November 7, 2017

CS 361: Probability & Statistics

Multidimensional data

The transformation

Now if we write $\mathbf{u}_i = \mathbf{x}_i - \text{mean}(\{\mathbf{x}\})$ we will have a dataset with a mean of 0. And if we then transform that dataset with the matrix of ordered eigenvectors of {**x**}, U

$$\mathbf{n}_i = \mathcal{U}^T \mathbf{u}_i$$

We will have a transformation worth looking at. Now, the mean of the dataset is 0. And our result from a few slides ago about covariances, tells us that the covariance of this dataset is

$$\text{Covmat}(\{\mathbf{n}\}) = \mathcal{U}^T \text{Covmat}(\{\mathbf{x}\}) \mathcal{U}$$

But we just said on the last slide that

$$\text{Covmat}(\{\mathbf{x}\}) = \mathcal{U} \Lambda \mathcal{U}^T$$

Or

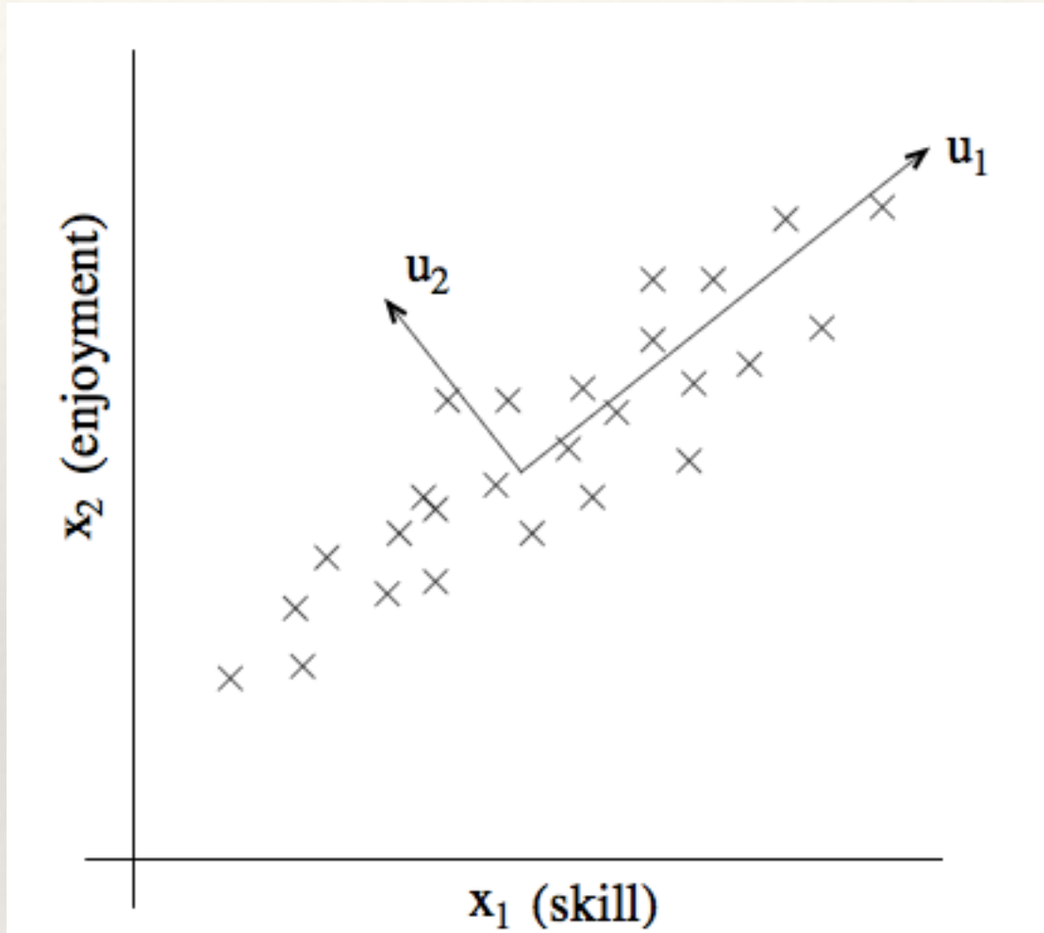
$$\text{Covmat}(\{\mathbf{n}\}) = \Lambda$$

How did we do?

We said we wanted a transformation.

