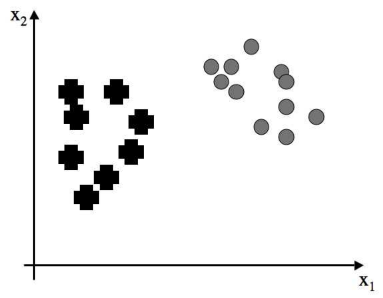

FIGURE 10.2: A set of datapoints in two dimensions, with two classes.

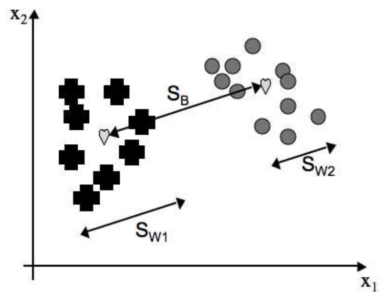

FIGURE 10.3: The meaning of the between-class and within-class scatter. The hearts mark the means of the two classes.

Many of the algorithms that we will see in this chapter are unsupervised. The disadvantage of this is that we are not then able to use the knowledge of their classes in order to reduce the problem further. However, we will start off by considering a method of dimensionality reduction that is aimed at supervised learning, Linear Discriminant Analysis. This method is credited to one of the best-known statisticians of the 20th century, R.A. Fisher, and dates from 1936.

10.1 Linear Discriminant Analysis (LDA)

Figure 10.2 shows a simple two-dimensional dataset consisting of two classes. We can compute various statistics about the data, but we will settle for the means of the two classes in the data, $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, the mean of the entire dataset ($\boldsymbol{\mu}$), and the covariance of each class with itself (see Section 8.2.2 for a description of covariance), which is $\sum_j (\mathbf{x}_j - \boldsymbol{\mu})(\mathbf{x}_j - \boldsymbol{\mu})^T$. The question is what we can do with these pieces of data. The principal insight of LDA is that the covariance matrix can tell us about the scatter within a dataset, which is the amount of spread that there is within the data. The way to find this scatter is to multiply the covariance by the p_c, the probability of the class (that is, the number of datapoints there are in that class divided by the total number). Adding the values of this for all of the classes gives us a measure of the within-class scatter of the dataset:

$$S_W = \sum_{\text{classes } c} \sum_{j \in c} p_c (\mathbf{x}_j - \boldsymbol{\mu}_c)(\mathbf{x}_j - \boldsymbol{\mu}_c)^T. \tag{10.1}$$

If our dataset is easy to separate into classes, then this within-class scatter should be small, so that each class is tightly clustered together. However, to be able to separate the data, we also want the distance *between* the classes to be large. This is known as the between-classes scatter and is a significantly simpler computation, simply looking at the difference in the means:

$$S_B = \sum_{\text{classes } c} (\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T. \tag{10.2}$$

The meanings of these two measurements is shown in Figure 10.3. The argument about good separation suggests that datasets that are easy to separate into the different classes (i.e., the classes are discriminable) should have S_B/S_W as large as possible.

This seems perfectly reasonable, but it hasn't told us anything about dimensionality reduction. However, we can say that the rule about making S_B/S_W as large as possible is something that we want to be true for our data when we reduce the number of dimensions. Figure 10.4 shows two projections of the dataset onto a straight line. For the projection on the left it is clear that we can't separate out the two classes, while for the one on the right we can. So we just need to find a way to compute a suitable projection.

Remember from Chapter 2 that any line can be written as a vector $\mathbf{w}$ (which we used as our weight vector in Section 2.3; it is one row of weight matrix $\mathbf{W}$). The projection of the data can be written as $z = \mathbf{w}^T \cdot \mathbf{x}$ for datapoint $\mathbf{x}$. This gives us a scalar that is the distance along the $\mathbf{w}$ vector that we need to go to find the projection of point $\mathbf{x}$. To see this, remember that $\mathbf{w}^T \cdot \mathbf{x}$ is the sum of the vectors multiplied together element-wise, and is equal to the size of $\mathbf{w}$ times the size of $\mathbf{x}$ times the cosine of the angle between them. We can make the size of $\mathbf{w}$ be 1, so that we don't have to worry about it, and all that is then described is the amount of $\mathbf{x}$ that lies along $\mathbf{w}$.

So we can compute the projection of our data along $\mathbf{w}$ for every point, and we will have projected our data onto a straight line, as is shown in the two examples in Figure 10.4. Since the mean can be treated as a datapoint, we can project that as well: $\mu'_c = \mathbf{w}^T \cdot \boldsymbol{\mu}_c$. Now we just need to work out what happens to the within-class and between-class scatters. Replacing $\mathbf{x}_j$ with $\mathbf{w}^T \cdot \mathbf{x}_j$ in Equations (10.1) and (10.2) we can use some linear algebra (principally the fact that $(\mathbf{A}^T\mathbf{B})^T = \mathbf{B}^T\mathbf{A}^{T^T} = \mathbf{B}^T\mathbf{A}$) to get:

$$\sum_{\text{classes } c} \sum_{j \in c} p_c(\mathbf{w}^T \cdot (\mathbf{x}_j - \boldsymbol{\mu}_c))(\mathbf{w}^T \cdot (\mathbf{x}_j - \boldsymbol{\mu}_c))^T = \mathbf{w}^T S_W \mathbf{w} \tag{10.3}$$

$$\sum_{\text{classes } c} \mathbf{w}^T(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T \mathbf{w} = \mathbf{w}^T S_B \mathbf{w}. \tag{10.4}$$

