A tutorial on Principal Components Analysis

Lindsay I Smith

February 22, 2002

Chapter 1

Introduction

This tutorial is designed to give the reader an understanding of Principal Components Analysis (PCA). PCA is a useful statistical technique that has found application in fields such as face recognition and image compression, and is a common technique for finding patterns in data of high dimension.

Before getting to a description of PCA, this tutorial first introduces mathematical concepts that will be used in PCA. It covers standard deviation, covariance, eigenvectors and eigenvalues. This background knowledge is meant to make the PCA section very straightforward, but can be skipped if the concepts are already familiar.

There are examples all the way through this tutorial that are meant to illustrate the concepts being discussed. If further information is required, the mathematics textbook "Elementary Linear Algebra 5e" by Howard Anton, Publisher John Wiley & Sons Inc, ISBN 0-471-85223-6 is a good source of information regarding the mathematical background.

Chapter 2

Background Mathematics

This section will attempt to give some elementary background mathematical skills that will be required to understand the process of Principal Components Analysis. The topics are covered independently of each other, and examples given. It is less important to remember the exact mechanics of a mathematical technique than it is to understand the reason why such a technique may be used, and what the result of the operation tells us about our data. Not all of these techniques are used in PCA, but the ones that are not explicitly required do provide the grounding on which the most important techniques are based.

I have included a section on Statistics which looks at distribution measurements, or, how the data is spread out. The other section is on Matrix Algebra and looks at eigenvectors and eigenvalues, important properties of matrices that are fundamental to PCA.

2.1 Statistics

The entire subject of statistics is based around the idea that you have this big set of data, and you want to analyse that set in terms of the relationships between the individual points in that data set. I am going to look at a few of the measures you can do on a set of data, and what they tell you about the data itself.

2.1.1 Standard Deviation

To understand standard deviation, we need a data set. Statisticians are usually concerned with taking a *sample* of a *population*. To use election polls as an example, the population is all the people in the country, whereas a sample is a subset of the population that the statisticians measure. The great thing about statistics is that by only measuring (in this case by doing a phone survey or similar) a sample of the population, you can work out what is most likely to be the measurement if you used the entire population. In this statistics section, I am going to assume that our data sets are samples of some bigger population. There is a reference later in this section pointing to more information about samples and populations.

Here's an example set:

$$X = [\ 1\ 2\ 4\ 6\ 12\ 15\ 25\ 45\ 68\ 67\ 65\ 98\ ]$$

I could simply use the symbol X to refer to this entire set of numbers. If I want to refer to an individual number in this data set, I will use subscripts on the symbol X to indicate a specific number. Eg. X_3 refers to the 3rd number in X, namely the number 4. Note that X_1 is the first number in the sequence, not X_0 like you may see in some textbooks. Also, the symbol n will be used to refer to the number of elements in the set X

There are a number of things that we can calculate about a data set. For example, we can calculate the mean of the sample. I assume that the reader understands what the mean of a sample is, and will only give the formula:

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Notice the symbol $\bar{X}$ (said "X bar") to indicate the mean of the set X. All this formula says is "Add up all the numbers and then divide by how many there are".

Unfortunately, the mean doesn't tell us a lot about the data except for a sort of middle point. For example, these two data sets have exactly the same mean (10), but are obviously quite different:

$$[\ 0\ 8\ 12\ 20\ ]\ and\ [\ 8\ 9\ 11\ 12\ ]$$

So what is different about these two sets? It is the *spread* of the data that is different. The Standard Deviation (SD) of a data set is a measure of how spread out the data is.

How do we calculate it? The English definition of the SD is: "The average distance from the mean of the data set to a point". The way to calculate it is to compute the squares of the distance from each data point to the mean of the set, add them all up, divide by $n-1$, and take the positive square root. As a formula:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{(n-1)}}$$

Where s is the usual symbol for standard deviation of a sample. I hear you asking "Why are you using $(n-1)$ and not n?". Well, the answer is a bit complicated, but in general, if your data set is a *sample* data set, ie. you have taken a subset of the real-world (like surveying 500 people about the election) then you must use $(n-1)$ because it turns out that this gives you an answer that is closer to the standard deviation that would result if you had used the

Set 1:

X	$(X - \bar{X})$	$(X - \bar{X})^2$
0	-10	100
8	-2	4
12	2	4
20	10	100
Total		208
Divided by (n-1)		69.333
Square Root		8.3266

Set 2:

X_i	$(X_i - \bar{X})$	$(X_i - \bar{X})^2$
8	-2	4
9	-1	1
11	1	1
12	2	4
Total		10
Divided by (n-1)		3.333
Square Root		1.8257

Table 2.1: Calculation of standard deviation

entire population, than if you'd used n. If, however, you are not calculating the standard deviation for a sample, but for an entire population, then you should divide by n instead of $(n-1)$. For further reading on this topic, the web page *http://mathcentral.uregina.ca/RR/database/RR.09.95/weston2.html* describes standard deviation in a similar way, and also provides an example experiment that shows the difference between each of the denominators. It also discusses the difference between samples and populations.

So, for our two data sets above, the calculations of standard deviation are in Table 2.1.

