cs339, Artificial Intelligence

Dimensionality and Basis

1 Dimensionality

Here we work with a set of samples $X = \{x^{(1)}, ..., x^{(P)}\}$ where we have P different samples x where each $x^{(i)}$ is a column vector in N-dimensional space, i.e. $x^{(i)} \in \Re^N$. We store these P points in an NxP matrix X:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & ... & x^{(P)} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & ... & x_1^{(P)} \\ x_2^{(1)} & x_2^{(2)} & ... & x_2^{(P)} \\ ... & ... & ... & ... \\ x_N^{(1)} & x_N^{(2)} & ... & x_N^{(P)} \end{bmatrix} \quad (1)$$

For example, we create a random set of points using matlab

```
X = randn(1,10)
```

creates a set of $P = 10$ samples, each sample being a 1-dimensional point ($N = 1$). We would clearly say that this dataset lies in 1-dimensional space; there is little confusion about that. Now lets create a second data set:

```
X = [randn(1,10); zeros(1,10)]
```

We can plot this data as shown in Figure 2.

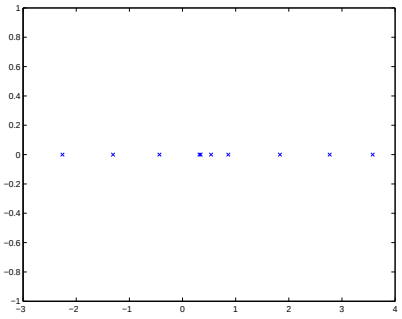


Figure 1: Is Data Dimension 1?

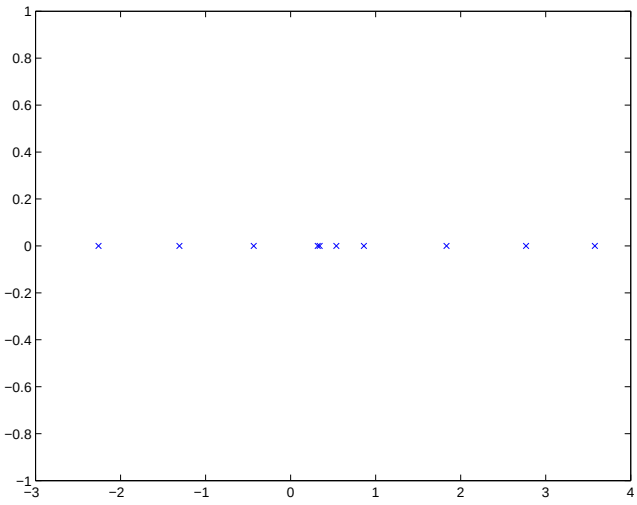


Figure 2: Linear Data?

Now, what dimensionality does this data possess? There are two proper answers: it certainly is 2-dimensional data, but one of those dimensions contains zero information. We can do a simple transformation to convert the data to 1-dimensional without losing any important information. Thus we say the *inherent dimensionality* is 1.

We can do other manipulations to the data, such as translation, scaling, and rotation. These are all linear manipulations. We can also perform non-linear transformations, which again, do not affect the inherent dimensionality, but we focus on linear transformations primarily here.

2 PCA

We can also create a data set like the following:

```
X = randn(2,10)
X =  [2, 0; 0, 1] * X;            %scaling matrix
X(1,:) = X(1,:) + 10              %translation operation
x1 = [0.8, -0.6; 0.6, 0.8] * x1   %rotation matrix
plot(x1(1,:),x1(2,:),'x');
set(gca,'ylim',[-25,25],'xlim',[-25,25]);
```

