CSE 521: Design and Analysis of Algorithms I **Spring 2016**

Lecture 7: Nearest Neighbor Search, Locally Sensitive Hashing

Lecturer: Shayan Oveis Gharan *April 18th* *Scribe: Yvonne Chen*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

In this lecture we discuss the nearest neighbor search problem. The material is based on the work of Indyk and Motwani [IM98] for which they received a Gödel prize in 2012.

7.1 Introduction

Given a set $P \in \mathbb{R}^d$ of n points, in the nearest neighbor search problem (NNS) we are interested in constructing a data structure such that for any given point $q \in \mathbb{R}^d$ we can return the point in P that is closest to q with respect to the distance function $\text{dist}(p, q)$ as fast as possible.

If $d = 1$, we can simply sort the points and do a binary search; so we can answer the query in time $O(\log n)$. We can extend this idea to d dimensions; this leads to famous data structures known as k-d trees and quad trees. Unfortunately, the memory that we need for any of these data structures grows exponentially in d, i.e., n^d. So, even if d is about 30 it is impossible to run these algorithms on today's computers.

7.1.1 Approximate NNS Problem

Indyk and Motwani decided to study the approximate version of the problem. Roughly speaking the goal is to design an algorithm that can approximately solve the NNS using a memory of size $\text{poly}(n)$ and query time which is sublinear in n. The approximate NNS is defined as follows:

Definition 7.1 (c, r approximate NNS)**.** *Given a set of n points $P \in \mathbb{R}^d$, and parameters $r, \delta > 0$, construct a data structure that for a query point $q \in \mathbb{R}^d$, if there is a point p such that $\text{dist}(p, q) \leq r$, return a point $\tilde{p}$ such that $dist(\tilde{p}, q) \leq c \cdot r$ with probability at least $1 - \delta$.*

The main idea of Indyk and Motwani is to reduce the NNS problem to the construction of Locally sensitive hash functions. Roughly speaking they show that if there is a locally sensitive hash function for the distance function dist(.,.) then one can use that to design an algorithm for the NNS problem.

Definition 7.2 ($(r, c \cdot r, p_1, p_2)$-locally sensitive hash function)**.** *Let $\mathcal{H} = \{h : P \to Z\}$ be a set of hash functions that map the points P to some discrete set, e.g., the set of all integers. We say $\mathcal{H}$ is $(r, c \cdot r, p_1, p_2)$-LSH if for any $p, q \in \mathbb{R}^d$ it satisfies the following two properties,*

i) if $\text{dist}(p, q) \leq r$, then $\mathbb{P}_{h \sim \mathcal{H}}[h(p) = h(q)] \geq p_1$, and

ii) if $\text{dist}(p, q) > c \cdot r$, then $\mathbb{P}_{h \sim \mathcal{H}}[h(p) = h(q)] \leq p_2$.

Note that ideally we like $p_1 \gg p_2$. In other words, we would like that the hash functions map close points to the same value with high probability and map *distant* points to different values with high probability. As we will see, the larger value of p_1/p_2 leads to more efficient algorithms for NNS.

Let us give a motivating example to better understand the above definition. Let $P \in \{0, 1\}^d$, be a subset of the points of a d-dimensional hypercube, and let $\text{dist}(p, q) = \|p - q\|_1$ be the number of bits that p and q differ in their bitwise representation.

Let $\mathcal{H} := \{h_i(p) = p_i \forall 1 \leq i \leq d\}$, i.e., h_i is the function that returns the i-th bit of p. For any $p, q \in \{0,1\}^d$ if $\text{dist}(p,q) \leq r$, then

$$\mathbb{P}[h(p) = h(q)] \geq 1 - \frac{r}{d} \approx e^{-r/d} = p_1,$$

where the approximation follows assuming $r \ll d$. On the other hand, if $\text{dist}(p,q) > c \cdot r$ then

$$\mathbb{P}[h(p) = h(q)] \leq 1 - \frac{r \cdot c}{d} \approx e^{-c \cdot r/d} = p_2.$$

As we mentioned above, we would like $p_1 \gg p_2$, so the plan is to use many samples of $\mathcal{H}$ to amplify p_1/p_2. We are going to design a 2-level hash function where the first level tries to reduce the probability that distance points hash to the same value and the second level tries to increase the probability that close points hash to the same value.