And so, as expected, the first set has a much larger standard deviation due to the fact that the data is much more spread out from the mean. Just as another example, the data set:

$$[\ 10\ 10\ 10\ 10\ ]$$

also has a mean of 10, but its standard deviation is 0, because all the numbers are the same. None of them deviate from the mean.

2.1.2 Variance

Variance is another measure of the spread of data in a data set. In fact it is almost identical to the standard deviation. The formula is this:

$$s^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{(n-1)}$$

You will notice that this is simply the standard deviation squared, in both the symbol (s^2) and the formula (there is no square root in the formula for variance). s^2 is the usual symbol for variance of a sample. Both these measurements are measures of the spread of the data. Standard deviation is the most common measure, but variance is also used. The reason why I have introduced variance in addition to standard deviation is to provide a solid platform from which the next section, covariance, can launch from.

Exercises

Find the mean, standard deviation, and variance for each of these data sets.

- [12 23 34 44 59 70 98]
- [12 15 25 27 32 88 99]
- [15 35 78 82 90 95 97]

2.1.3 Covariance

The last two measures we have looked at are purely 1-dimensional. Data sets like this could be: heights of all the people in the room, marks for the last COMP101 exam etc. However many data sets have more than one dimension, and the aim of the statistical analysis of these data sets is usually to see if there is any relationship between the dimensions. For example, we might have as our data set both the height of all the students in a class, and the mark they received for that paper. We could then perform statistical analysis to see if the height of a student has any effect on their mark.

Standard deviation and variance only operate on 1 dimension, so that you could only calculate the standard deviation for each dimension of the data set *independently* of the other dimensions. However, it is useful to have a similar measure to find out how much the dimensions vary from the mean *with respect to each other*.

Covariance is such a measure. Covariance is always measured *between* 2 dimensions. If you calculate the covariance between one dimension and *itself*, you get the variance. So, if you had a 3-dimensional data set (x, y, z), then you could measure the covariance between the x and y dimensions, the x and z dimensions, and the y and z dimensions. Measuring the covariance between x and x, or y and y, or z and z would give you the variance of the x, y and z dimensions respectively.

The formula for covariance is very similar to the formula for variance. The formula for variance could also be written like this:

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

where I have simply expanded the square term to show both parts. So given that knowledge, here is the formula for covariance:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

It is exactly the same except that in the second set of brackets, the X's are replaced by Y's. This says, in English, "For each data item, multiply the difference between the x value and the mean of x, by the the difference between the y value and the mean of y. Add all these up, and divide by $(n-1)$".

How does this work? Lets use some example data. Imagine we have gone into the world and collected some 2-dimensional data, say, we have asked a bunch of students how many hours in total that they spent studying COSC241, and the mark that they received. So we have two dimensions, the first is the H dimension, the hours studied, and the second is the M dimension, the mark received. Figure 2.2 holds my imaginary data, and the calculation of $cov(H,M)$, the covariance between the Hours of study done and the Mark received.

So what does it tell us? The exact value is not as important as it's sign (ie. positive or negative). If the value is positive, as it is here, then that indicates that both dimensions *increase together*, meaning that, in general, as the number of hours of study increased, so did the final mark.

If the value is negative, then as one dimension increases, the other decreases. If we had ended up with a negative covariance here, then that would have said the opposite, that as the number of hours of study increased the the final mark *decreased*.

In the last case, if the covariance is zero, it indicates that the two dimensions are independent of each other.

The result that mark given increases as the number of hours studied increases can be easily seen by drawing a graph of the data, as in Figure 2.1.3. However, the luxury of being able to visualize data is only available at 2 and 3 dimensions. Since the covariance value can be calculated between any 2 dimensions in a data set, this technique is often used to find relationships between dimensions in high-dimensional data sets where visualisation is difficult.

You might ask "is $cov(X,Y)$ equal to $cov(Y,X)$"? Well, a quick look at the formula for covariance tells us that yes, they are exactly the same since the only difference between $cov(X,Y)$ and $cov(Y,X)$ is that $(X_i - \bar{X})(Y_i - \bar{Y})$ is replaced by $(Y_i - \bar{Y})(X_i - \bar{X})$. And since multiplication is commutative, which means that it doesn't matter which way around I multiply two numbers, I always get the same number, these two equations give the same answer.

	Hours(H)	*Mark(M)*
Data	9	39
	15	56
	25	93
	14	61
	10	50
	18	75
	0	32
	16	85
	5	42
	19	70
	16	66
	20	80
Totals	167	749
Averages	13.92	62.42

Covariance:

H	M	$(H_i - \bar{H})$	$(M_i - \bar{M})$	$(H_i - \bar{H})(M_i - \bar{M})$
9	39	-4.92	-23.42	115.23
15	56	1.08	-6.42	-6.93
25	93	11.08	30.58	338.83
14	61	0.08	-1.42	-0.11
10	50	-3.92	-12.42	48.69
18	75	4.08	12.58	51.33
0	32	-13.92	-30.42	423.45
16	85	2.08	22.58	46.97
5	42	-8.92	-20.42	182.15
19	70	5.08	7.58	38.51
16	66	2.08	3.58	7.45
20	80	6.08	17.58	106.89
Total				1149.89
Average				104.54

Table 2.2: 2-dimensional data set and covariance calculation

includegraphicscovPlot.ps

Figure 2.1: A plot of the covariance data showing positive relationship between the number of hours studied against the mark received

2.1.4 The covariance Matrix

Recall that covariance is always measured between 2 dimensions. If we have a data set with more than 2 dimensions, there is more than one covariance measurement that can be calculated. For example, from a 3 dimensional data set (dimensions x, y, z) you could calculate $cov(x, y)$, $(cov(x, z)$, and $cov(y, z)$. In fact, for an n-dimensional data set, you can calculate $\frac{n!}{(n-2)!*2}$ different covariance values.

