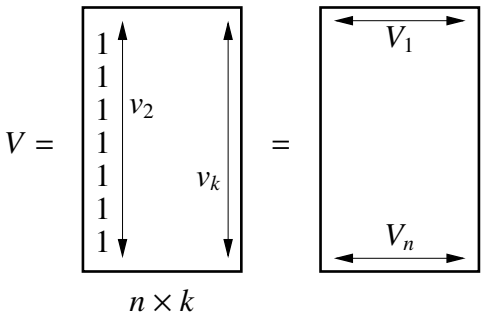
23 Multiple Eigenvectors; Random Projection; Latent Factor Analysis

Clustering w/Multiple Eigenvectors

[When we use the Fiedler vector for spectral graph clustering, it tells us how to divide a graph into two graphs. If we want more than two clusters, we can use divisive clustering: we repeatedly cut the subgraphs into smaller subgraphs by computing their Fiedler vectors. However, there are several other methods to subdivide a graph into k clusters in one shot that use multiple eigenvectors rather than just the Fiedler vector v_2. These methods are usually faster and sometimes give better results. They use k eigenvectors in a natural way to cluster a graph into k subgraphs.]

For k clusters, compute first k eigenvectors $v_1 = \mathbf{1}, v_2, \ldots, v_k$ of generalized eigensystem $Lv = \lambda Mv$.

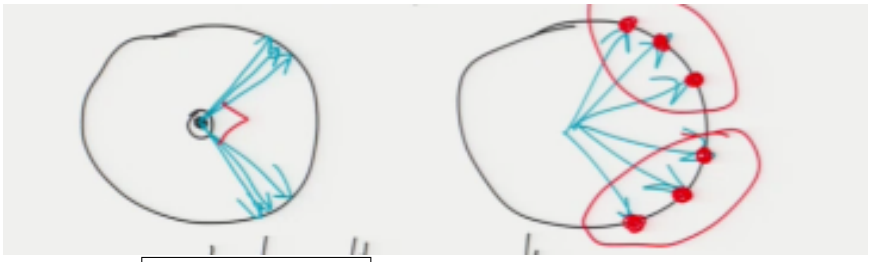


[V's columns are the eigenvectors with the k smallest eigenvalues.]
[Yes, we do include the all-1's vector v_1 as one of the columns of V.]

[Draw this by hand. eigenvectors.pdf]

Row V_i is spectral vector [my name] for vertex i. [The rows are vectors in a k-dimensional space I'll call the "spectral space." When we were using just one eigenvector, it made sense to cluster vertices together if their components were close together. When we use more than one eigenvector, it turns out that it makes sense to cluster vertices together if their spectral vectors point in similar directions.]

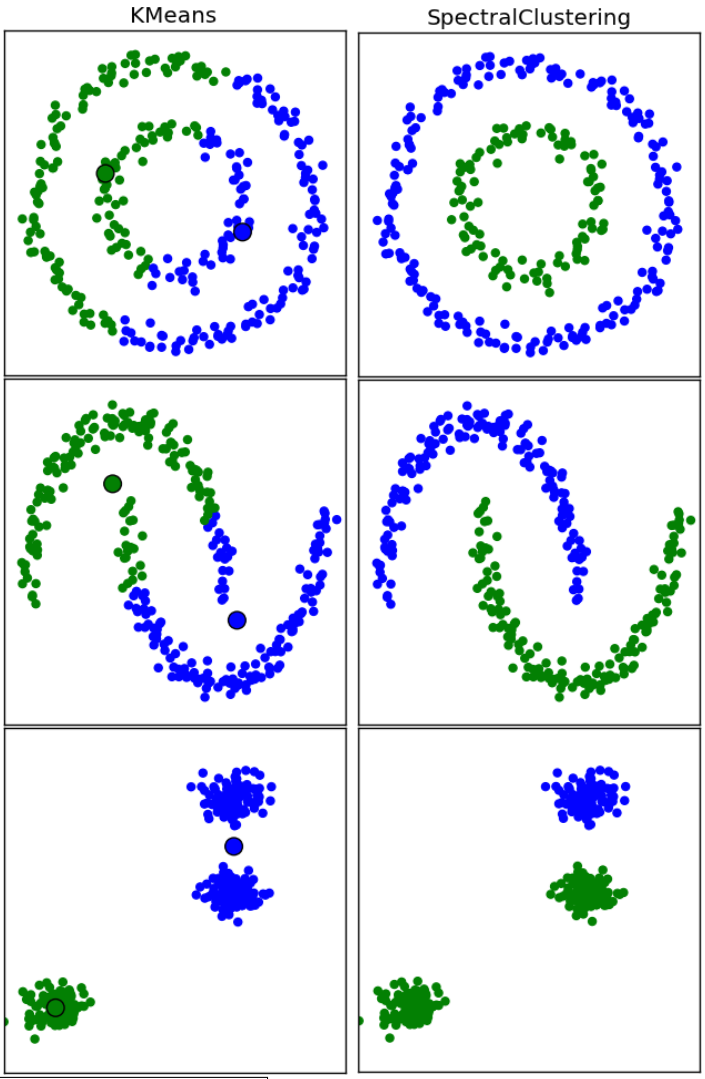
Normalize each row V_i to unit length.
[Now you can think of the spectral vectors as points on a unit sphere centered at the origin.]



