Chapter 6

The Best Basis

In this chapter, we will consider the problem of compact data representations. In particular, we will look at the implications of defining what we mean by "best", and the resulting expansion is the Karhunen-Loéve (KL) expansion. This goes by several other names, but they all refer to the same basic procedure: KL, Principal Components Analysis (PCA), Principal Orthogonal Decomposition (POD).

In this chapter, we will see that not only does the SVD of a matrix give *a* basis for the four fundamental subspaces, but it actually gives us the *best* basis for these subspaces.

6.1 The Karhunen-Loéve Expansion

Let us assume we have a set of data,

$$\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(p)}\right\}$$

where each $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$. Furthermore, let X be the $p \times n$ matrix formed from the data:

$$X = [\boldsymbol{x}^{(1)} \ \ldots \ \boldsymbol{x}^{(p)}]^T$$

Given any basis of $\mathbb{R}^n$, say the vectors in $\Phi = [\boldsymbol{\phi}_1 \ \boldsymbol{\phi}_2 \ \ldots \boldsymbol{\phi}_n]$, we can expand any $\mathbf{x} \in \mathbb{R}^n$ in terms of this basis:

$$\boldsymbol{x} = \sum_{k=1}^{n} \alpha_k \boldsymbol{\phi}_k = \Phi\vec{\alpha} \tag{6.1}$$

We will assume our basis is orthonormal, so that the coefficients α_k are easy to compute using the inner product:

$$\alpha_k = \boldsymbol{\phi}_k^T \boldsymbol{x} \quad \Rightarrow \quad \vec{\alpha} = \begin{bmatrix} \boldsymbol{\phi}_1^T \\ \boldsymbol{\phi}_2^T \\ \vdots \boldsymbol{\phi}_n^T \end{bmatrix} \mathbf{x} = \Phi^T \boldsymbol{x}$$

Notice that if we now substitute this expression in for $\vec{\alpha}$ in Equation 6.1, we get:

$$\boldsymbol{x} = \Phi\Phi^T \boldsymbol{x}$$

Recall that if Φ is a square matrix, then $\Phi\Phi^T$ is the identity, but if it is not, then this represents a projection of $\mathbf{x}$ into the subspace spanned by the columns of Φ.

Let's consider this second example- Suppose we are interested in the subspace spanned by the first k columns of Φ (where $k << n$). Then we can write any point $\mathbf{x}$ in the usual way, but split the expression after the k^{th} column:

$$\boldsymbol{x} = \sum_{j=1}^{k} \alpha_j \boldsymbol{\phi}_j + \sum_{j=k+1}^{n} \alpha_j \boldsymbol{\phi}_j$$

then the second term of the sum is our **error** in using k basis vectors to express $\mathbf{x}$:

$$\boldsymbol{x}_{\text{err}} = \sum_{j=k+1}^{n} \alpha_j \phi_j \quad \Rightarrow \quad \|\boldsymbol{x}_{\text{err}}\|^2 = \alpha_{k+1}^2 + \cdots + \alpha_n^2$$

Now we're ready to proceed with determining the best basis. The assumptions are:

1. Since the best basis will naturally have its origin at the centroid of the data, **we will assume that the data has been mean subtracted**.

2. The best basis should be orthonormal.

3. The data does not take up all of $\mathbb{R}^n$- It lies in some linear subspace.

4. If we have p data points, the error using k columns will be defined by the mean squared error of the data expansion:

$$\text{Error} = \frac{1}{p} \sum_{j=1}^{p} \|\boldsymbol{x}_{\text{err}}^{(j)}\|^2 \tag{6.2}$$

Now let us examine the error, Equation 6.2. We will construct the best basis by means of the following exercises:

6.2 Exercises: Finding the Best Basis

1. Show that one coordinate for the data point $\mathbf{x}$ can be written as:

$$\alpha_j^2 = \boldsymbol{\phi}_j^T \boldsymbol{x} \boldsymbol{x}^T \boldsymbol{\phi}_j$$

SOLUTION: From the equation following 6.1,

$$\alpha_j^2 = (\boldsymbol{\phi}_j^T \boldsymbol{x})^2 = \boldsymbol{\phi}_j^T \boldsymbol{x} \boldsymbol{\phi}_j^T \boldsymbol{x} = \boldsymbol{\phi}_j^T \boldsymbol{x} \boldsymbol{x}^T \boldsymbol{\phi}_j$$

