\mathcal{X}, d_{\mathcal{X}})$ is (r, cr, p_1, p_2)-***sensitive** *if the following two conditions are satisfied:*

1. $\Pr[h(x) = h(y)] \geq p_1$ *for all $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \leq r$,*
2. $\Pr[h(x) = h(y)] \leq p_2$ *for all $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \geq cr$,*

where the probability is over sampling a uniformly random $h \in \mathcal{H}$.

1 LSH for Unit Vectors and Cosine Similarity

Cosine similarity refers to a special case of Euclidean distance for unit vectors. We consider datasets $X \subseteq \mathcal{S}^{d-1}$, where we use $\mathcal{S}^{d-1}$ to denote the d-dimensional real vectors with unit norm, that is,

$$\mathcal{S}^{d-1} = \{x \in \mathbb{R}^d \mid \|x\|_2 = 1\}.$$

The distance metric will be the usual ℓ_2 distance, but we can also consider the angle between the data points based on the identity

$$\|x - y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y\rangle.$$

Notice that when $x, y \in \mathcal{S}^{d-1}$, we have that $\|x\|_2^2 = \|y\|_2^2 = 1$. Therefore,

$$\langle x, y\rangle = 1 - \frac{\|x - y\|_2^2}{2}. \tag{1}$$

In this way, we can view the inner product $\langle x, y\rangle$ as a notion of similarity between two vectors, which always has value between -1 and 1 for $x, y \in \mathcal{S}^{d-1}$. This is sometimes called the *cosine similarity*.

We now present an LSH for this, which is known as SimHash, from a paper of Charikar [1].

1.1 Random hyperplanes

Consider choosing a random vector $b \in \mathbb{R}^d$, which has each coordinate sampled according to the standard normal distribution $\mathcal{N}(0,1)$. We could normalize by $\|b\|_2$ to get a uniformly random point on the sphere $\mathcal{S}^{d-1}$ if we wanted to.

We will analyze the hash function $h_b : \mathbb{R}^d \to \{-1,1\}$ that is defined by

$$h_b(x) = \mathsf{sign}(\langle x, b\rangle).$$

The sign function outputs $+1$ if its input is ≥ 0, and otherwise it outputs -1. By choosing the normal vector b uniformly at random, we can choose a random hash function that assigns each vector x a value in $\{-1,1\}$.

This will be the basis of our LSH family for cosine similarity. We will also have a parameter k, which is based on concatenating k copies of independent hash functions. More precisely, choose k random vectors $b_1, b_2, \ldots, b_k$, each with i.i.d. entries from $\mathcal{N}(0,1)$. Then, define

$$h^k(x) = [h_{b_1}(x), h_{b_2}(x), \ldots, h_{b_k}(x)] = [\mathsf{sign}(\langle x, b_1\rangle), \mathsf{sign}(\langle x, b_2\rangle), \ldots, \mathsf{sign}(\langle x, b_k\rangle)].$$

Note that $h^k(x)$ maps vectors in $\mathbb{R}^d$ to k-dimensional sign vectors in $\{-1,1\}^k$. This is analogous to how the LSH family for Hamming distance mapped vectors to k bits.

1.2 Understanding these hash functions

As we are using k independent copies to build h_k, it suffices to analyze a single hash function h_b.

It will be convenient to talk about angles, and for this, we introduce the notation

$$\angle(xy) = \arccos\left(\frac{\langle x, y\rangle}{\|x\|_2\|y\|_2}\right).$$

When we are talking about $x, y \in \mathcal{S}^{d-1}$, we can drop the norms, so $\angle(xy) = \arccos(\langle x, y\rangle)$. We have that $0 \leq \angle(xy) \leq \pi$. Also, recall that the angle between x and y is equal to $\pi/2$ if they are orthogonal, and it is zero if they are identical, and it is π is they are opposite.