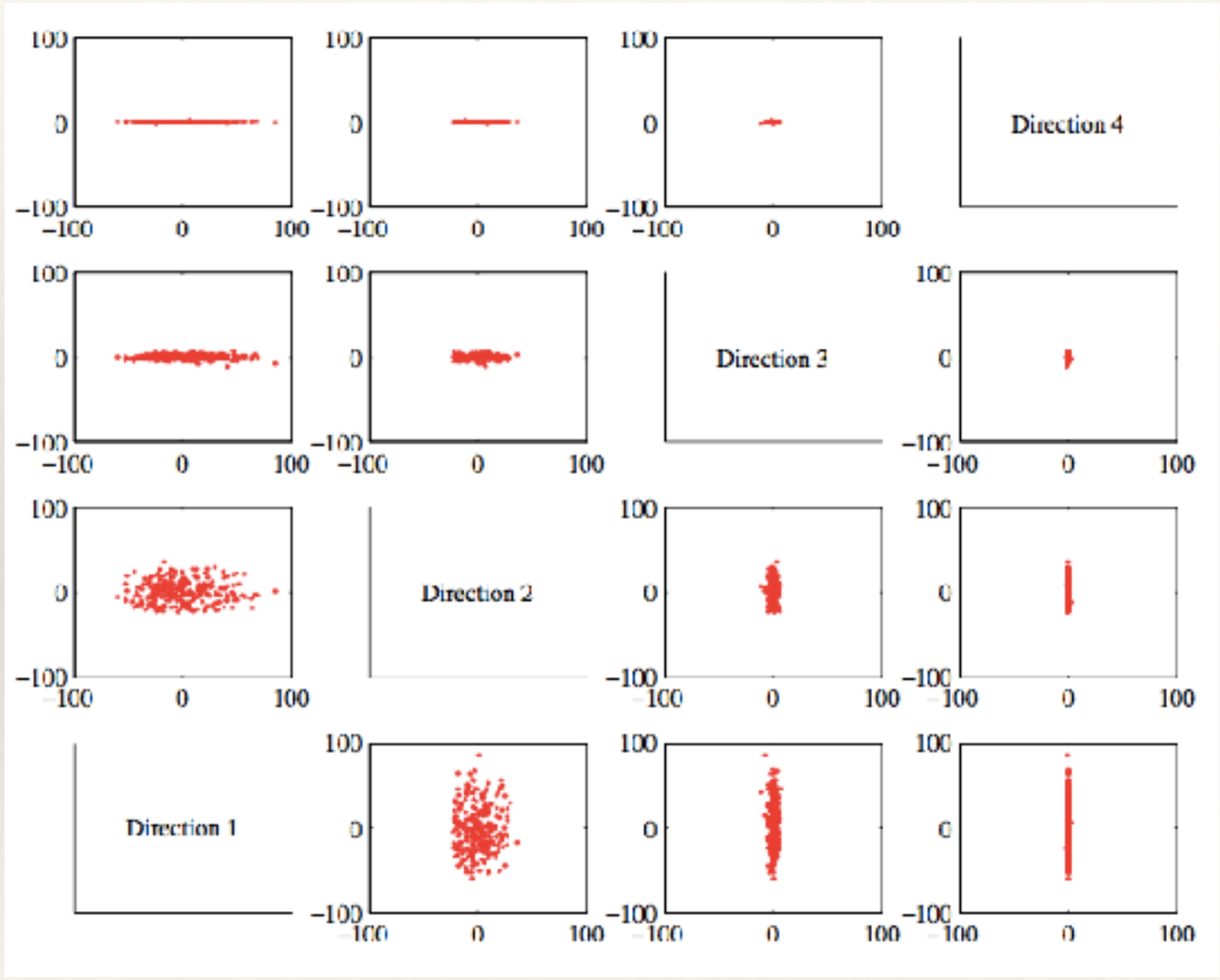
That will be a transformation which gives us data with 0 mean

That will be a transformation which gives us data with 0 covariance between dimensions



Source: Andrew Ng machine learning notes

The transformation



Example

This process will give us the ability to construct a new dataset by giving us a matrix of eigenvectors U and a matrix of eigenvalues. How should we interpret these matrices?

$$\text{Covmat}(\{\mathbf{x}\}) = \mathcal{U}\Lambda\mathcal{U}^T$$

Suppose we looked at data on countries in the world and the data had dimensions for population, area, GDP, infant mortality, and life expectancy. Suppose that we found the eigenvalues and eigenvectors of the covariance matrix of this data and got for the first two components

$$u_1 = \begin{bmatrix} 0.931 \\ -0.351 \\ -0.096 \\ -0.004 \\ -0.008 \end{bmatrix} \quad \Lambda_1 = 23973.32 \qquad u_2 = \begin{bmatrix} -0.338 \\ -0.931 \\ 0.125 \\ 0.030 \\ 0.041 \end{bmatrix} \quad \Lambda_2 = 16949.71$$

PCA dimensionality reduction

So we have created a new dataset with that is still d dimensional and has mean 0, uncorrelated dimensions, and dimensions that have decreasing variance

$$\mathbf{u}_i = \mathbf{x}_i - \text{mean}(\{\mathbf{x}\})$$

$$\mathbf{n}_i = \mathcal{U}^T \mathbf{u}_i$$

The variance in the first component is the largest eigenvalue of Covmat({x}), the variance in the second component is the second largest eigenvalue of Covmat({x}) and so-on

Which means we might get a good approximation of the dataset in only *r* dimensions if we only keep around the first *r* components of **n**. These are the components with the highest variance in our transformation, after all

PCA dimensionality reduction

We will call this representation of the data the rank-r PCA projection of the data

If we write Π_r for the *dxd* matrix which is the identity matrix, keeping only the r upper left most 1s and zeroing out the lower right part of the diagonal, then the rank-r PCA projection of data point x_i is given as

$$\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) = \Pi_r \mathcal{U}^T (\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))$$

Translate the data

PCA dimensionality reduction

We will call this representation of the data the rank-r PCA projection of the data

If we write Π_r for the *dxd* matrix which is the identity matrix, keeping only the r upper left most 1s and zeroing out the lower right part of the diagonal, then the rank-r PCA projection of data point x_i is given as

$$\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) = \Pi_r \mathcal{U}^T \left(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right)\right)$$

Rotate the data

PCA dimensionality reduction

We will call this representation of the data the rank-r PCA projection of the data

If we write Π_r for the *dxd* matrix which is the identity matrix, keeping only the r upper left most 1s and zeroing out the lower right part of the diagonal, then the rank-r PCA projection of data point x_i is given as

$$\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) = \Pi_r \mathcal{U}^T \left(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right)\right)$$

