Chapter 5

The Best Basis

5.1 Introduction

The problem we want to consider is this: We're given p points in $\mathbb{R}^n$. Find the "best" $k-$dimensional basis for the data.

There are a couple of things that will make our job easier:

- We will assume that the data has been mean-subtracted, so that the mean is zero (in $\mathbb{R}^n$).
- The basis is orthonormal (each basis vector is in $\mathbb{R}^n$).
- To find the "best" basis will require an error function. We will then minimize it.

At the end of this section, you'll see that the best $k-$dimensional basis for your data (regardless of k) is given by the first k **eigenvectors of the covariance matrix**, which are typically computed using the Singular Value Decomposition (SVD).

5.1.1 The Covariance Matrix, Revisited

Suppose we have p data points in $\mathbb{R}^n$, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p\}$, and they are organized column-wise in an $n \times p$ matrix X.

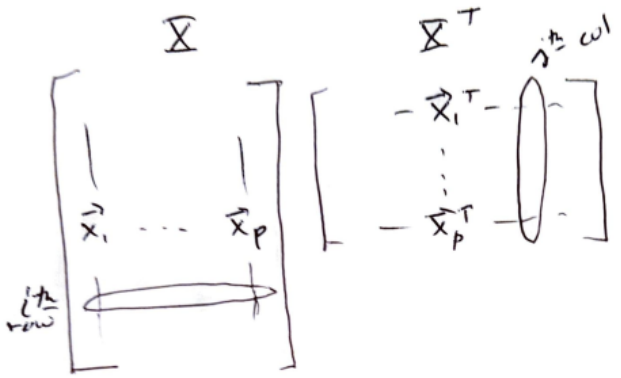
As we recall, the $n \times n$ covariance matrix for data in $\mathbb{R}^n$ measures the covariance between the data in coordinate i and the data in coordinate j. Using the $n \times p$ matrix X, then define $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ as the mean, then the $(i,j)^{\text{th}}$ entry of the covariance matrix is given by the following, where we're taking the covariance between the i^{th} and j^{th} row of X.

$$C_{ij} = \frac{1}{p-1} \sum_{k=1}^{p} (x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)$$

We will typically assume the mean is zero, so be sure and mean-subtract your data matrix before finding a basis for your data! With zero mean,

$$C_{ij} = \frac{1}{p-1} \sum_{k=1}^{p} x_{ik} x_{jk}$$

If we think of this computation in terms of the matrix X as in the figure below, we see that C_{ij} can be computed using a dot product between row i of X and column j of X^T:



Therefore, we see that the $n \times n$ matrix C can be computed one of two equivalent ways:

$$C = \frac{1}{p-1} XX^T \qquad \text{or} \qquad C = \frac{1}{p-1} \sum_{k=1}^{n} \boldsymbol{x}_k \boldsymbol{x}_k^T$$

Finally, we recognize that the covariance matrix is symmetric, so the **Spectral Theorem** applies. In particular, there is an orthonormal matrix P and a diagonal matrix D so that

$$C = PDP^T$$

where the columns of P form the eigenvectors associated with the diagonal elements of D (which are typically written largest to smallest).

To connect this to the SVD of the data matrix X, if X is $n \times p$ (so that data is stored column-wise), and we write the **reduced** SVD as:

$$X = U\Sigma V^T$$

Then

$$C = \frac{1}{p-1} XX^T = \frac{1}{p-1} U\Sigma V^T V \Sigma^T U^T = U \left(\frac{1}{p-1} \Sigma^2 \right) U^T$$

We see a relationship between the singular values of X, σ_i, and the eigenvalues of the covariance matrix, $\hat{\lambda}_i$:

$$\frac{1}{p-1} \sigma_i^2 = \hat{\lambda}_i$$

So far, we have defined a data matrix X, and we've looked at its covariance matrix C, and we've discovered that the eigenvectors of the covariance matrix are the left singular vectors of the data matrix (when the data is written column-wise and has been mean subtracted).

We'll be getting back to the best basis in a moment, but first we want to make a few more observations.

Projections and the Mean

Suppose you have your p data points in $\mathbb{R}^n$ that have *not* been mean subtracted, and you have a vector $\boldsymbol{u}$ onto which we want to project the data.

First, if we project one point $\boldsymbol{x}$ onto our (unit) vector $\boldsymbol{u}$, then the projection is $(\boldsymbol{x}^T\boldsymbol{u})\boldsymbol{u}$, and the scalar projection is the number $(\boldsymbol{x}^T\boldsymbol{u})$. Similarly, projecting all the data gives us p real numbers (the scalar projections):

$$\left\{ \boldsymbol{u}^T\boldsymbol{x}_1, \boldsymbol{u}^T\boldsymbol{x}_2, \ldots, \boldsymbol{u}^T\boldsymbol{x}_p \right\}$$

so the mean of the projected data is given by

$$\frac{1}{p} \left(\boldsymbol{u}^T\boldsymbol{x}_1 + \boldsymbol{u}^T\boldsymbol{x}_2 + \ldots + \boldsymbol{u}^T\boldsymbol{x}_p \right) = \boldsymbol{u}^T \frac{(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \ldots + \boldsymbol{x}_p)}{p} = \boldsymbol{u}^T \bar{\boldsymbol{x}}$$

Therefore, **the mean of the projection is the projection of the mean**. In particular, if the mean of a data set is zero, and the data is projected to a vector (or subspace), then the new mean is also zero.

