CS369G: Algorithmic Techniques for Big Data Spring 2015-2016

Lecture 16: Approximate Nearest Neighbor Search

Prof. Moses Charikar *Scribe: AmirMahdi Ahmadinejad*

1 Overview

In this lecture, we talk about the *nearest neighbor search* (NNS) problem.

NNS Given points $p_1, p_2, \ldots, p_n \in R^d$, and a query point $q \in R^d$, find $\arg\min_{p_i} d(p_i, q)$, that is, find the closest point to q among $p_1, \ldots, p_n$.

The storage and query time for a brute force algorithm is shown in Table 1. In the settings when the number of points is very large, it is not affordable to check every point for a single query. So assuming that we have the dataset in advance, the real goal is to preprocess and store some data so that when the query comes it can be answered quickly.

Things that matter in our analysis: amount of *storage* used, *preprocessing time*, *query time*.

	Brute Force (BF)	BF with Reduced Dimension
Storage	nd	$n\log(n)/\epsilon^2$
Preprocessing Time		$nd\log(n)/\epsilon^2$
Query Time	nd	$n\log(n)/\epsilon^2$

Table 1: Two basic approaches for answering the Nearest neighbor search queries.

2 Curse of Dimensionality

Trying to solve the nearest neighbor search problem exactly for high dimensional data is a challenge. There are some methods that answer the queries fast for low dimension data. However, their dependence on dimension is often exponential! These methods are not scalable in terms of dimension. And this is a negative result which is sometimes called the "curse of dimensionality".

It is also good to note that, by Johnson-Lindenstrauss lemma [1] one can always reduce the dimension to $\log(n)/\epsilon^2$, by compromising a $1+\epsilon$ approximation in distances. But we can observe that it is still not good when the the dependence to dimension is exponential, for example $n^{log(n)/\epsilon^2}$. The storage and running time for a brute force algorithm using JL lemma is also shown in Table 1.

3 Approximate Nearest Neighbor Search

In the approximate nearest neighbor search problem, we relax our goal of exactly finding the closest point p_i. Instead we look for a p_i, so that $d(q, p_i) \leq c \cdot \min_{p_j} d(q, p_j)$. In this section by using Locality

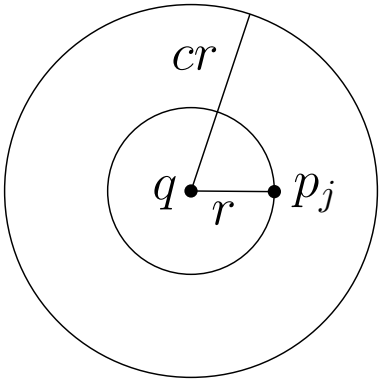

Figure 1: p_j is the nearest point to q. In the c-approx nearest neighbor problem any point within the radius cr is accepted.

Sensitive Hashing (LSH), we introduce schemes that can answer to approximate NNS in sub-linear query time.

3.1 "Intrinsic" dimensionality of data

Usually in real-world datasets even though the data is represented in a large number of dimensions, the actual complexity is low. For such datasets, we should be able to design algorithms that are tailored to their intrinsic (complexity of the dataset) rather than ambient (host space) dimension. In order to be able to talk about the true dimensionality of data, we define the concept of "Intrinsic" dimensionality which is closely related to *doubling dimension* (used in mathematical contexts).

Definition 1. *Given a set of points $\mathcal{X}$, and their pairwise distances , let $B(p,r) \subseteq \mathcal{X}$ be the set of all points within distance r from point p. Let d' be the smallest integer such that*

$$2^{d'} \geq \max_{r,p} \textit{minimum \# of balls of radius } r/2 \textit{ needed to cover } B(p,r).$$

d' is called the doubling dimension (intrinsic dimensionality) of data.

3.2 Locality Sensitive Hashing

Kushilevitz et al. [2] show that one can answer $1+\epsilon$ approximate nearest neighbor with polylogarithmic query time and $poly(n)$ preprocessing time and storage.

Indyk and Motwani [3] consider the following problem: Can we answer the approximate NNS with query time $o(n)$ with small amount of storage? To achieve this they use Locality Sensitive Hashing (LSH).

Definition 2. *A family of hash functions is (r, cr, p_1, p_2)-LSH with $p1 \geq p2$ and $c > 1$ if:*

(a) $Pr[h(x) = h(y)] \geq p_1$ when $d(x,y) \leq r$ (close points).

(b) $Pr[h(x) = h(y)] \leq p_2$ when $d(x,y) \geq cr$ (distant points).

An (r, cr, p_1, p_2)-LSH can be used to design an algorithm for approximate NNS. For now instead of NNS we focus on the following problem: If there is some point within distance r of q, we want to report a point within distance cr (For the NNS we can store multiple copies of this scheme for different values of r).