Truncate after first r components

PCA dimensionality reduction

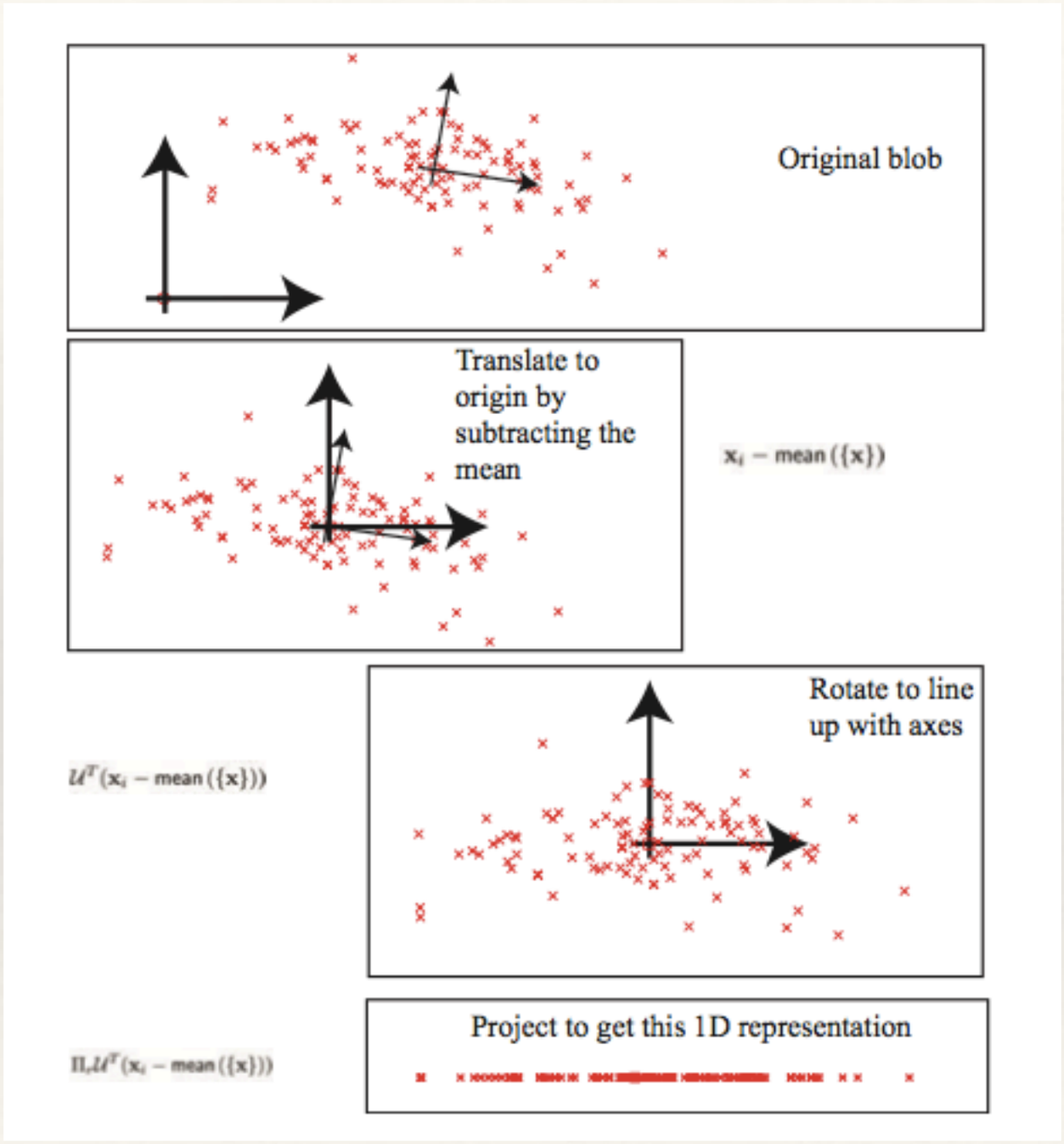
We will call this representation of the data the rank-r PCA projection of the data

If we write Π_r for the *dxd* matrix which is the identity matrix, keeping only the r upper left most 1s and zeroing out the lower right part of the diagonal, then the rank-r PCA projection of data point x_i is given as

$$\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) = \Pi_r \mathcal{U}^T \left(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right)\right)$$

If we transform our whole dataset our data will now be *r* dimensional—the last *d-r* components will be 0. This will save us *(d-r)N* space in storing the data. In a few slides we will show how good of an approximation of the data this is. And in the homework we will see that we can often use a small *r* and still capture most of the variation in a dataset

