20 Unsupervised Learning and Principal Components Analysis (PCA)

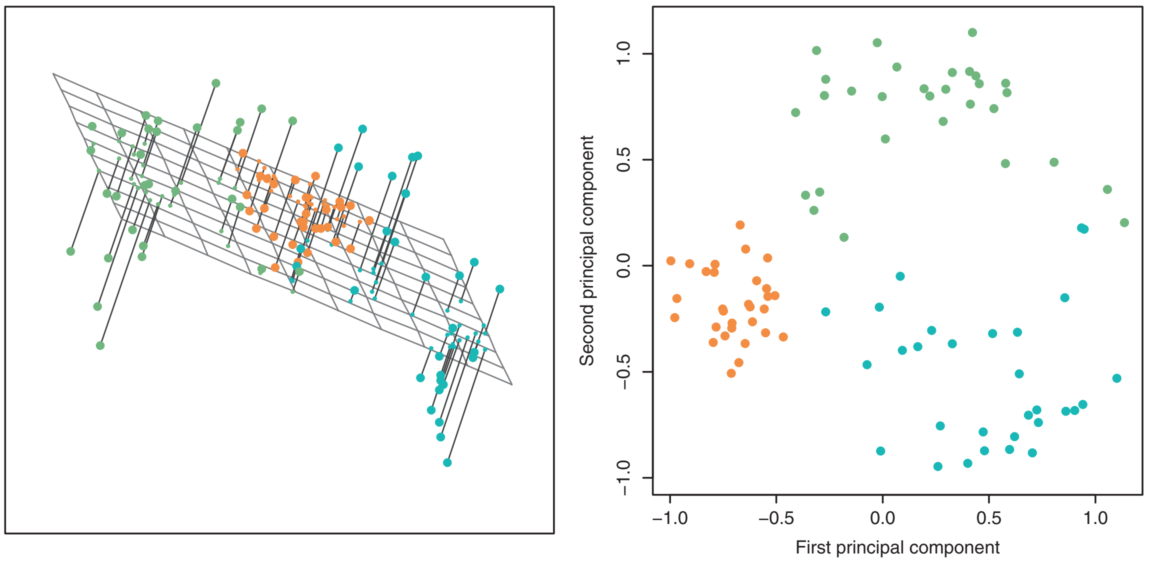
UNSUPERVISED LEARNING

We have sample points, but no labels!
No classes, no y-values, nothing to predict.
Goal: Discover structure in the data.

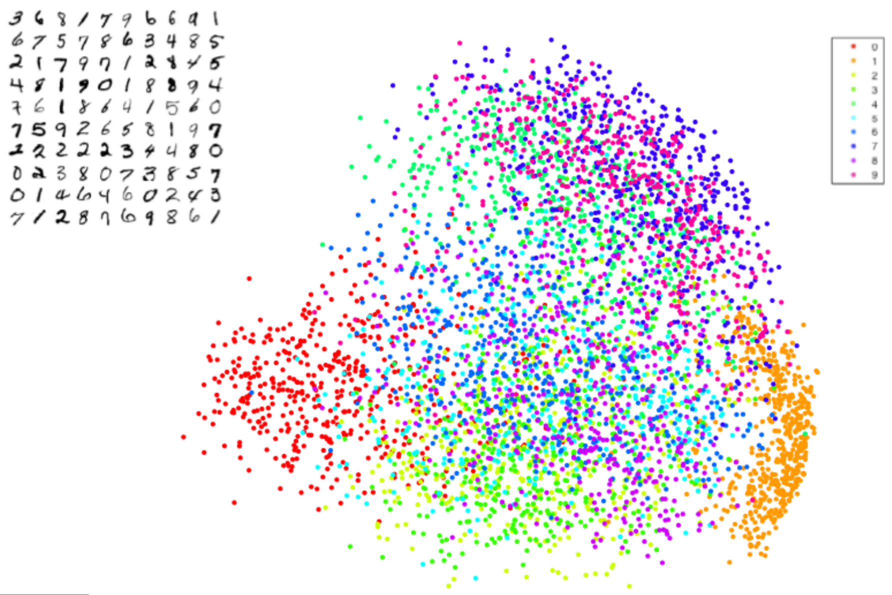
Examples:
- Clustering: partition data into groups of similar/nearby points.
- Dimensionality reduction: data often lies near a low-dimensional subspace (or manifold) in feature space; matrices have low-rank approximations.
  [Whereas clustering is about grouping similar sample points, dimensionality reduction is more about identifying a continuous variation from sample point to sample point.]
- Density estimation: fit a continuous distribution to discrete data.
  [When we use maximum likelihood estimation to fit Gaussians to sample points, that's density estimation, but we can also fit functions more complicated than Gaussians, with more local variation.]