A useful way to get all the possible covariance values between all the different dimensions is to calculate them all and put them in a matrix. I assume in this tutorial that you are familiar with matrices, and how they can be defined. So, the definition for the covariance matrix for a set of data with n dimensions is:

$$C^{n \times n} = (c_{i,j},\ c_{i,j} = cov(Dim_i, Dim_j)),$$

where $C^{n \times n}$ is a matrix with n rows and n columns, and Dim_x is the xth dimension. All that this ugly looking formula says is that if you have an n-dimensional data set, then the matrix has n rows and columns (so is square) and each entry in the matrix is the result of calculating the covariance between two separate dimensions. Eg. the entry on row 2, column 3, is the covariance value calculated between the 2nd dimension and the 3rd dimension.

An example. We'll make up the covariance matrix for an imaginary 3 dimensional data set, using the usual dimensions x, y and z. Then, the covariance matrix has 3 rows and 3 columns, and the values are this:

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Some points to note: Down the main diagonal, you see that the covariance value is between one of the dimensions and itself. These are the variances for that dimension. The other point is that since $cov(a, b) = cov(b, a)$, the matrix is symmetrical about the main diagonal.

Exercises

Work out the covariance between the x and y dimensions in the following 2 dimensional data set, and describe what the result indicates about the data.

Item Number:	1	2	3	4	5
x	10	39	19	23	28
y	43	13	32	21	20

Calculate the covariance matrix for this 3 dimensional set of data.

Item Number:	1	2	3
x	1	-1	4
y	2	1	3
z	1	3	-1

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 11 \\ 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Figure 2.2: Example of one non-eigenvector and one eigenvector

$$2 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 24 \\ 16 \end{pmatrix} = 4 \times \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

Figure 2.3: Example of how a scaled eigenvector is still and eigenvector

2.2 Matrix Algebra

This section serves to provide a background for the matrix algebra required in PCA. Specifically I will be looking at eigenvectors and eigenvalues of a given matrix. Again, I assume a basic knowledge of matrices.

2.2.1 Eigenvectors

As you know, you can multiply two matrices together, provided they are compatible sizes. Eigenvectors are a special case of this. Consider the two multiplications between a matrix and a vector in Figure 2.2.

In the first example, the resulting vector is not an integer multiple of the original vector, whereas in the second example, the example is exactly 4 times the vector we began with. Why is this? Well, the vector is a vector in 2 dimensional space. The vector $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ (from the second example multiplication) represents an arrow pointing from the origin, $(0, 0)$, to the point $(3, 2)$. The other matrix, the square one, can be thought of as a transformation matrix. If you multiply this matrix on the left of a vector, the answer is another vector that is transformed from it's original position.

It is the nature of the transformation that the eigenvectors arise from. Imagine a transformation matrix that, when multiplied on the left, reflected vectors in the line $y = x$. Then you can see that if there were a vector that lay *on* the line $y = x$, it's reflection it *itself*. This vector (and all multiples of it, because it wouldn't matter how long the vector was), would be an eigenvector of that transformation matrix.

What properties do these eigenvectors have? You should first know that eigenvectors can only be found for *square* matrices. And, not every square matrix has eigenvectors. And, given an $n \times n$ matrix that does have eigenvectors, there are n of them. Given a 3×3 matrix, there are 3 eigenvectors.

Another property of eigenvectors is that even if I scale the vector by some amount before I multiply it, I still get the same multiple of it as a result, as in Figure 2.3. This is because if you scale a vector by some amount, all you are doing is making it longer, not changing it's direction. Lastly, all the eigenvectors of a matrix are *perpendicular*, ie. at right angles to each other, no matter how many dimensions you have. By the way, another word for perpendicular, in maths talk, is *orthogonal*. This is important because it means that you can express the data in terms of these perpendicular eigenvectors, instead of expressing them in terms of the x and y axes. We will be doing this later in the section on PCA.

Another important thing to know is that when mathematicians find eigenvectors, they like to find the eigenvectors whose length is exactly one. This is because, as you know, the length of a vector doesn't affect whether it's an eigenvector or not, whereas the direction does. So, in order to keep eigenvectors standard, whenever we find an eigenvector we usually scale it to make it have a length of 1, so that all eigenvectors have the same length. Here's a demonstration from our example above.

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

is an eigenvector, and the length of that vector is

$$\sqrt{(3^2 + 2^2)} = \sqrt{13}$$

so we divide the original vector by this much to make it have a length of 1.

