CS590STA Sublinear Time Algorithms February 24, 2015

Approximate Near Neighbor Search in High Dimensions

Lecturer: Nitin *Scribe: Nitin*

1 Introduction

In this lecture [1], we are going to study the problem of approximate near neighbor search. Given a set of points, the problem of near neighbor search is to find a near point p (say within a distance r) to an input query point q (See Fig. 1). For a choice of the parameter r, the problem is sometimes also called a r-near neighbor problem. Formally, the problem can be stated as under.

Definition 1 *r-near Neighbor Problem : Given a metric space (X, D), a point subset P and $r > 0$. Let $q \in X$, find any $p \in P$ with $D(q, p) \le r$.*

This problem is of significant importance in diverse fields of image recognition, data compression, database and data mining, information retrieval, machine learning et al. Efficient algorithms to the problem are available for low dimension d. However, with the increase in dimension (d) of data, the problem requires either storage or query time that is exponential in d. In the several areas of usage, it is very common to have high dimensional ($\sim$ 1,000,000) data. This lecture will talk about some ways to quickly solve the problem with such high dimensional data.

To obviate the burden of exponential complexity, several researchers (e.g. [7, 13, 12, 9]) have proposed to use approximation. In such a setting, the algorithm is allowed to return a point that is at most $c = (1 + \epsilon)$ times the distance between an input query and its near points (See Fig. 2). This allows us to filter out far away points while allowing some approximation for near points. Moreover, an efficient approximation algorithm can be used to quickly solve the near(est) neighbor problem as well by enumerating all the approximate neighbors and choosing the near(est) neighbors. Formally, the approximate near neighbor problem is as follows :

Definition 2 *c-approximate r-near Neighbor Problem : Given a metric space (X, D), a point subset P, $r >$ 0 and $c >$ 1. Let $q \in X$, find any p' $\in P$ with D(q, p') $\le$ cr provided that $\exists$ $p \in P$ s.t. D(q, p) $\le r$.*

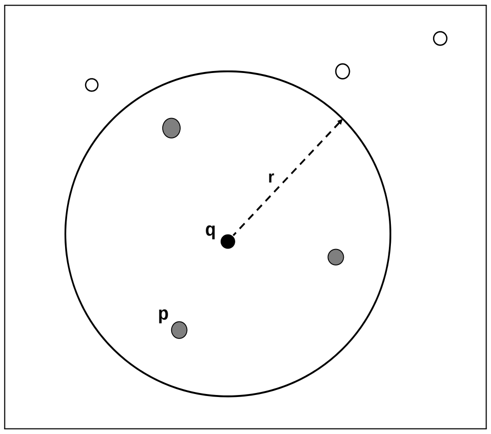


Figure 1: Near Neighbor problem, given q find p s.t. D(q, p) $\le$ r

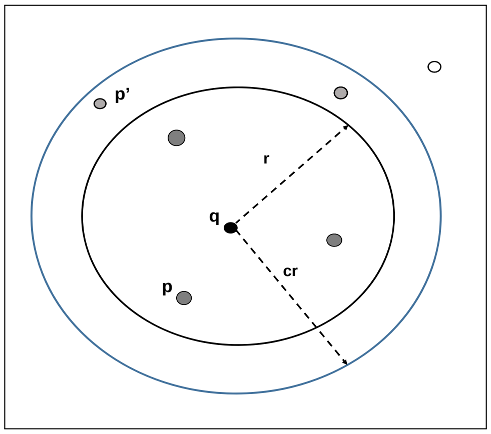


Figure 2: Approx Near Neighbor problem, given q find a p' s.t. D(q, p') $\le$ cr

[1]Notes prepared with the help of presentations [2, 3] by Alexandr Andoni at http://www.mit.edu/∼andoni

2 Approximate Near Neighbor (ANN) Algorithm

For the approximate near neighbor (ANN) problem, the goal is to find an algorithm that works in polynomial space and time, specifically near-linear in n storage, and sublinear in n query time. The only known technique that works to provide such a guarantee is that of locality sensitive hashing (LSH) [9]. The basic idea is to use several hash functions[2] that map points such that the probability of collison between far-away points is negligible and that of near points is high. Then, a query point can use such functions to retrieve nearby points from its hash bucket.

