COMS E6998-9: Algorithmic Techniques for Massive Data — Sep 8, 2015

Lecture 11 – Nearest Neighbor Search

Instructor: *Alex Andoni* — Scribes: *Marshall Ball*

1 Recall: Nearest Neighbor Search

As in the previous lecture, we are concerned here with the problem of Nearest Neighbor Search.

Recall 1. A *(c-approximate, r-near) near neighbor (of q in D)* is the following:

- Fix a set, D, of points in some space $\mathcal{X}$ with distance d. (We allow preprocessing of D.)
- For any query q, if $\exists p^* \in D$ such that $d(q, p^*) < r$, we want to return some $p \in D$ such that $d(q, p) < cr$.

Our aim in nearest neighbor search is to minimize the space complexity of our data structure, as well as the query-time complexity.

In the exact version, $c = 1$.

For the remainder of this document, we take $\mathcal{X} := \mathbb{R}^d$ and $n := |D|$.

Last lecture we defined a sketching method, W, that is useful for NNS:

- $W : \mathbb{R}^d \to \{0, 1\}^k$.
- Given $W(x), W(y)$ we can distinguish between:
  - $\|x - y\| < r$ (x, y are "close"),
  - $\|x - y\| > cr = (1 + \varepsilon)r$ (x, y are "far"),

  with high probability. In particular, the probability that our test did not succeed was $\leq \delta = 1/n^3$.
- Moreover, to acheive such an error bound (for ℓ_1-norm), we only required $k = O(1/\varepsilon^2 \log(n))$ bits. Although we did not see it in class, we can achieve the same sketch length for the ℓ_2-norm.

Given our sketch, we looked at the following two methodologies for solving NNS:

1. *Linear Scan*
   - Precompute $W(p)$ for all $p \in D$.
   - Given query q, compute $W(q)$.
   - Compare $W(q)$ to $W(p)$ for all $p \in D$.

   Note that while this gives us a near-linear space complexity, $O(1/\varepsilon^2 \log(n)n)$, it has poor query-time complexity, $O(nk)$.

2. *Exhaustive Storage*

   - For $\sigma \in \{0,1\}^k$, $A[\sigma] = p \in D$ such that $d(W(p), \sigma) < cr = (1+\varepsilon)r$.
   - On query q, output $A[W(q)]$.

3. This gives us $O(d/\varepsilon^2 \log(n))$ query time (the time to compute $W(q)$) and $O(2^{O(1/\varepsilon^2 \log(n))} \log(n)) = O(n^{O(1/\varepsilon^2)} \log(n))$ space.

In this lecture, our goal is to attempt to get the best of both worlds above: near-linear space complexity with sub-linear query time.

2 Locality Sensitive Hashing [?]

With the above aim in mind, consider the following primitive:

Definition 1 (informal)**.** A *locality senstive hash function, LSH* is a random hash function $h : \mathbb{R}^d \to U$ (h drawn from a family $\mathcal{H}$, U some finite set) such that

1. $d(q,p) \le r \implies \Pr[h(q) = h(p)] = P_1$ is "not-so-small," (p close to q implies they collide, under h, with higher probability)

2. $d(q,p) > cr \implies \Pr[h(q) = h(p)] = P_2$ is "small." (q far from p implies they collide, under h, with lower probability)

We will specify later what "small" and "not-so-small" actually mean. In general, $P_1 < P_2$ and we associate the following parameter with $\mathcal{H}$ to characterize this gap:

$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)}.$$

If we had an LSH such that P_1 was "large," then we could simply compute the hash table of D, A. Then on query q, simply compute $A[h(q)]$

Remark 1. Unfortunately, it is not possible to have P_1 high and P_2 low.

Roughly, suppose we have p_1, p_2 such that $d(p_1, p_2) = cr + \varepsilon'$. Now consider a series of points $q_1, \dots, q_m$ on the line through p_1, p_2 such that any neighbor points in $\{p_1, p_2, q_1, \dots, q_m\}$ are less than distance r apart.

Consider $m \approx c - 1$ to be not too large (say $c = 2$). Then with probability P_1^{m+1} we have $h(p_1) = h(q_1) = \cdots = h(p_m) = h(p_1)$. But, on the other hand with probability $h(p_1) = h(p_2)$ with probability P_2. So, $P_1^c \lesssim P_2$.

So instead of a single hash table, we will use $L = n^\rho$ hash tables for independent $h_1, \dots, h_L \in \mathcal{H}$. (We will justify this choice of L later.) Note that $\rho = \frac{\log 1/P_1}{\log 1/P_2} < 1$, so $n^\rho < n$.

