High dimensional geometry and dimension reduction

Huy L. Nguyễn

In most modern dataset, we have to deal with data in high dimensions. Many classical algorithms do not deal with the high dimensionality very well: the running time is typically exponential in the number of dimensions. In this note, we will look at how objects behave in high dimensions and the tools for dealing with them.

For a vector x, its ℓ_2 norm is $\|x\|_2 = \sqrt{\sum_i x_i^2}$ and its ℓ_1 norm is $\|x\|_1 = \sum_i |x_i|$. For any two vectors x, y, their Euclidean distance is $\|x - y\|_2$ and their Manhattan distance is $\|x - y\|_1$.

Next we explore a few facts in high dimension to get used to the geometry.

Example 1. How many nearly orthogonal vectors can we have in a d dimensional space, such that all pairwise angles are between 89 and 91 degrees?

In $\mathbb{R}^2$, we can have 2. In $\mathbb{R}^3$, we can have 3. It turns out in $\mathbb{R}^d$, we can have $\exp(cd)$ for some constant $c > 0$.

Example 2. What are the volumes of the cube $\{x : 0 \le x_i \le 1\}$ and the unit ball $\{x : \|x\|_2 \le 1\}$ in d dimensions?

The volume of the cube is easy to compute, it is 1.

It turns out that the volume of the unit ball is $\frac{\pi^{d/2}}{(d/2)!}$. This is a $2^{\Theta(d \log d)}$ factor smaller than the volume of the cube.

Example 3. If we sample a random point from a unit ball in d dimension, what is the expected distance from that point to the center?

In 1 dimension, that expected distance is 1/2. It turns out that in high dimensions, the expected distance is very close to 1. To see why, let's compare the volume of the unit ball and the volume of the ball with radius 0.99. The smaller ball accounts for only $0.99^d \ll 1$ fraction of the larger ball. Thus, most points are at distance larger than 0.99 from the center.

1 Number of nearly orthogonal vectors

We will show that there are exponentially many nearly orthogonal vectors in $\mathbb{R}^d$. Recall that the cosine of the angle between two vectors x, y is $\frac{\langle x,y\rangle}{\|x\|\cdot\|y\|}$.

Lemma 1.1. *Let x be a fixed vector of length 1 and $|x_i| \le 1/\sqrt{d}\ \forall i$. Let y be a random vector where each coordinate is chosen randomly from $\{1/\sqrt{d}, -1/\sqrt{d}\}$. We have*

$$\Pr[\langle x, y\rangle > c] \le \exp(-c^2 d/3)$$

Proof. We have $\mathbb{E}[\langle x, y\rangle] = 0$. The lemma follows from the Chernoff bound. □

By the Lemma, we see that if we pick two random vectors whose coordinates are chosen randomly from $\{1/\sqrt{d}, -1/\sqrt{d}\}$ then with probability $1 - 1/n^2$, their dot product is at most $O(\sqrt{\log n/d})$. If we pick n vectors, by the union bound, we have with probability at least 1/2, the dot products among them are all bounded by $O(\sqrt{\log n/d})$. We see that n can be as large as $\exp(\Theta(d))$ while keeping the dot products bounded by 0.01.

2 Dimension reduction

Given n points in d dimensional space, we would like to find a representation of them in lower dimensions m while preserving all distances up to $1 \pm \varepsilon$ factor.

Lemma 2.1 (Johnson-Lindenstrauss lemma)**.** *It suffices to have $m = O(\log n/\varepsilon^2)$ and in fact, the mapping can be a linear map.*

Some basic strategies that do not work: sample coordinates (e.g. vectors with mostly zeroes and only few non-zeroes), group coordinates and add them up (e.g. coordinates of opposite signs canceling each others).