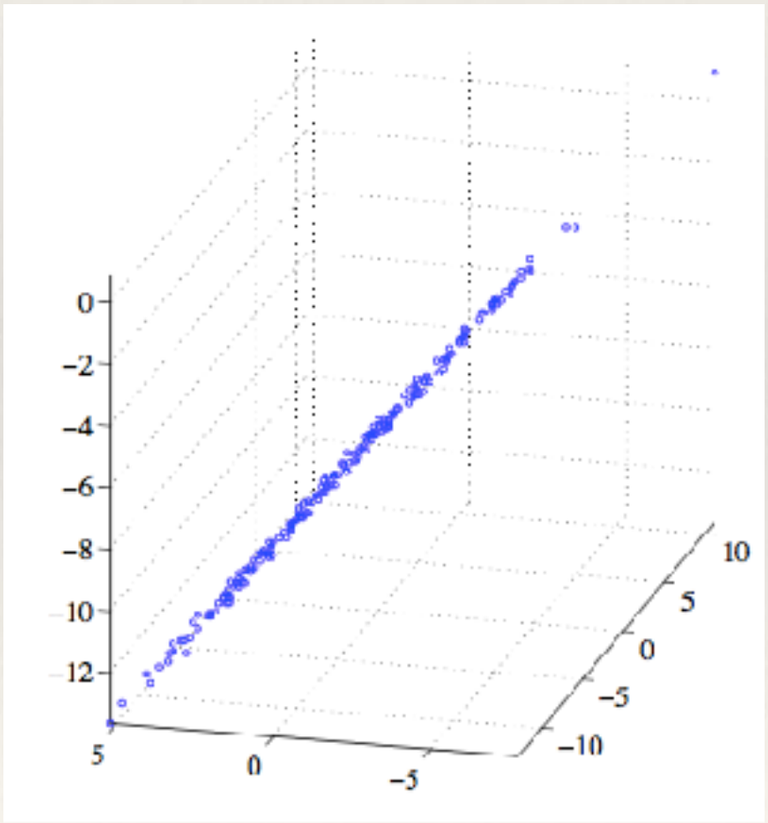
PCA dimensionality reduction



PCA smoothing

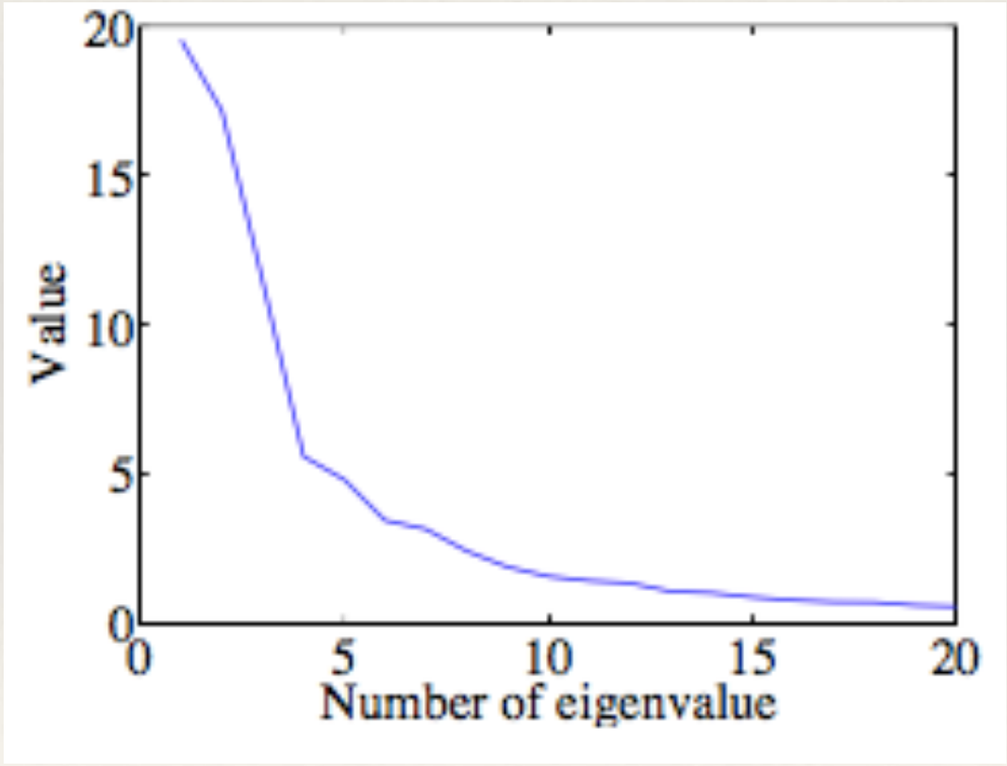
Supposing that space isn't a problem for us and we have no need to reduce the number of dimensions, this transformation can still be useful

In particular, if we are dealing with real-world data, we probably believe that of the variation in our data, some of it is due to noise



