CS265/CME309: Randomized Algorithms and Probabilistic Analysis
Lecture #9: Dimension Reduction and Nearest Neighbor Search

Gregory Valiant*

October 21, 2019

1 Dimension Reduction

In the previous lecture notes, we saw that any metric (X, d) with $|X| = n$ can be embedded into $R^{O(\log^2 n)}$ under any the ℓ_1 metric (actually, the same embedding works for any ℓ_p metic), with distortion $O(\log n)$. Here, we describe an extremely useful approach for reducing the dimensionality of a Euclidean (ℓ_2) metric, while incurring very little distortion. Such dimension reduction is useful for a number of reasons: on the practical side, many geometric algorithms have runtimes that scale poorly with the dimension of the space in which they operate. From a theoretical perspective these dimension-reduction procedures have been used numerous times as components within other algorithms (e.g. Locality Sensitive Hashing).

2 Johnson-Lindenstrauss Transformation

The randomized dimension reduction approach is essentially due to Johnson and Lindenstrauss in 1984 [4], and many variants (and de-randomizations) have been explored in the past 30 years.

Theorem 1. *Given any $\epsilon \in (0, 1)$, and a set $X \subset \mathbb{R}^d$ with $|X| = n$, there exists a randomized linear map $f : \mathbb{R}^d \to \mathbb{R}^m$ with $m = O(\frac{\log n}{\epsilon^2})$ that embeds (X, ℓ_2) into $(\mathbb{R}^m, \ell_2)$ with distortion at most $(1 + \epsilon)$.*

Proof. Let A be an $m \times d$ matrix with entries chosen independently from $N(0, 1/m)$, and define the map $f : \mathbb{R}^d \to \mathbb{R}^m$ by $f(x) = Ax$. Hence each of the m coordinates of $f(x)$ are given by the projection of x onto a d-dimensional Gaussian.

One useful trick in analyzing such Gaussian projections is the spherical symmetry of the Gaussian. Hence, to analyze the distortion, for a given pair of vectors x, y, we could imagine rotating the coordinate system so that $x - y$ is a basis vector. Such a rotation does not change the distribution,

*©2019, Gregory Valiant. Not to be sold, published, or distributed without the authors' consent.

and simplifies the analysis. (Exercise: prove this by leveraging the fact that sums of Gaussians are Gaussian...)

Given that $x - y$ is a basis vector, $A(x-y)$ only depends on one row of A, and hence $||f(x) - f(y)||_2^2 = ||x-y||_2^2\sqrt{\sum_{i=1}^m X_i^2}$, where the X_i's are independent Gaussians of variance $1/m$. Since the expected square of a zero-mean Gaussian is its variable, $\mathbf{E}[||f(x)-f(y)||_2^2] = ||x-y||_2^2$. Hence the theorem will follow provided we show that $\sum X_i^2$ is sufficiently tightly concentrated about its expectation. Specifically, if this is within a $(1+\epsilon)$ factor of its expectation, namely $\|f(x)-f(y)\|_2^2 = (1\pm\epsilon)\|x-y\|_2^2$, then that will imply that $\|f(x)-f(y)\|_2 = (1\pm\epsilon)\|x-y\|_2$. [The reason we are analyzing this squared norm instead of directly analyzing the actual distance is that the square-root would complicate the analysis.]

The probability that $\sum_{i=1}^m X_i^2$ is concentrated within a factor of $(1\pm\epsilon)$ of its expectation is identical to the probability that $\sum_{i=1}^m Z_i$ is within $(1\pm\epsilon)$ of its expectation, where Z_i is drawn from $N(0,1)$, so we will analyze this slightly simpler expression. To do this, we will prove a Chernoff-style bound, leveraging the easily verified fact that for $t < 1/2$, $\mathbf{E}[e^{tZ^2}] = 1/\sqrt{1-2t}$ for $Z \sim N(0,1)$. We begin by bounding the probability that $\sum_i Z_i^2 > (1+\epsilon)m$; a similar argument will show an analogous bound for $\Pr[\sum Z_i^2 < (1-\epsilon)m]$.

$$\begin{aligned}\Pr[\sum Z_i^2 > (1+\epsilon)m] &= \Pr[e^{t\sum Z_i^2} > e^{t(1+\epsilon)m}], \text{ [for } t>0]\\ &\leq \frac{1/(1-2t)^{m/2}}{e^{t(1+\epsilon)m}} \text{ [by Markov's inequality, for } t\in(0,1/2)]\\ &= e^{-m(t(1+\epsilon)+(1/2)\log(1-2t))}.\end{aligned}$$