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} \div \sqrt{13} = \begin{pmatrix} 3/\sqrt{13} \\ 2/\sqrt{13} \end{pmatrix}$$

How does one go about finding these mystical eigenvectors? Unfortunately, it's only easy(ish) if you have a rather small matrix, like no bigger than about 3×3. After that, the usual way to find the eigenvectors is by some complicated iterative method which is beyond the scope of this tutorial (and this author). If you ever need to find the eigenvectors of a matrix in a program, just find a maths library that does it all for you. A useful maths package, called newmat, is available at *http://webnz.com/robert/* .

Further information about eigenvectors in general, how to find them, and orthogonality, can be found in the textbook "Elementary Linear Algebra 5e" by Howard Anton, Publisher John Wiley & Sons Inc, ISBN 0-471-85223-6.

2.2.2 Eigenvalues

Eigenvalues are closely related to eigenvectors, in fact, we saw an eigenvalue in Figure 2.2. Notice how, in both those examples, the amount by which the original vector was scaled after multiplication by the square matrix was the same? In that example, the value was 4. 4 is the *eigenvalue* associated with that eigenvector. No matter what multiple of the eigenvector we took before we multiplied it by the square matrix, we would always get 4 times the scaled vector as our result (as in Figure 2.3).

So you can see that eigenvectors and eigenvalues always come in pairs. When you get a fancy programming library to calculate your eigenvectors for you, you usually get the eigenvalues as well.

Exercises

For the following square matrix:

$$\begin{pmatrix} 3 & 0 & 1 \\ -4 & 1 & 2 \\ -6 & 0 & -2 \end{pmatrix}$$

Decide which, if any, of the following vectors are eigenvectors of that matrix and give the corresponding eigenvalue.

$$\begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

Chapter 3

Principal Components Analysis

Finally we come to Principal Components Analysis (PCA). What is it? It is a way of identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences. Since patterns in data can be hard to find in data of high dimension, where the luxury of graphical representation is not available, PCA is a powerful tool for analysing data.

The other main advantage of PCA is that once you have found these patterns in the data, and you compress the data, ie. by reducing the number of dimensions, without much loss of information. This technique used in image compression, as we will see in a later section.

This chapter will take you through the steps you needed to perform a Principal Components Analysis on a set of data. I am not going to describe exactly *why* the technique works, but I will try to provide an explanation of what is happening at each point so that you can make informed decisions when you try to use this technique yourself.

3.1 Method

Step 1: Get some data

In my simple example, I am going to use my own made-up data set. It's only got 2 dimensions, and the reason why I have chosen this is so that I can provide plots of the data to show what the PCA analysis is doing at each step.

The data I have used is found in Figure 3.1, along with a plot of that data.

Step 2: Subtract the mean

For PCA to work properly, you have to subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension. So, all

$$\text{Data} = \begin{array}{c|c} x & y \\ \hline 2.5 & 2.4 \\ 0.5 & 0.7 \\ 2.2 & 2.9 \\ 1.9 & 2.2 \\ 3.1 & 3.0 \\ 2.3 & 2.7 \\ 2 & 1.6 \\ 1 & 1.1 \\ 1.5 & 1.6 \\ 1.1 & 0.9 \end{array} \qquad \text{DataAdjust} = \begin{array}{c|c} x & y \\ \hline .69 & .49 \\ -1.31 & -1.21 \\ .39 & .99 \\ .09 & .29 \\ 1.29 & 1.09 \\ .49 & .79 \\ .19 & -.31 \\ -.81 & -.81 \\ -.31 & -.31 \\ -.71 & -1.01 \end{array}$$

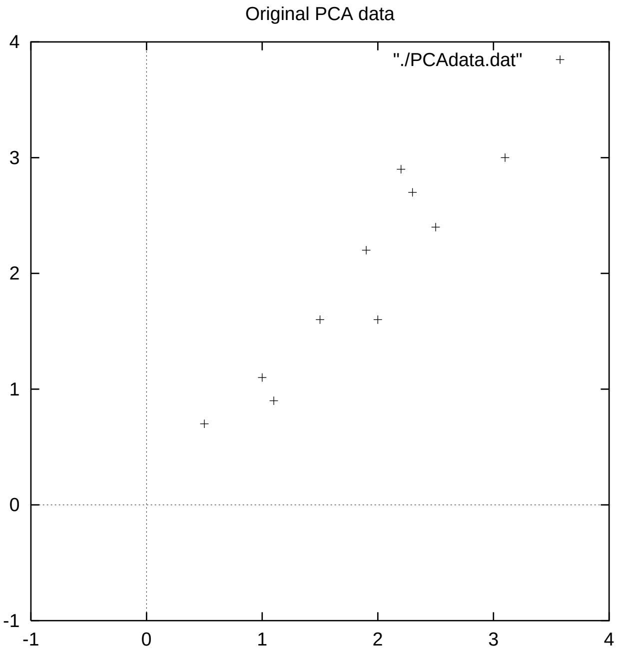

Figure 3.1: PCA example data, original data on the left, data with the means subtracted on the right, and a plot of the data

the x values have $\bar{x}$ (the mean of the x values of all the data points) subtracted, and all the y values have $\bar{y}$ subtracted from them. This produces a data set whose mean is zero.

Step 3: Calculate the covariance matrix

This is done in exactly the same way as was discussed in section 2.1.4. Since the data is 2 dimensional, the covariance matrix will be 2×2. There are no surprises here, so I will just give you the result:

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

So, since the non-diagonal elements in this covariance matrix are positive, we should expect that both the x and y variable increase together.

Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix

Since the covariance matrix is square, we can calculate the eigenvectors and eigenvalues for this matrix. These are rather important, as they tell us useful information about our data. I will show you why soon. In the meantime, here are the eigenvectors and eigenvalues:

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

It is important to notice that these eigenvectors are both *unit* eigenvectors ie. their lengths are both 1. This is very important for PCA, but luckily, most maths packages, when asked for eigenvectors, will give you unit eigenvectors.

So what do they mean? If you look at the plot of the data in Figure 3.2 then you can see how the data has quite a strong pattern. As expected from the covariance matrix, they two variables do indeed increase together. On top of the data I have plotted both the eigenvectors as well. They appear as diagonal dotted lines on the plot. As stated in the eigenvector section, they are perpendicular to each other. But, more importantly, they provide us with information about the patterns in the data. See how one of the eigenvectors goes through the middle of the points, like drawing a line of best fit? That eigenvector is showing us how these two data sets are related along that line. The second eigenvector gives us the other, less important, pattern in the data, that all the points follow the main line, but are off to the side of the main line by some amount.

So, by this process of taking the eigenvectors of the covariance matrix, we have been able to extract lines that characterise the data. The rest of the steps involve transforming the data so that it is expressed in terms of them lines.

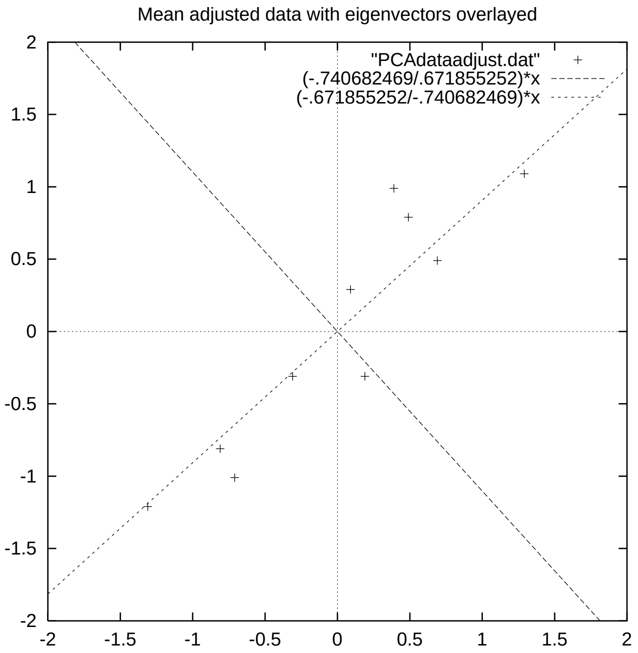

Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlayed on top.

Step 5: Choosing components and forming a feature vector

Here is where the notion of data compression and reduced dimensionality comes into it. If you look at the eigenvectors and eigenvalues from the previous section, you will notice that the eigenvalues are quite different values. In fact, it turns out that the eigenvector with the *highest* eigenvalue is the *principle component* of the data set. In our example, the eigenvector with the larges eigenvalue was the one that pointed down the middle of the data. It is the most significant relationship between the data dimensions.

In general, once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest. This gives you the components in order of significance. Now, if you like, you can decide to *ignore* the components of lesser significance. You do lose some information, but if the eigenvalues are small, you don't lose much. If you leave out some components, the final data set will have less dimensions than the original. To be precise, if you originally have n dimensions in your data, and so you calculate n eigenvectors and eigenvalues, and then you choose only the first p eigenvectors, then the final data set has only p dimensions.

What needs to be done now is you need to form a *feature vector*, which is just a fancy name for a matrix of vectors. This is constructed by taking the eigenvectors that you want to keep from the list of eigenvectors, and forming a matrix with these eigenvectors in the columns.

$$FeatureVector = (eig_1 \ eig_2 \ eig_3 \ .... \ eig_n)$$

Given our example set of data, and the fact that we have 2 eigenvectors, we have two choices. We can either form a feature vector with both of the eigenvectors:

$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

or, we can choose to leave out the smaller, less significant component and only have a single column:

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

We shall see the result of each of these in the next section.

Step 5: Deriving the new data set

This the final step in PCA, and is also the easiest. Once we have chosen the components (eigenvectors) that we wish to keep in our data and formed a feature vector, we simply take the transpose of the vector and multiply it on the left of the original data set, transposed.

$$FinalData = RowFeatureVector \times RowDataAdjust,$$

where $RowFeatureVector$ is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and $RowDataAdjust$ is the mean-adjusted data *transposed*, ie. the data items are in each column, with each row holding a separate dimension. I'm sorry if this sudden transpose of all our data confuses you, but the equations from here on are easier if we take the transpose of the feature vector and the data first, rather that having a little T symbol above their names from now on. $FinalData$ is the final data set, with data items in columns, and dimensions along rows.