Assume we have a family of hash functions with (r, cr, p_1, p_2)-LSH property. We sample k hash functions independently from the family and produce a for each element a single hash value:

$$g(x) = h_1(x)h_2(x)\cdots h_k(x)$$

which is a concatenation of the k hash values. We say that we have a collision between $x \neq y$ if $g(x) = g(y)$. Since, g is a concatenation of k hash functions this can happen only if $h_i(x) = h_i(y)$ for all $i \in [k]$. Now let us compute the probability of collision:

- If $d(x, y) \leq r$ then the probability of collision would be at least p_1^k.
- If $d(x, y) \geq cr$ then the probability of collision would be at most p_2^k.

We choose k so that $p_2^k = 1/n$. Then the probability that a bad point p, i. e. $d(q, p) > c \cdot r$, maps to the same bucket as query q is small. To achieve this one should set $k = \log(n)/\log(1/p2)$.

Now let us compute the collision probability for a close point. We can write $p_1 = p_2^\rho$, for some $\rho < 1$, then:

$$p_2^k = 1/n \Rightarrow p_1^k = 1/n^\rho.$$

Which is quite small. To correct this we produce multiple hash tables, namely $L \approx n^\rho$ hash tables, so that with high probability we get some close point in at least one of the hash tables. To summarize, this method uses:

- n^ρ hash tables
- n^ρ query time
- $n^{1+\rho}$ storage

Here ρ is the best parameter that we can find over all sets of locality sensitive hashing schemes, and is therefore a function of the constant c and metric d, i.e. $\rho = f(c, d(.,.))$. And since usually ρ is not small, both storage and query time are not too small in this scheme.

In two papers by Panigrahy [4] and Lv et al. [5] the authors try to reduce the amount of storage. The underlying idea is the same in the sense that they try to look up more than one entry of a hash function for a query. In Panigrahy's paper [4], the algorithm adds random perturbations to the original query and looks up for all generated points in a single hash function. In Lv et al.'s [5] paper they introduce an indexing scheme called multi-probe which uses a probing sequence to look up multiple entries that have a high probability of containing the nearest neighbors of a query point. In these schemes the number of hash tables is reduced to $\tilde{O}(1)$, and the query time is $n^{\rho'}$. In a paper by Kapralov [6], it is shown that one can also compromise (smoothly) on storage in order to reduce the query time.

3.3 ρ for Hamming Metric

In this section we compute ρ for hamming metric in $\{0,1\}^d$. A simple hash function is to pick the value of a random coordinate. By this hash function we have:

- If $d(x,y) \leq r$: $pr[h(x) = h(y)] \geq 1 - r/d \approx e^{-r/d}$
- If $d(x,y) \geq cr$: $pr[h(x) = h(y)] \leq 1 - cr/d \approx e^{-cr/d}$

Therefore $p_1 = p_2^{1/c}$ and $\rho = 1/c$. For hamming metric this turns out to be the best possible ρ for c-approx NNS. Using a similar strategy one can show the same result for ℓ_1 norm. That is if $d(x,y) = ||x-y||_1$, then $\rho = 1/c$. We also know for $d(x,y) = ||x-y||_2$ (ℓ_2 norm), $\rho = 1/c^2$.

4 Some other hash functions

4.1 Min Hash Scheme [7]

In this subsection we introduce a hash scheme which has application in finding similar objects, like similar pages over the web. Given a collection of sets, the goal is to find a hash function such that for two sets A and B,

$$Pr[h(A) = h(B)] = \frac{|A \cap B|}{|A \cup B|}$$

Let $f : u \to 2^{64}$ with no collision, then let $h(A) = \min_{a \in A} f(a)$. Then $Pr[\min f(A) = \min f(B)] = |A \cap B|/|A \cup B|$. This is called the min hash scheme.

4.2 Sim Hash Scheme [8]

In the sim hash scheme, we are given a collection of vectors and the distance between two vectors is the angular distance between them, $dist(u,v) = \angle(u,v)$. Consider the following hash function:

- Projection: take a random vector r and compute the dot product of vectors $\langle r, u \rangle$
- Rounding: return $h_r(u) = sign(\langle r, u \rangle)$.

Then $Pr[h(u) \neq h(v)] = \angle(u,v)/\pi$. This is called sim-hash scheme.

References

[1] Johnson, William B., and Joram Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space." *Contemporary mathematics* 26.189-206 (1984): 1.

[2] Kushilevitz, Eyal, Rafail Ostrovsky, and Yuval Rabani. "Efficient search for approximate nearest neighbor in high dimensional spaces." *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* ACM, 1998.

[3] 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.

[4] Panigrahy, Rina. "Entropy based nearest neighbor search in high dimensions." *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. Society for Industrial and Applied Mathematics*, 2006.

[5] Lv, Qin, et al. "Multi-probe LSH: efficient indexing for high-dimensional similarity search." *Proceedings of the 33rd international conference on Very large data bases.* VLDB Endowment, 2007.

[6] Kapralov, Michael. "Smooth tradeoffs between insert and query complexity in nearest neighbor search." *Proceedings of the 34th ACM Symposium on Principles of Database Systems.* ACM, 2015.

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

[8] Charikar, Moses S. "Similarity estimation techniques from rounding algorithms." Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, 2002.