PRINCIPAL COMPONENTS ANALYSIS (PCA) (Karl Pearson, 1901)

Goal: Given sample points in $\mathbb{R}^d$, find k directions that capture most of the variation. (Dimensionality reduction.)



3dpca.pdf [Example of 3D points projected to 2D by PCA.]



$\boxed{\text{pcadigits.pdf}}$ [The (high-dimensional) MNIST digits projected to 2D. Two dimensions aren't enough to fully separate the digits, but observe that the digits 0 (red) and 1 (orange) are well on their way to being separated.]

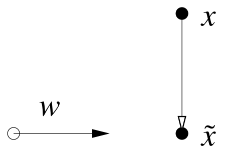
Why?

- Find a small basis for representing variations in complex things, e.g. faces, genes.
- Reducing # of dimensions makes some computations cheaper, e.g. regression.
- Remove irrelevant dimensions to reduce overfitting in learning algs.
  Like subset selection, but the "features" aren't axis-aligned;
  they're linear combos of input features.

[Sometimes PCA is used as a preprocess before regression or classification for the last two reasons.]

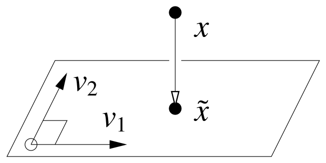
Let X be $n \times d$ design matrix. [No fictitious dimension.]
From now on, assume X is centered: mean X_i is zero.
[As usual, we can center the data by computing the mean x-value, then subtracting the mean from each sample point.]

[Let's start by seeing what happens if we pick just one principal direction.]
Let w be a unit vector.
The $\underline{\text{orthogonal projection}}$ of point x onto vector w is $\tilde{x} = (x \cdot w)\, w$
If w not unit, $\tilde{x} = \frac{x \cdot w}{|w|^2} w$



[The idea is that we're going to pick the best direction w, then project all the data down onto w so we can analyze it in a one-dimensional space. Of course, we lose a lot of information when we project down from d dimensions to just one. So, suppose we pick several directions. Those directions span a subspace, and we want to project points orthogonally onto the subspace. This is easy *if* the directions are orthogonal to each other.]

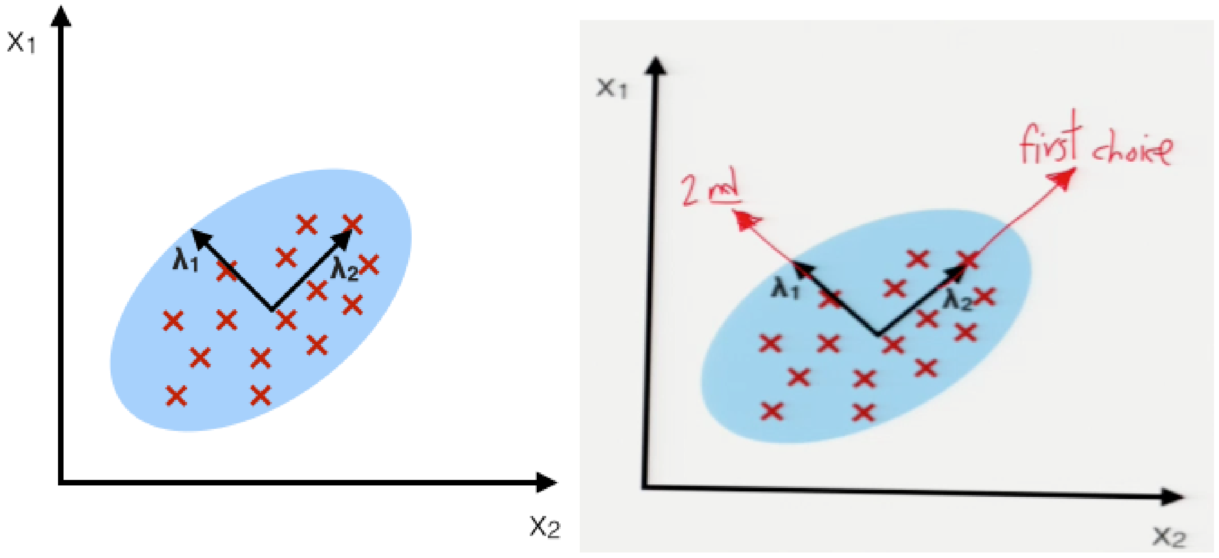
Given orthonormal directions $v_1, \ldots, v_k$, $\tilde{x} = \sum_{i=1}^{k} (x \cdot v_i)\, v_i$
[The word "orthonormal" implies they're mutually orthogonal and length 1.]