What will this give us? It will give us the original data *solely in terms of the vectors we chose*. Our original data set had two axes, x and y, so our data was in terms of them. It is possible to express data in terms of any two axes that you like. If these axes are perpendicular, then the expression is the most efficient. This was why it was important that eigenvectors are always perpendicular to each other. We have changed our data from being in terms of the axes x and y, and now they are in terms of our 2 eigenvectors. In the case of when the new data set has reduced dimensionality, ie. we have left some of the eigenvectors out, the new data is only in terms of the vectors that we decided to keep.

To show this on our data, I have done the final transformation with each of the possible feature vectors. I have taken the transpose of the result in each case to bring the data back to the nice table-like format. I have also plotted the final points to show how they relate to the components.

In the case of keeping both eigenvectors for the transformation, we get the data and the plot found in Figure 3.3. This plot is basically the original data, rotated so that the eigenvectors are the axes. This is understandable since we have lost no information in this decomposition.

The other transformation we can make is by taking only the eigenvector with the largest eigenvalue. The table of data resulting from that is found in Figure 3.4. As expected, it only has a single dimension. If you compare this data set with the one resulting from using both eigenvectors, you will notice that this data set is exactly the first column of the other. So, if you were to plot this data, it would be 1 dimensional, and would be points on a line in exactly the x positions of the points in the plot in Figure 3.3. We have effectively thrown away the whole other axis, which is the other eigenvector.

So what have we done here? Basically we have transformed our data so that is expressed in terms of the patterns between them, where the patterns are the lines that most closely describe the relationships between the data. This is helpful because we have now classified our data point as a combination of the contributions from each of those lines. Initially we had the simple x and y axes. This is fine, but the x and y values of each data point don't really tell us exactly how that point relates to the rest of the data. Now, the values of the data points tell us exactly where (ie. above/below) the trend lines the data point sits. In the case of the transformation using *both* eigenvectors, we have simply altered the data so that it is in terms of those eigenvectors instead of the usual axes. But the single-eigenvector decomposition has removed the contribution due to the smaller eigenvector and left us with data that is only in terms of the other.

Transformed Data=

x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287

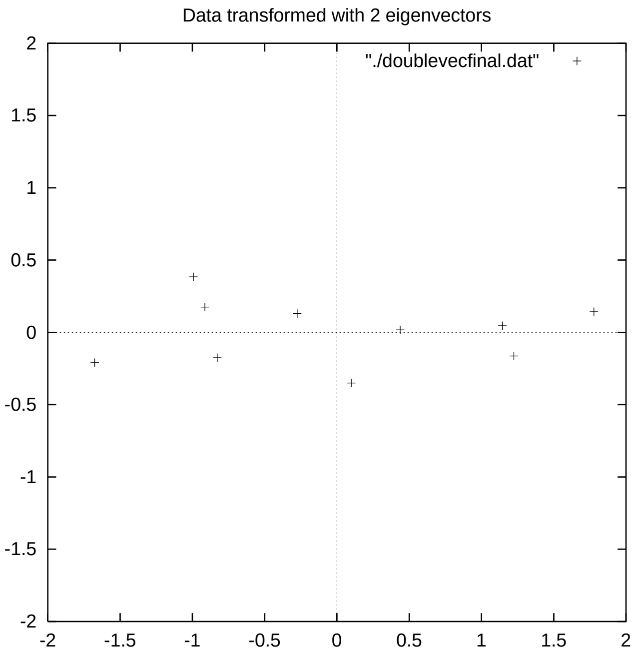

Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

Transformed Data (Single eigenvector)
x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056

Figure 3.4: The data after transforming using only the most significant eigenvector

3.1.1 Getting the old data back

Wanting to get the original data back is obviously of great concern if you are using the PCA transform for data compression (an example of which to will see in the next section). This content is taken from http://www.vision.auc.dk/ sig/Teaching/Flerdim/Current/hotelling/hotelling.html

So, how do we get the original data back? Before we do that, remember that only if we took *all* the eigenvectors in our transformation will we get *exactly* the original data back. If we have reduced the number of eigenvectors in the final transformation, then the retrieved data has lost some information.

Recall that the final transform is this:

$$FinalData = RowFeatureVector \times RowDataAdjust,$$

which can be turned around so that, to get the original data back,

$$RowDataAdjust = RowFeatureVector^{-1} \times FinalData$$

where $RowFeatureVector^{-1}$ is the inverse of $RowFeatureVector$. However, when we take *all* the eigenvectors in our feature vector, it turns out that the inverse of our feature vector is actually equal to the transpose of our feature vector. This is only true because the elements of the matrix are all the unit eigenvectors of our data set. This makes the return trip to our data easier, because the equation becomes

$$RowDataAdjust = RowFeatureVector^{T} \times FinalData$$

But, to get the actual original data back, we need to add on the mean of that original data (remember we subtracted it right at the start). So, for completeness,