Using the fact that $\log(1-2t) > -2t-4t^2$ (for $0 < t < \frac{1}{3}$), the above probability is at most $e^{-m\left(t\delta-2t^2\right)}$, for every $t \in (0,1/3)$. Optimizing this quadratic function $t\epsilon - 2t^2$ for t yields $t = \epsilon/4$. Plugging this in and simplifying yields that the above probability is at most $e^{-m\frac{\epsilon^2}{8}}$. Hence by choosing $m > \frac{17\log n}{\epsilon^2}$, this probability is $o(1/n^2)$, and hence we may perform a union bound over all $< n^2/2$ possible pairs of points x, y to argue that with constant probability, the embedding does not significantly distort any of the $O(n^2)$ distances. $\square$

2.1 Recent Advances: Fast Johnson-Lindenstrauss Transformations

To achieve distortion ϵ for a set of n points in $\mathbb{R}^d$, using the above scheme would require time $O(dm) = O(\frac{d\log n}{\epsilon^2})$ to compute the embedding of each datapoint. Can we hope to speed this up?

The most naive hope would be that there is a variance of the Johnson-Lindenstrauss scheme in which the random projection matrix, A, can be chosen to be sparse—if most of the entries are 0, then we can multiply by A in time proportional to the number of nonzero entries, instead of time (which is less than dm if A is sparse). Unfortunately, the guarantees of the theorem erode for sparse A, and this won't work.

However, in 2006 Ailon and Chazelle introduced the *Fast Johnson-Lindenstrauss* which is a clever twist on this. Roughly, instead of multiplying by a sparse A, we transform our data into the Fourier basis (recall that the Fourier transform is just multiplication by a $d\times d$ matrix, which has the property that this can be computed in time $O(d\log d)$), and then multiply by our sparse A. This corresponds to multiplying by a dense A in the standard basis, and the analysis works out (intuitively, as long as A is a bit random, and not sparse, then things are okay). This ends up with runtime $O(d \log d + \frac{polylogn}{\epsilon^2})$, which is much better than $O(d \log n/\epsilon^2)$ in the many cases when $\frac{\log n}{\epsilon^2} \gg \log d$. Feel free to check out the original paper [1] for the details.

2.2 Recent Advances: Stronger Johnson-Lindenstrauss Transformations

Recall the theorem we proved above, slightly restated:

Theorem 2. *Given any $\epsilon \in (0,1)$, and a set $X \subset \mathbb{R}^d$ with $|X| = n$, there exists a map $f : \mathbb{R}^d \to \mathbb{R}^m$ with $m = O(\frac{\log n}{\epsilon^2})$ such that for all $x, y \in X$,*

$$\|x - y\|_2 \le \|f(x) - f(y)\|_2 \le (1+\epsilon)\|x - y\|_2.$$

A sequence or recent works, culminating in the 2019 paper [5] established that an analog of the above theorem holds, where we replace "for all $x, y \in X$", with "for all $x \in X$ and for all $y \in \mathbb{R}^d$. Hence there is a mapping that preserves the distances from *any* point in $\mathbb{R}^d$ to any point in X. The construction of this mapping is much more complicated than the random projection that we analyzed above, and, in particular, the mapping is non-linear!

3 Intro to Nearest Neighbor Search

A useful primitive in many data analysis and machine learning algorithms is the ability to efficiently find similar data points to a given point of interest. For example, given a the text from a webpage, or an image, the ability to quickly figure out similar documents or images seems extremely useful. (Of course, one must make sure that the comparison metric and the feature space we are looking in is appropriate.) This problem is known as "nearest neighbor search".

Naively, given a set of n vectors in d dimensions, and a vector of interest, v, by simply computing the distance between v and each of the n vectors, one can clearly find v's nearest neighbor in time $O(nd)$. In the case that $d >> O(\log n)$, one can first apply the Johnson-Lindenstrauss transformation, and then any subsequent nearest neighbor search can be answered *approximately* in time $O(n \log n)$.

Could we hope for a significantly better runtime? Clearly we must consider each datapoint at some point, which would imply that we would need time at least $O(n)$; however, the hope is that there is a reasonable preprocessing step we could do, so as to enable us to perform subsequent nearest neighbor searches in *sublinear* time.

To give a slightly silly example, suppose the points are just real numbers ($d = 1$). We could preprocess the points by sorting them, and then given any number v, one can find its nearest neighbor in time $O(\log n)$ by performing a binary search in the sorted list. The analog of such a scheme in higher dimension suffers a "curse of dimensionality": the amount of space it would take to store such a lookup table scales exponentially with the dimension.