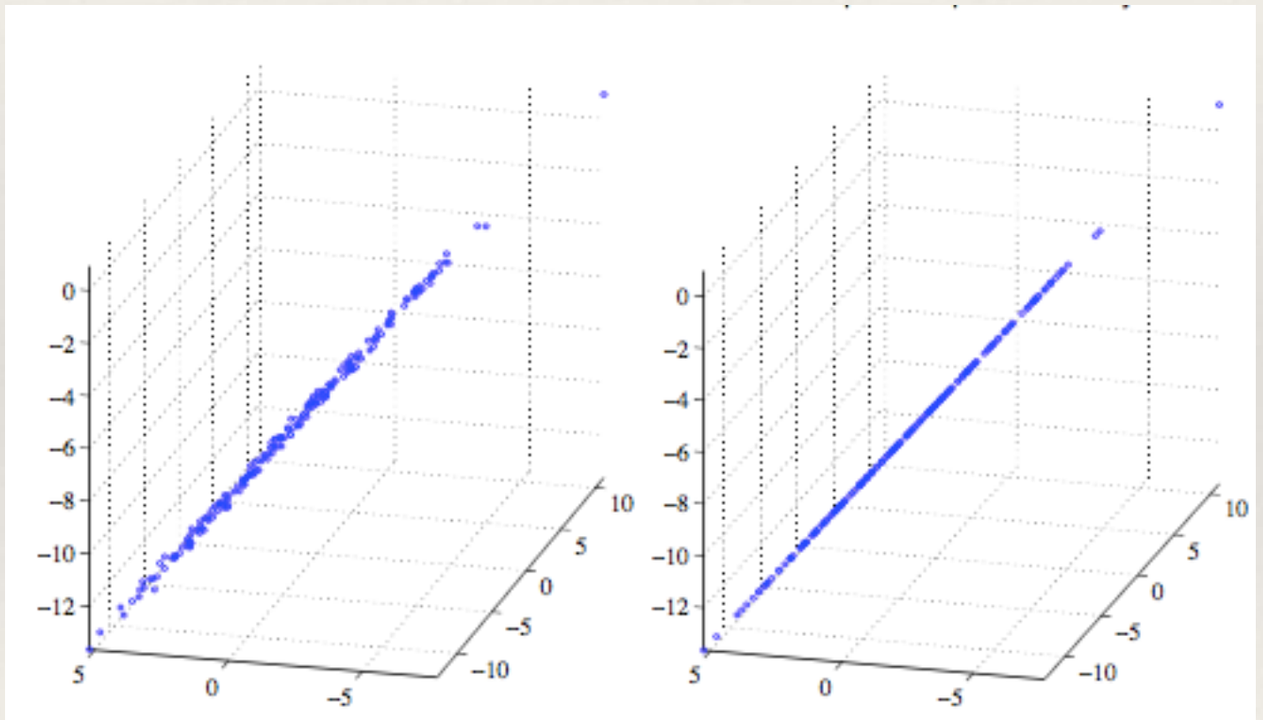
PCA Smoothing

If we examine or plot the eigenvalues of the data, they will be decreasing because we put them in decreasing order in our transformation. Furthermore, there is often a "knee" to the curve. We may assume that any variance after this knee is inconsequential and may be due to noise



PCA Smoothing

So just like dimensionality reduction we will keep only the top *r* components of the transformed data. Only since our concern this time is with smoothing the data, we finish up by putting the smoothed data back into its original coordinates



PCA Smoothing

If Π_r is as before, then we calculate the rank-r PCA smoothing of the data point x_i with

$$\begin{aligned}\text{pcasmooth}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) &= \mathcal{U}\Pi_r^T\left(\Pi_r\mathcal{U}^T(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))\right) + \text{mean}\left(\{\mathbf{x}\}\right) \\ &= \mathcal{U}\Pi_r^T\,\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) + \text{mean}\left(\{\mathbf{x}\}\right)\end{aligned}$$

PCA Smoothing

If Π_r is as before, then we calculate the rank-r PCA smoothing of the data point x_i with

$$\begin{aligned}\text{pcasmooth}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) &= \mathcal{U}\Pi_r^T\left(\Pi_r\mathcal{U}^T(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))\right) + \text{mean}\left(\{\mathbf{x}\}\right) \\ &= \mathcal{U}\Pi_r^T\,\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) + \text{mean}\left(\{\mathbf{x}\}\right)\end{aligned}$$

Does nothing

PCA Smoothing

If Π_r is as before, then we calculate the rank-r PCA smoothing of the data point x_i with

$$\begin{aligned}\text{pcasmooth}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) &= \mathcal{U}\Pi_r^T\left(\Pi_r\mathcal{U}^T(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))\right) + \text{mean}\left(\{\mathbf{x}\}\right) \\ &= \mathcal{U}\Pi_r^T\,\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) + \text{mean}\left(\{\mathbf{x}\}\right)\end{aligned}$$

Undo the rotation

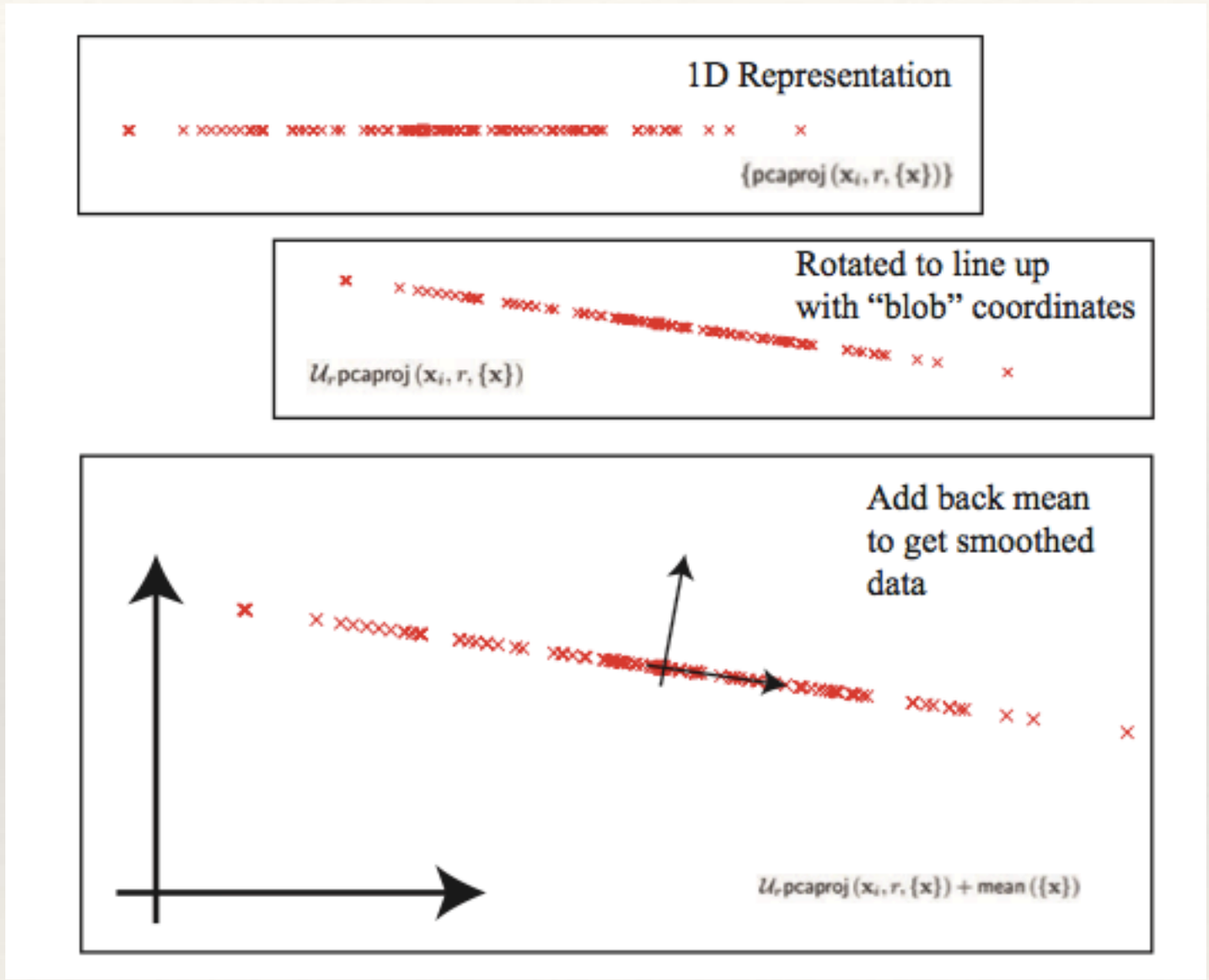
PCA Smoothing

If Π_r is as before, then we calculate the rank-r PCA smoothing of the data point x_i with

