CSIS8601: Probabilistic Method & Randomized Algorithms
Lecture 6: Locality Sensitive Hashing: Nearest Neighbor Search
Lecturer: Hubert Chan
Date: 7 Oct 2009

These lecture notes are supplementary materials for the lectures. They are by no means substitutes for attending lectures or replacement for your own notes!

1 Nearest Neighbor Search in Hamming Space

Definition 1.1 *Let $H = \{0, 1\}$. The d-dimensional Hamming Space H^d consists of bit strings of length d. Each point $x \in H^d$ is a string $x = (x_0, x_1, \ldots, x_{d-1})$ of zero's and one's. Given two points $x, y \in H^d$, the Hamming distance $d_H(x, y)$ between them is the number of positions at which the corresponding strings differ, i.e.,*

$d_H(x, y) := |\{i : x_i \neq y_i\}|$.

Example.

The DNA double helix consists of two strands. The four bases found in DNA are A, C, G and T. Each type of base on one strand forms a bond with just one type of base on the other strand, with A bonding only to T, and C bonding only to G. If we use 0 to represent an A-T bond and 1 to represent a C-G bond, each DNA can be represented as a point in Hamming Space.

Remark 1.2 Given two points $x, y \in H^d$, their Hamming distance $d_H(x, y)$ can be computed in $O(d)$ time.

Definition 1.3 (Nearest Neighbor Search) *Given a set P of n points in Hamming Space H^d, and a query point $q \in H^d$, a nearest neighbor for q in P is a point $p \in P$ such that the Hamming distance $d_H(p, q)$ is minimized.*

Remark 1.4 (Naive Method) Given a query point q, the Hamming distance $d_H(p, q)$ for each $p \in P$ can be computed in time $O(d)$. Hence, it takes $O(nd)$ time to find a nearest neighbor. Observe that running time is linear in the number n of points in P.

Definition 1.5 (Approximate Nearest Neighbor Search) *Given a set P of n points in Hamming Space H^d, approximation ratio $c > 1$ and a query point $q \in H^d$, a c-nearest neighbor for q in P is a point $p \in P$ such that the Hamming distance $d_H(p, q) \leq c d_H(p^*, q)$, where p^* is a nearest neighbor of q.*

Given a set P of n points in H^d, the goal is to design a data structure to store P such that given a query point q, an approximate nearest neighbor can be returned in **sublinear** time $o(n)$. We next look at a sub-problem that can help us achieve this goal.

Definition 1.6 (Approximate Range Search) *Suppose P is a set of n points in Hamming Space H^d. A range parameter $r > 0$ and an approximation ratio $c > 1$ are given. Given a query point $q \in H^d$, the search with specified range r and approximation ratio c does the following: if there is some point $p^* \in P$ such that $d_H(p^*, q) \leq r$, return a point $p \in P$ that satisfies $d_H(p, q) \leq cr$.*

Remark 1.7 If there is no point $p^* \in P$ such that $d_H(p^*, q) \leq r$, approximate range search could still return a point $p \in P$ that satisfies $r < d_H(p, q) \leq cr$.

2 Locality Sensitive Hashing

The algorithm that we describe is due to Indyk and Motwani. Suppose $\mathcal{F}$ is a family of hash function of the form $h : H^d \to \{0, 1\}$. A hash function h is picked uniformly at random from $\mathcal{F}$. The idea is that if two points x and y in H^d are close with respect to their Hamming distance, then the probability that $h(x) = h(y)$ should be higher.

Definition 2.1 *For $r_1 < r_2$ and $p_1 > p_2$, a family $\mathcal{F}$ of hash functions is (r_1, r_2, p_1, p_2)-sensitive for H^d if for any $x, y \in H^d$,*

1. *if $d_H(x, y) \leq r_1$, then $Pr_{\mathcal{F}}[h(x) = h(y)] \geq p_1$;*
2. *if $d_H(x, y) > r_2$, then $Pr_{\mathcal{F}}[h(x) = h(y)] \leq p_2$.*

We next define a family $\mathcal{F}_d$ of d hash functions. Each is of the form $h^{(i)} : H^d \to \{0, 1\}$, where $h^{(i)}(x) := x_i$. In other words, $h^{(i)}(x)$ returns the ith position of a point x.

Lemma 2.2 *For $x, y \in H^d$, $Pr_{\mathcal{F}_d}[h(x) = h(y)] = 1 - \frac{d_H(x,y)}{d}$.*

Proof: Let $\delta := d_H(x, y)$. This means x and y differ at exactly δ positions. Hence, when a position i is chosen uniformly at random, the probability that $x_i = y_i$ is $1 - \frac{\delta}{d}$. It follows that $Pr_{\mathcal{F}_d}[h(x) = h(y)] = 1 - \frac{d_H(x,y)}{d}$. ■

Corollary 2.3 *For $r > 0$, $c > 1$ and $rc \leq d$, the family $\mathcal{F}_d$ is $(r, rc, 1 - \frac{r}{d}, 1 - \frac{rc}{d})$-sensitive for the Hamming Space H^d.*

We take $r_1 := r$, $r_2 := cr$, $p_1 := 1 - \frac{r}{d}$ and $p_2 := 1 - \frac{rc}{d}$. Since $1 > p_1 > p_2$, we have $\rho := \frac{\log p_1}{\log p_2} < 1$.

2.1 Hashing by Dimension Reduction

We use the family $\mathcal{F}_d$ of hash functions to build another family $\mathcal{G}_K$ of hash functions, where K is a width parameter to be determined later. A hash function from $\mathcal{G}_K$ is of the form $g : H^d \to \{0, 1\}^K$. A hash function g can be picked uniformly at random from $\mathcal{G}_K$ in the following way.

