

5 Linear Dimensionality Reduction (PCA)

5.1 Reducing the Problem

The first order of business is to show that instead of considering a general q by p matrix \vec{E} and a general p by q matrix \vec{F} we can restrict ourselves (without loss of generality) to cases where the rows of \vec{E} are orthonormal and $\vec{F} = \vec{E}^T$. What does “without loss of generality” mean? Simply that for *any* pair of matrices \vec{E} and \vec{F} there exists an orthonormal matrix \vec{G} such that if we take $\vec{E} = \vec{G}$ and $\vec{F} = \vec{G}^T$ then the total reconstruction error using \vec{E} and \vec{F} is less than or equal to the total reconstruction error if we had used the original \vec{E} and \vec{F} .

1. There are several ways of showing this. One way is to notice that we can scale the rows of our original matrix \vec{E} by another matrix \vec{K} shown below. This scaling can produce a matrix whose rows are unit vectors if the k values are properly chosen. Notice also that this matrix is invertible. Now we can show that for any original matrix \vec{E} for which there existed a corresponding matrix \vec{F} , we can create a new matrix \vec{E}_{new} with unit vectors as rows and corresponding matrix \vec{F}_{new} , and that the new pair of matrices will perform the same function as the old pair. We do this by letting $\vec{E}_{new} = \vec{K}\vec{E}$ and $\vec{F}_{new} = \vec{F}\vec{K}^{-1}$.

$$\vec{K} = \begin{pmatrix} 1/k_1 & 0 & \dots & 0 \\ 0 & 1/k_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/k_q \end{pmatrix} \vec{K}^{-1} = \begin{pmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & k_q \end{pmatrix} \quad (1)$$

$$\vec{F}_{new}\vec{E}_{new}\vec{x} \quad (2)$$

$$= (\vec{F}\vec{K}^{-1})(\vec{K}\vec{E})\vec{x} \quad (3)$$

$$= \vec{F}\vec{E}\vec{x} \quad (4)$$

$$= \hat{x} \quad (5)$$

2. There are several ways of phrasing this:

- We are projecting from one subspace to an equal- or lesser-dimensional subspace, with q dimensions. To do this, we can use any matrix \vec{E} whose rows span that subspace (whether the rows are orthogonal or not). Can we always find a corresponding \vec{F} , however? Yes, because as long as you haven't reduced the number of linearly independent dimensions onto which your matrix projects, you can always tack on a rotation transformation to rotate your old \vec{F} into your preferred \vec{F}_{new} : $\vec{F}_{new} = \vec{F}\vec{T}$.
- Consider the subspace spanned by the columns of \vec{F} . The reconstructed vector \tilde{x} always lies somewhere in this subspace (that's what "span" means). Which point within this subspace is the *best* reconstruction in the squared error sense? Clearly it is the *perpendicular projection* of the original point into this subspace. No matter what compression matrix \vec{E} is used with this \vec{F} , it can never do better than making \tilde{x} equal to this projection. But we can show an easy way to do this using a matrix $\vec{E} = \vec{G}$ with orthonormal rows and a matrix $\vec{F} = \vec{G}^T$. Here's how: take the original matrix \vec{F} . Construct an orthonormal basis for the space spanned by its columns. Use these basis vectors as the rows of $\vec{E} = \vec{G}$ and as the columns of $\vec{F} = \vec{G}^T$. Now what is the reconstruction $\tilde{x} = \vec{G}^T \vec{G}\vec{x}$ of any point \vec{x} ? Just the projection of \vec{x} into the space of \vec{G} which we know is just as good as we could ever have done with the original \vec{F} and any \vec{E} .

The proof is as follows: The Gram-Schmidt process tells us that a set of n linearly independent vectors can be represented by another set of n orthonormal vectors spanning the same space. This doesn't change our vectors; it only changes the basis with which we represent the vectors.