Lemma 3. *For any $x, y \in \mathcal{S}^{d-1}$, we have that*

$$\Pr_b[h_b(x) \neq h_b(y)] = \frac{\angle(xy)}{\pi}.$$

and

$$\Pr_b[h_b(x) = h_b(y)] = 1 - \frac{\angle(xy)}{\pi}.$$

Proof. Proof by picture (e.g., draw a 2 dimensional circle). □

We can rewrite the collision probability in a more "friendly" form. We combine Eq. (1) and the identity for $0 \leq z \leq 1$

$$2\arcsin(z) = \arccos(1 - z^2).$$

This leads to

$$\Pr_b[h_b(x) = h_b(y)] = 1 - \frac{2}{\pi}\arcsin\frac{\|x-y\|_2}{\sqrt{2}}. \tag{2}$$

As before, we define p_1 to be this probability when $\|x-y\|_2 = r$, and we let p_2 be the probability when $\|x-y\|_2 = cr$.

Then, recall that the LSH parameter of interest is $\rho = \log(p_1)/\log(p_2)$. Using Eq. (2), we have that this is

$$\rho = \frac{\log(p_1)}{\log(p_2)} \;=\; \frac{\log\left(1 - \frac{2}{\pi}\arcsin\frac{r}{\sqrt{2}}\right)}{\log\left(1 - \frac{2}{\pi}\arcsin\frac{cr}{\sqrt{2}}\right)} \leq \frac{1}{c}. \tag{3}$$

For the last inequality, it can be shown that $\rho \leq 1/c$.

1.3 Wrapping up the ANNS for Cosine similarity

To finish the ANNS algorithm, we have shown that we can use k to control the hash bucket size (recall that we defined the actual hash function to be $h^k(x)$, which concatenates k independent copies of the random hyperplane sign hash).

Then, the probability of close collisions is p_1^k and far collisions is p_2^k. Therefore, using the above Lemma, we can set k such that $p_2^k \leq 1/n$. By Eq. (3), we have that $p_1^k \geq 1/n^\rho \geq 1/n^{1/c}$.

As before, we can set $L = 1/p_1^k = n^\rho \leq n^{1/c}$ to be the number of hash tables that we use. The rest of the algorithm is the same. We use space $Ln = n^{1+\rho}$ because we have L hash tables. And the query time also scales with $L = n^\rho$. Assuming we can store each d dimensional vector with $O(d)$ bits, then the total space is $O(dn^{1+1/c})$, and query time is $O(dn^{1/c})$.

At this point, the details, algorithm, and analysis are essentially identical to the Hamming distance analyze from last lecture, and we leave them to the reader.

2 LSH for Euclidean distance

We now present another LSH family that doesn't make any assumptions about being unit vectors. On query $q \in \mathbb{R}^d$, we want to find a vector with $\|x - q\|_2 \leq cr$ if there is a vector with $\|x - q\|_2 \leq r$ in the dataset. Notice that by rescaling the dataset and the query, we can WLOG assume $r = 1$. That is, we consider finding x with $\|x - q\|_2 \leq c$ if there is a vector with $\|x - q\|_2 \leq 1$ in the dataset.

The hash function will again use a random vector b that is normally distributed. But instead of taking the sign of the inner product, we will use a random offset t and we will break the line into segments of width w.

Intuitively, we will project an input vector x onto a random line (with angle based on the random normal vector b), and we will look at where it falls on this line. We will have infinite potential buckets based partitioning the random line into segments of width w. We will randomly shift these segments by an offset (because it would be bad to always start/end at 0, among other reasons).

More precisely: We fix w, which will be a (somewhat magical) parameter to set at the very end (it will not have a closed form, and it will depend on c). Then, we choose $b \sim \mathcal{N}(0,1)^d$ and $t \sim \texttt{Uniform}[0, w]$. Then, the hash function $h_{b,t}$ is defined as

$$h_{b,t}(x) = \left\lfloor \frac{\langle x, b\rangle + t}{w} \right\rfloor.$$

The hash family will be all possible functions of this form, where we choose b and t uniformly at random. The intuition for why this is a good LSH is that if x and y are close, then they will be more likely to project only the same segment of width w.