Proof. Consider m vectors $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^d$ whose coordinates are i.i.d. from $\{\frac{-1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\}$. The embedding of a vector v is

$$f(v) = (\langle x^{(1)}, v\rangle, \ldots, \langle x^{(m)}, v\rangle)$$

Let $z = u - v$. Let's compute the expected value of the squared distance between $f(u)$ and $f(v)$:

$$\begin{aligned}
\mathbb{E}[\|f(u) - f(v)\|^2] &= \mathbb{E}\left[\sum_{i=1}^{m} (\langle x^{(i)}, z\rangle)^2\right] \\
&= m\,\mathbb{E}\left[(\langle x^{(1)}, z\rangle)^2\right] \\
&= m\,\mathbb{E}\left[\sum_{i=1}^{d}\sum_{j=1}^{d} x_i^{(1)} x_j^{(1)} z_i z_j\right]
\end{aligned}$$

Notice that for $i \neq j$, $\mathbb{E}[x_i^{(1)} x_j^{(1)}] = 0$. For $i = j$, we have $\mathbb{E}[x_i^{(1)} x_j^{(1)}] = \frac{1}{m}$. Thus,

$$\mathbb{E}[\|f(u) - f(v)\|^2] = m\,\mathbb{E}\left[\sum_{i=1}^{d} \frac{1}{m} z_i^2\right] = \|z\|^2 = \|u - v\|^2$$

Next we need to show that it is concentrated around this value. To do this we need a variant of the Chernoff bound.

Lemma 2.2. *Let $\mu = \mathbb{E}[\|f(x)\|^2]$ and let $\beta \in (0,1)$. There exists constants c_1, c_2 such that*

$$\begin{aligned}
\Pr[\|f(x)\|^2 \geq (1+\beta)\mu] &\leq \exp(-c_1\beta^2\mu) \\
\Pr[\|f(x)\|^2 \leq (1-\beta)\mu] &\leq \exp(-c_2\beta^2\mu)
\end{aligned}$$

We can choose $\beta = \varepsilon/2$ and $m = O(\log n/\varepsilon^2)$. We have that for any two vectors u, v, their distance is preserved with probability at least $1 - 1/n^3$. By the union bound over all $\binom{n}{2}$ pairs of vectors, the distances among n vectors are all preserved with probability at least $1 - 1/n$. □

3 Nearest neighbor search

Next we consider the problem of finding the nearest neighbor in a database of points. That is, we are given a dataset with n points. For any query point q, we would like to find the point in the database that is closest to q. In low dimension, a classical solution is the Voronoi diagram. The

Voronoi cell of a point p is simply the set of points that are closer to p than all other points. In 2D, the diagram consists of $O(n)$ segments. Thus, we can store the diagram and given a query point q, use (a complicated variant of) binary search trees to find the nearest point in the database.

Question: how complicated is a Voronoi diagram in 3D? can it be described using $O(n)$ segments?

Unfortunately the description of the Voronoi diagram scales as $n^{\lceil d/2 \rceil}$ so it quickly becomes infeasible to store in higher dimensions.

One approach to deal with the higher dimension case is to relax the problem: we are only required to find an approximate solution rather than the nearest point. That is, given a query point q and an approximation factor $c > 1$. If the nearest neighbor to q in the database is at distance r then the algorithm needs to return a point in the database at distance at most $c \cdot r$.

In fact, we will focus only on a *near* neighbor variant. There is a reduction from the nearest version to this version. Given an approximation factor c and a distance threshold r, the algorithm needs to do the following: if there exists a point within distance r from the query then the algorithm needs to return a point within distance $c \cdot r$. If no such point exists, the algorithm can do anything.

Our algorithm will use inspiration from the solution for finding exact match in a database: *hashing*. In hashing, we would like to have the property that exact match are hashed to the same value with probability 1 and different keys are hashed to the same value with very small probability. In our setting, we would like near points to be hashed to the same value with high probability and far points are hashed to the same value with low probability. The following definition of *locality sensitive hashing* (LSH) formalizes this idea.

Definition 3.1. *A family of hash functions $h : \mathbb{R}^d \to U$ is (p_1, p_2, r, cr)-sensitive for the distance metric D if for any p, q:*

- *If $D(p,q) \le r$ then $\Pr[h(p) = h(q)] \ge p_1$*
- *If $D(p,q) \ge cr$ then $\Pr[h(p) = h(q)] \le p_2$*

To illustrate the idea, we will consider the Hamming metric. That is, the points we consider are bit strings of length d and the distance between two bit strings is simply the number of bits where they differ.

3.1 An LSH family for Hamming distance

Our hash family is very simple. There are d hash function, the ith function simply returns the ith bit of the string.