[Usually we don't actually want the projected point in $\mathbb{R}^d$; usually we want the principle coordinates $x \cdot v_i$ in principal components space.]

$X^\top X$ is square, symmetric, positive semidefinite, $d \times d$ matrix. [As it's symmetric, its eigenvalues are real.]
Let $0 \le \lambda_1 \le \lambda_2 \le \ldots \le \lambda_d$ be its eigenvalues. [sorted]
Let $v_1, v_2, \ldots, v_d$ be corresponding orthogonal **unit** eigenvectors.
[It turns out that the principal directions will be these eigenvectors, and the most important ones will be the ones with the greatest eigenvalues. I will show you this in three different ways.]

PCA derivation 1: Fit a Gaussian to data with maximum likelihood estimation.
Choose k Gaussian axes of greatest variance.

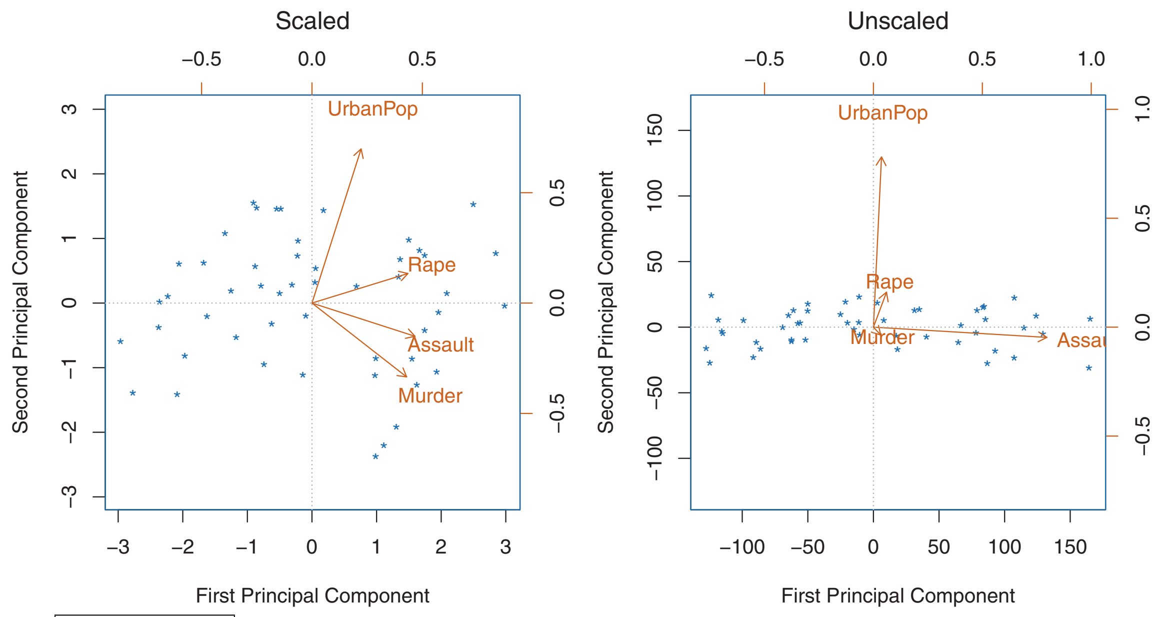


gaussfitpca.png [A Gaussian fitted to sample points.]