Projections and the Variance

In the last section, we saw how the mean and the projection interacted. In this section, let's see how the variance is affected. We'll keep our previous general data set, p points in $\mathbb{R}^n$, and we'll suppose that **the data has zero mean** in $\mathbb{R}^n$. By what we showed, the scalar projections would also have zero mean.

If we project the p vectors onto an arbitrary unit vector $\boldsymbol{u}$, and consider the **scalar projections**, then the resulting variance (in the direction of $\boldsymbol{u}$ will be:

$$S_u^2 = \frac{1}{p-1}\sum_{k=1}^{p}(\boldsymbol{u}^T\boldsymbol{x}_k)^2 = \frac{1}{p-1}\sum_{k=1}^{p}\boldsymbol{u}^T\boldsymbol{x}_k\boldsymbol{x}_k^T\boldsymbol{u} = \boldsymbol{u}^T\left(\frac{1}{p-1}\sum_{k=1}^{p}\boldsymbol{x}_k\boldsymbol{x}_k{}^T\right)\boldsymbol{u} = \boldsymbol{u}^T C\boldsymbol{u}$$

This is actually a very key quantity, and will come up in the next section. We will look at this quantity more closely in a bit, but let's look at what happens in one **special case**: Suppose that $\boldsymbol{u}$ is an eigenvector of the covariance C corresponding to the first eigenvalue, $\hat{\lambda}_1$ using our previous notation. Then:

$$\boldsymbol{u}^T C\boldsymbol{u} = \boldsymbol{u}^T\hat{\lambda}_1\boldsymbol{u} = \hat{\lambda_1}$$

Therefore, if we project all the data to the first eigenvector of C, the new variance will be the first eigenvalue of C.

Reconstruction Error and the Basis

Given a specific vector $\boldsymbol{x} \in \mathbb{R}^n$ and an arbitrary orthonormal basis, $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_n$, we can write

$$\boldsymbol{x} = (\boldsymbol{\phi}_1^T\boldsymbol{x})\boldsymbol{\phi}_1 + (\boldsymbol{\phi}_2^T\boldsymbol{x})\boldsymbol{\phi}_2 + \ldots + (\boldsymbol{\phi}_n^T\boldsymbol{x})\boldsymbol{\phi}_n$$

so that the magnitude of $\boldsymbol{x}$ can be written as:

$$\|\boldsymbol{x}\|^2 = (\boldsymbol{\phi}_1^T\boldsymbol{x})^2 + (\boldsymbol{\phi}_2^T\boldsymbol{x})^2 + \ldots + (\boldsymbol{\phi}_n^T\boldsymbol{x})^2.$$

We can use the same algebraic manipulation that we used in the last section to rewrite this as:

$$\|\boldsymbol{x}\|^2 = \boldsymbol{\phi}_1^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{\phi}_1 + \boldsymbol{\phi}_2^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{\phi}_2 + \ldots + \boldsymbol{\phi}_n^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{\phi}_n.$$

We can break this up and define the error using one vector $\boldsymbol{\phi}_1$:

$$\|\boldsymbol{x}\|^2 = \boldsymbol{\phi}_1^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{\phi}_1 + \|\boldsymbol{x}_{\text{err}}\|^2$$

Now do this for all p data points. For any single vector $\boldsymbol{\phi}_1$, we sum these together:

$$\begin{aligned}
\|\boldsymbol{x}_1\|^2 &= \boldsymbol{\phi}_1^T\boldsymbol{x}_1\boldsymbol{x}_1^T\boldsymbol{\phi}_1 + \|\boldsymbol{x}_{\text{err}}^{(1)}\|^2 \\
+\|\boldsymbol{x}_2\|^2 &= \boldsymbol{\phi}_1^T\boldsymbol{x}_2\boldsymbol{x}_2{}^T\boldsymbol{\phi}_1 + \|\boldsymbol{x}_{\text{err}}^{(2)}\|^2 \\
&\vdots \\
\|\boldsymbol{x}_p\|^2 &= \boldsymbol{\phi}_1^T\boldsymbol{x}_p\boldsymbol{x}_p{}^T\boldsymbol{\phi}_1 + \|\boldsymbol{x}_{\text{err}}^{(p)}\|^2
\end{aligned}$$

$$\sum_{k=1}^{p}\|\boldsymbol{x}_k\|^2 = \boldsymbol{\phi}_1^T\left(\sum_{k=1}^{p}\boldsymbol{x}_k\boldsymbol{x}_k{}^T\right)\boldsymbol{\phi}_1 + \sum_{k=1}^{p}\|\boldsymbol{x}_{\text{err}}^{(k)}\|^2$$