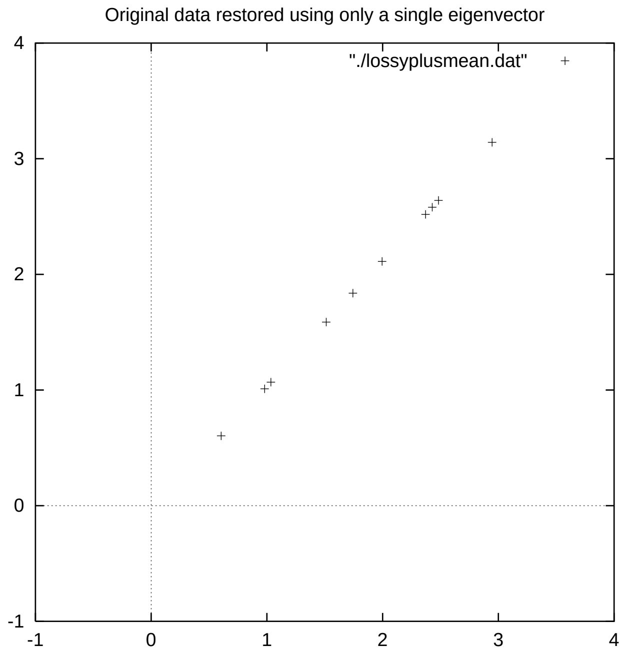

Figure 3.5: The reconstruction from the data that was derived using only a single eigenvector

$$RowOriginalData = (RowFeatureVector^T \times FinalData) + OriginalMean$$

This formula also applies to when you do not have all the eigenvectors in the feature vector. So even when you leave out some eigenvectors, the above equation still makes the correct transform.

I will not perform the data re-creation using the *complete* feature vector, because the result is exactly the data we started with. However, I will do it with the reduced feature vector to show you how information has been lost. Figure 3.5 show this plot. Compare it to the original data plot in Figure 3.1 and you will notice how, while the variation along the principle eigenvector (see Figure 3.2 for the eigenvector overlayed on top of the mean-adjusted data) has been kept, the variation along the other component (the other eigenvector that we left out) has gone.

Exercises

- What do the eigenvectors of the covariance matrix give us?
- At what point in the PCA process can we decide to compress the data? What effect does this have?

- For an example of PCA and a graphical representation of the principal eigenvectors, research the topic 'Eigenfaces', which uses PCA to do facial recognition

Chapter 4

Application to Computer Vision

This chapter will outline the way that PCA is used in computer vision, first showing how images are usually represented, and then showing what PCA can allow us to do with those images. The information in this section regarding facial recognition comes from “Face Recognition: Eigenface, Elastic Matching, and Neural Nets”, Jun Zhang et al. Proceedings of the IEEE, Vol. 85, No. 9, September 1997. The representation information, is taken from “Digital Image Processing” Rafael C. Gonzalez and Paul Wintz, Addison-Wesley Publishing Company, 1987. It is also an excellent reference for further information on the K-L transform in general. The image compression information is taken from *http://www.vision.auc.dk/ sig/Teaching/Flerdim/Current/hotelling/hotelling.html*, which also provides examples of image reconstruction using a varying amount of eigenvectors.

4.1 Representation

When using these sort of matrix techniques in computer vision, we must consider representation of images. A square, N by N image can be expressed as an N^2-dimensional vector

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & . & . & x_{N^2} \end{pmatrix}$$

where the rows of pixels in the image are placed one after the other to form a one-dimensional image. E.g. The first N elements ($x_1 - x_N$ will be the first row of the image, the next N elements are the next row, and so on. The values in the vector are the intensity values of the image, possibly a single greyscale value.

4.2 PCA to find patterns

Say we have 20 images. Each image is N pixels high by N pixels wide. For each image we can create an image vector as described in the representation section. We can then put all the images together in one big image-matrix like this:

$$ImagesMatrix = \begin{pmatrix} ImageVec1 \\ ImageVec2 \\ . \\ . \\ ImageVec20 \end{pmatrix}$$

which gives us a starting point for our PCA analysis. Once we have performed PCA, we have our original data in terms of the eigenvectors we found from the covariance matrix. Why is this useful? Say we want to do facial recognition, and so our original images were of peoples faces. Then, the problem is, given a new image, whose face from the original set is it? (Note that the new image is not one of the 20 we started with.) The way this is done is computer vision is to measure the difference between the new image and the original images, but not along the original axes, along the new axes derived from the PCA analysis.

It turns out that these axes works much better for recognising faces, because the PCA analysis has given us the original images *in terms of the differences and similarities between them*. The PCA analysis has identified the statistical patterns in the data.

Since all the vectors are N^2 dimensional, we will get N^2 eigenvectors. In practice, we are able to leave out some of the less significant eigenvectors, and the recognition still performs well.

4.3 PCA for image compression

Using PCA for image compression also know as the Hotelling, or Karhunen and Leove (KL), transform. If we have 20 images, each with N^2 pixels, we can form N^2 vectors, each with 20 dimensions. Each vector consists of all the intensity values from the *same* pixel from each picture. This is different from the previous example because before we had a vector for *image*, and each item in that vector was a different pixel, whereas now we have a vector for each *pixel*, and each item in the vector is from a different image.

Now we perform the PCA on this set of data. We will get 20 eigenvectors because each vector is 20-dimensional. To compress the data, we can then choose to transform the data only using, say 15 of the eigenvectors. This gives us a final data set with only 15 dimensions, which has saved us 1/4 of the space. However, when the original data is reproduced, the images have lost some of the information. This compression technique is said to be *lossy* because the decompressed image is not exactly the same as the original, generally worse.