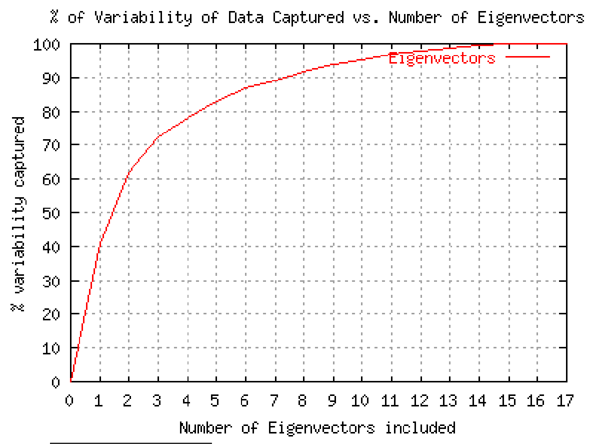
Recall that MLE estimates a covariance matrix $\hat{\Sigma} = \frac{1}{n} X^\top X$. [Presuming X is centered.]

PCA Alg:
- Center X.
- Optional: Normalize X. Units of measurement different?
  - Yes: Normalize.
    [Bad for principal components to depend on arbitrary choice of scaling.]
  - No: Usually don't.
    [If several features have the same unit of measurement, but some of them have smaller variance than others, that difference is usually meaningful.]
- Compute unit eigenvectors/values of $X^\top X$.

– Optional: choose k based on the eigenvalue sizes.
– For the best k-dimensional subspace, pick eigenvectors $v_{d-k+1}, \ldots, v_d$.
– Compute the coordinates $x \cdot v_i$ of training/test data in principal components space.
[When we do this projection, we have two choices: we can un-center the input training data before projecting it, OR we can translate the test data by the same vector we used to translate the training data when we centered it.]



normalize.pdf [Normalized data at left; unnormalized data at right. The arrows show the original axes projected on the top two principal components. When the data are not normalized, rare occurrences like murder have little influence on the principal directions. Which is better? It depends on whether you think that low-frequency events like murder and rape should have a disproportionate influence.]

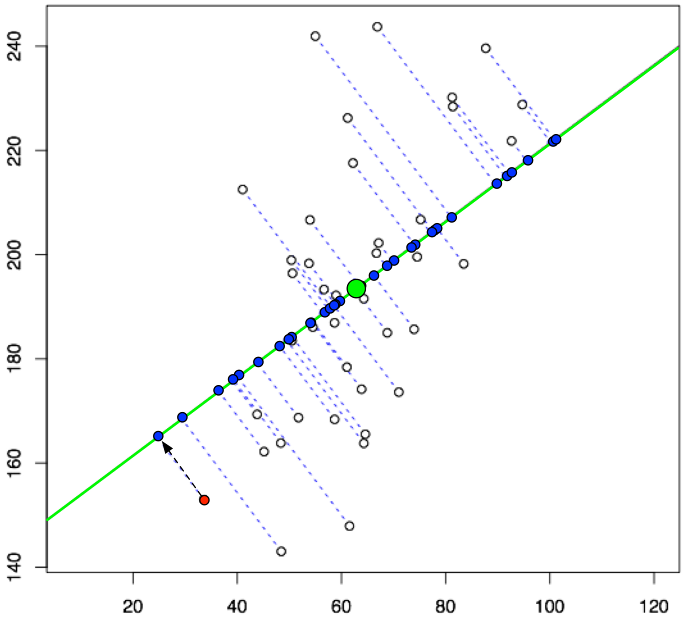


$$\boxed{r_k = \sum_{i=1}^{k} \lambda_i^2 \Big/ \sum_{i=1}^{n} \lambda_i^2}$$

variance.pdf [Plot of # of eigenvectors vs. percentage of variance captured . In this example, just 3 eigenvectors capture 70% of the variance.]

[If you are using PCA as a preprocess for a supervised learning algorithm, there's a more effective way to choose k: (cross-)validation.]

PCA derivation 2: Find direction w that maximizes variance of projected data
[In other words, when we project the data down, we don't want it all to bunch up; we want to keep it as spread out as possible.]