To put this in equation form, we use a derivative of Gram-Schmidt (sometimes named "QR Factorization") which states that if you take a matrix \vec{E}^T which you hope to orthogonalize, you can separate it into two matrices $\vec{E}^T = \vec{Q}\vec{R}$ where \vec{Q} is a $p \times q$ matrix whose columns form an orthonormal basis for the column space of \vec{E} , and \vec{R} is a $q \times q$ upper

triangular invertible matrix with positive entries on the diagonal. Now:

$$\vec{E} = (\vec{Q}\vec{R})^T = \vec{R}^T\vec{Q}^T \quad (6)$$

$$\hat{x} = \vec{F}\vec{E}\vec{x} = \vec{F}(\vec{R}^T\vec{Q}^T)\vec{x} \quad (7)$$

$$(8)$$

Now if you let $\vec{F}_o = \vec{F}\vec{R}^T$ and $\vec{E}_o = \vec{Q}^T$ such that:

$$\hat{x} = \vec{F}_o\vec{E}_o\vec{x} \quad (9)$$

We already said that \vec{Q} would have orthogonal columns, so \vec{Q}^T has orthogonal rows, and thus $\vec{E}_o = \vec{Q}^T$ is orthogonal as desired. And we've shown how to create a corresponding \vec{F}_o .

3.

$$\vec{E} = \begin{pmatrix} \vec{e}_1 \\ \vec{e}_2 \\ \vdots \\ \vec{e}_q \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & & 0 \\ 0 & 0 & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} \vec{e}_1 \\ \vec{e}_2 \\ \vdots \\ \vec{e}_q \\ \vdots \\ \vec{e}_p \end{pmatrix} = \vec{P}\vec{R} \quad (10)$$

Where \vec{P} is the projection matrix from p dimensions to q dimensions, and \vec{R} is a unitary rotation matrix (where all the \vec{e}_i are orthonormal). Since \vec{E} can be written as the product of a projection and a rotation matrix, we can see that \vec{E} does in fact perform a unitary transformation and projection onto the first q vectors of the coordinate system.

5.2 Picking a good rotation

We have shown that we only need to consider compression matrices \vec{E} with orthonormal rows and reconstruction matrices $\vec{F} = \vec{E}^T$; the optimal linear transformation can always be written as a rotation and/or flip followed by chopping off some components. We now want to ask what the optimal rotation for \vec{E} is, in other words what directions should the rows of \vec{E} have?

1. We want to undo *exactly* what we have done. So the best thing to do is to reverse our previous steps by using $\vec{F} = \vec{E}^{-1}$. Since we have already shown that we can assume that \vec{E} is orthonormal, and that

assumption implies that $\vec{E}^T = \vec{E}^{-1}$, we would like to use $\vec{F} = \vec{E}^T$. With this choice of \vec{F} : $\hat{x} = \vec{F}\vec{E}\vec{x} = \vec{E}^T\vec{E}\vec{x} = \vec{E}^{-1}\vec{E}\vec{x} = \vec{x}$ and the error is zero.

Numerically, we would like to find where the square error is minimum, or equivalently where the first derivative of this error is zero:

$$\begin{aligned}
err &= (\vec{x} - \hat{x})^T (\vec{x} - \hat{x}) \\
&= (\vec{x} - \vec{F}\vec{E}\vec{x})^T (\vec{x} - \vec{F}\vec{E}\vec{x}) \\
&= (\vec{x}^T - \vec{x}^T \vec{E}^T \vec{F}^T) (\vec{x} - \vec{F}\vec{E}\vec{x}) \\
&= \vec{x}^T \vec{x} - \vec{x}^T \vec{F}\vec{E}\vec{x} - \vec{x}^T \vec{E}^T \vec{F}^T \vec{x} + \vec{x}^T \vec{E}^T \vec{F}^T \vec{F}\vec{E}\vec{x} \quad (11)
\end{aligned}$$

$$\frac{d}{d\vec{F}} err = -\vec{x}^T \vec{E}\vec{x} + \vec{x}^T \vec{E}^T \vec{F}^T \vec{E}\vec{x} = 0 \quad (12)$$

$$\begin{aligned}
\vec{x}^T \vec{E}\vec{x} &= \vec{x}^T \vec{E}^T \vec{F}^T \vec{E}\vec{x} \\
\vec{E} &= \vec{E}^T \vec{F}^T \vec{E} \\
\vec{E}\vec{E}^T &= \vec{E}\vec{E}^T \vec{F}^T \vec{E}\vec{E}^T \\
\vec{E} &= \vec{F}^T \\
\vec{E}^T &= \vec{F} \quad (13)
\end{aligned}$$

where in the last few steps we have made use of the assumption of orthonormality, which allows us to cancel out $\vec{E}\vec{E}^T = \vec{I}$.