Definition 3 *LSH functions for near-neighbor : A family of hash functions $\mathcal{H} = \{h : S \to U\}$ on a metric (X, D) s.t. for every p and $q \in S$*

- *if D(q, p) $\leq r$ $P_{h \sim \mathcal{H}}$ [h(p) = h(q)] $\geq p_1$ is high.*
- *if D(q, p) $\geq cr$ $P_{h \sim \mathcal{H}}$ [h(p) = h(q)] $\leq p_2$ is low.*

ANN algorithm

For k specified later, let $\mathcal{G} = \{g : S \to U^k\}$ be a function family such that $g = \{h_1(p), h_2(p), ..., h_k(p)\}$ consists of k primitive hash functions that satisfy definition 3. We choose L hash functions $g_1 \cdots g_L$ from $\mathcal{G}$ independently and uniformly at random. Each such function corresponds to a hash table. Now, the algorithm consists of only two steps :

1 Hash all points into L hash tables.

2 Given a query q, check for collisions in all the hash tables until we get a point within distance cr.

ANN complexity

1 Let $\rho = \frac{\log 1/p_1}{\log 1/p_2}$ and $L = n^\rho$. L would also be the number of hash functions or hash tables used.

2 Then, the algorithm requires $O(n \cdot L + nd) = O(n^{1+\rho} + nd)$ space and query time $O(L \cdot d) = O(dn^\rho)$ (in expectation).

[2]The term hash functions and hash tables are used interchangebly in this lecture.

2.1 Locality Sensitive Hashing (LSH) Example

In the above definition ρ is known as the quality of the hash function family $\mathcal{G}$. It determines the number of hash tables required to get a good probability of collision for nearby points.

Let us now consider an example of a LSH function to get a better understanding of the hashing. The family g of LSH functions described above is a concatenation of several primitive hash functions. For k such primitive hash functions, $g =< h_1(p), h_2(p), ..., h_k(p) >$. Let $\{0,1\}^d$ be a Hamming space and $h(x) = x_i$ where i is uniformly randomly chosen from $[d]$; i.e. we choose i_{th} bit for a random i. Then g chooses k bits at random from x. Also, let $Hamm(p,q)$ be the Hamming distance between p and q. Then, from definition 3 we have

$$\begin{aligned}
p_1 \geq P[h(p) = h(q)] &= 1 - \frac{Hamm(p,q)}{d} \\
&= 1 - \frac{r}{d} \approx e^{-\frac{r}{d}} && \text{(r << d)} \\
p_2 &= 1 - \frac{cr}{d} \approx e^{-\frac{cr}{d}} \\
\Rightarrow \rho &= \frac{\log 1/p_1}{\log 1/p_2} = \frac{1}{c}
\end{aligned}$$

$\Rightarrow$ we get a query time of $O(d \cdot n^{1/c})$. Specifically, for a 2-approximate near neighbor we get a time of $O(d \cdot \sqrt{n})$

2.2 Analysis of LSH algorithm

We now wish to evaluate what values of k should we choose for a choice of $L = n^\rho$ hash tables. Each of those hash tables uses the function $g_i =< h_{i,1}(p), h_{i,2}(p), ..., h_{i,k}(p) >$. Recall from definition 3 that, if $D(q,p) \geq cr$, $P_{h \sim \mathcal{H}}[h(p) = h(q)] \leq p_2$. For k such hash functions, the probability of far points colliding is simply p_2^k. We choose k s.t. $p_2^k = \frac{1}{n}$; then,

$$\begin{aligned}
Pr[\text{collison of far points}] &\leq p_2^k \\
Pr[\text{collison of near points}] &\geq p_1^k \\
&= (p_2^\rho)^k \\
&= \frac{1}{n^\rho}
\end{aligned}$$

$\Rightarrow L = O(n^\rho)$ tables suffice to find a near neighbor with constant probability.