Nevertheless, if one is willing to tolerate some approximation—in the sense that we might only be able return the approximate closest vector (e.g. a vector whose distance is a factor of ϵ larger than that of the closest vector), then nontrivial schemes with interesting theoretical properties and good practical performance exist.

Locality sensitive hashing schemes, as their name suggests, are hashing schemes with the property that points that are close have a higher probability of hashing to the same bucket. Given such a scheme, a nearest-neighbor search can be performed by simply hashing the query vector v, and then checking which vectors are hashed to the same bucket. Since their introduction in the late 1990's in a paper of Indyk and Motwani (from Stanford) [3], there has been a huge body of research describing locality sensitive hashing schemes for various metrics, with various tradeoffs between the various parameters (storage space, preprocessing time, etc.). Despite the volume of research, many of the most basic questions are still open—we currently do not know if the schemes we have are near optimal or not. See [2] for a (only slightly outdated) survey.

We now explore an extremely simple locality sensitive hashing scheme, based on the Johnson-Lindenstrauss transformation, that illustrates some of the properties and intuitions of more complex schemes:

Algorithm 1. RANDOM HYPERPLANE HASHING
Input: n points in d dimensions, integer s representing the number of hash tables we will construct, and an integer k representing the length of each hash.

- *Pick s matrices of dimension $k \times d$, denoted $A_1 \ldots, A_s$, by drawing each entry of A_i independently from $N(0,1)$.*
- *For every point x, hash it to each of the s hash tables as follows: for $i \in [1, \ldots, s]$, set x's ith hash to be the length k vector $sign(A_i v)$ whose jth index is ± 1 according to the sign of the jth coordinate of the vector $A_i v$.*

First we argue that the points that get hashed to the same bucket as x will tend to have a small angle with x:

Claim 2. *For $x, y \in \mathbb{R}^d$, the probability that they are hashed to the same bucket in the ith hash table is $(1 - \frac{angle(x,y)}{\pi})^k$, where $angle(x,y)$ denotes the angle, in radians between the vectors x and y.*

Proof. Consider the jth index of the hash of x and y. The entries corresponding to x and y will agree if, and only if points x and y lie on the same side of the hyperplane defined by the the jth row of matrix A. Because of the spherical symmetry of the d dimensional Gaussian, the probability that this random hyperplane splits the points x and y is exactly equal to the probability that a random line in the 2-dimensional plane spanned by x, y, passing through the origin, splits points x and y. This probability is exactly $angle(x,y)/\pi$. The claim now follows from the independence of the k coordinates. □

Hence for $x, y \in \mathbb{R}^d$, the probability that they are not hashed to the same bucket in any of the s hash tables is approximately $(1-(1-\frac{angle(x,y)}{\pi})^k)^s \approx e^{-s \cdot e^{-k \cdot angle(x,y)/\pi}}$. To see the implications of this statement, consider setting $k = \frac{\pi \log n}{2\epsilon}$, and $s = \sqrt{n}$. For this setting of parameters, if $angle(x,y) \leq \epsilon$, then the probability that they will hash to the same bucket in at least one of the hash tables is roughly $1 - e^{-s \cdot e^{-k \cdot angle(x,y)/\pi}} \geq 1 - e^{-s/\sqrt{n}} = 1 - 1/e > 1/2$. On the other hand, if $angle(x,y) \geq 5\epsilon$, then the probability that they will hash to the same bucket in at least one of the hash tables is roughly $1 - e^{-s \cdot e^{-k \cdot angle(x,y)/\pi}} \leq 1 - e^{-sn^{5/2}} \approx 1/n^2$. Hence, if we have hashed n points, via a union bound, with constant probability, *no* pair x, y with $angle(x,y) > 5\epsilon$ will collide in any of the s hash tables.

The above reasoning shows that this hashing approach will allow us to construct a set of hash tables with the following properties: 1) given a point $x \in \mathbb{R}^d$, it takes time $O(\frac{d\sqrt{n}\log n}{\epsilon})$ to hash x, and 2) if there exists y with $angle(x,y) < \epsilon$, then with constant probability, by checking each of the

$\sqrt{n}$ hashes, we will find a y' s.t. $angle(x, y') \leq 5\epsilon$. While this factor of 5 slop might not be ideal, we get a sublinear dependence on n in the computation time—which is a big deal in typical settings for which $d \approx O(\log n)$.

References

[1] Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009.

[2] A. Andoni and P. Indyk. near-optimal hashing algorithms for approximate nearest neighbors in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[3] P. Indyk and R. Motwani. approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

[4] W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[5] Shyam Narayanan and Jelani Nelson. Optimal terminal dimensionality reduction in euclidean space. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1064–1069. ACM, 2019.