- Remember we're only solving for the case where $q = 1$ (we project onto a single direction, namely \vec{E} is a row vector). Writing the total error over the whole dataset we get:

$$\begin{aligned}
\sum_{i=1}^N \|\vec{x}^i - \hat{x}^i\|^2 &= \sum_{i=1}^N \|\vec{x}^i - \vec{E}^T \vec{E}\vec{x}^i\|^2 \\
&= \sum_{i=1}^N (\vec{x}^i - \vec{E}^T \vec{E}\vec{x}^i)^T (\vec{x}^i - \vec{E}^T \vec{E}\vec{x}^i) \quad (14)
\end{aligned}$$

Now we expand out this quadratic form and do some simplification. An important things to remember is that $\vec{E}\vec{E}^T = 1$ and $\vec{E}\vec{x} = \vec{x}^T \vec{E}^T$.

Here are the details (we have left out the index i for notational clarity):

$$\begin{aligned}
& \sum (\vec{x} - \vec{E}^T \vec{E} \vec{x})^T (\vec{x} - \vec{E}^T \vec{E} \vec{x}) \\
& \Rightarrow \sum (\vec{x}^T - \vec{x}^T \vec{E}^T \vec{E}) (\vec{x} - \vec{E}^T \vec{E} \vec{x}) \\
& \Rightarrow \sum \left[\vec{x}^T \vec{x} - \vec{x}^T \vec{E}^T \vec{E} \vec{x} - \vec{x}^T \vec{E}^T \vec{E} \vec{x} + \vec{x}^T \vec{E}^T \vec{E} \vec{E}^T \vec{E} \vec{x} \right] \\
& \Rightarrow \sum \left[\vec{x}^T \vec{x} - \vec{x}^T \vec{E}^T \vec{E} \vec{x} \right] \\
& \Rightarrow \sum \left[\|\vec{x}\|^2 - \vec{E} \vec{x} \vec{x}^T \vec{E}^T \right] \\
& \Rightarrow N \left[\langle \|\vec{x}\|^2 \rangle - \vec{E} \langle \vec{x} \vec{x}^T \rangle \vec{E}^T \right] \\
& \Rightarrow N \left[\langle \|\vec{x}\|^2 \rangle - \vec{E} \vec{C} \vec{E}^T \right]
\end{aligned}$$

since the covariance matrix C (again leaving out the superscript on x^i is:

$$C = \sum_{i=1}^N \frac{1}{N} x x^T \quad (15)$$

3. We want to pick \vec{E} to **minimize** this error subject to the constraint that \vec{E} is a unit vector. Since $\langle \|\vec{x}\|^2 \rangle$ does not depend on \vec{E} (and neither does N !) this is equivalent to picking \vec{E} to **maximize** the term $\vec{E} \vec{C} \vec{E}^T$ subject to $\vec{E} \vec{E}^T = 1$, or equivalently $\vec{E} \vec{E}^T - 1 = 0$ (this is the constraint denoted G in the footnote of your problem set). Using Lagrange multipliers, we set the gradient of the term we want to maximize equal to some multiple of the gradient of the constraint:

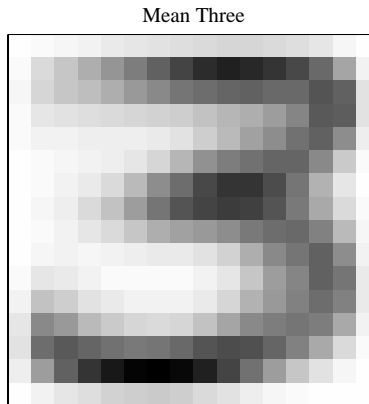
$$\begin{aligned}
\frac{\delta F}{\delta x} &= \lambda \frac{\delta G}{\delta x} \\
\nabla_{\vec{E}} \vec{E} \vec{C} \vec{E}^T &= \lambda \nabla_{\vec{E}} (\vec{E} \vec{E}^T - 1) \\
\vec{C} \vec{E}^T &= 2\lambda \vec{E}^T
\end{aligned}$$

(where the gradients are with respect to the elements of \vec{E}). What does the result above tell us? That that *all local extrema of $\vec{E} \vec{C} \vec{E}^T$ occur when \vec{E} is an eigenvector of the covariance matrix \vec{C}* . From which it follows that the Lagrangian multiplier λ is the corresponding eigenvalue. So we know \vec{E} has to be an eigenvector, the only question is which one? We want to maximize $\vec{E} \vec{C} \vec{E}^T = \vec{E} (\vec{C} \vec{E}^T) = \vec{E} (\lambda \vec{E}^T) = \lambda \vec{E} \vec{E}^T = \lambda$. So whichever eigenvector we pick to be \vec{E} , we would like to maximize its corresponding λ ; i.e. the answer is to make \vec{E} point in the direction of the eigenvector of \vec{C} with the largest eigenvalue.