2. Show that the inner product and sum can be switched. That is, for a single vector $\boldsymbol{\phi}$, show that we can write:

$$\sum_{i=1}^{p} \boldsymbol{\phi}^T \boldsymbol{x}^{(i)} \left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{\phi} = \phi^T \left[\sum_{i=1}^{p} \boldsymbol{x}^{(i)} \left(\boldsymbol{x}^{(i)}\right)^T\right] \boldsymbol{\phi}$$

3. Show that we can write the covariance matrix of X in the following way:

$$C = \frac{1}{p} X^T X = \sum_{i=1}^{p} \boldsymbol{x}^{(i)} \left(\boldsymbol{x}^{(i)}\right)^T$$

We're now just about ready to find the best basis. First, let's summarize what we have shown in the previous exercises:

$$\begin{aligned}
\frac{1}{p}\sum_{i=1}^{p}\|\boldsymbol{x}_{\text{err}}^{(i)}\|^2 &= \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{j=k+1}^{n}\left(\alpha_j^{(i)}\right)^2\right) \\
&= \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{j=k+1}^{n}\boldsymbol{\phi}_j^T\boldsymbol{x}^{(i)}\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{\phi}_j\right) \\
&= \sum_{j=k+1}^{n}\left(\frac{1}{p}\sum_{i=1}^{p}\boldsymbol{\phi}_j^T\boldsymbol{x}^{(i)}\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{\phi}_j\right) \\
&= \sum_{j=k+1}^{n}\boldsymbol{\phi}_j^T\left[\frac{1}{p}\sum_{i=1}^{p}\boldsymbol{x}^{(i)}\left(\boldsymbol{x}^{(i)}\right)^T\right]\boldsymbol{\phi}_j \\
&= \sum_{j=k+1}^{n}\boldsymbol{\phi}_j^T C\boldsymbol{\phi}_j
\end{aligned}$$

Conclusion so far: To minimize the mean-squared error on the $k-$term expansion $\{\boldsymbol{\phi}_i\}_{i=1}^k$, we need to minimize:

$$\sum_{j=k+1}^{n}\boldsymbol{\phi}_j^T C\boldsymbol{\phi}_j$$

with the constraints that the ϕ_i form an orthonormal set.

One way we might do this is to find the best 1-dimensional subspace first, then reduce the dimensionality of the data by 1. Now find the best 1-dimensional subspace for the remaining data, and so on. To do this, we minimize the quantity we found:

$$\min_{\phi_2,\cdots,\phi_n}\sum_{j=2}^{n}\boldsymbol{\phi}_j^T C\boldsymbol{\phi}_j$$

and since the basis is orthonormal, this is equivalent to:

$$\max_{\phi_1\neq 0}\frac{\boldsymbol{\phi}_1^T C\boldsymbol{\phi}_1}{\boldsymbol{\phi}_1^T\boldsymbol{\phi}_1}$$

Theorem: The maximum (over non-zero ϕ) of the quantity

$$\frac{\phi^T C\phi}{\phi^T\phi}$$

where C is the covariance matrix of a data set X occurs at $\phi = \boldsymbol{v}_1$, the first eigenvector of C, where λ_1 is the largest eigenvalue of C.

The proof will proceed as several exercises:

1. Justify the following:

   Let $\{\lambda_i\}_{i=1}^n$, $\{\boldsymbol{v}_i\}_{i=1}^n$, be the eigenvalues and eigenvectors (respectively) of the covariance matrix C, where

   $$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$$

   Then we can write C as:

   $$C = \lambda_1\boldsymbol{v}_1\boldsymbol{v}_1^T + \ldots + \lambda_n\boldsymbol{v}_n\boldsymbol{v}_n^T = V\Lambda V^T$$

   with $V^TV = VV^T = I$.

2. We can write any vector ϕ as:
$$\phi = a_1\boldsymbol{v}_1 + \ldots + a_n\boldsymbol{v}_n = V\boldsymbol{a}$$

3. From the previous exercise, show that:
$$\phi^T\phi = a_1^2 + a_2^2 + \ldots + a_n^2$$

4. Show that:
$$\phi^T C\phi = \lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_n a_n^2$$

5. Justify the following: For any nonzero vector ϕ,
$$\frac{\phi^T C\phi}{\phi^T\phi} = \frac{\lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_n a_n^2}{a_1^2 + c_2^2 + \ldots + a_n^2} \leq \lambda_1$$
with equality if and only if $\phi = \boldsymbol{v}_1$.
By the way, this works in both directions:
$$\lambda_n = \lambda_{\min} \leq \frac{\phi^T C\phi}{\phi^T\phi} =\leq \lambda_{\max} = \lambda_1$$

So far, we have shown how to find the best one dimensional basis vector! How would we go about finding the next one? Formally, we would say that the next basis vector satisfies the following:
$$\max_{\phi\perp\boldsymbol{v}_1} \frac{\phi^T C\phi}{\phi^T\phi}$$
which, by the same argument as in the previous theorem, is going to be the second eigenvector of C- this is a consequence of the Spectral Theorem, which says that our eigenvectors are orthogonal.