3 NNS/LSH in Hamming Space

3.1 LSH for Hamming Space

We construct a LSH for Hamming Space, $\{0,1\}^d$ with distance metric $\mathrm{Ham}(x,y) = |\{x_i \ne y_i\}|$.

Our hash family, $\{g : \{0,1\}^d \to \{0,1\}^k\}$, is defined as follows:

$$g(p) := (h_1(p), h_2(p), \ldots, h_k(p)),$$

where

$$h_i(p) := p_j \text{ for random } j \leftarrow [d].$$

Note 1.

$$\Pr[g(p) = g(q)] = \prod_{i=1}^{k} \Pr[h_i(p) = h_i(q)].$$

Fact 1. $\rho_g = \rho_h$

Proof.

$$\Pr[g(p) = g(q)] = \prod_{i=1}^{k} \Pr[h_i(p) = h_i(q)] \implies \begin{cases} P_{1,g} = P_{1,h}^k \\ P_{2,g} = P_{2,h} \end{cases}$$

$$\rho_g = \frac{\log 1/P_{1,g}}{\log 1/P_{2,g}} = \frac{\log 1/P_{1,h}^k}{\log 1/P_{2,h}^k} = \frac{k \log 1/P_{1,h}}{k \log 1/P_{2,h}} = \rho_h$$

□

Claim 2. $\rho \approx \frac{1}{c}$

Proof. Notice that

$$\forall i,\ \Pr[h_i(p) = h_i(q)] = 1 - \frac{\mathrm{Ham}(p,q)}{d}.$$

For simplicity we assume $r \ll d$. This assumption is justified because (1) we can always embed in a higher dimension, and (2) the analysis goes through without the following approximation.

From the taylor series of $e^x = 1 + x + \frac{x^2}{2!} + \cdots$, we get the following approximation (within additive factor $O((cr/d)^2)$):

$$P_{1,h} = 1 - \frac{r}{d} \approx e^{-r/d}$$
$$P_{2,h} = 1 - \frac{cr}{d} \approx e^{-cr/d}$$

This implies

$$\rho_g = \frac{\log 1/P_{1,h}}{\log 1/P_{2,h}} \approx \frac{r/d}{cr/d} = \frac{1}{c}.$$

□

3.2 Using LSH for NNS

We now present an algorithm for NNS in Hamming Space via the above LSH. We will use the technique outlined earlier.

3.2.1 Algorithm for NNS in Hamming Space

- Data Structure:

- Allocate $L = n^\rho$ hash tables, $A_1, \ldots, A_L$ each with a fresh Hamming-LSH g_i. (choice of k for $g_i = (h_1, \ldots, h_k)$, and implicitly L, below.)
- Hash all of D into tables.
- We will want each hash table to have size n. So, we will think of hash table size as simply the number of the non-empty buckets.

• Query:

On q,

- Compute $g_1(q), \ldots, g_L(q)$.
- Each table, $A_1[g_1(q)], \ldots, A_L[g_L(q)]$, for collisions.
- For each collision $p \in D$ under g_i, check if $d(p,q) < cr$. If so, output p. If none found, FAIL.

(Assuming as usual that $\exists p \in D : d(p,q) < r$. Our promise problem is only concerned with this case.)

For each table/hash function we have success probability $P_{1,h}^k$: probability of a "good" or (close) collision. We have L tables total. So, taking a union bound we want to choose $L = O(1/P_{1,h}^k)$.

Suppose it takes time T_g to compute $g_i(q)$. Notice that we expect $nP_{2,g} = nP_{2,h}^k$ "bad" (or far) collisions So in expectation, our runtime will be

$$O\left(\frac{1}{P_{1,h}^k}(T_g + nP_{2,h}^k)\right).$$

T_g we think of as a constant (ignoring $\log(n)$ factors). So, we want $nP_{2,h}^k = O(1)$ as well. Thus, we take $P_{2,h}^k = 1/n$ so that the expected number of far points encountered is 1. This implies:

$$P_{2,h}^k = 1/n \implies k\log(1/P_{2,h}) = \log n \implies k = \frac{\log n}{\log(1/P_{2,h})}.$$

For this choice we also get,

$$P_{1,h}^k = P_{1,h}^{\frac{\log(n)}{\log 1/P_{2,h}}} = n^{\frac{-\log 1/P_{1,h}}{\log 1/P_{2,h}}} = n^{-\rho}.$$

So for g we have:

$$P_{1,g} = \Pr[g(p) = g(q) | d(p,q) < r] = P_{1,h}^k = (P_{2,h}^\rho)^k = \frac{1}{n^\rho}$$
$$P_{2,g} = \Pr[g(p) = g(q) | d(p,q) < cr] = P_{2,h}^k = 1/n.$$