Appendix A

Implementation Code

This is code for use in Scilab, a freeware alternative to Matlab. I used this code to generate all the examples in the text. Apart from the first macro, all the rest were written by me.

```
// This macro taken from
// http://www.cs.montana.edu/~harkin/courses/cs530/scilab/macros/cov.sci
// No alterations made

// Return the covariance matrix of the data in x, where each column of x
// is one dimension of an n-dimensional data set.  That is, x has x columns
// and m rows, and each row is one sample.
//
// For example, if x is three dimensional and there are 4 samples.
//   x = [1 2 3;4 5 6;7 8 9;10 11 12]
//   c = cov (x)

function [c]=cov (x)
// Get the size of the array
sizex=size(x);
// Get the mean of each column
meanx = mean (x, "r");
// For each pair of variables, x1, x2, calculate
// sum ((x1 - meanx1)(x2-meanx2))/(m-1)
for var = 1:sizex(2),
        x1 = x(:,var);
        mx1 = meanx (var);
        for ct = var:sizex (2),
                x2 = x(:,ct);
                mx2 = meanx (ct);
                v = ((x1 - mx1)' * (x2 - mx2))/(sizex(1) - 1);
```

```
                cv(var,ct) = v;
                cv(ct,var) = v;
                // do the lower part of c also.
        end,
end,
c=cv;


// This a simple wrapper function to get just the eigenvectors
// since the system call returns 3 matrices
function [x]=justeigs (x)
// This just returns the eigenvectors of the matrix

[a, eig, b] = bdiag(x);

x= eig;



// this function makes the transformation to the eigenspace for PCA
// parameters:
// adjusteddata = mean-adjusted data set
// eigenvectors = SORTED eigenvectors (by eigenvalue)
// dimensions  = how many eigenvectors you wish to keep
//
// The first two parameters can come from the result of calling
// PCAprepare on your data.
// The last is up to you.

function [finaldata] = PCAtransform(adjusteddata,eigenvectors,dimensions)
finaleigs = eigenvectors(:,1:dimensions);
prefinaldata = finaleigs'*adjusteddata';
finaldata = prefinaldata';



// This function does the preparation for PCA analysis
// It adjusts the data to subtract the mean, finds the covariance matrix,
// and finds normal eigenvectors of that covariance matrix.
// It returns 4 matrices
// meanadjust = the mean-adjust data set
// covmat = the covariance matrix of the data
// eigvalues = the eigenvalues of the covariance matrix, IN SORTED ORDER
// normaleigs = the normalised eigenvectors of the covariance matrix,
// IN SORTED ORDER WITH RESPECT TO
// THEIR EIGENVALUES, for selection for the feature vector.
```

```
//
// NOTE: This function cannot handle data sets that have any eigenvalues
// equal to zero.  It's got something to do with the way that scilab treats
// the empty matrix and zeros.
//
function [meanadjusted,covmat,sorteigvalues,sortnormaleigs] = PCAprepare (data)
// Calculates the mean adjusted matrix, only for 2 dimensional data
means = mean(data,"r");
meanadjusted = meanadjust(data);
covmat = cov(meanadjusted);
eigvalues = spec(covmat);
normaleigs = justeigs(covmat);
sorteigvalues = sorteigvectors(eigvalues',eigvalues');
sortnormaleigs = sorteigvectors(eigvalues',normaleigs);



// This removes a specified column from a matrix
// A = the matrix
// n = the column number you wish to remove
function [columnremoved] = removecolumn(A,n)
inputsize = size(A);
numcols = inputsize(2);
temp = A(:,1:(n-1));
for var = 1:(numcols - n)
        temp(:,(n+var)-1) = A(:,(n+var));
end,
columnremoved = temp;


// This finds the column number that has the
// highest value in it's first row.
function [column] = highestvalcolumn(A)
inputsize = size(A);
numcols = inputsize(2);
maxval = A(1,1);
maxcol = 1;
for var = 2:numcols
        if A(1,var) > maxval
                maxval = A(1,var);
                maxcol = var;
        end,
end,
column = maxcol
```

```
// This sorts a matrix of vectors, based on the values of
// another matrix
//
// values = the list of eigenvalues (1 per column)
// vectors = The list of eigenvectors (1 per column)
//
// NOTE:  The values should correspond to the vectors
// so that the value in column x corresponds to the vector
// in column x.
function [sortedvecs] = sorteigvectors(values,vectors)
inputsize = size(values);
numcols  = inputsize(2);
highcol = highestvalcolumn(values);
sorted = vectors(:,highcol);
remainvec = removecolumn(vectors,highcol);
remainval = removecolumn(values,highcol);
for var = 2:numcols
        highcol = highestvalcolumn(remainval);
        sorted(:,var) = remainvec(:,highcol);
        remainvec = removecolumn(remainvec,highcol);
        remainval = removecolumn(remainval,highcol);
end,
sortedvecs = sorted;


// This takes a set of data, and subtracts
// the column mean from each column.
function [meanadjusted] = meanadjust(Data)
inputsize = size(Data);
numcols = inputsize(2);
means = mean(Data,"r");
tmpmeanadjusted = Data(:,1) - means(:,1);
for var = 2:numcols
        tmpmeanadjusted(:,var) = Data(:,var) - means(:,var);
end,
meanadjusted = tmpmeanadjusted
```