We can multiply everything by $1/(p-1)$ to make things work. That is,

$$\frac{1}{p-1}\sum_{k=1}^{p}\|\boldsymbol{x}_k\|^2 = \boldsymbol{\phi}_1^T C\boldsymbol{\phi}_1 + \frac{1}{p-1}\sum_{k=1}^{p}\|\boldsymbol{x}_{\text{err}}^{(k)}\|^2. \tag{5.1}$$

In light of this equation, let us now define an error function using an arbitrary orthonormal basis, $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_n$. The error we get when using a one dimensional representation of our data is given by

$$E(\boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_n) = \frac{1}{p-1} \sum_{n=1}^{p} \|\boldsymbol{x}_{\text{err}}^{(n)}\|^2.$$

Notice that the left side of Equation 5.1 is constant (it is the sum of the magnitudes of all the known data). Therefore, **minimizing** the error function (the second term) is equivalent to **maximizing** the first term, $\boldsymbol{\phi}_1^T C \boldsymbol{\phi}_1$.

Here now is our algorithm for find the "best" basis:

1. Find the unit vector $\boldsymbol{\phi}_1$ so that $\boldsymbol{\phi}_1^T C \boldsymbol{\phi}_1$ is maximized.
2. We "project out" this vector so that the i^{th} data point now becomes:
$$\underline{\boldsymbol{x}}^{(i)} = \boldsymbol{x}^{(i)} - \text{Proj}_{\boldsymbol{\phi}_1}(\boldsymbol{x}^{(i)})$$
3. Re-compute C.
4. Repeat from Step 1 until we have enough basis vectors.

In practice, we will not need to do this- there is an easier way!

5.2 The Best Basis and the Eigenvectors

We have shown that finding the best basis reduces to maximizing the quantity:

$$\max_{\boldsymbol{\phi} \neq \mathbf{0}} \frac{\boldsymbol{\phi}^T C \boldsymbol{\phi}}{\boldsymbol{\phi}^T \boldsymbol{\phi}}$$

where we divide by the magnitude (squared) to enforce the fact that we want a unit vector, and we want to stay away from the zero vector.

We know that the eigenvectors of the covariance matrix form an orthonormal basis for $\mathbb{R}^n$, so we can write any vector as a linear combination of them:

$$\boldsymbol{\phi} = c_1 \boldsymbol{v}_1 + c_2 \boldsymbol{v}_2 + \cdots + c_n \boldsymbol{v}_n = V \boldsymbol{c}$$

Using the eigenvector-eigenvalue factorization $C = VDV^T$, we can now write the numerator as:

$$\boldsymbol{\phi}^T C \boldsymbol{\phi} = (V\boldsymbol{c})^T (VDV^T)(V\boldsymbol{c}) = \boldsymbol{c}^T (V^T V) D (V^T V) \boldsymbol{c} = \boldsymbol{c}^T D \boldsymbol{c} = c_1^2 \lambda_1 + c_2^2 \lambda_2 + \cdots + c_n^2 \lambda_n$$

Similarly, the denominator is:

$$\boldsymbol{\phi}^T \boldsymbol{\phi} = c_1^2 + c_2^2 + \cdots + c_n^2$$

Let's look at the coefficients of our expansion now. For λ_i, the coefficient in front is

$$\rho_i = \frac{c_i^2}{c_1^2 + c_2^2 + \cdots + c_n^2}$$

where $\rho_i \geq 0$ and $\sum_{i=1}^{n} \rho_i = 1$ (like a probability distribution). Let's summarize where we are. We now see that maximizing $\boldsymbol{\phi}^T C \boldsymbol{\phi}$ is equivalent to choosing $\rho_1, \rho_2, \cdots, \rho_n$ so that each $\rho_i \geq 0$ and they sum to 1, to maximize the quantity:

$$\rho_1 \lambda_1 + \rho_2 \lambda_2 + \cdots + \rho_n \lambda_n$$

It is easy to see that, if the $\lambda_i \geq 0$ and are ordered from largest to smallest, then:

$$\lambda_n \leq \rho_1\lambda_1 + \rho_2\lambda_2 + \cdots + \rho_n\lambda_n \leq \lambda_1.$$

To maximize our given quantity, we set $c_1 = 1$ and the rest of the coefficients to zero. This leads us to our main conclusion. The vector $\boldsymbol{\phi}$ that maximizes the quantity $\max_{\boldsymbol{\phi}\neq\mathbf{0}} \frac{\boldsymbol{\phi}^T C\boldsymbol{\phi}}{\boldsymbol{\phi}^T\boldsymbol{\phi}}$ is given by $\boldsymbol{v}_1$, the eigenvector corresponding to the largest eigenvalue of C. This is summarized by the theorem below:

The Best Basis Theorem

Given p points in $\mathbb{R}^n$, the *best $k-$dimensional basis* is found by taking the first k eigenvectors of the covariance matrix C. Equivalently, given the data in an $n \times p$ matrix X, the best $k-$dimensional basis is found by taking the first k columns of the U, the left singular vectors of the SVD of X. Further, this is the "best" basis for $k = 1, 2, \ldots, r$, where r is the rank of X.