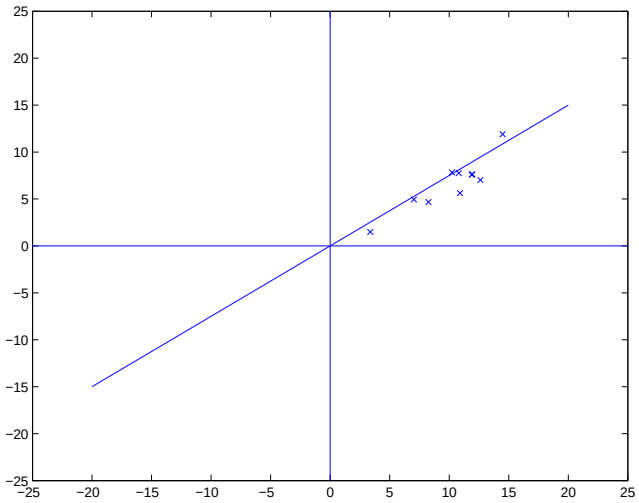


Figure 3: Linear Data?

This is a data set where one dimension (the initial x dimension) is much "larger" than the initial y dimension. We also translate and rotate the data. The figure shows a line passing through the data. We can map the data down on to this line by taking the dot product of the data with the vector producing the line: $\begin{bmatrix} 4 \\ 3 \end{bmatrix}$. This allows us to "compress" the data onto 1 dimension, but we realize that we lose some of the information in the other direction. We still say this data has inherent dimensionality of 1 since there is one direction that contains the preponderance of the information. What exactly do we mean by *information*? And how do we find the line/vector on which to map the data?

We measure the information of the data by the spread – the variance. This captures the data points' average squared distance from the data mean. Our X data set had large spread in the x-direction and small spread in the y-direction (before the rotation operation). The eigenvectors give the direction of maximum spread for data that has been rotated. We compute it using the covariance matrix. But we must center the data first (mean subtract) so that we don't pick up the mean as the first eigenvector.

```
avg = mean(X')';
for i = 1:10, X(:,i) = X(:,i) - avg; end;
hold on; plot(X(1,:),X(2,:),'ro');
```

```
C = X * X';
[Evec, Eval] = eig(C);
v1 = Evec(:,2);
v2 = Evec(:,1);
line([v1(1)*10, v1(1)*-10],[v1(2)*10,v1(2)*-10])
line([v2(1)*3, v2(1)*-3],[v2(2)*3,v2(2)*-3])
```

The blue x's are the original points. The red o's are the points centered (mean subtracted). The two blue lines show the first two principle components, the first two eigenvectors.

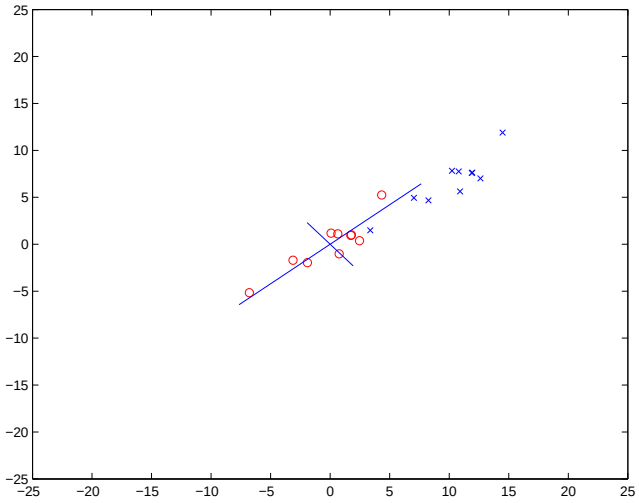


Figure 4: Linear Data?

This process is known by many names, but machine learning folks call it *principle component analysis*, or PCA for short. We find the principle components or principle directions for the data, and then we can project the data down on these directions.

