COMS 4995-2: Advanced Algorithms (Spring'20) Apr 28, 2020

Lecture Nearest Neighbor Search (part 2)

Instructor: *Alex Andoni* Scribes: *Bryant Hugo Zhang, Silva Xiaobin Lin, Declan O'Reilly*

1 Recap

Last lecture, we looked at the nearest neighbor search. Today we will look at research from the last decade as well as earlier research.

Last time, we looked at the approximate nearest neighbor search, specifically, that we can solve theoretically and have guarantees, which allows us to compare different algorithms. For a fixed c-approximate and a fixed threshold r, if we have a guarantee that there is something within distance r, the algorithm will find something that is within a distance of cr.

We looked at the LSH algorithms, that is based upon the locality of hashing functions, where we build n^ρ hash tables, where ρ depends on how well the space is partitioned. For hamming space we had $\rho = 1/c$, where the hashes are projected on random bins. For Euclidean space we use a similar ρ and space/times through we can improve $\rho \geq 1/c^2$ when we move from cells to cells.

We have

$$\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$$

$$p_1 = \text{Probability that similar pair of points collide}$$

$$p_2 = \text{Probability that dissimilar pair of points collide}$$

2 Euclidean LSH: Hyperplane ($\rho = 1/c$)

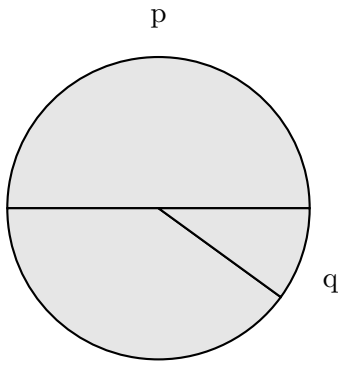
In this case, the vectors are unit-norm vectors. To define the hash functions, we have a circle which is the shape of the hyper-plane, i.e. all the vectors of unit norm. We pick a random direction r and position the circle with a line perpendicular to the direction r and going through the center, with +1 on one side and -1 on the other.

We hash p into $h(p) = sgn < r, p >$. $Pr[h(p) = h(q)]$ depends on the angle between p and q and is $1 - \frac{\alpha}{\pi}$ where α is the angle between p and q.

We have $p_1 = \frac{3}{4}$ and $p_2 = \frac{1}{2}$ with $p \approx 0.42$

Recall hash function $g = (h_1, h_2, ..., h_k)$, now we perform k such projections on k random directions.

This algorithm is not one of the most efficient, but it is one of the simplest.



3 LSH ≈ random dimension reductions

In the hamming space rather than using the full vector, we sample from it, which gives us an approximation to the real hamming distance. This corresponds to the hash function we are using in LSH. The previous Euclidean algorithm is also in a similar vain, where we lower the dimension while trying to maintain distance, with probability

$$1 - e^{\frac{-k\epsilon^2}{c}}$$

For nearest neighbors search, this gives us a reduction

$$k = O(\log n)$$

4 Detour: Fruit Fly Olfactory System (and how it compares to LSH)

The olfactory system is made up of four layers:

1. Odorant receptors ∼50 - normalized to release flow to nearest layer
2. Projection neurons ∼50 - random space connections (6/50) to nearest layer
3. Keyon cells ∼2000 - only 51 made it to next layer
4. Final layer

The random sparse connections from the projection neurons to the Keyon cells is a form of random dimension increasing, but the outermost layer is very sparse. So the paper by *Dasgupta, Stevens, and Navlakha [2018]* suggests that the transformation is similar to Locality-Sensitive Hashing, though of a different kind. The best transformation they suggest is a form of Winner-Take-All (WTA).

5 Back to LSH

In Euclidean space you can define certain hash functions that are data dependent, which have better performance i.e.

$$\rho \approx \frac{1}{2c^2 - 1}$$

in the exponent.

Data dependent LHS has 2 components:

1. If data is "pseudo-random", we get better LSH partitions
2. Can always reduce any worst case dataset to the "pseudo-random" case (Regularity lemma). Though this is not practical at the moment, i.e. it is theoretical

6 Data Dependent Hashing

We choose the random hash function that partitions the space after seeing the data. For part 2, we could build a Voronoi diagram using the data, which partitions the space around each point where each partition contains the space that is closer to the point than any other point. We have smaller partitions where the points are denser. This allows us to build the hash function from the diagram. The problem is that building this diagram is as inefficient as linear nearest neighbors search.