5.3 PCA on handwritten digits

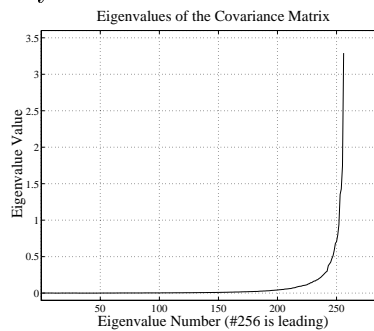
Throughout, keep in mind that the matrix `threes` has one picture per row (a 16 by 16 image expanded into a 256 wide row vector).

1. The “mean three” can be obtained in MATLAB by saying `M=mean(threes)` and looks like this:



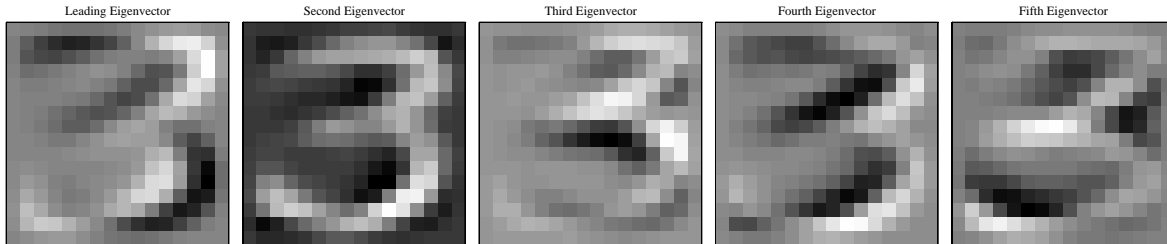
2. To get the covariance matrix, simply say `C=cov(threes)` and then use `[V,D]=eig(C)` to get its eigenvalues and eigenvectors. Notice that `eig` returns the eigenvectors in the *columns* of `V` and the eigenvalues on the *diagonal* of `D` and also that the leading eigenvector/value occupy the last column instead of the first.

Here is a plot of the eigenvalues from `plot(diag(D))`, you can see that they drop off quickly:



3. Here are the first few eigenvectors (namely columns 256,255,254, etc. from `V`). Remember that these show the directions of maximum vari-

ation in the dataset of all three:



- Now to do compression and reconstruction, all we need to do is multiply the (row) vector to be compressed by the last few columns of V (that's the compression) and then the transpose of those columns (reconstruction). We have to remember to subtract the mean vector off before we compress and add it back in before we reconstruct.

Here is a function which does exactly that (it works for both single row vectors and for matrices where each row is to be compressed and reconstructed).