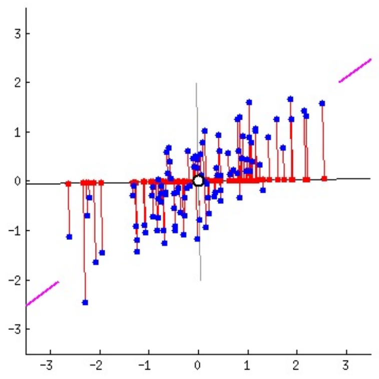


$\boxed{\text{project.jpg}}$ [Points projected on a line. We wish to choose the orientation of the green line to maximize the variance of the blue points.]

$$\boxed{\text{Maximize } \operatorname{Var}(\{\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n\}) = \frac{1}{n}\sum_{i=1}^{n}\left(X_i \cdot \frac{w}{|w|}\right)^2 = \frac{1}{n}\frac{|Xw|^2}{|w|^2} = \frac{1}{n}\underbrace{\frac{w^\top X^\top X w}{w^\top w}}_{\underline{\text{Rayleigh quotient}} \text{ of } X^\top X \text{ and } w}}$$

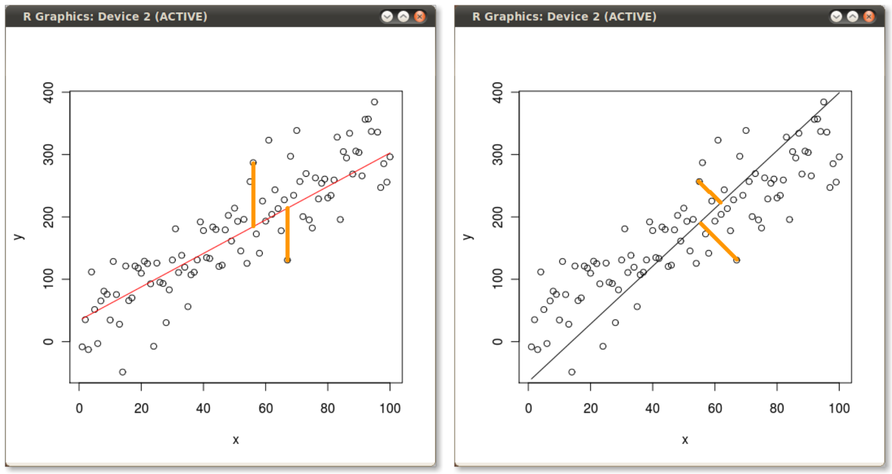
[This fraction is a well-known construction called the Rayleigh quotient. When you see it, you should smell eigenvectors nearby. How do we maximize this?]
If w is an eigenvector v_i, Ray. quo. = λ_i
$\rightarrow$ of all eigenvectors, v_d achieves maximum variance λ_d/n.
One can show v_d beats every other vector too.
[Because every vector w is a linear combination of eigenvectors, and so its Rayleigh quotient will be a convex combination of eigenvalues. It's easy to prove this, but I don't have the time. For the proof, look up "Rayleigh quotient" in Wikipedia.]
[So the top eigenvector gives us the best direction. But we typically want k directions. After we've picked one direction, then we have to pick a direction that's orthogonal to the best direction. But subject to that constraint, we again pick the direction that maximizes the variance.]
What if we constrain w to be orthogonal to v_d? Then v_{d-1} is optimal.
[And if we need a third direction orthogonal to v_d and v_{d-1}, the optimal choice is v_{d-2}. And so on.]

PCA derivation 3: Find direction w that minimizes "projection error"



PCAanimation.gif [This is an animated GIF; unfortunately, the animation can't be included in the PDF lecture notes. Find the direction of the black line for which the sum of squares of the lengths of the red lines is smallest.]