[Draw this by hand vectorclusters.png] [A 2D example showing two clusters on a circle. If the graph has k components, the points in each cluster will have identical spectral vectors that are exactly orthogonal to all the other components' spectral vectors (left). If we modify the graph by connecting these components with small-weight edges, we get vectors more like those at right—not exactly orthogonal, but still tending toward distinct clusters.]

k-means cluster these vectors.

[Because all the spectral vectors lie on the sphere, k-means clustering will cluster together vectors that are separated by small angles.]



compkmeans.png, compspectral.png [Comparison of point sets clustered by k-means—just k-means by itself, that is—vs. a spectral method. To create a graph for the spectral method, we use an exponentially decaying function to assign weights to pairs of points, like we used for image segmentation but without the brightnesses.]

Invented by [our own] Prof. Michael Jordan, Andrew Ng [when he was still a student at Berkeley], Yair Weiss.

[This wasn't the first algorithm to use multiple eigenvectors for spectral clustering, but it has become one of the most popular.]

RANDOM PROJECTION

A cheap alternative to PCA as preprocess for clustering, classification, regression.
Approximately preserves distances between points!

[We project onto a random subspace instead of the "best" subspace, but take a fraction of the time of PCA. It works best when you project a very high-dimensional space to a medium-dimensional space. Because we roughly preserve the distances, algorithms like k-means clustering and nearest neighbor classifiers will give similar results to what they would give in high dimensions, but they run much faster.]

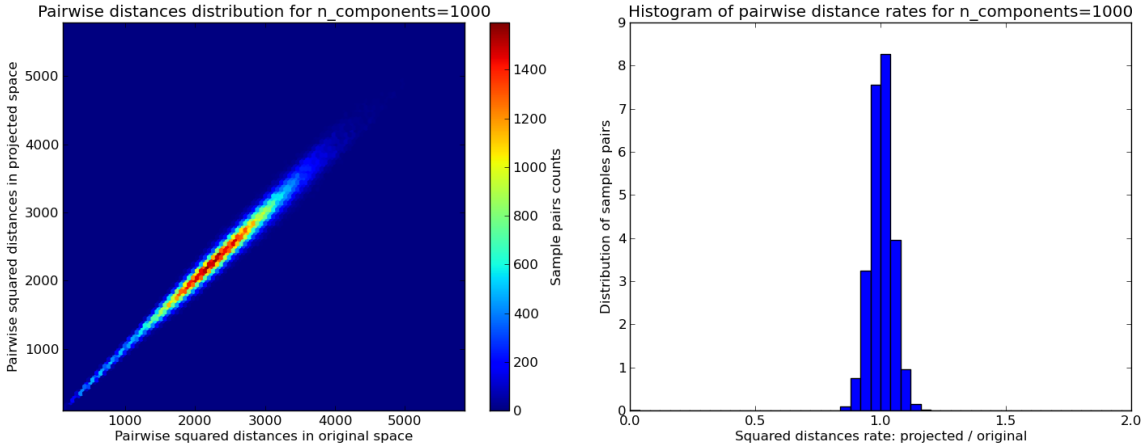
Pick a small ϵ, a small δ, and a random subspace $S \subset \mathbb{R}^d$ of dimension k, where $k = \dfrac{2\ln(1/\delta)}{\epsilon^2/2 - \epsilon^3/3}$.

For any pt q, let $\hat{q}$ be orthogonal projection of q onto S, multiplied by $\sqrt{\frac{d}{k}}$.

[The multiplication by $\sqrt{d/k}$ helps preserve the distances between points after you project.]

