COMS E6998-8: Advanced Data Structures (Spring'19) February 28, 2019

Lecture 5: Approximate Nearest Neighbor Search through LSH

Instructor: *Omri Weinstein* Scribes: *Evan Ziebart, Ruiqi Zhong*

1 Introduction

1.1 More on predecessor search

- Exponential trees: branching factor is exponentially increasing w.r.t. depth. Interesting running time and space: interpolation between Van Emde Boas Tree and Fusion Tree.
- 2D point location: finding which of n partitions of the 2D plane a given point lies within. The map can be represented by lines, curves, or even points if the map is a **Voronoi Diagram**. The best static DS is $S = O(n), T = O(\frac{\log n}{\log \log n})$. The fact that we can reduce below $\log n$ is interesting. The technique is to reduce to problems where all subdivisions are trapezoid, which can be solved by predecessor search on point pairs/lines.

1.2 Plan for this lecture

- Nearest Neighbor Search. High dimension (e.g. $d = \log n$) and low dimension (e.g. $d = 2$) NNS are fundamentally different. Note that predecessor search is a special case of nearest neighbor.
- LSH: Here we use a non-trivial technique called local sensitive hashing (LSH) to solve it.
- Black-box method to dynamize static data structure to solve nearest neighbor problem.

2 NNS in high dimensions

2.1 Preliminaries on NNS

Definition 1. NNS pre-process a dataset $S = \{x_1, x_2 ... x_n\}$ in some **metric space** X (e.g. $\mathbb{R}^d, S^d, \{0,1\}^d$), s.t. given query $y \in X$, we need to find $x^* = argmin_{x \in S'} ||y - x||$.

Such algorithm is a backbone of machine learning, image processing, recommendation systems, etc. The correct notion of similarity / definition of metrics depends on specific applications. Some example includes: ℓ_1, ℓ_2, ℓ_p editting distance, Jaccard distance. In this lecture we focus on ℓ_1.

Definition 2. $\ell_p(x,y) = (\sum_{i=1}^{d} |x_i - y_i|^p)^{1/p}$

Here we suppose $X = \{0,1\}^d$. There are two naive solutions:

- no pre-processing, achieving space $O(s)$ and search time $O(n)$ by iterating through the data set and finding the nearest neighbor (we usually assume that we can calculate the distance between two points in constant time).
- pre-compute solution for every point in the domain. $s = O(2^d)$, $t = O(1)$.

Curse of Dimensionality: No sub-exponential space data structure for fast query time is known for exact NNS.

Solution: Relax the problem by allowing an approximate answer.

2.2 Relaxation: Approximate Nearest Neighbor Search

Definition 3. Approximate Nearest Neighbor Search Problem: given an approximation factor c, dataset $S = \{x_1, x_2 ... x_n\} \in X$, $c > 1$, return $\hat{x} \in S$ s.t. $||\hat{x} - y|| \leq c||x^* - y||$, where x^* is the optimal solution (exact NN) as defined in definition **??**.

Definition 4. (c, r)-ANN: given y, if $\exists x \in S$ s.t. $||x - y|| \leq r$, then return a point $\hat{x} \in S$ s.t. $||\hat{x} - y|| \leq cr$. If $\forall x \in S, ||x - y|| \geq cr$, return $\emptyset$. Otherwise the algorithm can return anything.

Remark: In this setting the algorithm only needs to distinguish between "at least one point lies within distance r" and "all points lie out of distance cr", and do not need to consider cases in between. Note that there is also a decision formulation which returns true if $\exists x \in S$ s.t. $||x - y|| \leq r$, and false if $\forall x \in S, ||x - y|| \geq cr$.

