COMS E6998-9: Algorithms for Massive Data (Spring'19) Apr 23, 2019

Lecture 25: Dimension Reduction, Sketching, NNS, and Fruit Flies

Instructor: *Alex Andoni* Scribes: *Qingtian Gong, Gleb Posobin*

1 Similarity Search

One typical example of similarity search problem is handwritten digit recognition:

Example 1. *Handwritten digit recognition.*
You have a set of labeled images, each of which is an image of handwritten digit. Then for a new image, recognize the digit it represents.

For a new image, one solution is to find the one in the labeled dataset with the highest similarity with it and use its label as the output. Now the question is: how can we measure similarity, and how can we encode the objects?
For handwritten digit recognition, we can use binary vectors to represent the images. In general, we encode the objects as high-dimensional vectors, and we use distances between vectors to measure similarity. For example, Hamming distance can be used when the objects $x_i \in \{0,1\}^d$. If $x_i \in \mathbb{R}^d$, then we can use Euclidean distances.

2 Nearest Neighbor Search

2.1 Definition

Taking advantage of similarity search, we can define nearest neighbor search problem:

Definition 2. *Nearest Neighbor Search problem (NNS): Given a set P of n points, for any point q, report a point $p^* \in P$ with the smallest distance to q.*

To solve this problem, we can simply store the set P in memory, and for a new point q, conduct a linear scan in P to compute distances between each $x \in P$ and q. But if we allow some approximation, we can definitely improve the query time and space. By modifying NNS, we have the ANN problem:

Definition 3. *Approximate Near Neighbor Search problem (ANN): Given a set P of n points, a factor $c > 1$, and threshold r, for any point q, if $\exists p^* \in P$ s.t. $\|p^* - q\| \leq r$, then report a point $p \in P$ s.t. $\|p - q\| \leq cr$.*

The applications of ANN include speech/image/video/music recognition, signal processing, bioinformatics, etc. So how can we solve this problem?

2.2 Idea 1: Random Dimension Reduction

One idea is to reduce the dimensionality by sampling k coordinates in the d dimensions. Recall the JL theorem in Lecture 6 [Johnson-Lindenstrauss '84].

Theorem 4. *In Euclidean space,* $\forall \epsilon > 0$, $\forall k \in \mathbb{N}$, $\exists$ *linear function* $S(p) = (G_1 p, G_2 p, ..., G_k p) = \mathbf{G}p$, *where each* G_i *is (scaled)* d*-dimensional Gaussian vector, s.t.* $\|S(p) - S(q)\| = (1 \pm \epsilon) \cdot \|p - q\|$ *with probability* $1 - e^{-k\epsilon^2/c}$

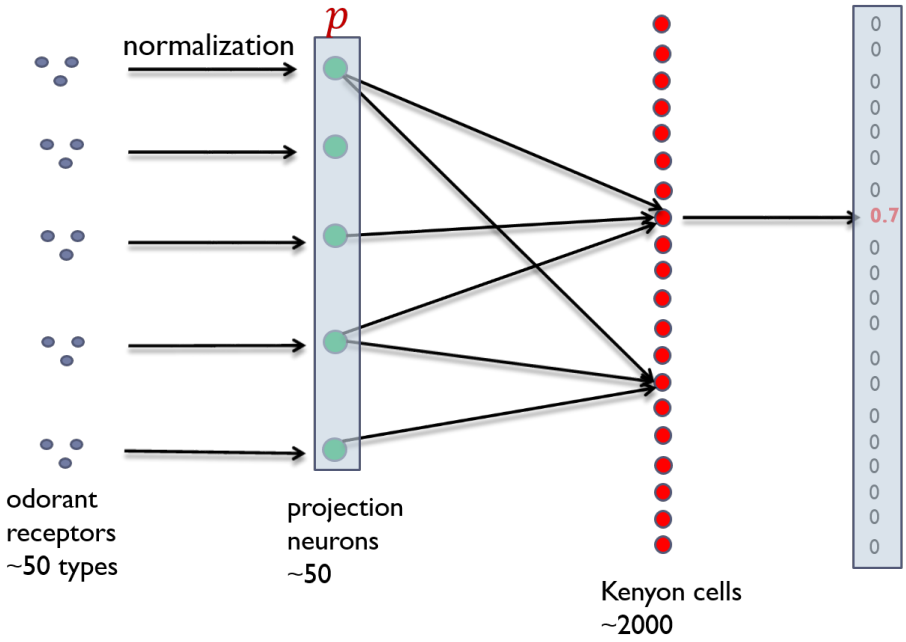
Then for NNS, it's enough to reduce to dimension $k = \frac{c \log n}{\epsilon^2}$. Space complexity is thus $O(nk) + O(dk) = O(\frac{n \log n}{\epsilon^2})$.

2.3 Idea 2: Sketching

We can generalize dimension reduction to the following sketch:
$S : \mathbb{R}^d \to \{0,1\}^k$ (or $\mathbb{R}^k$) s.t. $\exists$ some procedure R: given $S(p)$ and $S(q)$, $R(S(p), S(q))$ is a $(1 \pm \epsilon)$ approximation to $\|p - q\|$ with some probability.
Is this sketch more powerful than dimension reduction? Sometimes it is. In ℓ_1, sketching is possible, but dimension reduction is not. But still both ℓ_1 and ℓ_2 require $k \geq \frac{c \log n}{\epsilon^2}$ (for probability $1 - 1/n$). And also the best constant c is not known.