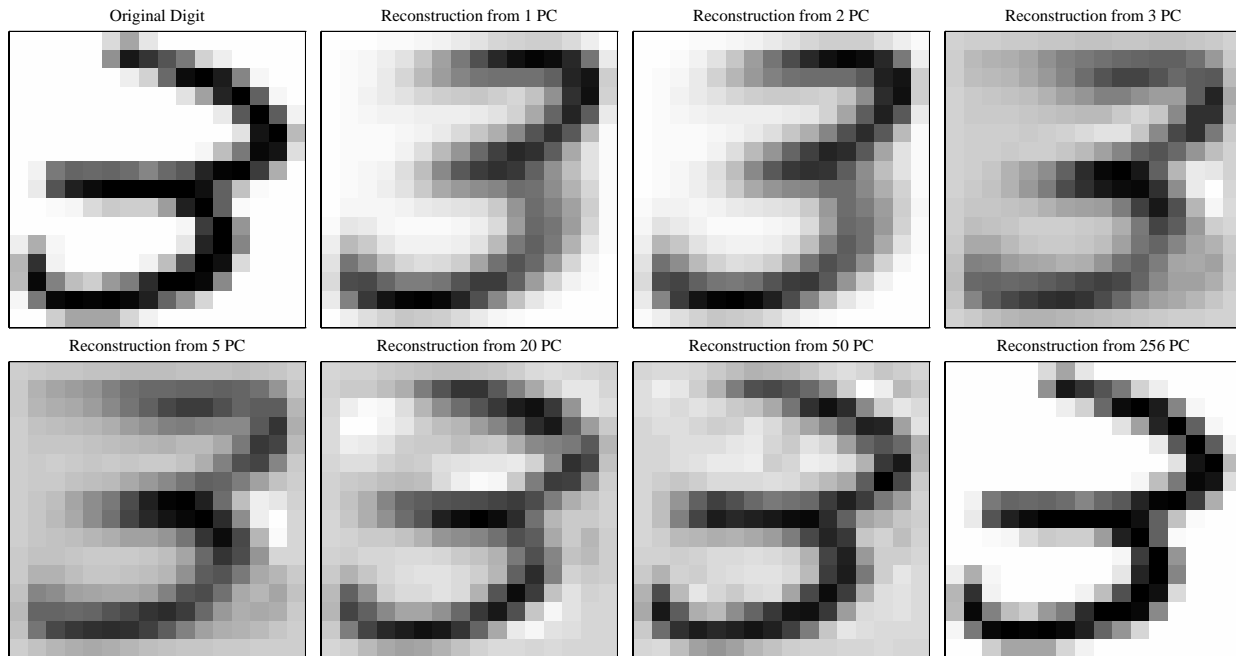
```
function [new_vecs] = reconvec(orig_vecs,mean_vec,V,compressed_dimension)
% [new_vecs] = reconvec(orig_vecs,mean_vec,V,compressed_dimension)
%
% RECONVEC compresses and reconstructs a vector (or set of vectors)
%         in orig_vecs by projecting onto the first compressed_dimension
%         principal components.
%
% orig_vecs are the original vectors, one per row
% mean_vec is the mean vector of the dataset, a row vector
% V is the matrix of eigenvectors of the covariance matrix, V=cov(all_data);
% compressed_dimension is the number of principal components to use

[num_vecs,dimens_vecs] = size(orig_vecs);

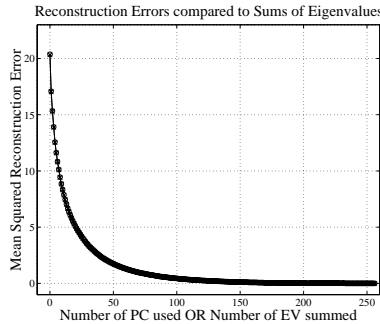
X = (orig_vecs-ones(num_vecs,1)*mean_vec)'; % subtract off mean from each row
E = fliplr(V(:,(dimens_vecs-
    compressed_dimension+1):dimens_vecs))'; % make E from the last
                                                % compressed_dimension
                                                % columns of V
Z = E*X;                                     % compress
Xhat = E'*Z;                                 % reconstruct
newvecs = Xhat'+ones(numvecs,1)*mean_vec;   % add mean back in
```

If we do that to the first three in the dataset, here is what we get for various numbers of principal components used. Notice that if we use

all the principal components, the reconstruction is perfect as we would expect:



5. If we run the above function on the entire matrix `threes` then we get the whole dataset compressed and reconstructed. We can check the mean squared reconstruction error by saying `sum(sum((threeshat-threes).^2))/500`. If we do this for various values of k (called `compressed_dimension` in the function above) then we can make a plot of the error as a function of the number of principal components used during the compression and reconstruction. Using *zero* principal components corresponds to just guessing the mean three as our reconstruction no matter what the original was. This should in theory give us the maximum reconstruction error which should be proportional to the total variance of the dataset.



6. Using 256 principal components corresponds to perfect reconstruction and so in theory should give us zero error¹ which compared to the scale of errors being about 10 is quite good. The difference is just numerical noise in MATLAB. We can predict what this plot should be for any intermediate value of k by noticing² that the following very interesting and important fact: the squared reconstruction error incurred by *not using* any principal component is proportional to its eigenvalue. That is why if the eigenvalues fall off quickly then projecting onto the first few gives very small error because the sum of the eigenvalues we are not using is not very large.
7. We can compare our first plot to a plot of a vector whose i^{th} element is the sum of *all but* the largest i eigenvalues. When $i = 0$ this is just the sum of all the eigenvalues – this corresponds to using zero principal components, i.e. just guessing the mean. When $i = 256$ it is just zero – this corresponds to using all the principal components and getting perfect reconstruction. In the figure above, you can see that the theoretical prediction agrees excellently with actual error values (to within numerical roundoff errors).

¹In fact it gives a value of $3.1616\text{e-}28$

²This is not immediately obvious but it is easy to prove if you think about the relationship between variance and squared error.