7 "Pseudo-random" case (Isotropic)

Ignore the data dependent part, we will assume the following two additional structures:

1. Far Pair is near orthogonal @distance cr. This is equivalent to when dataset is random on sphere. Pick two random points on unit sphere: $cr = \sqrt{2}$. All points are orthogonal to the query.
2. Close Pairs' distance is r.

Lemma 1. *There exists LSH with*

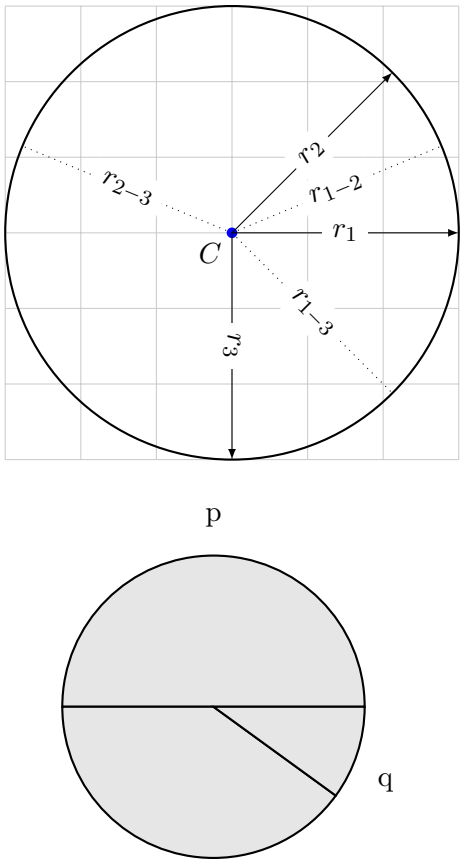
$$\rho = \frac{1}{2c^2 - 1}$$

(not data dependent). The Pseudo random case is the best possible, from the isoperimetry argument. We cannot further improve on such ρ.

8 Better LSH for Isotropic case

In contrast with Hyperplane LSH, sample T i.i.d standard d-dimensional Gaussian $r_1, r_2, ..., r_t$. Hash a point p into $g(p) = argmax_{1 \leq i \leq T} < p, r_i >$. (Winner-Take-All)

1. We have three partitions, each has a set of points.



2. Mathematically speaking, the dot product of such point with r_i is assigned to r_i if the product is the biggest among all i.
3. Geometrically speaking, the half angle line r_{j-k} between r_j and r_k separates the neighboring regions.
4. This is a better LSH for the isotropic case.
5. It begins to be very similar to Fruit Fly Olfactory System at least when there is only one winner.

Example:

- When T=2. This is the Hyperplane LSH.
- As T gets larger, start to differ from Hyperplane LSH.

9 WTA LSH for Isotropic case

Issue: Issue here is mainly two-fold. First, fruit fly can have more than one winner. And what does it mean to have "more than one winner", for example, top 5 percent? And does it have any equivalence in nearest neighbor search? Second, Fruit Fly can be said to only use one hash table.

Multi-probe LSH: Driven by the idea of doing nearest neighbor search using fewer hash tables to reduce space, [Lv-Josephson-Wang-Charikar-Li'07] suggests another nearest neighbor idea called multi-probe LSH. For a classic LSH, p_1 is very small, $\approx n^{-\rho}$, hence build n^{ρ} iid hash tables. While the pre-processing remains the same, everything is put in one hash table. Every time an element is checked, its nearby bucket will also be looked up as well.

However, we do need to define properly what "nearby" means. Normally starting from $g(p)$,if $g(q)! = g(p)$ and $g(q)$ is ruled out, $g_1(q)$ will be the next most likely option and it will be looked at then. After that, $g_2(q)$ subs in once $g_1(q)$ is ruled out. And the logic remains for the rest of the search.

Oftentimes in WT LSH, $g(q), g_1(q)...g_k(q)$ are top k max's.

10 Back to LSH: the data-dependent part

We can reduce any worst-case dataset to the "pseudorandom" case.

Lemma 2. *Any pointset $P \in R^d$ can be decomposed into clusters, where one cluster is pseudo-random and the rest have smaller diameter. (A form of "regularity lemma")*