COMS 4995-3: Advanced Algorithms — Feb 13, 2017

Lecture 8 – Sketching and Nearest neighbor search

Instructor: *Alex Andoni* — Scribes: *Mehul Kumar, Yanlin Duan*

1 Introduction

Today's lecture is mainly about Nearest Neighbor search, starting at the introduction to the problem, various approaches to solve it and corresponding space and time bounds. We also discussed dimension reduction and sketching. A couple of relevant theorems are presented. The main problem is to avoid the curse of dimensionality, which serves as a motivator for creating algorithms that find object approximately close to a given query, usually in a high-dimensional space.

2 Review - Dimension Reduction ($\ell 2$-norm)

Recall the dimension reduction for $\ell 2$-norm, Johnson-Lindenstrauss Lamma, from last time:

Lemma 1. *$\forall \epsilon > 0, \exists$ randomized mapping ϕ s.t.*

$$\phi : \mathbb{R}^d \to \mathbb{R}^k : \phi(X) = \frac{1}{\sqrt{k}} GX \tag{1}$$

Where $G = k \times d$ matrix of i.i.d. Gaussian random variables.

Here are the probability bounds for the Johnson - Lindenstrauss lemma:

Claim 2.

$$\forall x, y \in \mathbb{R}^d,\ Pr(\|\phi(x) - \phi(y)\|_2 \in (1 \pm \epsilon) \|x - y\|_2) \geq 1 + e^{-\frac{\epsilon^2 k}{9}} \tag{2}$$

$$\textit{If we further bound } k = O(\frac{logn}{\epsilon^2}) \implies Pr(\|\phi(x) - \phi(y)\|_2 \in (1 \pm \epsilon) \|x - y\|_2) \geq 1 - \frac{1}{n^3} \tag{3}$$

3 Review - Sketching ($\ell 1$-norm/Manhattan space)

Definition 3. *Pick a randomized $\psi_i : \mathbb{R}^k \to \mathbb{R}^d$ s.t.*

$$\forall x, y \in \mathbb{R}^d, Pr(median(\psi_l(x) - \psi_l(y)) \in (1 \pm \epsilon) \|x - y\|_1) \leq 1 - e^{-\frac{\epsilon^2 k}{9}} \tag{4}$$

Theorem 4. *$\exists \psi' : \mathbb{R}^d \to \mathbb{R}^k$ s.t.*

$$\forall n, D \subset \mathbb{R}^d, \forall x, y \in D,\ \left\|\psi'(x) - \psi'(y)\right\| \in (1 \pm \epsilon) \|x - y\|_1 \tag{5}$$

$$\textit{for } \ell_1,\ k = O(\frac{n}{\epsilon^2}) \tag{6}$$

$$\textit{for } \ell_2,\ k = \binom{n}{2} \tag{7}$$

4 Nearest-neighbor Search

Assuming the preprocessing of the data as a seperate step, we define Nearest Neighbor search as

Definition 5. *Given $D \subset \mathbb{R}^d, |D| = n$*

$$\forall q \in \mathbb{R}^d,\ \textit{find } p^* \in D \textit{ s.t. } \|p^* - q\| = \min_{p \in D} \|p - q\| \tag{8}$$

4.1 Motivation

This idea of finding points nearest to the query point is utilized in a lot of situations, like k-nearest neighbors, nearest neighbor search, similarity search, etc. Figure 1 shows how we may vectorize a picture and perform nearest neighbor to find similar graphs.