Speaking of Rank...

We discussed this briefly in an earlier section, but it is worth thinking about again.

It is useful to look at the rank as that number of basis vectors required to preserve some percentage of the variance in the data. From our previous section on the covariance matrix, we had a relationship between the eigenvalues of the covariance matrix, $\hat{\lambda}_i$ and the singular values of X:

$$\frac{1}{p-1}\sigma_i^2 = \hat{\lambda}_i$$

so normalizing the set of eigenvalues is equivalent to doing it to the squared singular values:

$$\lambda_i = \frac{\hat{\lambda}_i}{\sum_{j=1}^n \hat{\lambda}_j} = \frac{\frac{1}{p-1}\sigma_i^2}{\sum_{j=1}^n \frac{1}{p-1}\sigma_j^2} = \frac{\sigma_i^2}{\sum_{j=1}^n \sigma_j^2}$$

Now the λ_i are positive and sum to 1. The idea is to keep enough dimensions r so that

$$\sum_{i=1}^{r} \lambda_i \geq \tau \qquad \text{but} \qquad \sum_{i=1}^{r-1} \lambda_i < \tau.$$

In this case, we would say that it takes r dimensions to explain or encapsulate τ percent of the variance in the data.

What should τ be? This is problem dependent. In some very noisy problems, you may only want to keep $\tau \approx 0.6$, while with very little noise, you might take $\tau \approx 0.99$.

5.2.1 The Dimensionality Reduction Step

Once we have our k basis vectors, what do we do with them? First, we create our low dimensional representation of the data. Initially, the data represents p points in $\mathbb{R}^n$, and we want to reduce that to p points in $\mathbb{R}^k$- These are the coordinates of each point using our $k-$dimensional basis. That is, if $U\Sigma V^T$ is the svd of X (mean subtracted), then the k dimensional data is created by the following, where U is $n \times k$, X is $n \times p$, and the low-dimensional representation X_{coords} is $k \times p$.

$$X_{\text{coords}} = U^T X$$

Especially if $k = 2$ or $k = 3$, we can then plot the low dimensional points in the plane or in 3-d. The "reconstruction" of the data is the representation back in $\mathbb{R}^n$ using the k basis vectors.

$$X_{\text{recon}} = UX_{\text{coords}} = UU^T X$$

Remember that earlier we said that $U^T U = I$, but UU^T is the projection matrix taking data in $\mathbb{R}^n$ and projecting it into the column space of U (so $X \neq UU^T X$ unless the columns of X are already contained within the column space of U).

5.3 Connecting to Principal Components Analysis (PCA)

In PCA, the principal components are defined to be a sequence of k direction vectors, where the i^{th} vector is the direction of a line that best fits the data while being orthogonal to the first $i-1$ vectors[1].

You can see that the principal components of set of data are then equivalent to the k basis vectors we've constructed (the first k eigenvectors of the covariance matrix). While PCA and the best basis are the same, you will typically hear the language of statistics used in PCA, while we use the language of linear algebra in constructing the best basis.

5.4 Exercises

Before doing the computer problems below, you should write down (using linear algebra notation) what computations you want to make. If you have questions (especially with the coding), I'm happy to help.

1. Suppose we have p data points in $\mathbb{R}^n$. Show that the variance of the data, projected to a standard basis vector $\boldsymbol{e}_i$, returns the usual variance for the data in that dimension. (I want you to look back at the computations we made for this in the text, "Projections and the Variance").

2. Suppose we have two o.n. vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$. Given our p points in $\mathbb{R}^n$, compute the covariance between the data projected to $\boldsymbol{u}$ and the data projected to $\boldsymbol{v}$, and (i) show that the result is

$$\boldsymbol{u}^T C \boldsymbol{v}$$

   (ii) In the special case that $\boldsymbol{u}, \boldsymbol{v}$ are eigenvectors of the covariance matrix, how does this quantity simplify?