Now we're ready for the big pay-off:

The Best Basis Theorem: Suppose that:

- X is an $p \times n$ matrix of p points in $\mathbb{R}^n$.
- X_m is the $p \times n$ matrix of mean-subtracted data.
- C is the covariance of X, $C = \frac{1}{p}X_m^T X_m$

Then the best (in terms of the mean-squared error defined earlier) orthonormal $k-$dimensional basis is given by the leading k eigenvectors of C, for any k.

Definition: The Karhunen-Loéve (KL) expansion vectors are the eigenvectors of the covariance. Therefore, the statement "Put the data into its KL expansion" means to find the eigenvectors of the covariance, and use these as the basis vectors. These vectors are also known as the Principal Components of the data (and so KL expansion is also called Principal Component Analysis, or PCA). You might infer this from the name, but this type of expansion has also been called Principal Orthogonal Decomposition (POD).

6.3 Connections to the SVD

Consider the (thin) SVD of the matrix X,
$$X = U\Sigma V^T$$
so that Σ is $k \times k$, where k is the rank of X. By formulating the covariance, we see that:
$$C = \frac{1}{p}X^T X = \frac{1}{p}V\Sigma U^T U\Sigma V^T = V\left(\frac{1}{p}\Sigma^2\right)V^T$$
Comparing this to the Best Basis Theorem, we see that:

The best basis vectors for the rowspace of X are the right singular vectors for X.

Similarly, what happens if we formulate the other covariance (thinking of X, which is $p \times n$, as n points in $\mathbb{R}^p$):

$$C_2 = \frac{1}{n} XX^T = \frac{1}{n} U\Sigma V^T V\Sigma U^T = U\left(\frac{1}{n}\Sigma^2\right)U^T$$

Again, comparing this to the Best Basis Theorem, we see that:

The best basis vectors for the columnspace of X are the left singular vectors for X.

Furthermore, the SVD gives a beautiful symmetry to the problem: That is, knowledge of the best basis vectors of the rowspace gives the best basis vectors of the columnspace (and vice-versa) by the relationships:

$$X\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i \qquad X^T\boldsymbol{u}_i = \sigma_i \boldsymbol{v}_i$$

where σ_i can be computed from the eigenvalues of the covariance:

$$\lambda_i = \frac{\sigma_i^2}{p}, \text{ or } \lambda_i = \frac{\sigma_i^2}{n}$$

This author has seen this symmetry relegated to being referred to as a "trick" in at least one engineering paper!

Additional Exercises, Best Basis

1. If we need to compute an arbitrary orthonormal basis for $\mathbb{R}^n$, we can use the QR algorithm[1]. In Matlab, this is straightforward:

   ```
   %Let n be assigned the desired dimension
   A=randn(n,n);
   [Q,R]=qr(A);
   %The columns of Q are the orthonormal basis
   ```

   Implement this as a new function, `getbasis.m`, where you input the dimension and the number of basis vectors, and these are returned as the columns of a matrix.

2. If the data takes up all of $\mathbb{R}^n$, then any basis of $\mathbb{R}^n$ will suffice- there are no easier ones to deal with than the standard basis. What makes the best basis problem non-trivial is that we assume that our data does not take up all of $\mathbb{R}^n$.

6.4 Computation of the Rank

Most of the time, we will not have zero eigenvalues to drop for the rank. Therefore, we need to decide on a couple of issues:

- If there is a large gap in the graph of the eigenvalues, we'll use that as our rank (the index of the eigenvalue just prior to the gap).
- The unscaled eigenvalues are hard to compare- we've seen that they might be scaled by p or n, and the exact value is not as important as their relative sizes. We should always consider the normalized eigenvalues- let $\tilde{\lambda}_i$ be the unscaled eigenvalues of C. Then:

  $$\lambda_i = \frac{\tilde{\lambda}_i}{\sum_{j=1}^n \tilde{\lambda}_j}$$

  are the normalized eigenvalues of C. Note that they are all positive, and sum to one.