$$\begin{aligned}\text{pcasmooth}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) &= \mathcal{U}\Pi_r^T\left(\Pi_r\mathcal{U}^T(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))\right) + \text{mean}\left(\{\mathbf{x}\}\right) \\ &= \mathcal{U}\Pi_r^T\,\text{pcaproj}\left(\mathbf{x}_i, r, \{\mathbf{x}\}\right) + \text{mean}\left(\{\mathbf{x}\}\right)\end{aligned}$$

Translate back

PCA Smoothing



PCA Smoothing

The mean of the PCA smoothed data will be the same as the mean of the original data

The rank of the covariance matrix of the PCA smoothed data will be r which means that all of the data lies on an r dimensional hyperplane

The covariance matrix of the PCA smoothed data is the best rank r approximation to the covariance matrix of the original data

How much error?

We can write the error in the smoothed dataset for data point *i* as

$$\|x_i - \text{pcasmooth}(x_i, r, \{x\})\|^2$$

Since rotations and translations don't change the lengths of vectors and we invert our rotation and our translation during smoothing, we know the only time that we lose information in smoothing the data is when we cut off all but the first *r* components

If n_i is the translated and rotated data before we cut off the components and n_r,i is the data after we cut it off, then we have

$$\|x_i - \text{pcasmooth}(x_i, r, \{x\})\|^2 = \|n_i - n_{r,i}\|^2$$

How much error?

$$\|x_i - \text{pcasmooth}(x_i, r, \{x\})\|^2 = \|n_i - n_{r,i}\|^2$$

But we can rewrite the right hand side

$$\|x_i - \text{pcasmooth}(x_i, r, \{x\})\|^2 = \sum_{j=r+1}^{d} (n_i^{(j)})^2$$

If we average this quantity over our whole dataset, the average error of our approximation is

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{j=r+1}^{d} (n_i^{(j)})^2$$

How much error?

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j=r+1}^{d}(n_i^{(j)})^2$$

But since the mean of {n} is zero, this is just the formula for variance and our average approximation error is given by

$$\sum_{j=r+1}^{d}\mathrm{var}(\{n^{(j)}\})$$

If we've chosen our *r* well, this sum—the sum of the eigenvalues from Λ that we chopped off—will be small

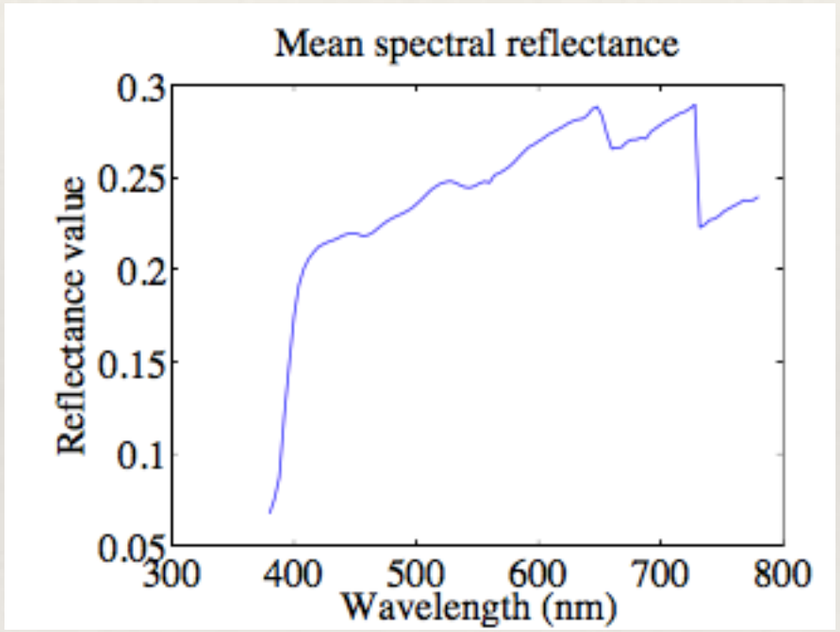
Example

Spectral reflectance data—how much light a surface reflects for a given range of wavelengths. Have in our dataset reflectance data on 1995 different surfaces represented by 101 4 nm ranges of wavelength in the visible spectrum from 380nm to 784nm

This is much more precision than we need given that surfaces have properties that means that reflectances can’t change too much from wavelength to wavelength