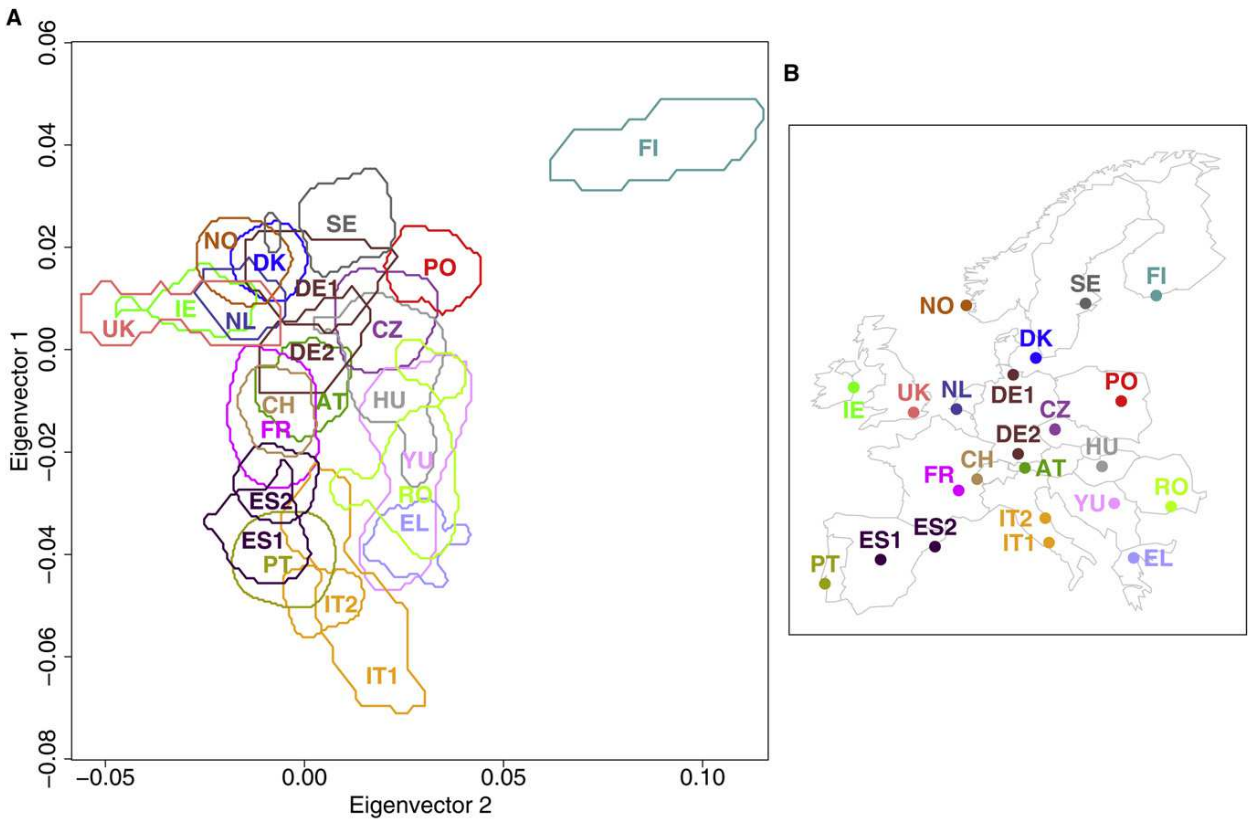
[You can think of this as a sort of least-squares linear regression, with one subtle but important change. Instead of measuring the error in a fixed vertical direction, we're measuring the error in a direction orthogonal to the principal component direction we choose.]



mylsq.png, mypca.png [Least-squares linear regression vs. PCA. In linear regression, the projection direction is always vertical; whereas in PCA, the projection direction is orthogonal to the projection hyperplane. In both methods, however, we minimize the sum of the squares of the projection distances.]

$$\boxed{\begin{aligned}\text{Minimize } \sum_{i=1}^{n} \left|X_i - \tilde{X}_i\right|^2 &= \sum_{i=1}^{n} \left|X_i - \frac{X_i \cdot w}{|w|^2} w\right|^2 = \sum_{i=1}^{n} \left(|X_i|^2 - \left(X_i \cdot \frac{w}{|w|}\right)^2\right) \\ &= \text{constant} - n\,(\text{variance from derivation 2}).\end{aligned}}$$

Minimizing projection error = maximizing variance.
[From this point, we carry on with the same reasoning as derivation 2.]