Let's start with the first level. For a point $p \in \mathbb{R}^d$ let $f(p) = [h_1(p), h_2(p) ... h_k(p)]$ where $h_1, \ldots, h_k$ are independent samples of $\mathcal{H}$ and k is an integer that we fix shortly. Now, if $\text{dist}(p,q) > c \cdot r$, by the independence of $h_1, \ldots, h_k$ we can write

$$\mathbb{P}[f(p) = f(q)] \leq p_2^k.$$

We choose k such that $p_2^k = \frac{1}{n}$. Assuming that we can write

$$\sum_{p \in P} \mathbb{P}[f(p) = f(q)] = 1,$$

in other words, in expectation only one distant point is mapped to $f(q)$. For that matter it is enough to let $k = \log(n) \cdot \log \frac{1}{p_2}$.

Now, observe that for such a large k, the probability that a close point p is mapped to $f(q)$ is nothing but p_1^k. Let ρ be chosen such that

$$p_2^\rho = p_1. \tag{7.1}$$

Then, if $\text{dist}\, p, q \leq r$ we have

$$\mathbb{P}[f_i(p) = f_i(q)] = p_1^k = p_2^{\rho \cdot k} = \frac{1}{n^\rho},$$

where we used that $p_2^k = 1/n$.

We will see later that ρ is parameter that depends on the c the approximation factor of algorithm for the NNS problem. Now, we want to boost up the above probability to an absolute constant, to make sure that we can indeed find a point that is close to q. The idea is to use multiple independent chosen copies of $f(p)$. We are going to construct $L = n^\rho$ hash tables, $T_1, T_2 .. T_L$ and for any point $p \in P$ we store $f_i(p)$ in T_i for all $1 \leq i \leq L$, For each i,

$$f_i(p) = [h_{i,1}(p), h_{i,2}(p), \ldots, h_{i,k}(p)],$$

where $h_{i,1}(.), \ldots, h_{i,k}$ are independently sampled from $\mathcal{H}$. To store the has values $\{f_i(p)\}_{p \in P}$ we can simply sort them. In that case we need to use $O(kn)$ amount of memory and search takes $O(k \log n)$. A faster algorithm is to hash these values; because we are hashing only n different values we only need to use $O(n)$ bits and searching takes $O(1)$ steps.

Given a query point q, we go over all $i \in L$ hash tables and every point p where $f_i(p) = f_i(q)$ and we return p if $\text{dist}(p,q) \leq c \cdot r$. For any point p where $\text{dist}(p,q) \leq r$ we have

$$\begin{aligned}
\mathbb{P}[\exists i : f_i(p) = f_i(q)] \geq 1 - \mathbb{P}[\forall i : f_i(p) \neq f_i(q)] &= 1 - (1 - p_1^k)^L \\
&= 1 - \left(1 - \frac{1}{n^\rho}\right)^L = 1 - (1 - \frac{1}{L})^L = 1 - \frac{1}{e}
\end{aligned}$$

So with a constant probability, we find p. On the other hand, as alluded to above, for each $1 \leq i \leq L$ we only see one distant point that is mapped to $f(q)$. So, overall we see $O(L)$ distant points in expectation.

Now, let us complete the analysis by analyzing the memory usage and the running time of our NNS algorithm.

Memory usage. We need to store L different values for each point $p \in P$ This needs $O(Ln)$ bits of memory if we hash the values. Recall that $L = n^\rho$. By equation 7.1 we can write $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.