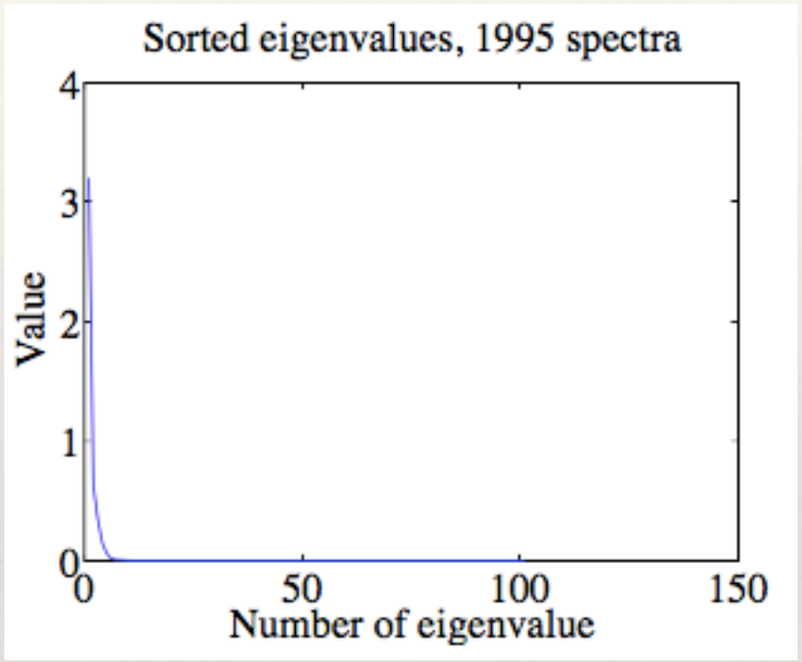
Example

Here's what the mean over 1995 different surfaces in our dataset looks like



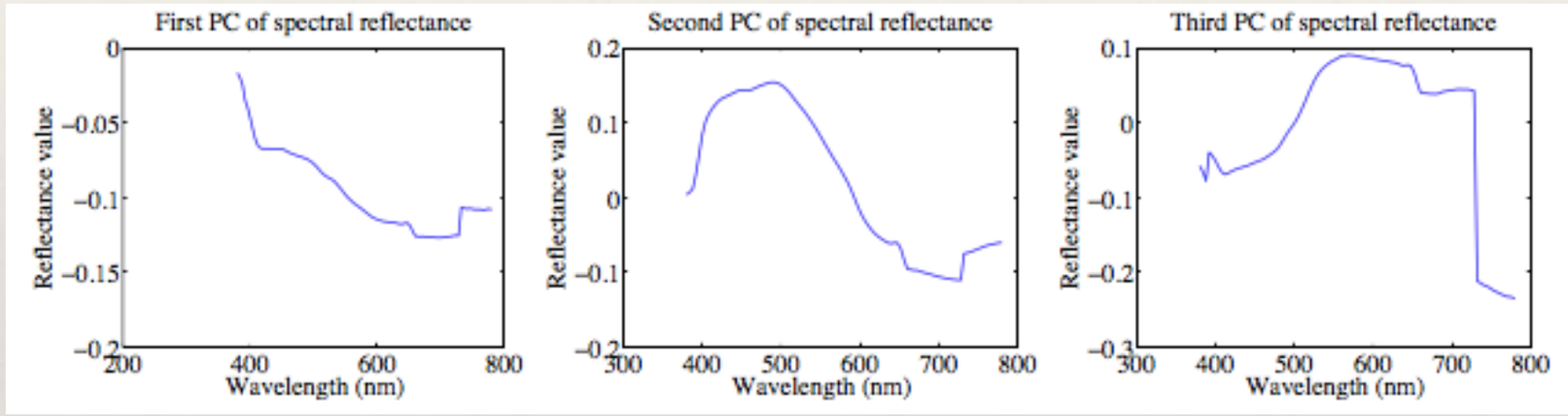
Example

Here's what the sorted eigenvalues look like



Example

It is useful to inspect the eigenvectors to get a sense of what coordinates of the original dataset they vary with or represent