3. Suppose we have 4 points in $\mathbb{R}^3$ as organized in the matrix X (left and below), and let $\boldsymbol{\phi}_1 = (1/\sqrt{3})[1,1,1]^T$. Use a computer (Octave/Matlab, Python or R) to compute the three quantities given in the formula to the right and below. In your script, be sure you're actually computing the covariance matrix and each quantity separately.

$$X = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & 0 & 1 & 1 \\ -1 & 1 & 2 & 1 \end{bmatrix}, \qquad \frac{1}{p-1}\sum_{k=1}^{p} \|\boldsymbol{x}_k\|^2 = \boldsymbol{\phi}_1^T C \boldsymbol{\phi}_1 + \frac{1}{p-1}\sum_{k=1}^{p} \|\boldsymbol{x}_{\text{err}}^{(k)}\|^2.$$

4. Using the data (and vector $\boldsymbol{\phi}_1$) in the previous exercise, compuationally verify our statements: The projection of the mean is the mean of the projection, and the variance of the data projected to $\boldsymbol{\phi}_1$ is $\boldsymbol{\phi}_1^T C \boldsymbol{\phi}_1$.

5. Verify numerically that the variance of the projected data to the first best basis vector (first one) is given by the first eigenvalue of the covariance matrix. (Careful- if you use the `eig` command, the eigenvalues are not ordered).

6. Continuing with the data from Problem 3, if we retained two of the basis vectors, how much variance (as a percentage) is "explained" by them? (This refers to the discussion in the text about how to compute the rank).

7. Load the clown data, we obtain a matrix X that is 200×320. Treat this as 320 vectors in $\mathbb{R}^{200}$.

   (a) Double center the data in X (call the result X_m).

   (b) Find the best two dimensional basis for the vectors in X_m, then project the data to two dimensions and plot the result.
   (Question to think about, you don't need to answer: Did you expect a pattern or not?)

   (c) Reconstruct the data back in $\mathbb{R}^{200}$, and show the result as an image (don't add the means back in).

[1] Wikipedia, pulled March 2021

5.5 A Summary

Before we continue, let's summarize the process for PCA (or best basis), and let's discuss what you can get out of the PCA. First, the process- It's short.

Principal Components Analysis

Let X store p points, each in $\mathbb{R}^n$ so X is $n \times p$. To find the best basis (in $\mathbb{R}^n$):

1. Compute the mean so that $\bar{\boldsymbol{x}} \in \mathbb{R}^n$.
2. Mean subtract the data, put in the $n \times p$ matrix X_m.
3. For the resulting data, take the SVD:
$$X_m = U\Sigma V^T$$
4. The best basis (or principal components) are the first k columns of U.

Once this is done, what can we do with it? Here is a list of common tasks.

- Visualize the mean and eigenvectors.

  The eigenvectors of the covariance matrix (the columns of U) typically carry very interesting information- What do they look like? In the application below, the eigenvectors can be visualized as photos, but you can also simply plot an eigenvector if that has meaning (plot the vector index along the horizontal axis, and the vector values along the vertical axis. In Matlab, if $\boldsymbol{v}$ is a vector, this would be `plot(v)`.

- Determine the rank.

  We talked about this earlier- The rank of a matrix is critical to know if you'll be constructing the pseudo-inverse, or if you simply want to reduce the dimensionality of the data (see the next item). One way to determine the rank is to look for large breaks in the plot of the singular values (the diagonal elements of Σ). A second way is to construct the eigenvalues of the covariance matrix, and retain enough dimensions so that the resulting space encapsulates a certain percentage of the variance.

- Reduce the dimensionality of the data.

  This is the primary goal of most applications of PCA. If X_m is $n \times p$ (p points in $\mathbb{R}^n$, with mean subtracted), then the following is the $k \times p$, or $k-$dimensional, representation of the p points- matrix L (for low dimensional) is $k \times p$:
$$L = U(:, 1:k)^T X_m$$
  The Matlab-esque notation means to take all rows, but only the first k columns of U.

- Visualize the data that has been reduced in dimension.

  It's often a good idea to plot the first rows of L in $\mathbb{R}^2$ just to be sure it gives you what you expect. There might be even more interesting information using other basis vectors down further in the decomposition (corresponding to lower rows of L)- We'll actually see an example below.

- Project new data using the new basis.

  If D is $m \times n$ (m points in $\mathbb{R}^n$), and D_m is the matrix with the mean of X subtracted from the columns, then the projected data in matrix P is also $m \times n$ and is given by:
$$P = U(:, 1:k)U(:, 1:k)^T D_m$$
  You can then compare P to D_m (or add the mean back to both matrices) to see if the new data is well represented by the basis.

Before we leave this discussion for an application, have you considered what the matrix V (from the SVD of X) represents? It represents a scaled version of the coordinates (or low dimensional representation). How? If $X_m = U\Sigma V^T$ is the reduced SVD, then

$$L = U^T X_m = U^T(U\Sigma V^T) = \Sigma V^T$$

And remember that Σ is just a diagonal matrix ($k \times k$) and V is $p \times k$. Multiplying by the diagonal just scales each column of V, so V is a scaled version of the $k-$dimensional representation of the data!