3 A Change of Bases

Our original data set X is cast in the standard basis. We use this basis so regularly that we don't even think about using it. Our space of $\Re^2$ where the data lives

is spanned by the vectors that comprise the x-axis $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and y-axis $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The components of each data point in X are the *coefficients* of these data points in this basis. Here we make that explicit:

$$x^{(1)} = \begin{bmatrix} 3.3717 \\ 1.4793 \end{bmatrix} = 3.3717 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 1.4793 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2}$$

$$x^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3.3717 \\ 1.4793 \end{bmatrix} \tag{3}$$

$$\text{let} \quad \alpha^{(1)} = \begin{bmatrix} \alpha_1^{(1)} \\ \alpha_2^{(1)} \end{bmatrix} = \begin{bmatrix} 3.3717 \\ 1.4793 \end{bmatrix} \tag{4}$$

$$\text{then} \quad x^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1^{(1)} \\ \alpha_2^{(1)} \end{bmatrix} \tag{5}$$

$$\text{and} \quad X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} | & | & & | \\ \alpha^{(1)} & \alpha^{(2)} & ... & \alpha^{(P)} \\ | & | & & | \end{bmatrix} \tag{6}$$

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} A \tag{7}$$

where A is the alpha matrix of coefficients. But we could pick some other basis besides the orthonormal one. Let us pick u_1 and u_2 to span the space. Any two orthogonal vectors will do. Then we could rewrite x_1 in this new space with new coefficients given by α.

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} = \alpha_1^{(i)} u_1 + \alpha_2^{(i)} u_2 \tag{8}$$

$$X = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & ... & u^{(P)} \\ | & | & & | \end{bmatrix} A = UA \tag{9}$$

But what basis, U, should we choose? If our goal is to map data into a lower dimensional space, then we could use the eigenvectors as the basis. Mapping the data onto the eigenvectors captures the maximal information in the residual data at each step. If the data has an inherent dimensionality less than the actual dimensionality, we can ignore some of the the trailing eigenvectors.

4 Mapping

If we use the eigenvectors V as our orthonormal basis, we need to compute the coefficients of the data points. Some simple linear algebra allows us to arrive at:

$$X = VA \tag{10}$$
$$V^{-1}X = V^{-1}VA \tag{11}$$
$$V^{-1}X = A \tag{12}$$

We typically have to compute the pseudo inverse of V instead of V^{-1}. The following Matlab code gives the results. Recall that X contains the mean-centered data. V is the column eigenvectors; notice matlab provides them in reverse order (first eigenvector corresponds to the smallest eigenvalue, the last eigenvector corresponds to the biggest eigenvalue).

```
A = pinv(V) * X;    % compute coefficients for new basis
Xr = V * A;         % compute reconstructed dataset
```

We have used Xr as the reconstructed dataset. We should find that $Xr = X$ as the reconstruction should be perfect (perhaps just a slight bit of round off error might creep in to our calculations).

5 Snapshot Method

Recall that our dataset X is NxP where there are P vectors each living in $\Re^N$. In this last example, we had $P = 10$ and $N = 2$, or more specifically $P > N$. The covariance matrix C is NxN.

In some problem domains, we might find that our data lives in very high dimensional space. That is, we have $N >> P$. This presents an issue because the computation of C becomes prohibitive if N gets too large. For example, suppose we store a series of grayscale images that are 100 x 100. Thus each image has 10,000 pixels and is therefore a 10,000-dimensional vector. We cannot compute the eigenvectors for a matrix that is 10,000 x 10,000.

We also notice another peculiarity. The data only contains P samples, thus we will find (if we could compute the eigenvectors) that most of the eigenvalues are 0. There should be at most P non-zero eigenvalues and their corresponding eigenvectors. We can see this by imagining that we choose as our basis each of the P data points. Thus the coefficients become trivial and we are able to fully reconstruct the original images. Thus there must be at most a P-dimensional subspace in the input

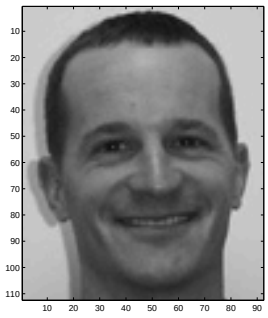
space that spans all of our data. We cannot have data lying in a space greater than P. We can use this observation to get around the computational problem.

Let us create the *sample covariance matrix* $L = C' * C$. This is a PxP matrix. We can compute the eigenvectors of this P-space and then project them back up to N-space to find the coefficients.

We turn to the following real-life example to illustrate the snapshot method in use.