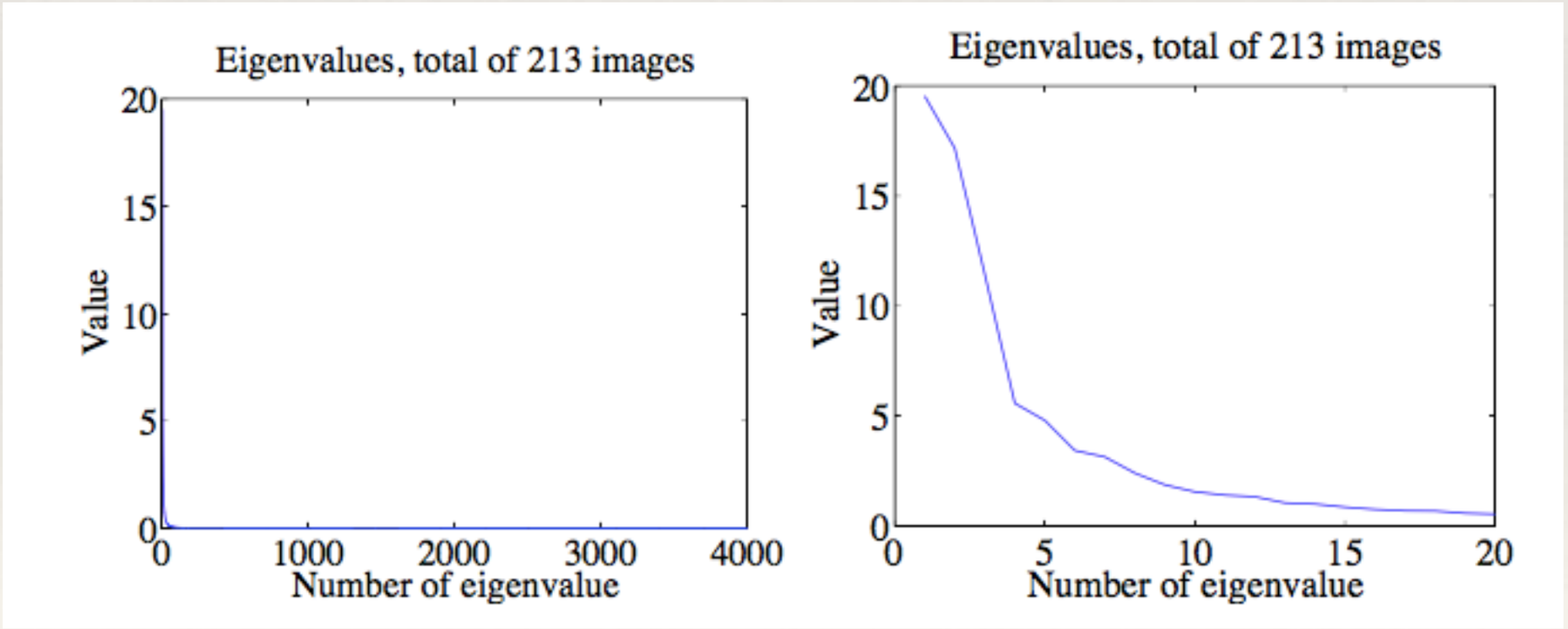


We can get decent approximations with the first 7 components or so

Example

We have a facial dataset encoding facial expressions of 213 Japanese women. They are 64x64 images and we only store a grayscale value, so our dataset is 4096 dimensional

We can view our data as vectors and proceed as normal. This is what the eigenvalues of the covariance matrix looks like



Example

Our mean and eigenvectors will be 4096 dimensional vectors. What if we interpret them as images?

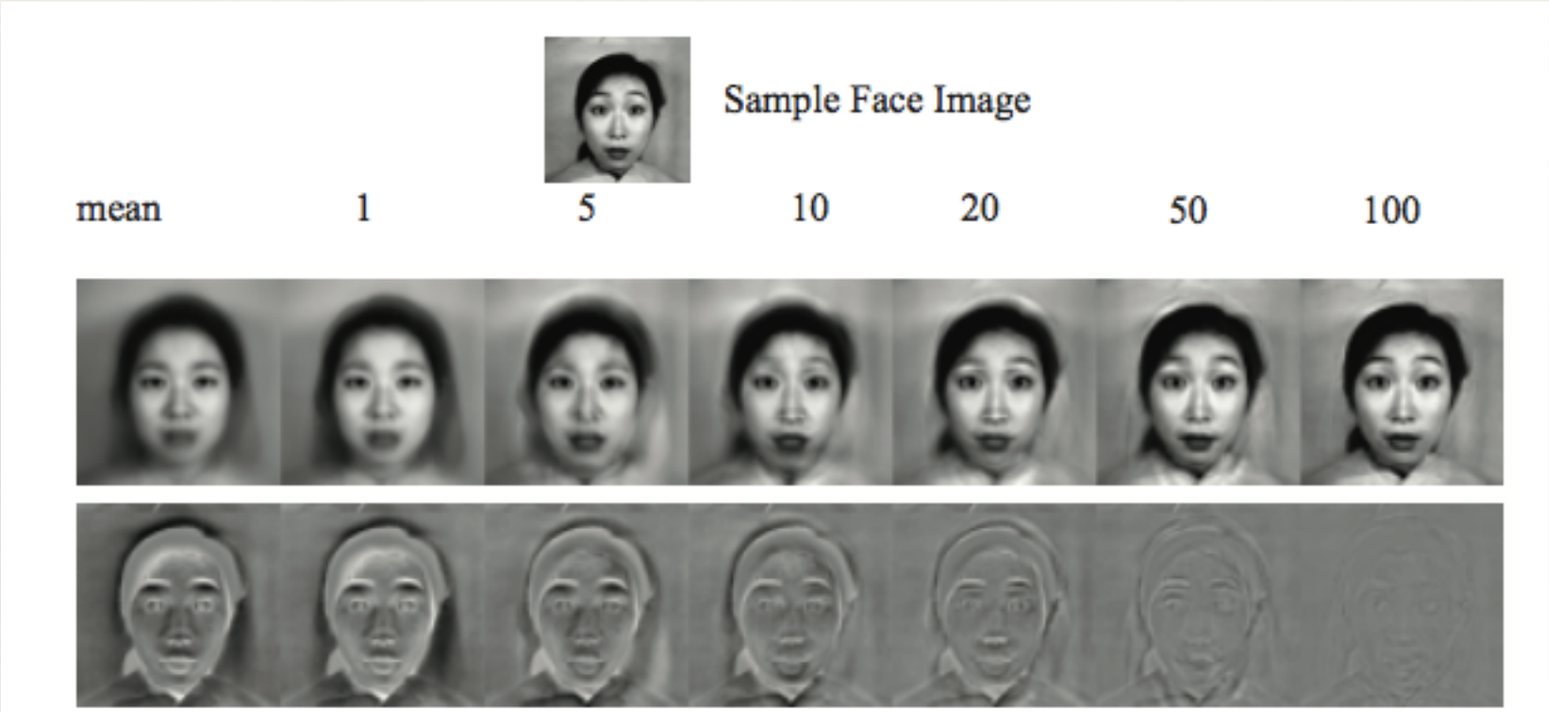
Example

Mean image from Japanese Facial Expression dataset

First sixteen principal components of the Japanese Facial Expression dat

Example

Having only 213 images to vary over 4096 dimensions seems like a very underdetermined problem. How well can we hope to approximate a face?



sample - approximation

Example

The mean face is kind of blurry because faces all have a different shape

Thinking of data as being a mean plus a weighted sum of the principal components is useful in general and here it helps us to understand how the first few principal components are capturing the shape of the hair, then the height of the face, then height of the eyebrows, and so on

If we had a larger dataset that included beards and people wearing glasses, these features would have been apparent in the first dozen or so principal components