3.2.2 Analysis

Claim 3. *The above algorithm gives us the following guarrantees:*

1. *Space:* $O(nL) = O(nn^{1+\rho})$ *(or actually* $O(nL\log(n))$ *to store pointers).*
2. *Query time:* $O(L(k+d)) = O(n^\rho d)$ *in expectation.*
3. $> 50\%$ *success probability.*

	Space	Time	Exponent	$c = 2$	Ref
Hamming	$n^{1+\rho}$	n^ρ	$\rho = 1/c$	$\rho = 1/2$	[?]
Space			$\rho \geq 1/c$		[?, ?]
Euclidean	$n^{1+\rho}$	n^ρ	$\rho = 1/c$	$\rho = 1/2$	[?, ?]
Space			$\rho = 1/c^2$	$\rho = 1/4$	[?]
			$\rho \geq 1/c^2$		[?, ?]

Proof. (1) and (2) are clear from above.

For (3) *Correctness*:

Let p^* be an r-near neighbor to some query q. (Recall that we have no requirements if some p^* does not exist.) Then, the probability that the algorithm fails is bounded from above by

$$\begin{aligned}
\Pr[p^* \notin \{g_1(q), \ldots, g_L(q)\}] &= \prod_{i=1}^{L} \Pr[h_i(p^*) \neq h_i(q)] \\
&\leq (1 - \tfrac{1}{n^\rho})^L \\
&= (1 - \tfrac{1}{n^\rho})^{\frac{1}{n^\rho}} \\
&\leq 1/e < 1/2.
\end{aligned}$$

□

4 LSH Continued

In practice, we may be concerned with *exact* NNS ($c = 1$). Note that for the guarrantees on our algorithm to hold, all we require is that L, k are chosen such that

$$\Pr[\text{failure}] \leq (1 - P_{1,g})^L \leq 0.1 \text{ (small constant)}.$$

4.1 Table of LSH algorithms for NNS

Below we present a table of NNS algorithms using the framework defined above:

4.2 LSH for Other ℓ_1-type "distance" (Zoo(ℓ_1)

In general all of these LSH constructions have

$$g(p) := \langle h_1(p), \ldots, h_k(p) \rangle.$$

Below we specify a variety of "primitive" h for preserving locality under various notions of distance:

- Hamming Distance [?]

  h : project onto random coordinate (as seen above).

- ℓ_1 (Manhattan) Distance

  h : weight of cell in a randomly shifted grid.

- Jacard distance between sets.
  We define $J(A,B) := \frac{|A \cap B|}{|A \cup B|}$ where $A, B \subseteq U$ for some universe $U = [n]$.
  Min-wise Hashing [?]
  - Pick a random permutation $\pi : U \to U$.
  - $h(A) := \min_{a \in A} \pi(a)$. (Recall $U = [n]$. In general, simply impose some arbitrary ordering.)

  $$\Pr[h(A) = h(B)] = \Pr[\pi(A \cup B) \in A \cap B] = \frac{A \cap B}{A \cup B} = J(A,B).$$

  Note that Jacard Distance LSH can be used for Hamming Distance LSH (with a little work).

4.3 LSH for Euclidean Space [?]

For LSH for euclidean distance, consider the following primitive hash function: (Idea: project onto a randomly partitioned, random one dimensional subspace.)

- Pick a random gaussian vector ℓ.
- Pick random $b \in [0,1]$.
- w is a parameter that will quantize ℓ (size of partitions).

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

Claim 4. *For g constructed via the above primitive functions, $\rho = 1/c$*

Proof. Next time. □

References

[1] Andoni, Alexandr, and Piotr Indyk. "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions." *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on.* IEEE, 2006.

[2] Broder, Andrei Z. "On the resemblance and containment of documents." *Compression and Complexity of Sequences 1997.* Proceedings. IEEE, 1997.

[3] Datar, Mayur, et al. "Locality-sensitive hashing scheme based on p-stable distributions." *Proceedings of the twentieth annual symposium on Computational geometry.* ACM, 2004.

[4] Indyk, Piotr, and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality." *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* ACM, 1998.

[5] Motwani, Rajeev, Assaf Naor, and Rina Panigrahy. "Lower bounds on locality sensitive hashing." *SIAM Journal on Discrete Mathematics* 21.4 (2007): 930-935.

[6] Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality sensitive hashing (except when q is tiny). In *Proceedings of Innovations in Computer Science (ICS '2011)*, pages 275–283, 2011.