[1] We'll recall that the QR algorithm is just the matrix version of the Gram-Schmidt orthogonalization procedure.

- The normalized eigenvalue λ_i can be interpreted as the percentage of the total variance (or total energy) captured by the i^{th} eigenspace (the span of i^{th} eigenvector). Some people refer to this as the percentage of total energy captured by the i^{th} eigenspace.
- Using the previous ideas, we can think of the dimension as a function of the total energy (or total variance) that we want to encapsulate in a $k-$dimensional subspace.
- Define the KL dimension as the number of eigenvectors required to explain a given percentage of the energy:

  $$\text{KLD}_d = k$$

  where k is the smallest integer so that:

  $$\lambda_1 + \ldots + \lambda_k \geq d$$

  (Recall that these are the normalized, ordered eigenvalues!)
- The value you choose for d will be problem dependent. For example, if your data is the graph of a smooth function with no noise, you might choose $d = 0.99$. If there is a lot of noise corrupting the data, you might choose $d = 0.6$. The choice of d is not a simple one, and is the topic of some current research.

6.5 Matlab and the KL Expansion

As we have seen in the previous sections, there are many ways that we might compute the KL basis vectors. We'll look at several, and discuss the pros and cons.

KL using the SVD

In this case, if X has been previously loaded, and we assume our data is organized row-wise (X is $p \times n$), then we could use the following to compute the best basis (KL basis):

```
[p,n]=size(X);
mx=mean(X);
Xm=X-repmat(mx,p,1);
[U,S,V]=svd(Xm,0);
ss=diag(S)./sum(diag(S));  %Compute normalized singular values
ss=ss.^2;  %We want to compare to eigenvalues of Covariance
```

Some comments on this code:

- It is critical to mean subtract your data! Otherwise, your first eigenvector (right singular vector) points to the mean, which is a wasted dimension.
- The additional argument (0) in the SVD command can be used when p is much larger than n. Matlab will only return the first n columns of U, instead of the full $p \times p$ matrix U.
- The basis for the data is taken from the first k columns of V, where k is the dimension you want.
- To visualize the singular values, plot them- it's convenient to also plot the point values: `plot(ss,'k-*');`
- You can use `cumsum(ss)` to look at the cumulative sums of the normalized singular values to see how many you want to use.

This method will work well for "small" data sets, but we're working harder than necessary, since we don't use the vectors U. This is a very accurate method, however.

Matlab using the Covariance

In this case, we will explicitly compute the covariance matrix, although some authors warn of the possibility of introducing inaccuracies. This is true, but in some cases (for very large p, for example) is necessary.

The Matlab code will then look like:

```
[p,n]=size(X);
mx=mean(X);
Xm=X-repmat(mx,p,1);
C=(1/p)*Xm'*Xm;
[V,S]=eig(C);
[V,S]=sorteig(V,S);
ss=diag(S)./sum(diag(S));
```

There is a problem here- the eigenvalues do not come back ordered. The `sorteig` command is not a built-in Matlab function- we have to write it ourselves. It's a function that will sort the eigenvalues and eigenvectors out for us:

```
function [V1,L1]=sorteig(V,L)
%Sorts the eigenvalues in L to descending order, re-orders
%the eigenvectors in V accordingly
lambda=diag(L);
n=length(lambda);
[val,idx]=sort(lambda);
val=val(n:-1:1);
idx=idx(n:-1:1);
L1=diag(val);
V1=V(:,idx);
```

Another way around this problem is not to use the `eig` command. We might, for example, use the `svd` command. In this case, we would have:

```
[p,n]=size(X);
mx=mean(X);
Xm=X-repmat(mx,p,1);
C=(1/p)*Xm'*Xm;
[V,S,V]=svd(C);
ss=diag(S)./sum(diag(S));
```

where the diagonal elements of S come out of the SVD in a decreasing fashion already. Note that the SVD of the covariance is the eigenvector decomposition...

Matlab when the Basis is large

Lastly, consider the case where n is very much larger than p (the dimension is much larger than the number of points). We may not want to compute the vectors $\boldsymbol{v}_i$ directly. However, we can use the properties of the SVD to make the problem much faster:

```
[p,n]=size(X);
mx=mean(X);
Xm=X-repmat(mx,p,1);
C1=Xm*Xm';
[V,S]=eig(C);  %These are the right sing. vecs
[V,S]=sorteig(V,S);
U=X*V;  %These are the left sing vecs, unscaled
```

6.6 The Details

We give a secondary argument that the KL basis for the $D-$term expansion forms the best $D-$dimensional subspace for the data (in terms of minimizing the MSE). We do not argue that this basis is the *only* basis for which this is true- indeed, as a $D-$dimensional subspace, any orthonormal basis will produce the same MSE.