6 Photo Example

We start with a set of digital photographs of two people; there are 10 pictures of each person for a total of 20 images. Though they were taken with a 1 Megapixel camera, each image was downgraded (reduced resolution) to a 112 x 92 pixel grayscale. Thus each input image is a vector living in $N = 112x92 = 10,304$ space and there are $P = 20$ images. Two example images are shown here.



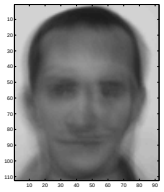
Here we start with the data images in a variable called `set` which is NxP. I have a brief routine called `dispImage` that takes a vector and displays it as an image (see matlab code below).

```
averageFace = mean(set')';
dispImage(averageFace)
for i = 1:20; X(:,i) = set(:,i) - averageFace; end;
dispImage(X(:,1))
dispImage(X(:,11))
C = X' * X;
[Evec, Eval] = eig(C);
```

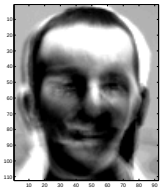
```
diag(Eval)  % see the relative sizes of the various directions
V = X * Evec; % project eigenvectors back to N space
dispImage(V(:,20)+averageFace)  % prints the "eigenface"
dispImage(V(:,19)+averageFace)  % prints the "eigenface"
alpha = pinv(V) * X;  % compute coefficients
Xr = V * alpha;  % compute reconstructed images
%compare original with reconstructed
figure(1); dispImage(X(:,1)+averageFace)
figure(2); dispImage(Xr(:,1)+averageFace)
```

Notice the step where we compute the sample covariance matrix of size PxP. After we get the eigenvectors in P-space, we can project them back to N-space by multiplying with the dataset.

Here we have the average face (left) and two eigenfaces (20 and 19).



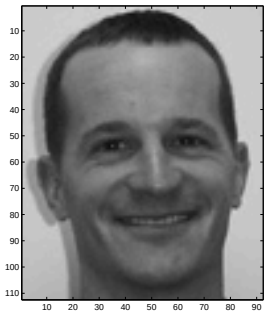
Average



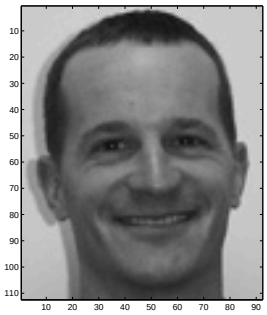
$Eig20$

$Eig19$

Here we have an original image and its reconstruction, do you see any difference?

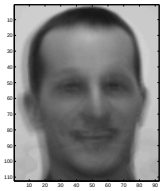


Original

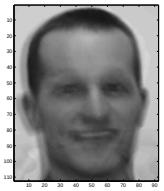


Reconstruction

We can truncate the reconstruction to two terms (using the two highest eigenvalues):



$2 - Term$



$5 - Term$

$10 - Term$

As you can see, we get pretty good results using only half the original dataset! Though the 2-term reconstruction isn't a good image, it has other useful properties. This allows us to graph each image in 2-space using only these two alpha coefficients; person 1 is x's while person 2 is o's. Can you see that the data is linearly seperable? It would be much preferable to train a perceptron for classification in 2-space rather than 10,304 space!

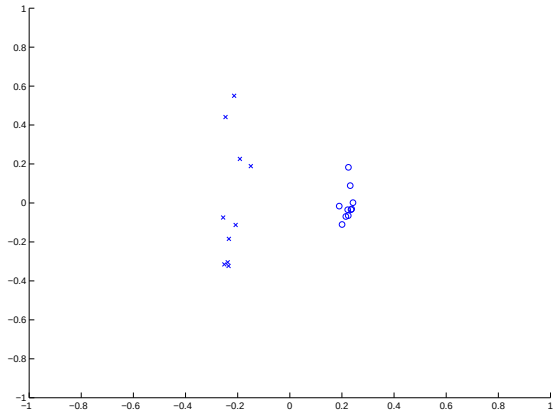


Figure 5: Images plotted using $\alpha^{(1)}$ and $\alpha^{(2)}$