We first understand their projection onto b by showing that their distance on this one-dimensional line can be written as a normal random variable. The punchline is that $\langle x-y, b\rangle$ is distributed as the normal distribution $\mathcal{N}(0, \|x-y\|_2^2)$ with variance $\|x-y\|_2^2$ scaling with their distance.

Lemma 4. *If $b \sim \mathcal{N}(0,1)^d$, then $\langle x, b\rangle$ is distributed according to $\|x\|_2 \cdot \mathcal{N}(0,1) = \mathcal{N}(0, \|x\|_2^2)$.*

Proof. Recall the 2-stable property of normal random variables: adding independent normal random variables also leads to a normal random variable; the means add; the variances also add. Each coordinate in b is distributed as $\mathcal{N}(0,1)$. Hence, each coordinate $x_i \cdot b_i$ is distributed as $\mathcal{N}(0, x_i^2)$. And we know that $\sum_i x_i^2 = \|x\|_2^2$, which completes the proof. □

By this lemma, we have that $\langle x-y, b\rangle$ is distributed as $\mathcal{N}(0, \|x-y\|_2^2)$. Therefore, the variance is smaller is x and y are closer, and so there distance when projected onto b is more likely to be small. This is good, because we now show that they are more likely to end up in the same segment when projected onto b:

Lemma 5. *Fix b, but choose $t \sim \texttt{Uniform}[0, w]$ at random. Then,*

$$\Pr_t[h_{b,t}(x) = h_{b,t}(y)] = \max\left(1 - \frac{|\langle x-y, b\rangle|}{w},\ 0\right).$$

In other words, the probability the the offset t separates x and y is proportional to their distance when projected onto b, and this one-dimensional distance is equal to $|\langle x, b\rangle - \langle y, b\rangle| = |\langle x-y, b\rangle|$. The max/zero is because if this distance is larger than w, then x and y will be hashed to different places regardless of t.

Now, we will sample both b and t at random. As before we define

$$p_1 = \Pr_{b,t}[h_{b,t}(x) = h_{b,t}(y)] \quad \text{when } \|x-y\|_2 = 1.$$

$$p_2 = \Pr_{b,t}[h_{b,t}(x) = h_{b,t}(y)] \quad \text{when } \|x-y\|_2 = c.$$

It can be shown that these probabilities are equal to an integral based on the normal distribution. Using this fact, and some numerical analysis (using a computer), it's possible to estimate p_1 and p_2 as a function of w. It has been shown that it is possible to solve for w such that we achieve

$$\rho = \frac{\log p_1}{\log p_2} \leq \frac{1}{c}.$$

Therefore, we get an ANNS data structure for Euclidean distance with $n^\rho \leq n^{1/c}$ hash tables (that is, the query time scales with this). Assuming we can store each d dimensional vector with $O(d)$ bits, then the total space is $O(dn^{1+1/c})$, and query time is $O(dn^{1/c})$.

Improvements. The best known LSH for Euclidean distance actually achieves $\rho = 1/c^2$. The result is due to Andoni and Indyk from 2006. And, it has been shown that this is basically optimal for LSH-based ANNS. Their hash family is more complicated in two ways. First, instead of projecting onto a one-dimensional line, it projects onto a higher dimensional subspace (with a carefully chosen dimension). Second, this subspace is partitioned into balls (based on a lattice that determines their centers). However, it isn't possible to partition the whole space into balls, so the hash function takes the first ball that x lands in (when the balls are ordered randomly). This is known as *ball carving* and it's an important and influential idea. The analysis is quite tricky, based on understanding when x, y falls into a ball after a projection. But it achieves much better results in terms of the ρ exponent.

References

[1] Moses Charikar. Similarity Estimation Techniques From Rounding Algorithms. STOC, 2002.