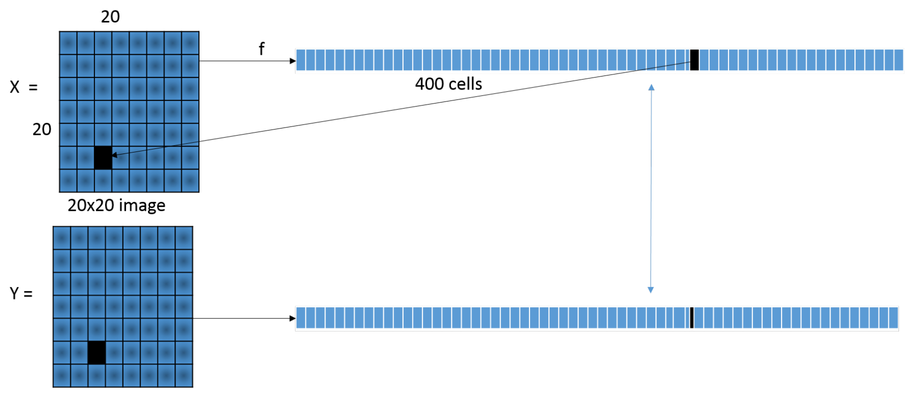

Figure 1: Example using 20x20 images

4.2 Solution #0

A naive approach to start with would be to go through all the points p in D.

- Space $= O(nd)$, which is good.
- Query Time $= O(nd)$, which is bad.

4.3 Solution #0.5: d = 1

For d = 1, we can use a Binary Search Tree to find the nearest neighbor. This takes:

- Space $= O(n)$, which is good.
- Query Time $= O(\log n)$, also good.

4.4 Solution #0.5: d $\geq$ 2

The approach, of partitioning the space, to recursively cut-down search space, can be extended to higher dimensions. We can use Voronoi Diagrams for the same. When d = 2:

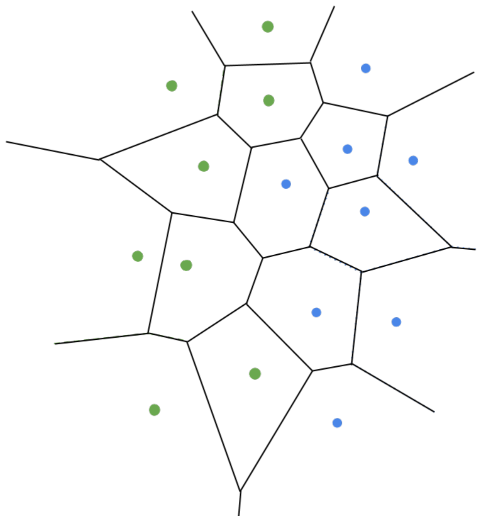

Figure 2: Voronoi Diagram in 2D [**2**]

To find which region q (the query point) belongs to is not a trivial task. This is called point location. When d=2, using Voronoi Diagram, we may achieve:

- Space = $O(n^2)$
- Query Time = $O(\log n)$

This idea can be extended to higher dimensions. Other data structures and algorithms, such as K-D tree, are also commonly used. **We note one problem though: as the number of dimensions increase, the query time grows exponentially in d.** This is a common problem with high dimensional representation - the Curse of Dimensionality.(See figure 3 for an intuitive explanation)

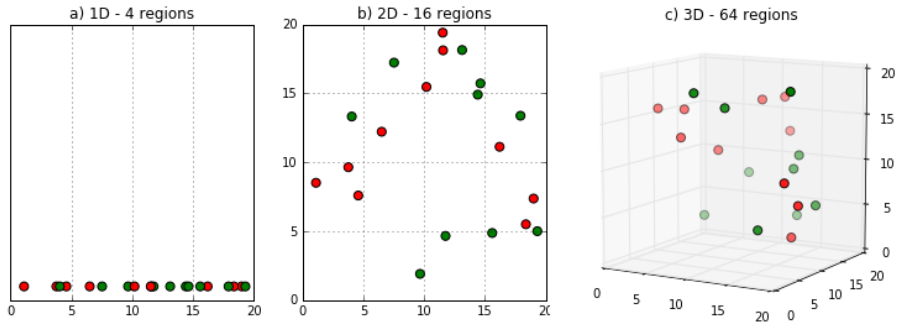

Figure 3: How partitions grow exponentially [**3**]

5 Approximate nearest neighbor (ANN)

Sometimes, it may be acceptable to obtain a "good guess" of the nearest neighbor. In those cases, we can use an algorithm which doesn't guarantee to return the actual nearest neighbor in every case, but will return a neighbor close enough. Those are called approximate nearest neighbor (ANN) problems.