1. For each $k \in [K]$, pick h_k uniformly at random from $\mathcal{F}_d$ independently.
2. Let $g := (h_k)_{k \in [K]}$ be the concatenation of the h_k's.

   For $x \in H^d$, $g(x) := (h_0(x), h_1(x), \ldots, h_{K-1}(x))$.

Hash Table T with Key g. For each point $x \in H^d$, a pointer to the point x can be stored in a hash table T with the key $g(x) \in \{0, 1\}^K$. The space for the hash table is proportional to the number of pointers stored. Hence, if there are n points, the space for the hash table T is $O(n)$. Each computation of $g(x)$ takes $O(K)$ time. Hence, the time for constructing a hash table is $O(nK)$.

2.2 Data Structure for Approximate Range Search

Recall we are given a set P of n points in H^d, a range $r > 0$ and an approximate ratio $c > 1$. Let $p_1 := 1 - \frac{r}{d}$ and $p_2 := 1 - \frac{rc}{d}$. Define $\rho := \frac{\log p_1}{\log p_2} < 1$.

Let $K := \log_{\frac{1}{p_2}} n$ be the width of hash family $\mathcal{G}_K$ and $L := n^\rho$ be the number of hash tables in our data structure. For ease of presentation, we assume that K and L are integers and do not worry about rounding off fractions.

Construction of Data Structure. For each $l \in [L]$, a hash table T_l storing pointers to points in P is constructed as follows.

1. Pick a function g_l from $\mathcal{G}_K$ uniformly at random.

   This requires $O(K \log d)$ random bits.

2. For each point $x \in P$, store the pointer to the point x in the hash table T_l using key $g_l(x)$.

The space used for the L hash tables is $O(Ln)$ words and the space to store the original n points is $O(nd)$. The construction time is $O(LnK)$.

The total space is $O(n(n^\rho + d))$ and the time is $\tilde{O}(n^{1+\rho})$. The notation $\tilde{O}$ hides logarithmic factors $\log n$.

Query Algorithm. Given a query point $q \in H^d$, a point $p \in P$ is returned by the following algorithm.

1. For each $l \in [L]$, compute the key $g_l(q)$ and use the key to retrieve points stored in the hash table T_l.
2. For each retrieved point p, if $d_H(p, q) \leq cr$, return the point p, and terminate.
3. At most $2L$ retrieved points will be looked at. If after $2L$ points, the algorithm still has not returned a point, the algorithm terminates and does not return a point.

Running time for Query. Since each computation $g_l(q)$ takes $O(K)$ time, and each computation $d_H(p, q)$ takes $O(d)$ time, the total time of the algorithm is $O(L(K + d)) = \tilde{O}(dn^\rho)$. Observe $\rho < 1$.

2.3 Correctness

We show that for each query point q, the algorithm performs correctly with probability at least $\frac{1}{2} - \frac{1}{e}$. Using standard repetition technique, the failure probability can be reduced to δ, if we repeat the whole data structure $\log \frac{1}{\delta}$ times.

The algorithm behaves correctly means that if there is a point $p^* \in P$ such that $d_H(p^*, q) \leq r$, a point p is returned such that $d_H(p, q) \leq cr$.

Suppose $p^* \in P$ is a point such that $d_H(p^*, q) \leq r$. The algorithm behaves correctly if both the following events happen.

1. E_1: there exists some $l \in [L]$ such that $g_l(p^*) = g_l(q)$.

2. E_1: there exist less than $2L$ points p such that $d_H(p,q) > cr$ and for some $l \in [L]$, $g_l(p) = g_l(q)$.

The event E_1 ensures that the point p^* would be a candidate point for consideration. The event E_2 ensures that there would not be too many bad collision points so that the algorithm does not terminate before the point p^* (or other legitimate points) can be found.

We show that $Pr[\overline{E_1}] \leq \frac{1}{e}$ and $Pr[\overline{E_2}] \leq \frac{1}{2}$. By union bound, we can conclude $Pr[E_1 \cap E_2] \geq \frac{1}{2} - \frac{1}{e}$.

Since $d_H(p,q) \leq r$, for some fixed l, the probability that $g_l(p) = g_l(q)$ is at least $p_1^K = p_1^{\log_{\frac{1}{p_2}} n} = n^{-\frac{\log p_1}{\log p_2}} = n^{-\rho} = \frac{1}{L}$.

Hence, $Pr[\overline{E_1}] \leq (1 - \frac{1}{L})^L \leq \frac{1}{e}$.

Fix $l \in [L]$ and a point p such that $d_H(p,q) > rc$. Then, the probability that $g_l(p) = g_l(q)$ is at most $p_2^k = \frac{1}{n}$. Hence, it follows that the expected number of points p such that $d_H(p,q) > rc$ and $g_l(p) = g_l(q)$ is at most 1.

Hence, the expected number of points p such that $d_H(p,q) > rc$ and $g_l(p) = g_l(q)$ for some l is at most L. Using Markov's Inequality, $Pr[\overline{E_2}] \leq \frac{1}{2}$, as required.

3 From Approximate Range Query to Approximate Nearest Neighbor

The data structure described above can be repeated for different values of r. In particular, define $I := \lceil \log_{1+\epsilon} \Delta \rceil$, where $\Delta := d = \max_{x,y} d_H(x,y)$. For each $i \in [I]$, let $r_i := (1+\epsilon)^i$, and we build a data structure D_i with range r_i and approximation ratio $c = 1+\epsilon$.

For the query algorithm, we can use binary search to find the smallest i for which the corresponding data structure D_i returns a point.