5.6 Application: Eigenfaces

Consider the set of all photos of some fixed width and length (say $a \times b$) and these are photos of faces. To be even more specific, let's set the coordinates of the eyes to be the same in all photos.

As you might imagine, the dimension of this "face space" should be quite large, since we're representing all possible faces here. However, should it fill up $\mathbb{R}^{ab}$? If you were to take an arbitary vector in $\mathbb{R}^{ab}$ and visualize it as an $a \times b$ photo, what would you see? The image would probably be just noise- the set of "faces" is then (hopefully) only a "small" dimensional subspace of $\mathbb{R}^{ab}$, and that gives us some reason to expect that we can find a smallish number of template faces so that every face is a linear combination of the templates.

What are these template faces? They are basis vectors in $\mathbb{R}^{ab}$, meaning that each basis vector can be visualized as an $a \times b$ image. We know that these basis vectors are the eigenvectors of the covariance matrix of the photo data, so we'll call them **eigenfaces**.

In out next example, we'll explore face space a bit and see some interesting things.

5.6.1 Project Example: The Yale Database

The data we'll look at is a portion of the Extended Yale Database below linked below if you want to have a look (good as of March 2021):

Extended Yale Database

Further, this little project is adapted from Brunton and Kutz' text on "Data Driven Science and Engineering", which is at `databookuw.com`. The had a very cute example!

As a side note, it's important to distinguish between an $m \times n$ matrix (which as m rows and n columns) versus an $m \times n$ image, which has a width of m pixels and a height of n pixels (not my fault!). The images below are said to be 168×192, but are typically stored in a 192×168 matrix (for future reference, $168 \times 192 = 32256$).

Discussion of the Data

We'll begin with a discussion of the dataset `allFaces2.mat`, which will be available from our class website. Here is a discussion of the datasets you'll see stored in this file.

- `faces` is 32256×2410 matrix, representing 2410 photos, each of which is 192×168 (as a matrix).
- Constants $n = 192$ and $m = 168$.
- `nfaces` is a vector with 38 elements, each representing one of the persons posing for photos. The datain `faces` is stored in order with all the poses for the first person first, then the second person, and so on. The vector `nfaces` stores the number of poses each person has. For example, the first person has 64 poses, so `nfaces(1)` is 64.
- `Dog` is an image of the same size as the photos of the people, but is the grayscale photo of a dog.

Figure 5.1: The first 36 of the 38 persons in the file are shown to the left. To the right, we show a sample of 36 poses for the first person (out of 64 included in the data file). Each photo is 168×192 (as an image).

Visualizing a Sample of Data

In Figure 5.1, the first 36 of the 38 persons in the file are shown to the left. To the right, we show a sample of 36 poses for the first person (out of 64 included in the data file).

Project Outline

We're going to find the best basis for our face data. Since we have so many photos, we're going to try "training" the PCA on just the photos from the first 36 people, leaving the last two people out to see how well we have captured "face space". Continuing, here's what we'll do on the computer:

1. Load the data. The "training" data will be the first 2282 columns.
2. Find the mean, then mean-subtract the data. Visualize the mean.
3. Find the SVD.
4. Look at the singular values to see if there is any clear break.
5. Look at some sample eigenfaces (columns of U).
6. Project persons 2 and 7 into the plane spanned by the 5th and 6th basis vector, and plot the result.
7. Look at reconstructions of person 37 using dimension: $50, 150, 500, 1000$.
8. Look at whether or not we can reconstruct the dog's face using a basis from people!