So our ratio of within-class and between-class scatter looks like $\frac{\mathbf{w}^T S_W \mathbf{w}}{\mathbf{w}^T S_B \mathbf{w}}$. In order to find the maximum value of this with respect to $\mathbf{w}$, we differentiate it and set the derivative equal to 0. This tells us that:

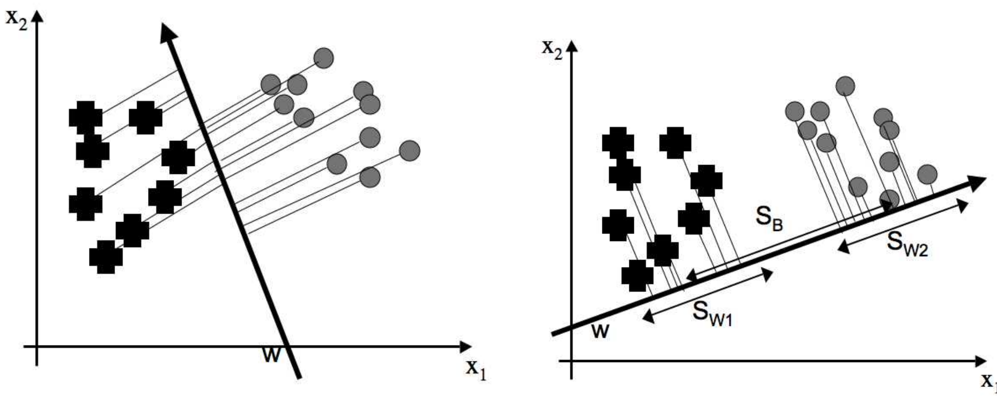

FIGURE 10.4: Two different possible projection lines. The one on the left fails to separate the classes.

$$\frac{S_B\mathbf{w}(\mathbf{w}^TS_W\mathbf{w}) - S_W\mathbf{w}(\mathbf{w}^TS_B\mathbf{w})}{(\mathbf{w}^TS_W\mathbf{w})^2} = 0. \tag{10.5}$$

So we just need to solve this equation for $\mathbf{w}$ and we are done. We start with a little bit of rearranging to get:

$$S_W\mathbf{w} = \frac{\mathbf{w}^TS_W\mathbf{w}}{\mathbf{w}^TS_B\mathbf{w}}S_B\mathbf{w}. \tag{10.6}$$

If there are only two classes in the data, then we just need to notice that S_B in Equation (10.2) reduces to $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$, which tells us that $S_B\mathbf{w}$ is in the direction $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, and so $\mathbf{w}$ is in the direction of $S_W^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ (we can ignore the ratio of within-class and between-class scatter, since it is a scalar and therefore does not affect the direction of the vector).

Unfortunately, this does not work for the general case. There, finding the minimum is not simple, and requires computing the generalised eigenvectors of $S_W^{-1}S_B$, assuming that S_W^{-1} exists. We will be discussing eigenvectors in the next section if you are not sure what they are.

Turning this into an algorithm is very simple. You simply have to compute the between-class and within-class scatters, and then find the value of $\mathbf{w}$. In NumPy, the entire algorithm can be written as (where the generalised eigenvectors are computed in SciPy rather than NumPy, which was imported using `from scipy import linalg as la`):

```
C = cov(transpose(data))

# Loop over classes
classes = unique(labels)
for i in range(len(classes)):
    # Find relevant datapoints
    indices = squeeze(where(labels==classes[i]))
```

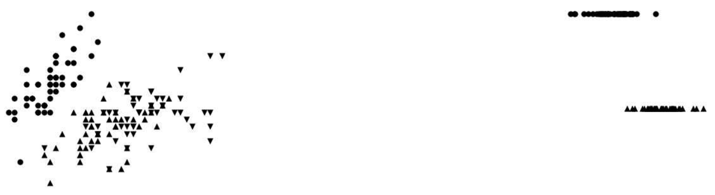

FIGURE 10.5: Plot of the iris data showing the three classes *left:* before and *right:* after LDA has been applied.

```
    d = squeeze(data[indices,:])
    classcov = cov(transpose(d))
    Sw += float(shape(indices)[0])/nData * classcov

Sb = C - Sw
# Now solve for W and compute mapped data
# Compute eigenvalues, eigenvectors and sort into order
evals,evecs = la.eig(Sw,Sb)
indices = argsort(evals)
indices = indices[::-1]
evecs = evecs[:,indices]
evals = evals[indices]
w = evecs[:,:redDim]
newData = transpose(dot(data,w))
```

As an example of using the algorithm, Figure 10.5 shows a plot of the first two dimensions of the iris data (with the classes shown as three different symbols) before and after applying LDA, with the number of dimensions being set to two. While one of the classes (the triangles) can already be separated from the others, all three are readily distinguishable after LDA has been applied (and only one dimension, the y one, is required for this).

10.2 Principal Components Analysis (PCA)

The next few methods that we are going to look at are also involved in computing transformations of the data in order to identify a lower-dimensional set of axes. However, unlike LDA, they are designed for unlabelled data. This