Recall that the algorithm outputs a point p', s.t. $D(q,p') \leq cr$ provided $\exists$ a p s.t. $D(q,p) \leq r$. The algorithm fails if for all L hash functions $(g_1(q) \cdots g_L(q))$, p' is not found in any of the hash buckets. The probability that q and p' do not collide in a hash table is $\leq$ 1 - p_1^k. Hence, the probability that they do not collide in any of the L hash tables is

$$\begin{aligned}
Pr[\forall i,\ g_i(q) \neq g_i(p')] &\leq (1 - p_1^k)^L \\
&= (1 - \frac{1}{n^\rho})^{n^\rho} \\
&\leq \frac{1}{e}
\end{aligned}$$

3 Dimension Reduction

In general, the problem of near neighbor becomes computationally expensive as the size of dimension (d) increases. One way to get around that issue is to reduce the dimension of the available data to a lower dimensional space $k << d$ and then solve the problem. If the time to transform the data into this lower dimensional space is low, we can find a fast method of solving the original problem. While transforming the data we need to take care that distances between points in $\mathbb{R}^d$ are preserved in $\mathbb{R}^k$.

In general, this is not possible due to the sphere packing bound. But for a fixed subset of $\mathbb{R}^d$, we have the Johnson Lindenstrauss (JL) lemma [10] that enables us such a transformation.

Lemma 4 *Johnson Lindenstrauss Lemma (variation) There is a randomized linear map $F\colon l_2^d \to l_2^k$, $k << d$, $0 < \epsilon < 1$, that preserves distance between two vectors x and y s.t.*

$$P\Big[\|x-y\| \leq \|F(x)-F(y)\| \leq (1+\epsilon)\|x-y\|\Big] \geq 1 - e^{-C\epsilon^2 k}$$

where C is some constant and $k = O(\frac{\log n}{\epsilon^2})$ for any set of n points.

To project $\mathbb{R}^d$ into $\mathbb{R}$, a choice of normal distribution as F satisfies the constraint provided above. For reduction into $\mathbb{R}^k$, we concatenate k-independently chosen normal distributions to form F. The time to compute the transformation the above lemma is $O(kd)$. A fast method to compute the JL dimension reduction [1] reduces the complexity to $O(d \log d + k^3)$.

4 ANN for Euclidean space

The paper [9] on LSH provided a $\rho = \frac{1}{c}$ bound for Hamming space. However, for Euclidean distances, we would have to first embed L_2 into Hamming space before applying the hash functions. A later paper [8] provided a better lower bound for Euclidean space. This is based on the idea of using dot product as a "locality sensitive" hash function. Dot product of a point with a random vector derived from a "p $-$ *stable*" (see [8] for details) distribution maps the point to a one dimensional $\mathbb{R}$ such that the properties as mentioned in lemma 4 hold. Specifically, for two vectors (v_1, v_2) the distance between their projections $(a.v_1 - a.v_2)$ is distributed as $\|v_1 - v_2\|X$; where $\mathbf{a}$ is a random vector in d dimensional space. For Euclidean distances, each entry [0-d] of $\mathbf{a}$ is chosen independently from a normal distribution and X is a random variable with a normal distribution. The hash functions used in this algorithm are :

$$h(p) = \left\lfloor \frac{p \cdot l + b}{r} \right\rfloor$$

Here l is a random normal vector, b is random in $[0, r]$ and r is a parameter similar to radius r in section 1. The hash family works directly in the Euclidean space, and it provides a $\rho = \frac{1}{c}$; similar to the one for Hamming space in [9].