Johnson–Lindenstrauss Lemma (modified):
For any two pts $q, w \in \mathbb{R}^d$, $(1-\epsilon)|qw|^2 \le |\hat{q}\hat{w}|^2 \le (1+\epsilon)|qw|^2$ with probability $\ge 1 - 2\delta$.
Typical values: $\epsilon \in [0.02, 0.5]$, $\delta \in [1/n^3, 0.05]$.

[So the distance between two points after projecting might change by 2 to 50%. In practice, you can experiment with k to find the best speed-accuracy tradeoff. If you want most inter-point distances to be accurate, you should set δ smaller than $1/n^2$, so you need a subspace of dimension $\Theta(\log n)$. Reducing δ doesn't cost much, but reducing ϵ costs more. You can bring 1,000,000 sample points down to a 10,000-dimensional space with a 10% error in the distances.]
[What is remarkable about this result is that the dimension d of the input points doesn't matter.]



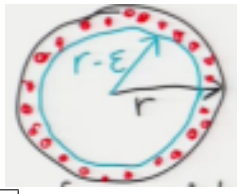
100000to1000.pdf [Comparison of inter-point distances before and after projecting points in 100,000-dimensional space down to 1,000 dimensions.]

[Why does this work? A random projection of a vector is like taking a random vector and selecting k components. The mean of the squares of those k components approximates the mean for the whole population.]

[How do you get a uniformly distributed random projection direction? You can choose each component from a univariate Gaussian distribution, then normalize the vector to unit length. How do you get a random subspace? You can choose k random vectors, then use Gram–Schmidt orthogonalization to make them mutually orthonormal. Interestingly, Indyk and Motwani show that if you skip the expensive normalization and Gram–Schmidt steps, random projection still works almost as well, because random vectors in a high-dimensional space are nearly equal in length and nearly orthogonal to each other with high probability.]

THE GEOMETRY OF HIGH-DIMENSIONAL SPACES

Consider shell between spheres of radii r & $r - \epsilon$.



[Draw this by hand concentric.png] [Concentric balls. In high dimensions, almost every point chosen uniformly at random in the outer ball lies outside the inner ball.]

Volume of outer ball $\propto r^d$
Volume of inner ball $\propto (r - \epsilon)^d$
Ratio of inner ball volume to outer =

$$\frac{(r-\epsilon)^d}{r^d} = \left(1 - \frac{\epsilon}{r}\right)^d \approx \exp\left(-\frac{\epsilon d}{r}\right) \qquad \text{which is small for large } d.$$

E.g. if $\frac{\epsilon}{r} = 0.1$ & $d = 100,$ inner ball has $90\%^{100} = 0.0027\%$ of volume.

Random points from uniform distribution in ball: nearly all are in outer shell.
" " " Gaussian " : nearly all are in some shell.

[If the dimension is very high, the majority of the random points generated from an isotropic Gaussian distribution are approximately at the same distance from the center. So they lie in a thin shell. Why? Consider a d-dimensional normal distribution with mean zero. By Pythagoras' Theorem, the squared distance from a random point P to the mean is]

$$||P||^2 = P_1^2 + P_2^2 + \ldots + P_d^2.$$

[Each component P_i is sampled independently from a univariate normal distribution with mean zero. When you add d independent random numbers, you scale the mean by d but you scale the standard deviation by only $\sqrt{d}$.]

$$\begin{aligned} E[||P||^2] &= d\,E[P_1^2]. \\ \text{std}(||P||^2) &= \sqrt{d}\,\text{std}(P_1^2). \end{aligned}$$

[So when d is large, the distance from P to the mean is concentrated in a narrow shell whose radius is proportional to $\sqrt{d}$ with a standard deviation proportional to $\sqrt[4]{d}$.]
[The principle here is that when you take the mean of a very large sample, you get a very accurate estimate of the population mean. When you sample one point from a high-dimensional normal distribution, it's like sampling d different scalars from one-dimensional normal distributions.]

Lessons:
- In high dimensions, sometimes nearest neighbor and farthest neighbor don't differ much.
- k-means clustering and nearest neighbor classifiers [and many other classifiers] are less effective for large d.

[Your head TA, Marc Khoury, has a nice short essay entitled "Counterintuitive Properties of High Dimensional Space", which you can read at]
https://marckhoury.github.io/counterintuitive-properties-of-high-dimensional-space/

LATENT FACTOR ANALYSIS [aka Latent Semantic Indexing]

[You can think of this as dimensionality reduction for matrices.]