Project Implementation: Matlab/Octave

Side Remark: This data will most likely be too large to run on Octave-online. However, if you have Octave set up on your home PC, it should work. For the homework, we'll be using a smaller data set.

```
%% Eigenface Example using Yale Database

%%Step 1: Load the data and create the training subset using 36 people.
%   The total number of columns needed is: sum(nfaces(1:36))

load allFaces2.mat
trainingFaces=faces(:,1:sum(nfaces(1:36)));

%% Step 2: Find the mean, mean-subtract the data.  Visualize the mean
%    as a photo!

avgFace=mean(trainingFaces,2);
Xm=trainingFaces-avgFace;

figure(1)
imagesc(reshape(avgFace,n,m));
colormap(gray); axis off; axis square

%% Step 3: Compute the SVD.
[U,S,V]=svd(Xm,'econ');

%% Step 4: Look at the singular values; usually a log plot for high dimensions.
plot(log(diag(S)));  % At approx 2262, we see a big drop- this is the rank.

%% Step 5:  Let's look at some eigenfaces!  Here are the first four:
figure(2)

for j=1:4
    subplot(2,2,j)
    imagesc(reshape(U(:,j),n,m));
    colormap(gray); axis off; axis square;
end

%% Step 6:  Project the poses from persons 2 and 7 into the plane spanned by
%          basis vectors 5 and 6.  Don't worry about the formulas used to determine
%          the columns.  They're included here in case you want to change 2 and 7
%          to other people.
P1=2;  P2=7;  Bas1=5; Bas2=6;
P1cols=sum(nfaces(1:P1-1))+1:sum(nfaces(1:P1));
P2cols=sum(nfaces(1:P2-1))+1:sum(nfaces(1:P2));
Data=faces(:,[P1cols,P2cols]) - avgFace;

Coords=U(:,[Bas1,Bas2])'*Data;  % This is two dimensional data
figure(3)
plot(Coords(1,1:nfaces(P1)),Coords(2,1:nfaces(P1)),'ro');
hold on
plot(Coords(1,nfaces(P1)+1:end),Coords(2,nfaces(P1)+1:end),'k^');
hold off

%% Step 6A: Optional subplot with the two faces and positive/negative
%   5th basis vector.
```

```
subplot(2,2,1)
imagesc(reshape(faces(:,65),n,m));
colormap(gray); axis off; axis square;
subplot(2,2,2)
imagesc(reshape(U(:,5),n,m));
colormap(gray); axis off; axis square;
subplot(2,2,3)
imagesc(reshape(faces(:,381),n,m));
colormap(gray); axis off; axis square;
subplot(2,2,4)
imagesc(reshape(-U(:,5),n,m));
colormap(gray); axis off; axis square;

%% Step 7: Look at partial reconstructions of person 37 using rank 50, 150, 500, 1000
Data=faces(:,sum(nfaces(1:36))+1);
Data=Data-avgFace;
kdim=[50,150,500,1000];

figure(4)
for j=1:4
    subplot(2,2,j)
    Recon=U(:,1:kdim(j))*(U(:,1:kdim(j))'*Data);
    imagesc(reshape(Recon+avgFace,n,m));
    colormap(gray); axis off; axis square
end

%% Step 8:  Repeat, for the photo of a dog!
Data=Dog;
Data=Data-avgFace;
kdim=[50,100,400,600,1600];

figure(5)
subplot(2,3,1)
imagesc(reshape(Dog,n,m));
colormap(gray); axis off; axis equal

for j=1:5
    subplot(2,3,j+1)
    Recon=U(:,1:kdim(j))*(U(:,1:kdim(j))'*Data);
    imagesc(reshape(Recon+avgFace,n,m));
    colormap(gray); axis off; axis equal
end
```

Project Implementation: Python

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io

# Step 0: set some parameters, load the original Matlab data
plt.rcParams['figure.figsize'] = [8, 8]
plt.rcParams.update({'font.size': 18})
```

```
mat_contents = scipy.io.loadmat('allFaces2.mat')
faces = mat_contents['faces']
m = int(mat_contents['m'])
n = int(mat_contents['n'])
Dog = mat_contents['Dog']
nfaces = np.ndarray.flatten(mat_contents['nfaces'])

#%%
# Step 1: Load the data
trainingFaces = faces[:,:np.sum(nfaces[:36])]

#%%
# Step 2: Find the mean, visualize it.
avgFace = np.mean(trainingFaces,axis=1) # size n*m by 1
Xm=trainingFaces-avgFace[:,np.newaxis];

plt.figure()
plt.imshow(np.reshape(avgFace,(m,n)).T)
plt.set_cmap('gray')
plt.title('Mean Face')
plt.axis('off')
plt.show()

#%%
# Step 3: Compute the SVD
U, S, VT = np.linalg.svd(Xm,full_matrices=0)

#%%
# Step 4: Plot the singular values.
plt.figure()
plt.plot(np.log(S))

#%%
# Step 5:  Let's look at some eigenfaces (first 4, pos and neg)

fig,axs = plt.subplots(2,2)
axs=axs.ravel()
for i in range(4):
    axs[i].imshow(np.reshape(U[:,i],(m,n)).T)

fig,axs = plt.subplots(2,2)
axs=axs.ravel()
for i in range(4):
    axs[i].imshow(np.reshape(-U[:,i],(m,n)).T)
    axs[i].set_title(str(i))


#%%
## Step 6:  Project person 2 and 7 onto basis vecs 5 and 6

P1 = 2 # Person number 2
```

```
P2 = 7 # Person number 7

P1data = faces[:,np.sum(nfaces[:(P1-1)]):np.sum(nfaces[:P1])]
P2data = faces[:,np.sum(nfaces[:(P2-1)]):np.sum(nfaces[:P2])]

P1data = P1data - avgFace[:,np.newaxis]
P2data = P2data - avgFace[:,np.newaxis]

# Project onto PCA modes 5 and 6
PCACoordsP1 = U[:,4:6].T @ P1data
PCACoordsP2 = U[:,4:6].T @ P2data

plt.figure()

plt.plot(PCACoordsP1[0,:],PCACoordsP1[1,:],'d',color='k',label='Person 2')
plt.plot(PCACoordsP2[0,:],PCACoordsP2[1,:],'^',color='r',label='Person 7')

plt.legend()
plt.show()


