CS 388R: Randomized Algorithms, Fall 2023 November 8th, 2023

Lecture 22: Nearest Neighbor Search

Prof. Eric Price *Scribe: Aaryan Prakash, Sameer Gupta*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Nearest Neighbor Search

Given some set of points $p_1, \ldots, p_n \in X$, we want to construct a data structure to find a nearby point to a given query point q. An approximation is fine, so we want to find $\hat{p}$ such that $\|\hat{p} - q\| \leq (1+\epsilon)\min_p \|p - q\|$. Today, we will cover $X = \{0,1\}^d$, and the distance is Hamming distance.

One strategy would be to just try every point, which requires $O(nd)$ time and space. We could create a lookup table for all 2^d inputs, so we have $O(d)$ lookup with $2^d \cdot d$ space required. Using the JL lemma, we can compress the dimensions to get that we need $O\left(\frac{\log n}{\epsilon^2}\right)$ dimensions to get an ϵ-approximation. Now, we need approximately $n^{O(1/\epsilon^2)}$ space and $d\frac{1}{\epsilon^2}\log n$ lookup time. This guarantee doesn't work if our queries are not independent since we can adverserially learn A and construct a q that does not work against it.

2 Locality Sensitive Hashing

Instead, if we use *locality sensitive hashing*, then we can solve this in $O(n^\rho d)$ time and $O(n^{1+\rho})$ space, where $\rho = 1/C$ and $C = 1+\epsilon$. If we want to use the L2 norm, then we can use $\rho = 1/C^2$.

In this lecture, we will be solving approximate *near* neighbor. Given $r \in \mathbb{R}$ and query q, if $\min_p \|p - q\| \leq r$, we want to find some p such that $\|p - q\| \leq Cr$. If we have the nearest neighbor, then we can solve r-near neighbor by just returning the output from that. For the other direction, we can pick some initial distance r and then test $(1+\epsilon)^k r$ for different non-negative values of k. We only need to try $\log_{1+\epsilon}(r)$ different costs until we find one that works.

Intuitively, we just want to hash our values, which allows us to check equality. If any elements are within r of each other, they should hash to the same value, but they should hash to different values if they are far away.

In the plot for the probability, the probability of a collision is high at a distance of r, but the probability should be low after Cr.

Definition 1. *h is a (p_1, p_2) LSH if and only if $\forall \|x - y\| < r$, then $\mathbb{P}[h(x) = h(y)] \geq p_1$. Additionally, $\forall \|x - y\| > Cr$, $\mathbb{P}[h(x) = h(y)] \leq p_2$.*

Definition 2. *A (p_1, p_2) LSH has* efficiency

$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \log_{p_2} p_1.$$

If some hash family H is ρ-efficient, then define H^2 to be $(h(x), h(x'))$ for $h, h' \in H$. For $g \sim H^2$, then

$$\mathbb{P}_{g\sim H^2}[g(x)=g(y)] = \mathbb{P}_{h\sim H}[h(x)=h(y)]^2.$$

If we evaluate the probabilities, we get H^2 is (p_1^2, p_2^2) LSH while still being ρ-efficient. Thus, we can take any algorithm with lower probabilties and amplify the individual probabilities by repeating the algorithm.

3 Using a Locality Sensitive Hash Function

Suppose we have a ρ-efficient hash function H such that $p_2 = \frac{1}{n}$ and $p_1 = \frac{1}{n^\rho}$. The obvious approach is to just build a hash table using this hash function. For the qth query, the probability of a false positive collision is at most np_2 in expectation, and at least a p_1 chance of a true positive. In $O(1)$ time, there is a $\frac{1}{n^\rho}$ chance of success. The output of the hash function will be very large, but we can store the values of the non-zero cells in a regular hash table, so we only need linear space. To increase our probability of success, we just repeat this $O(n^\rho \log(1/\delta))$ times. This gives us $O(n^\rho)$ time, $O(n^{1+\rho})$ space, and $1-\delta$ success probability.

4 Constructing a Locality Sensitive Hash Family

Define hash family H to be the set of hash functions $\{h_i(x) = x_i \mid i \in [d]\}$, i.e. we always look at a random single coordinate. The number of positions where x and y are the same each increase the probability of a match, so the probability of a hash collision is $1 - \frac{\|x-y\|_1}{d}$.

If we have $p_1 = 1 - \frac{r}{d}$ and $p_2 = 1 - \frac{Cr}{d}$, then

$$\begin{aligned}
\rho &= \frac{\log(1/p_1)}{\log(1/p_2)} \\
&= \frac{\log(1-r/d)}{\log(1-Cr/d)} \\
&\approx \frac{-r/d}{-Cr/d} \\
&= \frac{1}{C}.
\end{aligned}$$

Now, we can amplify the probabilities while keeping the same efficiency. We set $G = H^k$ for $k \approx \log_{p_2} \frac{1}{n}$, With this, our probability of failure is amplified to $\frac{1}{n}$. The number of steps required is approximately $\frac{\log n}{Cr/d} = \frac{d \log n}{Cr}$ since $\frac{Cr}{d}$ is the probability of failure.

5 Other Locality Sensentive Hash Functions

Suppose now that $x \in [\Delta]^d$, where our distance is measured by L1 norm. We can represent a coordinate x as a sequence of x 1s and $\Delta - x$ 0s. In this representation, the difference is the Hamming distance. Thus, we can transform the problem into a LSH on $\{0,1\}^{\Delta d}$. Our hash function performance does not have large dependence on the dimension, so this is fine.

We can also let the ith coordinate of the hash function be $\lfloor \frac{x_i - s_i}{w} \rfloor$, where s_i is some random shift. This transforms the problem from $[\Delta]^d$ to $\left[\frac{\Delta}{w}\right]^d$. We can show that we get $\rho = \frac{1}{C} + O(r/w)$, so picking a large w gives us an efficiency close to $\frac{1}{C}$. We can then pick a similar k as above to get a $\frac{1}{n}$ probability of failure.

For an L2 norm, we could let $h(x) = \text{sign}(\langle v, x \rangle)$ for some random vector v. This gives $\rho = \frac{1}{C}$. For another algorithm, we can pick $u_1, \ldots, u_T \in S^{d-1}$ uniformly. Then, $h(x) = \text{argmin}_i \|x - u_i\|$. In other words, we hash each point to the closest point on the sphere. This has $\rho = \frac{1}{C^2} + o_T(1)$.