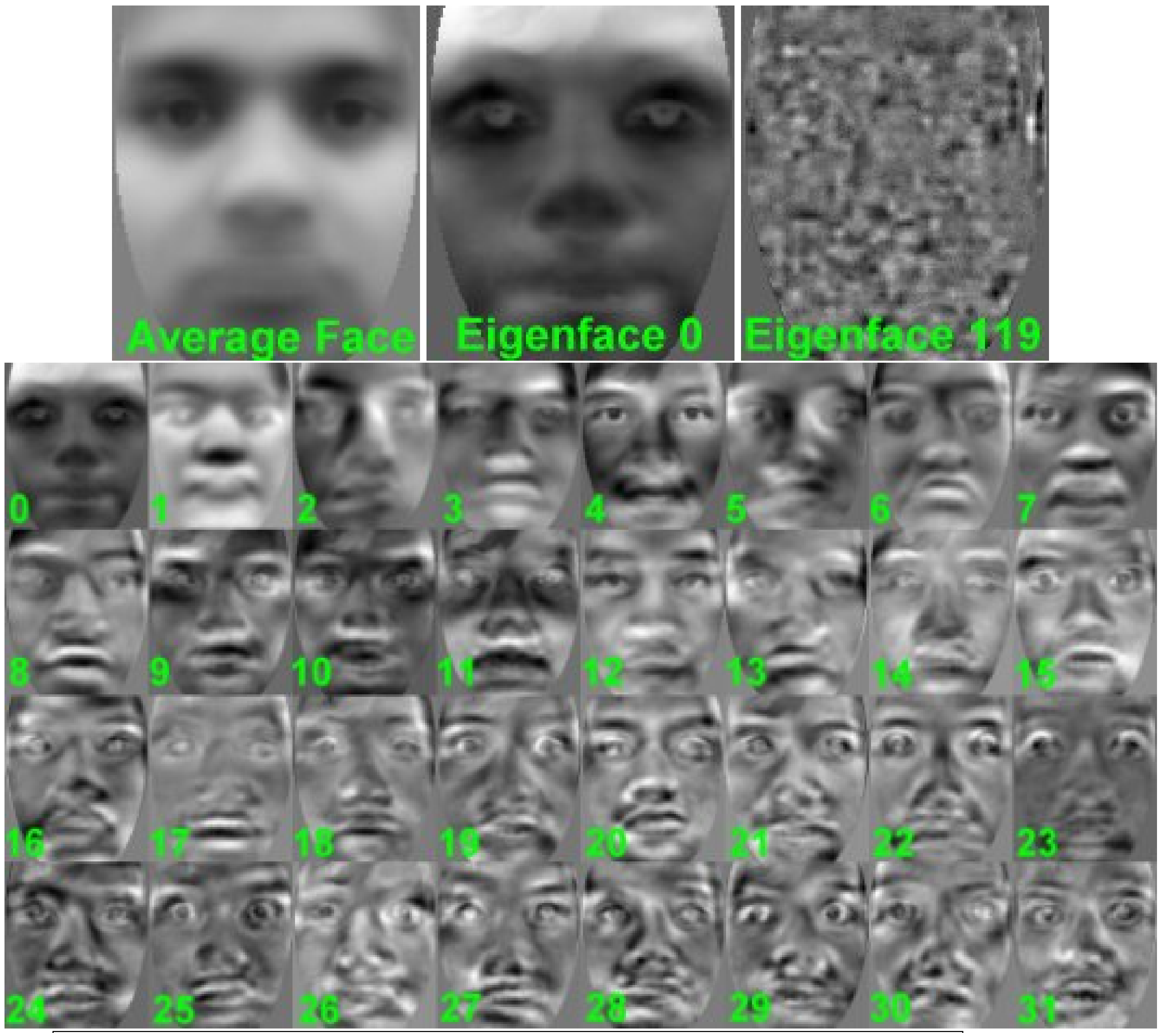
europegenetics.pdf (Lao et al., Current Biology, 2008.) [Illustration of the first two principal components of the single nucleotide polymorphism (SNP) matrix for the genes of various Europeans. The input matrix has 2,541 people from these locations in Europe (right), and 309,790 SNPs. Each SNP is binary, so think of it as 309,790 dimensions of zero or one. The output (left) shows spots on the first two principal components where the projected people from a particular national type are denser than a certain threshold. What's amazing about this is how closely the projected genotypes resemble the geography of Europe.]