More formally, we define ANN problems as following:

Definition 6. *Given $c > 1$, $D \in \mathbb{R}^d, |D| = N$, construct a data structure, such that:*

$$\forall q \in \mathbb{R}^d,\ \textit{returns any } p^* \textit{ s.t. } \|q - p^*\| \leq c \cdot min_{p \in D} \|q - p\| .$$

Note that the distance can be defined differently and the pre-processing on D is allowed.

To help us solve c-ANN problem, we define a similar problem called **c-approximate r-near neighbor problem**:

Definition 7. *Given $c > 1, r > 0, D \in \mathbb{R}^d, |D| = n$ (pre-processing on D allowed), A c-approximate, r-near near neighbor problem is defined as:*

For any query q, we want the following happen with probability $\geq 90\%$:

- *if $\exists p^* \in D$ s.t. $\|q - p^*\| \leq r$, output $p' \in D$ s.t. $\|q - p'\| \leq c \cdot r$.*
- *if no such p^* exists, then output "No".*

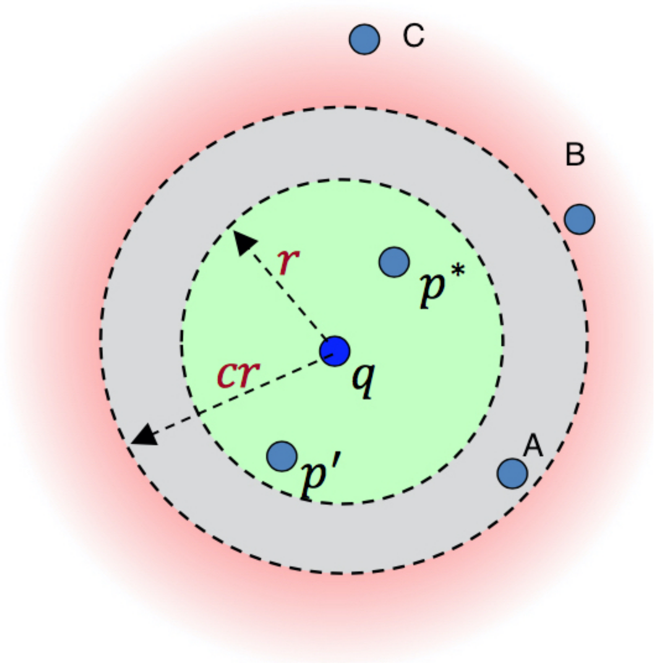

Figure 4: Illustration of c-approx. r-near neighbor problem [**4**]

As shown in Figure 4, suppose we have a point p^* such that the distance between q and p^* is less than r, then the algorithm should output either p' or point A. Point B and C will not get returned.

If, on the other hand, both p' and $p*$ do not exist, then the algorithm may report A or not (since there's no point within the r-radius circle).

Now we link the c-approx. nearest neighbor problem with the c-approx. near neighbor problem:

Lemma 8. *If we can solve c-approx. neighbor problem with space S and query time Q, then we can solve $(1+\epsilon)\cdot$ c-approx. nearest neighbor problem with*[1]*:*

- *Space:* $O(S \lg_{1+\epsilon} \frac{d_{max}}{d_{min}})$
- *Query time:* $O(Q \lg\lg_{1+\epsilon} \frac{d_{max}}{d_{min}})$

where d_{max} and d_{min} are the maximum and minimum inter-point distance in D, respectively.

Here is a proof sketch for the above lemma. The general idea is to build near-neighbor data structures for all possible thresholds (as shown in the below figure). During query time, we will essentially be doing binary search.