# %%
## Step 7:  Show eigenface reconstruction of image that was omitted from test set


plt.figure()
testFace = faces[:,np.sum(nfaces[:36])] # First face of person 37
plt.imshow(np.reshape(testFace,(m,n)).T)
plt.set_cmap('gray')
plt.title('Original Image')
plt.axis('off')
plt.show()


#%% Step 8: Reconstruct a dog from photos of people!
#
# Vector "Dog" needs to be reshaped for some reason...
Dog=np.reshape(Dog,(32256,))
testFace = Dog - avgFace
k_list = [50, 200, 500, 1000, 1600]

plt.figure()

fig,axs = plt.subplots(2,3)
axs=axs.ravel()

axs[0].imshow(np.reshape(Dog,(m,n)).T)
axs[0].axis('off')
for i in range(5):
    reconFace = avgFace + U[:,:k_list[i]]  @ ( U[:,:k_list[i]].T @ testFace)
    axs[i+1].imshow(np.reshape(reconFace,(m,n)).T)
    axs[i+1].axis('off')
```

Project: Eigenfaces

Description of the data: `Math350Homework.mat`

- Matrix X that is 24138 26 (26 photos, each with 162 rows and 149 columns)
- Constants $m = 162$ and $n = 149$.

If you want to visualize the photos, you would reshape the corresponding column vector. For example, to visualize the first four faces and put them together in one figure, we would type:

```
for jj=1:4
   subplot(2,2,jj)
   imagesc(reshape(X(:,jj),m,n));
   axis off; axis equal; colormap(gray)
   end
```

Our project is to use the provided code and previous example to do some analysis of this data set.

1. Compute the mean vector and represent it as a face (by plotting it).
2. Compute the first four eigenfaces (and represent them as faces). Put them together in one plot. In a second plot, show what the "photographic negatives" of the eigenfaces look like.
3. What might be a good approximation to the rank of the matrix X (if our goal is to make the faces recognizable, about 74% of the variance).
4. Plot the reconstruction of a randomly selected face using rank 2, 5, 10 and 15. Plot these (as photos) together in one figure.
5. Plot the data using the best two dimensional representation using a new figure. Plot the first 13 data points as asterisks and the remaining 13 as triangles.

Chapter 6

A Best Nonorthogonal Basis

In this section, we examine a particular question whose solution will involve getting an optimal *nonorthogonal* basis, which is quite contrary to our earlier chapters- in fact, the reader should ask why we would ever want to use a non-orthogonal basis when it would be quite easy (using Gram-Schmidt) to construct an orthogonal version of the same basis.

To answer this question, consider the synthetic data example in Figure 6.1. Here there is a definite "natural" basis appearing in the data- and the basis vectors are not orthogonal. While the data was synthetic, we do get similar types of data appearing in the problem of *Blind Signal Separation*. Consider the following tasks:

1. We have a patient that is pregnant. Our overall goal is to listen to the fetus heartbeat, but when we try, the sound of the mother's heartbeat is mixed with the heartbeat of the fetus. Symmetrically, if we were to try to listen to the heartbeat of the mother, we would also hear the heartbeat of the fetus. Is it possible to gather these sounds on microphones and manipulate the data so that the mother's (or fetus) heartbeat has been isolated?

2. We have two microphones placed at random, but distinct, places in a room. We also have two people speaking in the room (the placement of the people is distinct from the placement of the microphones- we do not assume that each microphone is placed in front of each speaker). Is it possible to manipulate the two mixtures of voices so that we can isolate each speaker's voice?

The answer lies in a fairly new technique called *Independent Component Analysis* (ICA) (versus what we studied earlier, *principal components analysis*, or PCA). In ICA, we assume that we have some underlying, statistically independent, processes and that we are observing mixtures of these processes. Our goal is to separate the mixtures.

This problem is also known as *Blind Signal Separation*, where we assume some unknown mixture of signals, and we attempt at separating them.

This process (the problem and the solution) can also be framed in other terms- we will focus on the geometric meaning of the problem, and will solve it using the techniques of linear algebra.

6.1 Set up the Signal Separation Problem

We will assume that there exists a "clean" separation of our observed mixture of two signals (we will be explicit in what we mean by that momentarily, and we will discuss the more general case in a moment). These signals, as time series, are two columns of a matrix S, so that $S \in \mathbb{R}^{p\times 2}$, where p is the length of the sample.

We will further assume that the mixtures we are observing are *linear* mixtures, so that the mixtures we observe may be modeled as:

$$\boldsymbol{x}_1 = \alpha_1 \boldsymbol{s}_1 + \alpha_2 \boldsymbol{s}_2$$