Lemma 3.2. *The family is $(1 - r/d, 1 - cr/d, r, cr)$-sensitive.*

Proof. When two points u and v are at distance r, that means they agrees on $d - r$ bits and disagree on r bits. Thus,

$$\Pr_{h \sim \mathcal{H}}[h(u) = h(v)] = \frac{d - r}{d} = 1 - r/d$$

The calculation is similar for distance $c \cdot r$. □

3.2 From LSH to approximate near neighbor

The basic hash function gives some separation in the collision probability between near and far points. However, in the dataset, there might be only 1 near point but a lot (up to n) of far points. Thus, we will still see a lot of collision with the far points. Our first idea is to reduce the collision probability by using many independent hash functions. We will use k independent hash functions $h_1, \ldots, h_k$ and the hash value is simply the concatenation of the hash value of each of these hash functions.

$$g(p) = (h_1(p), h_2(p), \ldots, h_k(p))$$

Using this concatenated hash function g, we can build a hash table containing all the points in the database. The algorithm will have l independent copies of the hash function g above: $g_1, g_2, \ldots, g_l$, each with their own hash table containing all the points in the database. Given the query point q, we do the following:

- retrieve the points from the buckets $g_1(q), g_2(q), \ldots, g_l(q)$ until either we get all the points from these buckets or we have retrieved more than $3l$ points in total.
- return the closest point to q among the points we found.

The space to store each hash table is $O(n)$ and we have l tables so the total space of the algorithm is $O(nl)$. The query time is dominated by the time to compute the distance between q and the retrieved points, which is $O(dl)$.

We will now show that the algorithm finds an approximate near neighbor to q with at least a constant probability.

Lemma 3.3. *For $l = n^\rho/p_1$ hash functions, where $\rho = \log(p_1)/\log(p_2)$, the algorithm solves the c-approximate near neighbor problem with constant probability.*

Proof. To solve the problem, we only need to consider the case where a near neighbor exists. Suppose there is a point p in the database P where $D(p, q) \leq r$. Let $FAR(q) = \{p' \in P : D(p', q) > cr\}$ the set of points far from q. Let $B_i(q) = \{p' \in P : g_i(p') = g_i(q)\}$ the set of points colliding with q.

We will show that the following two events happen simultaneously with constant probability:

- $E_1 : \sum_i |B_i(q) \cap FAR(q)| \leq 3L$: the event that the number of false positives is bounded by $3L$.
- $E_2 : g_i(p) = g_i(q)$ for some $i \in \{1, 2, \ldots, l\}$: the event that p collides with q in some hash table.

Let $k = \lceil -\log n/\log(p_2)\rceil$. Let's consider a fixed table i. What is the probability that a far point p' collides with q?

$$\Pr[g_i(p') = g_i(q)] = p_2^k = p_2^{\lceil -\log n/\log(p_2)\rceil} \leq 1/n$$

By linearity of expectation, we have $\mathbb{E}[|B_i(q) \cap FAR(q)|] \leq 1$. Again, by linearity of expectation, we have

$$\mathbb{E}[\sum_i |B_i(q) \cap FAR(q)|] \leq L$$

Thus, by Markov's inequality, event E_1 fails with probability at most $1/3$.

Next, what is the probability that the near point p collides with q?

$$\Pr[g_i(p) = g_i(q)] = p_1^k = p_1^{\lceil -\log n/\log(p_2)\rceil} \geq p_1^{1-\log n/\log(p_2)} = p_1 n^{-\rho} = 1/l$$

The probability that E_2fails i.e. the near point never collides with q in any table is at most $(1 - 1/l)^l \leq 1/e$.

By the union bound, the probability that either E_1 or E_2 fails is at most $1/3 + 1/e < 0.71$. Thus, the algorithm succeeds with probability at least 0.29. □

Notice that the parameter $\rho = \log(p_1)/\log(p_2)$ governs both the space and the query time of the algorithm: the space is $O(n^{1+\rho})$ and the query time is $O(dn^\rho)$.

Lemma 3.4. *For Hamming distance, $\rho \leq 1/c$.*

Proof. We would like to show $\log(p_1)/\log(p_2) \leq 1/c$ or equivalently $p_1^c \geq p_2$. Substituting in $p_1 = 1 - r/d$ and $p_2 = 1 - cr/d$, we obtain:

$$(1 - r/d)^c \geq 1 - cr/d$$

This is true because $(1 - x)^c \geq 1 - cx$ for all $x \in (0, 1), c \geq 1$. □

The value of ρ depends on the distance metric. It turns out that for Euclidean distance, it is possible to get $\rho \leq 1/c^2$.