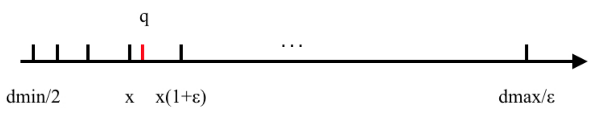

Figure 5: Transforming c-nearest neighbor problem to c-near neighbor problem using binary search

Note the range we care about is from $d_{min}/2$ to d_{max}/ϵ. The reason is that:

- For minimum, we know that if there is a point p whose distance with q is $\leq d_{min}/2$, then it must be the nearest neighbor as the closet two points in D has distance d_{min}, so any point in D (apart from p) has a distance to q larger than $d_{min}/2$.
- For maximum, we only need to consider up to d_{max}/ϵ, because (here p^* is the nearest neighbor candidate):

$$\begin{aligned}\forall p \in D, \|p-q\| &\leq \|p^*-q\| + d_{max} \text{(triangle inequality)}\\ &\leq \|p^*-q\| + \epsilon\,\|p^*-q\| \text{ (since } \|p^*-q\| \geq d_{max}\text{, given it is maximum)}\\ &= (1+\epsilon)\,\|p^*-q\|\end{aligned}$$

We are confident that p^* in this case will be the $(1+\epsilon)$ nearest neighbor.

6 Solution #1: Dimension Reduction / Sketching

We can approach the c-approx. nearest neighbor search problem through dimension reduction / sketching. Here is the algorithm:

[1] [Har-Peled, Indyk, Motwani'12] actually suggested a more efficient reduction.

Lemma 9. • *Pick a random* $\phi = \frac{1}{\sqrt{k}} G$, *where* $k = O(\frac{\lg n}{\epsilon^2})$.

- *Compute* $D^{'} = \{\phi(x), x \in D\}$.
- *At query time, compute* $q^{'} = \phi(q)$, *then compute distance.*

Proof. Apply the Johnson-Lindenstrauss Lamma to set $D \cup \{q\}$. For $\ell 1, \ell 2$:

- Space: $O(n \frac{\lg n}{\epsilon^2})$
- Query time: $O(n \frac{\lg n}{\epsilon^2})$

□

7 Solution #2 [Kushilevitz, Ostrovsky, Rabani'98]:

Theorem 10. $\exists \phi_{d,r} : \{0,1\}^d \to \{0,1\}^k, k = O(\frac{\lg n}{\epsilon^2})$, *such that:*

- *if* $\|x-y\|_1 \leq r$, *then* $\|\phi(x) - \phi(y)\|_1 \leq (\frac{1}{4} - \frac{\epsilon}{4}) \cdot k$
- *if* $\|x-y\|_1 > (1+\epsilon) \cdot r$, *then* $\|\phi(x) - \phi(y)\|_1 > (\frac{1}{4} + \frac{\epsilon}{4}) \cdot k$

with probability $\geq 1 - \frac{1}{n^3}$.

Here:

$$\phi(x)$$

is a $k \cdot d$ matrix whose (i, j) entry is:

$$\begin{cases} 1, \text{ with probability} \propto \text{r/d} \\ 0, \text{ otherwise} \end{cases}$$

8 Question for next class

How to use [Kushilevitz, Ostrovsky, Rabani'98] to get $(1+\epsilon)$-approx. near neighbor for hamming space, where:

- Query time: sub-linear
- Space: $n^{O_\epsilon(1)}$

References

[1] MIT Advanced Algorithm by Prof. Ankur Moitra; Scribed by: Rio LaVigne, Matthew Staib, Dimitris Tsipras : http://people.csail.mit.edu/moitra/docs/6854lec6.pdf

[2] Voronoi Diagrams : http://www.dma.fi.upm.es/personal/mabellanas/tfcs/fvd/

[3] Curse of dimensionality : http://cleverowl.uk/wp-content/uploads/2016/02/pca_1d_2d_3d.png/

[4] Algorithm Techniuqes For Massive Data by Prof. Alexandr Andoni: http://sublinear.wikischolars.columbia.edu/file/view/Lecture10.pdf/561810629/Lecture10.pdf