3 Locality-Sensitive Hashing (LSH)

3.1 Fruit Fly Olfactory System [Dasgupta-Stevens-Navlakha '18]



In the olfactory system of fruit flies, there are 50 types of odorant receptors. When the input smell comes, it is first sensed by the receptors. The signal is then normalized and passed to the corresponding

projection neurons. There are also 2000 types of Kenyon cells, which have random, sparse connections with the projection neurons. Each Kenyon cell is connected to about 6 projection neurons. Kenyon cells receive the signals, but only the top 5% strongest ones are activated in a Winner-Take-All (WTA) circuit. All other ones are inhibited.

3.2 Idea 3: LSH [Indyk-Motwani '98]

The idea of LSH is to find a random function $h : \mathbb{R}^d \to \{codes\}$ s.t.
• For close pair (when $\|q - p\| \leq r$), $P_1 = Pr[h(q) = h(p)]$ is "high".
• For far pair (when $\|q - p'\| > cr$), $P_2 = Pr[h(q) = h(p')]$ is "small".
It is usually not quite possible for P_1 to be "high", so we usually just make sure it is "not-so-small". Here we use n^ρ hash tables, where $\rho = \frac{\log 1/P_1}{\log 1/P_2}$ ($\Leftrightarrow P_1 = P_2^\rho$).

Example 5. *Hyperplane.*
Assume r, p, q are all unit-norm vectors. We sample unit r uniformly, and hash p into

$$h(p) = sgn\langle r, p\rangle$$

Then

$$Pr[h(p) = h(q)] = 1 - \alpha/\pi$$

where α is the angle between p and q.
Then $P_1 = 3/4$, $P_2 = 1/2$, $\rho \approx 0.42$.

The full LSH data structure looks like this: the whole data structure is $L = n^\rho$ hash tables, i-th having $g_i(p) = \langle h_{i,1}(p), \ldots, h_{i,k}(p)\rangle$ as its hash function, with fresh randomness for each of $h_{i,j}$. Algorithm hashes the whole dataset into each of these tables, and when asked to find nearest neighbor to a point q, it goes through each of the hash tables, hashes the point q with g_i and scans the bucket $g_i(q)$ for a point p within distance cr from q, repeating with the next table if no such point is found.

3.3 Analysis of LSH

This data structure requires $O(nL) = O(n^{1+\rho})$ space, plus space required to store points themselves (we store pointers in hash tables). Expected query time is $O(L(k + d)) = O(n^\rho d)$, and the probability of success is 50% (the algorithm may fail when none of the close points were hashed into the same bucket with q).

Let us understand where such parameters come from. If for each $h_{i,j}$ the probability of collision of a far pair is P_2 and the probability of collision of a close pair is P_1, then same probabilities for g_i are P_2^k and P_1^k respectively because the hash functions are sampled independently. So if we set k to be such that $P_2^k = 1/n$ (we want to have at most a constant number of far-away points in the bucket), and let $P_2^\rho = P_1$, then probability of collision of close points is at least

$$P_1^k = (P_2^\rho)^k = (P_2^k)^\rho = \frac{1}{n^\rho},$$

so if we make at least $L = \Theta(n^\rho)$ hash tables, with probability 50% we will find a close point if there is one.

What LSH functions are possible for the Euclidean space? It is possible to get $\rho = 1/c$ by taking an orthonormal lattice, applying a random rotation and scaling, and encoding each point by the cell of the lattice it fell into. We can get $\rho = 1/c^2$ by making a similar construction, but now looking not at a cell the point fell into, but at the closest lattice point, provided it is closer than some fixed radius. It can be shown that $\rho = 1/c^2$ is the best possible result for Euclidean spaces.

Space	Time	Exponent	$c = 2$	
$n^{1+\rho}$	n^ρ	$\rho = 1/c$	$\rho = 1/2$	Orthonormal lattices
		$\rho = 1/c^2$	$\rho = 1/4$	Shifts of balls
		$\rho = 1/(2c^2 - 1)$	$\rho = 1/7$	Data-dependent hashing

But we can get an improvement if we use data-dependent hash functions, and in this case we get $\rho = 1/(2c^2 - 1)$. This can be done in two steps: first looking at the case when the set of points is "pseudo-random," and then showing that it is possible to reduce any case to the "pseudo-random" one.

"Pseudo-random" case is isotropic: when far pairs are nearly orthogonal, which is equivalent to a random set of points on a sphere. For that case, there exists an LSH with $\rho = 1/(2c^2 - 1)$, and it can be shown that it is impossible to get a better ρ [A-Razenshteyn '16].

How to achieve such ρ? We sample T i.i.d. standard Gaussians $g_1, \dots, g_T$ in d dimensions, and hash point p into $h(p) = \arg\max_i \langle g_i, p \rangle$. Notice the similarity with the process happening in fly's olfactory system: here also a winner-take-all process is happening when we select the index of the largest inner product. Though there is a difference: fly's olfactory system takes top-k "winners," and we take only one, but this can be reconciled by taking k largest inner products and hashing point p into all of the corresponding buckets. So it turns out that the fly's olfactory system evolved to use the best possible algorithm for hashing vectors in Euclidean spaces!