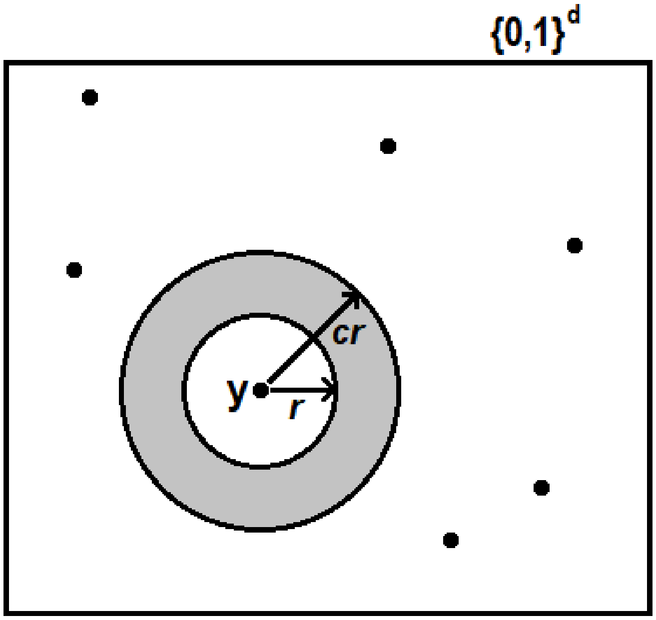
We observe that we can use (c,r)-ANN to solve the c-ANN problem. Given a $(c,r) - ANN$, choose radii whihc are powers of 2 and binary search on these. For a given radii: if (c,r)-ANN returns a point, eliminate upper half of radii; and if (c,r)-ANN returns $\emptyset$, eliminate the lower half of the radii. Hence we have a factor of $\log d = \log\log n$ increase in running time.

Results: Does the relaxation circumvent the "curse of dimensionality"? For many metrics, yes! The intuition is that the margin/approximation "gap" allows for dimension reduction via efficient geometric partitioning of the space which roughly preserves distances between points.

2.3 Local Sensitive Hashing

Theorem 1. For ℓ_1 over $\{0,1\}^d$, $\exists$ a randomized $(c,r) - ANN$ data structure, with

Figure 1: (C,r) approximate nearest neighbor search



- $s = O(n^{1+1/c})$
- $t = O(dn^{1/c})$
- success probability 0.9 (can be boosted arbitrarily)

Remark: Also, the data structure is fully dynamized with $t_q = t_n = n^{1/c}$. Note also that if we insist on $t = O(1)$, then for approximation factor $c = 1 + \epsilon$, $s = n^{O(1/\epsilon^2)}$.

Proof: We project the space into k (TBD) of random coordinates, where $K \subset [d], |K| = k$. Then we simply hash all the data points by their k coordinates (hence there are 2^k keys). Specifically, we first randomly sample a set $K = \{j_1, j_2 ... j_k\} \subset [d]$ of k indexes, then we create 2^k buckets, and each $x_i \in S$ is hashed to the bucket with key $= [x_{ij_1}, x_{ij_2} ... x_{ij_k}]$. The intuition is that closer points are more likely to be hashed together, while further points are not. So now it sounds like we can have $s = O(2^k)$ and $t = O(1)$? In fact, not quite.

Here is the analysis, $\forall y \in \{0, 1\}^d$. Let event $E_x =$ "$x|_k = y|_k$", then we want

$$\mathbb{P}[E_x| ||x - y||_1 \geq cr] = \mathbb{P}[x_i = y_i]^k = (1 - cr/d)^k \leq \delta/n^2 \tag{1}$$

Then we can take a union bound over all $x \in S$ and then the failure probability will be less than δ/n.

Now we can pick $k \geq \lceil \log n / \log(1 - cr/d) \rceil$.

$$\begin{aligned}
\mathbb{P}[E_x | ||x-y|| \leq r] &\geq (1 - r/d)^k] \\
&= (1-r/d)(1-r/d)^{\frac{\log n}{-\log(1-cr/d)}} \\
&= (1-r/d)2^{\frac{\log n \log(1-r/d)}{\log(1-cr/d)}} \\
&= (1-r/d)n^{-1/c} \\
&= O(n^{-1/c})
\end{aligned} \tag{2}$$

So far, for a single choice of K,

$$\mathbb{P}[E_x^y] = \begin{cases} \leq 1/n & \text{if all points are far from y} \\ \geq 1/n^c & \text{if some point is near} \end{cases} \tag{3}$$

We need to distinguish between these two cases, so we need $O(n^{1/c})$ trials. Therefore, we sample $O(n^{1/c})$ such K and hence we use $O(n^{1+1/c})$ space. In terms of running time, we need $t = O(n^{1/c})$ to distinguish between the two cases in equation **??**.