Query time. We just need to bound the time to go over the distant points. Since there are only $O(L)$ distant points in expectation, computing the distance between q and any such point takes $O(L \cdot d)$ operations. Furthermore, computing $f_1(q), f_2(q), \ldots, f_L(q)$ takes $O(Lk)$ times. Overall, the run time is $O(L(k+d))$

Let us measure the memory usage and query time for the special case of the points on a hypercube. In that case we have

$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \frac{r/d}{c \cdot r/d} = \frac{1}{c}.$$

and $k = \log(n) \cdot \log(1/p_2) = \log(n) \cdot \frac{d}{c \cdot r}$. Therefore, for $c = 2$ we $L = \sqrt{n}$. So, we need $O(n^{3/2})$ bits of memory and the query time is $O(\sqrt{n}(d + (d \log n)/cr))$.

We conclude this lecture by describing two additional examples of locally sensitive hash functions.

Jaccard Distance. Let $P \subseteq 2^{[n]}$, i.e., each point of P is a subset of $[n]$ where as usual $[n] = \{0, 1, \ldots, n-1\}$. We can think of $[n]$ as the set of all words and each point of P is a document. So we are modeling each document as a bag of words and ignoring the ordering of the words in the document.

For $A, B \in P$, the *Jaccard Distance* of A, B is defined as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Observe that if $J(A, B) \approx 1$, then A, B have almost identical words and conversely if $J(A, B) \approx 0$ then most of the words in the two documents are the same. We can defined a distance function letting

$$\text{dist}(A, B) = 1 - J(A, B).$$

Next, we describe a locally sensitive hash function for the above distance function.

Let $\sigma : [n] \to [n]$ be a permutation. The idea is to use the minhash function that we studied in PS 2. We let $h_\sigma(A) = \min_{i \in A} \sigma(i)$ be the smallest number that the elements of A are mapped to. We also let

$$\mathcal{H} = \{h_\sigma : \sigma : [n] \to [n]\}.$$

That is, we choose a uniformly random permutation and we map each set A to the smallest number that its elements are mapped to. Observe that

$$\mathbb{P}_{h \sim \mathcal{H}}\left[h(A) = h(B)\right] = \frac{|A \cap B|}{|A \cup B|} = J(A, B). \tag{7.2}$$

This is because for any A, B, we have $h_\sigma(A) = h_\sigma(B)$ only if

$$\min_{i \in A} \sigma(i) = \min_{i \in A \cap B} \sigma(i) = \min_{i \in B} \sigma(i).$$

The above event happens with probability exactly $J(A, B)$. Now, we can show that $\mathcal{H}$ is a $(r, c \cdot r, e^{-r}, e^{-cr})$-LSH.

i) If $\text{dist}(A,B) \leq r$, then $J(A,B) \geq 1-r$. S, by (7.2),

$$\mathbb{P}\left[h(A) = h(B)\right] \geq 1 - r \approx e^{-r} = p_1.$$

ii) If $\text{dist}(A,B) \geq c \cdot r$, then $J(A < B) \leq 1 - c \cdot r$. So, by (7.1),

$$\mathbb{P}\left[h(A) = h(B)\right] \leq 1 - c \cdot r \approx e^{-c \cdot r} = p_2.$$

ℓ_2 Distance. Let $P \subseteq \mathbb{R}^d$ and let

$$\text{dist}(p,q) = \|p - q\|_2.$$

The following hash function can be used as an LSH. Let g be a d-dimensional Gaussian vector, let w be a large finite number and b chosen uniformly at random in $[0, w)$. For a point p, we let

$$h(p) = \left\lfloor \frac{\langle g, p \rangle + b}{w} \right\rfloor.$$

It is a nice exercise to show that for the above LSH we have $\rho = O(1/c)$.

References

[IM98] P. Indyk and R. Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *STOC*. ACM. 1998, pp. 604–613 (cit. on p. 7-1).