5 Near-optimal hashing for ANN

The paper [14] provided optimal lower bounds of $\rho = \frac{1}{c}$ and $\rho = \frac{1}{c^2}$ for LSH in Hamming space and Euclidean space respectively. The original work on LSH [9] achieved the bound for Hamming space. Near-optimal hashing [4] presents an algorithm to almost achieve the optimal lower bound for Euclidean space. Specifically, for the c-approximate near neighbor problem, the algorithm achieves $\rho(c)$

$$\rho(c) = 1/c^2 + O(\log\log n / \log^{1/3} n)$$

This is an improvement over [8] where $\rho = \frac{1}{c}$. In [8], the hash functions project points on to random lines. The family of such projections leads to a space partitioning by a regular grid of hypercubes. To achieve the new optimal lower bound, [4] uses LSH funtions that partition space with a grid of spherical balls instead of a regular grid of hybercubes. Intuitively, because balls have minimum surface area for a given volume, they are able to "pack" nearby points better and consequently provide better LSH functions. The downside of using a grid of balls is that we require an exponential (in dimension d) number of balls to cover the metric space. In order to alleviate the exponential problem, it uses JL lemma 4 as a preprocessing step to project points in $\mathbb{R}^d$ to a reduced dimension t (high but constant). The hash functions then project the points into spheres in $\mathbb{R}^k$. With that as the backbone of the partitioning algorithm, it achieves $\rho \approx 1/c^2$. In the next subsection, a brief intuitive proof is provided. However, the time required to achieve the tight bound is not quick. Specifically, the time is bound by

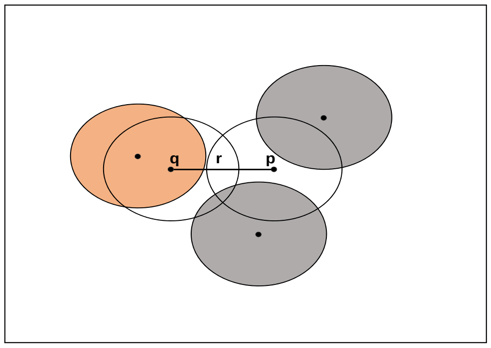
$$t^{O(t)} n^{1/c^2 + O(\log t)/t^{1/2}}$$

5.1 Intuitive proof to near-optimal ANN

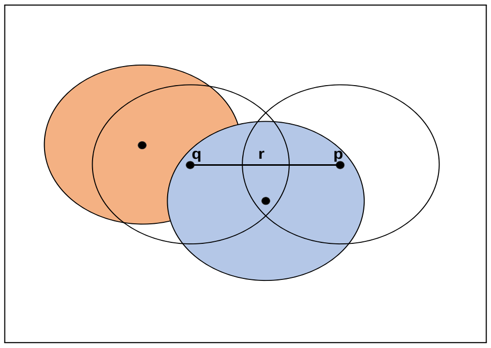
Claim 5

$$\rho \approx 1/c^2 \quad i.e.$$
$$P(r) \geq P(cr)^{1/c^2}$$

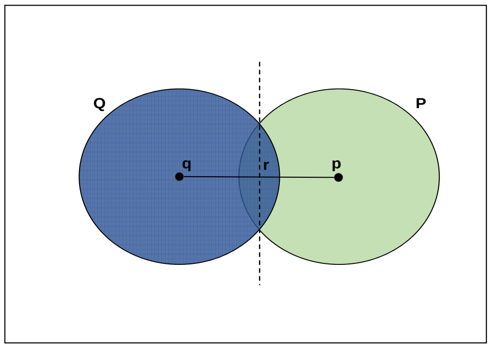
where P(r) is the probability of collision of points when $\|q - p\| = r$



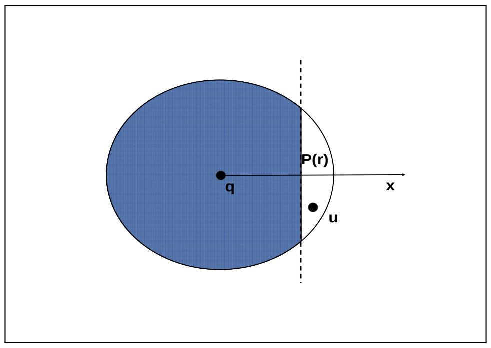
(a) Circles with centers outside P ∪ Q would not allow q and p to be colocated in the same hash bucket. Grey circles show such circles.



(b) Only circles with centers inside P ∩ Q would allow q and p to collide. The blue circle is an example of such a circle.