Eigenfaces

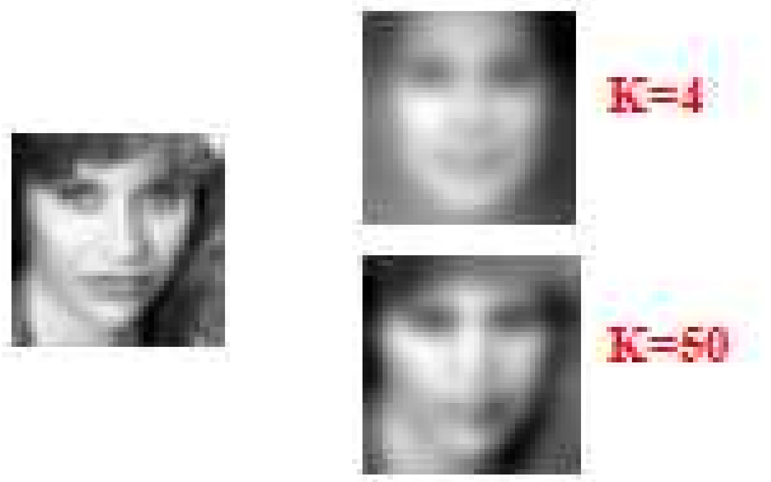
X contains n images of faces, d pixels each.
[If we have a 200×200 image of a face, we represent it as a vector of length 40,000, the same way we represent the MNIST digit data.]
Face recognition: Given a query face, compare it to all training faces; find nearest neighbor in $\mathbb{R}^d$.
[This works best if you have several training photos of each person you want to recognize, with different lighting and different facial expressions.]

Problem: Each query takes $\Theta(nd)$ time.
Solution: Run PCA on faces. Reduce to much smaller dimension d'.
Now nearest neighbor takes $O(nd')$ time.
[Possibly even less. We'll talk about speeding up nearest-neighbor search at the end of the semester. If the dimension is small enough, you can sometimes do better than linear time.]

[If you have 500 stored faces with 40,000 pixels each, and you reduce them to 40 principal components, then each query face requires you to read 20,000 stored principle coordinates instead of 20 million pixels.]

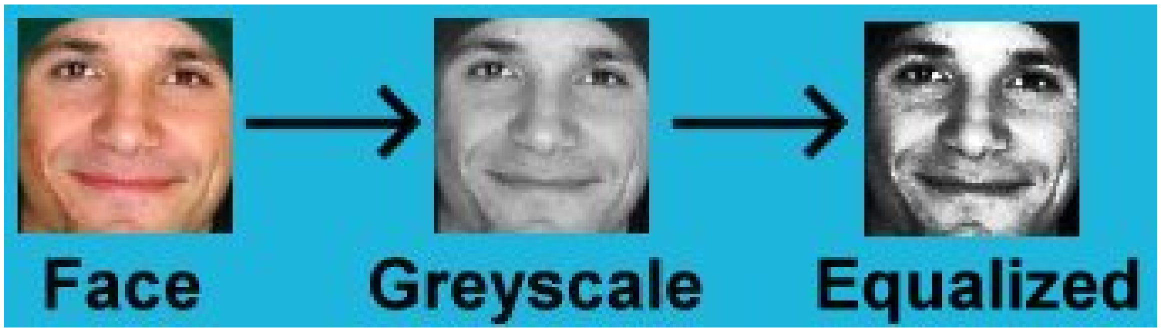


facerecaverage.jpg, facereceigen0.jpg, facereceigen119.jpg, facereceigen.jpg [Images of the the eigenfaces. The "average face" is the mean used to center the data.]



eigenfaceproject.pdf [Images of a face (left) projected onto the first 4 and 50 eigenvectors, with the average face added back. These last image is blurry but good enough for face recognition.]

For best results, equalize the intensity distributions first.



facerecequalize.jpg [Image equalization.]

[Eigenfaces are not perfect. They encode both face shape *and* lighting. Ideally, we would have some way to factor out lighting and analyze face shape only, but that's harder. Some people say that the first 3 eigenfaces are usually all about lighting, and you sometimes get better facial recognition by dropping the first 3 eigenfaces.]

[Show Blanz–Vetter face morphing video (morphmod.mpg).]

[Blanz and Vetter use PCA in a more sophisticated way for 3D face modeling. They take 3D scans of people's faces and find correspondences between peoples' faces and an idealized model. For instance, they identify the tip of your nose, the corners of your mouth, and other facial features, which is something the original eigenface work did not do. Instead of feeding an array of pixels into PCA, they feed the 3D locations of various points on your face into PCA. This works more reliably.]