First, we saw that, for any orthonormal basis $\{\psi_i\}_{i=1}^n$

$$\frac{1}{p}\sum_{i=1}^{p}\|\boldsymbol{x}^{(i)}\| = \sum_{j=1}^{D}\psi_j^T C\psi_j + \sum_{j=D+1}^{n}\psi_j^T C\psi_j$$

and this sum is a constant. Therefore, minimizing the second term of the sum is the same as maximizing the first term of the sum. Using the eigenvector basis, we get that:

$$\sum_{j=1}^{D}\phi_j^T C\phi_j = \lambda_1 + \lambda_2 + \ldots + \lambda_D$$

and we can evaluate the second term directly as well:

$$\frac{1}{p}\sum_{i=1}^{p}\|\boldsymbol{x}_{\text{err}}^{(i)}\|^2 = \sum_{j=D+1}^{n}\phi_j^T C\phi_j = \lambda_{D+1} + \ldots + \lambda_n$$

We also stated that first D eigenvectors of C form the best basis over all $D-$term expansions, and we will prove that now:

Let $\{\psi_i\}_{i=1}^D$ be any other $D-$dimensional basis. We can write each ψ_i in terms of its coordinates with respect to the eigenvectors of C,

$$\psi_i = \sum_{k=1}^{n}\left(\psi_i^T\phi_k\right)\phi_k = \Phi\boldsymbol{\alpha}_i$$

so that $\alpha_{ik} = \psi_i^T\phi_k$. Given this, we see that:

$$\psi_i^T C\psi_i = \boldsymbol{\alpha}_i^T\Lambda\boldsymbol{\alpha}_i$$

Now, consider the mean squared projection of the data onto this $D-$ dimensional basis:

$$\sum_{j=1}^{D}\psi_j^T C\psi_j = \sum_{j=1}^{D}\boldsymbol{\alpha}_j^T\Lambda\boldsymbol{\alpha}_j = \sum_{j=1}^{D}\lambda_1\alpha_{j1}^2 + \ldots\lambda_n\alpha_{jn}^2$$

so that:

$$\sum_{j=1}^{D}\psi_j^T C\psi_j = \lambda_1\sum_{j=1}^{D}\alpha_{j1}^2 + \lambda_2\sum_{j=1}^{D}\alpha_{j2}^2 + \ldots\lambda_n\sum_{j=1}^{D}\alpha_{jn}^2$$

We're just about there- what we want to do now is to show that each of these coefficients is less than one.

The nifty trick is to consider the coefficients:

$$\{\alpha_{1i},\ldots,\alpha_{Di}\} = \left\{\psi_1^T\phi_i,\ldots,\psi_D^T\phi_i\right\}$$

as the coefficients from the projection of ϕ_i onto the subspace spanned by the ψ's:

$$\text{Proj}_\Psi(\phi_i) = \alpha_{1i}\psi_1 + \ldots + \alpha_{Di}\psi_D$$

where we have what we need:

$$\sum_{j=1}^{D}\alpha_{ji}^2 = \|\text{Proj}_\Psi(\phi_i)\|^2 \leq 1$$

with equality iff ϕ_i is in the span of the columns of Ψ. Now we can say the the maximum of:

$$\lambda_1 \sum_{j=1}^{D} \alpha_{j1}^2 + \lambda_2 \sum_{j=1}^{D} \alpha_{j2}^2 + \ldots \lambda_n \sum_{j=1}^{D} \alpha_{jn}^2$$

is found by replacing the coefficients of the first D terms to 1, and the remaining to zero. This corresponds, of course, to the value of the error found by using the eigenvector basis.

We have thus shown that the first D eigenvectors of the covariance matrix forms the best D dimensional subspace for the data- but notice that *any* basis for that D dimensional subspace would work.

6.7 Sunspot Analysis, Part I

In this example, we show one method for constructing a matrix from a time series, and from that matrix, we can extract the cyclical information from the data. This is intimately related to the *Fourier analysis* that we will discuss later- and in fact, under certain conditions, it can be shown that the best basis vectors of oscillatory data *are* the sinusoids that come out of the Fourier analysis. See [23] for more details on the method.