Suppose X is a term-document matrix: [aka bag-of-words model]
row i represents document i; column j represents term j. [Term = word.]
[Term-document matrices are usually sparse, meaning most entries are zero.]
X_{ij} = occurrences of term j in doc i
better: log (1+ occurrences) [So frequent words don't dominate.]
[Better still is to weight the entries so rare words give big entries and common words like "the" give small entries. To do that, you need to know how frequently each word occurs in general. I'll omit the details, but this is the common practice.]

Recall SVD $X = UDV^\top = \sum_{i=1}^{d} \delta_i u_i v_i^\top$. Suppose $\delta_i \le \delta_j$ for $i \ge j$.

Unlike PCA, we usually don't center X.
For large δ_i, u_i and v_i represent a cluster of documents & terms.

- Large components in u_i mark docus using similar/related terms, i.e., a genre.
- " " " v_i mark frequent terms in that genre.
- E.g., u_1 might have large components for the romance novels,
- v_1 " " " " for terms "passion," "ravish," "bodice" ...

[... and δ_1 would give us an idea how much bigger the romance novel market is than the markets for every other genre of books.]

[v_1 and u_1 tell us that there is a large subset of books that tend to use the same large subset of words. We can read off the words by looking at the larger components of v_1, and we can read off the books by looking at the larger components of u_1.]

[The property of being a romance novel is an example of a latent factor. So is the property of being the sort of word used in romance novels. There's nothing in X that tells you explicitly that romance novels exist, but the similar vocabulary is a hidden connection between them that gives them a large singular value. The vector u_1 reveals which books have that genre, and v_1 reveals which words are emphasized in that genre.]

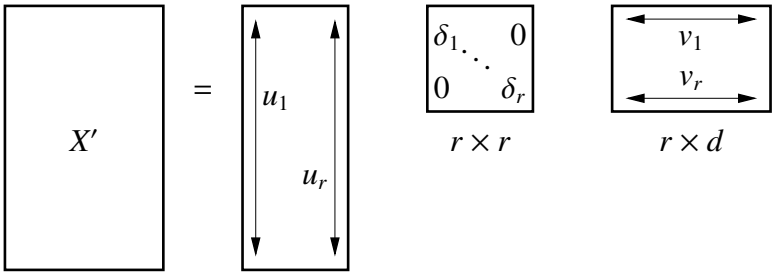
Like clustering, but clusters overlap: if u_1 picks out romances &
u_2 picks out histories, they both pick out historical romances.

[So you can think of latent factor analysis as a sort of clustering that permits clusters to overlap. Another way in which it differs from traditional clustering is that the u-vectors contain real numbers, and so some points have stronger cluster membership than others. One book might be just a bit romance, another a lot.]

Application in market research:
identifying consumer types (hipster, suburban mom) & items bought together.
[For applications like this, the first few singular vectors are the most useful. Most of the singular vectors are mostly noise, and they have small singular values to tell you so. This motivates approximating a matrix by using only some of its singular vectors.]

Truncated sum $X' = \sum_{i=1}^{r} \delta_i u_i v_i^\top$ is a low-rank approximation of X, of rank r. [Assuming $\delta_r > 0$.]

[We choose the singular vectors with the largest singular values, because they carry the most information.]



[Draw this by hand. truncate.pdf]

X' is the rank-r matrix that minimizes the [squared] Frobenius norm

$$\|X - X'\|_F^2 = \sum_{i,j} \left(X_{ij} - X'_{ij}\right)^2$$

Applications:

- Fuzzy search. [Suppose you want to find a document about gasoline prices, but the document you want doesn't have the word "gasoline"; it has the word "petrol." One cool thing about the reduced-rank matrix X' is that it will probably associate that document with "gasoline," because the SVD tends to group synonyms together.]
- Denoising. [The idea is to assume that X is a noisy measurement of some unknown matrix that probably has low rank. If that assumption is partly true, then the reduced-rank matrix X' might be better than the input X.]
- Matrix compression. [As you can see above, if we use a low-rank approximation with a small rank r, we can express the approximate matrix as an SVD that takes up much less space than the original matrix. Often this row-rank approximation supports faster matrix computations.]
- Collaborative filtering: fills in unknown values, e.g. user ratings.
  [Suppose the rows of X represents Netflix users and the columns represent movies. The entry X_{ij} is the review score that user i gave to movie j. But most users haven't reviewed most movies. We want to fill in the missing values. Just as the rank reduction will associate "petrol" with "gasoline," it will tend to associate users with similar tastes in movies, so the reduced-rank matrix X' can predict ratings for users who didn't supply any.]