(c) Probability of collision is intersection/union.



(d) Collision probability P(r) is roughly the probability that a random point u from the sphere falls into the white region.

Figure 3: All circles have radius w. Circles P and Q have centers p and q respectively.

Proof

1. Projection of points to dimension t preserves distance (JL lemma)
2. P(r) = $\frac{Intersection}{Union}$ see Figure 3 (a) through (c)
3. P(r) ≈ random point u from a sphere falls beyond the dashed line in Figure 3 (d)
4. Fact : In high dimensions, the x-coordinate of u has a nearly normal distribution.

$$\Rightarrow P(r) \approx exp(-Ar^2)$$
$$= [exp(-A(cr)^2)]^{1/c^2} = P(cr)^{1/c^2} \quad \blacksquare$$

5.2 Beyond Locality-Sensitive Hashing

The above LSH functions partition the space independent of the distribution of the input point subset P. A recent paper[5] improves upon the LSH bound for ANN from [4] by providing a hash function family that exploits the geometry of the input data.

The main idea behind this work is to first apply a few rounds of data-independent LSH to partition the input data into low-diameter clusters ($\sim O(cr)$). Subsequently, the points within the low-diameter clusters are hashed further, using the center of corresponding clusters as a parameter, to achieve finer space partitions. The second step is what makes the scheme a data-dependent LSH scheme. The scheme can further be extended to a multi-level hashing scheme instead of just two-levels of hashing.

For the low-diameter case, because all points are within a sphere of radius $O(cr)$, applying the method of "ball carving" [11] achieves a $\rho(c)$ of

$$\rho(c) = \frac{1-\Omega(1)}{c^2}$$

Data Dependent LSH algorithm

For a choice of k in section 2 we obtain a space complexity of $O(n^{1+\rho})$ and a query time complexity of $O(dn^{\rho})$. For the data-dependent LSH we use a two-level hashing. The steps in the algorithm are as follows :

1 Hash all points coarsely using a somewhat smaller k than before.

2 Argue that each part has a low-diameter ($O(cr)$).

3 Use the better hash family to partition the space finer for each of the low-diameter clusters.

Because of the outer data-independent hash functions ($\rho(c) = 1/c^2$) and inner data-dependent hash functions, we obtain a better $\rho(c) = 1 - \Omega(1)/c^2$.

The application of data-dependent LSH, for Euclidean distances, leads to a $\rho(c)$ of

$$\rho(c) = \frac{7}{8c^2} + O(\frac{1}{c^3}) + o_c(1)$$

Furthermore, by a simple reduction of l_1 into $l_2 - squared$, for hamming distances, we get a $\rho(c)$ of

$$\rho(c) = \frac{7}{8c} + O(\frac{1}{c^{3/2}}) + o_c(1)$$

5.2.1 Recent Updates

A very recent paper [6] (not covered in this lecture) further reduces the $\rho(c)$, for Euclidean distances, to

$$\rho(c) = \frac{1}{2c^2-1}$$

References

[1] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 557–563, New York, NY, USA, 2006. ACM.

[2] Andoni Alexandr. Dimension reduction. `http://www.mit.edu/~andoni/dim-reduction.pptx`. Accessed: 2-16-2015.

[3] Andoni Alexandr. Locality sensitive hashing. `http://www.mit.edu/~andoni/lsh_cph.pptx`. Accessed: 2-16-2015.

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, January 2008.

[5] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. *CoRR*, abs/1306.1547, 2013.

[6] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. *CoRR*, abs/1501.01062, 2015.

[7] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 573–582, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[9] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[10] W. B. Johnson and J Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *Contemp. Math. 26*, pages 189–206, 1984.

[11] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, March 1998.

[12] Jon M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 599–608, New York, NY, USA, 1997. ACM.

[13] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 614–623, New York, NY, USA, 1998. ACM.

[14] Rajeev Motwani, Assaf Naor, and Rina Panigrahi. Lower bounds on locality sensitive hashing. In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, SCG '06, pages 154–157, New York, NY, USA, 2006. ACM.