Notice that this data structure is inherently dynamic! All we need to do for insertion/deletion is to calculate its corresponding hashes and delete/add the elements from the corresponding bucket.

2.3.1 Discussion

The high level idea of local sensitive hashing in general is to design a randomized map $h : X \rightarrow \{0,1\}^k, k \ll dim(X)$ s.t. $Pr[h(x) = h(y)] \propto ||x-y||^{-1}$ (probability of collision is inversely proportional to distance) - creating an unbiased estimate of the distance. Whether such a map exists heavily depends on the metric.

There are 2 pre-requisite for LSH:

- Randomized hash.
- Triangular inequality satisfied by metric

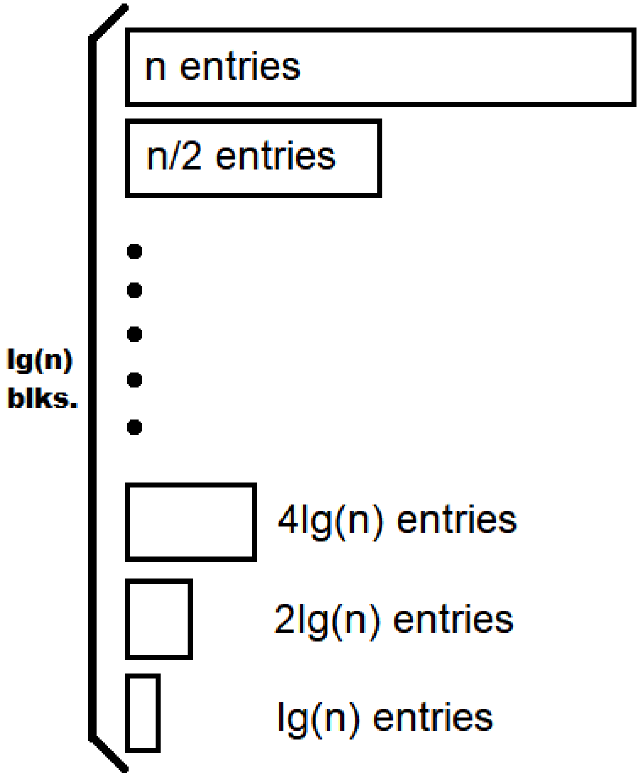
3 Black-box Dynamization

Theorem 2. Black-box Dynamization of ANN: $\forall$ static ANN w. space $s(n) \leq p(n)$ (pre-process time), query time $t(n)$, we can produce a dynamic ANN data structure with

- query time: $t_q \leq t(n) \log n$
- amortized update time: $t_u \leq O(p(n) \log n / n)$

.

Figure 2: black box dynamization with hierarchical sequence of blocks



The idea is as follows: we have $O(\log n)$ buckets of data, each containing $\{\log n, 2\log n ... n/2, n\}$ data points. i.e. bucket B_i contains 2^i data points. For each bucket we have a black-box data structure to answer ANN queries. When a data point arrive, we add to the first bucket; once the bucket B_i is full, we move all the data points in B_i to B_{i+1} and pre-process data points B_{i+1} again to answer ANN query. In this way, we only pre-process a bucket with 2^i data after 2^i insertaion queries. Therefore, the amortized cost is $\sum_{i=1}^{\log n} p(2^i)/2^i = O(\log n p(n)/n)$. During query time, we simply query each bucket, find ANN for each of the $\log n$ bucket, and then find the nearest neighbor in these $\log n$ candidates.

This is an instance of a **Decomposable DS**. A problem P with query y on space x is decomposable if $P(y, x_1 \cup x_2) = g(P(y, x_1), P(y, x_2))$ for some function g.