First, let us define some terms. We will say that a set of data, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is translationally invariant if a shift of the data of the form:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \longrightarrow \begin{bmatrix} x_n \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} \longrightarrow \begin{bmatrix} x_{n-1} \\ x_n \\ x_1 \\ \vdots \\ x_{n-2} \end{bmatrix} \longrightarrow \ldots$$

always forms another valid data point. A primary example is a signal time series of speech. Pieces of the data of length k from any initial time, t_0, all provide for equally valid samples of the voice.

In this example, we consider the sunspot data that comes with Matlab. This data represents a normalized number of sunspots, approximately every year from 1700 to 1987. The exact method of normalization is not important here- what is important is the relative cycling of the number of sunspots. Heavy sunspot activity (years with the peaks) causes problems with electronic communications on the earth, from cellular phones to cable TV.

There is a rule of thumb that says that the sunspot activity cycles approximately every 11 years. In this example, we will see how the KL eigenvectors correspond with this value- although remember that a full analysis will need to wait until we've discussed the Fourier analysis.

We first load the data and from this data we make a circulant matrix.

```
load sunspot.dat
npts=288;
t=sunspot(:,1);
x=sunspot(:,2);
plot(t,x);  %Do this if you haven't seen the data
meansun=mean(x);
x=x-meansun;  %Mean subtract the data

X=zeros(npts,npts);  %Will hold the matrix
X(:,1)=x;
idx=1:npts;

%Create the circulant matrix
for j=2:npts
```

```
   idx=[idx(npts) idx(1:npts-1)];
   X(:,j)=x(idx);
end

[U,S,V]=svd(X);
ss=diag(S)./sum(diag(S));
```

Type this into Matlab, and remark on the following:

- Plot the first 20 values of `ss`. Here you see that the singular values are doubled up, which corresponds to a doubling up of the eigenvalues of the covariance. Each of these two dimensional spaces correspond to a sinusoid of a certain period.
- To see the sinusoids, plot the first two eigenvectors, then the next two, then the next two. You'll see them line up like sine and cosine pairs for each period.
- The first two eigenvectors will correspond to the sine/cosine pair that best explain the data. It is this period that is approximately 11- Try to estimate the period from its graph (we will numerically estimate this period in the Fourier analysis section).

6.8 Eigenfaces

The application of KL to face recognition seems to have begun in the late 1980's with the paper by Sirovich and Kirby [34, 22]. Since then, there has been a quite extensive body of research in face recognition using similar techniques. A good starting point for locating additional resources and bibliographies are available on the internet; for example, the Face Recognition Homepage [25] and MIT's Artificial Intelligence Lab [31]. A recent text [23] also gives more details on KL, its application to faces, and more applications to other data analysis problems. We also note that current research in psychology and computer science is suggesting that perhaps the actual biological processes involved in familiarity are related to KL [10].

In face recognition problems, we are given a large collection of photos of faces. In this example, we assume that all photos are the same size, that they are grayscale, that they have basically the same background, and have been centered (these and many other issues can be discussed in the classroom setting, and are at the heart of the problem in applying KL in the real world).

To be more specific, suppose our photos all have 262×294 pixels. Then each photo is a vector in $\mathbb{R}^{77,028}$. We look for a small dimensional basis, Φ, in which to represent the space of all faces in the database, so that if $\boldsymbol{x}^{(i)}$ is a face, it can be represented by its (much lower dimensional) coordinates, $\left[\boldsymbol{x}^{(i)}\right]_{\Phi}$.

We would like to use the KL algorithm to compute this basis, but this is a very large eigenvector problem! If we have less photos than the number of pixels, it is faster to compute:

$$X^T X = V \Lambda V^T$$

and get the vectors $\Phi = U$ from this:

$$XV\Lambda^{-1} = U$$

Either way, the columns of U are in $\mathbb{R}^{77,028}$, and we will choose only a "small" number of them. Here is the interesting translation: Since each eigenvector $\mathbf{u}_i$ is a vector in $\mathbb{R}^{77,028}$, *it can be realized as a photograph (face) since it has* 262×294 *pixels!* Thus the term "Eigenface" was coined (I'm not sure who originally came up with it), and we get a set of orthonormal "faces" by which all faces in the database can be encoded using the projection coefficients. Of course, the question remains as to how many faces one will require, and it leads us to the question of how we might compute the rank of X- as we said previously, this can be a nontrivial question- A commercial package uses 128 basis vectors [35].

Once the dimension has been set, we can numerically encode every face in terms of the weights of the linear combinations- this is the key to actual face recognition. You encode someone's face, and search the database for the closest match.