Lecture 17 - Fisher's Linear Discriminant Function (FLDA or simply LDA)

(Parametric) Linear Models for Classification

So far we designed classifiers based on probability density or probability functions. In some cases, we saw that the resulting classifiers were equivalent to a set of linear discriminant functions.

We will now focus on the design of linear classifiers, irrespective of the underlying distributions describing the training data.

- The major advantage of linear classifiers is their simplicity and computational attractiveness.
- We will develop techniques for the computation of the corresponding linear functions. In the sequel we will focus on a more general problem, in which a linear classifier cannot classify correctly all feature vectors, yet we will seek ways to design an optimal linear classifier by adopting an appropriate optimality criterion.

Before, we designed generative classification algorithm that parametrize each class with a probabilistic model for the posterior probability of each class, i.e., $P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)}$.

Let's now consider a generalization of this model in which we transform the linear function of $\mathbf{w}$ using a nonlinear function $f(\bullet)$ so that:

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b)$$

In the machine learning literature $f(\bullet)$ is known as an **activation function**. The **decision surfaces** correspond to $y(\mathbf{x}) = \text{constant}$, so that $\mathbf{w}^T\mathbf{x} + b = \text{constant}$ and hence the decision surfaces are linear functions of $\mathbf{x}$, even if the function $f(\bullet)$ is nonlinear. For this reason, the class of models described by equation above are called **generalized linear models**.

Note, however, that in contrast to the models used for regression, they are no longer linear in the parameters due to the presence of the nonlinear function $f(\bullet)$. This will lead to more complex analytical and computational properties than for linear regression models. Nevertheless, these models are still relatively simple compared to the more general nonlinear models that will be studied later in the course.

The algorithm we will study will be equally applicable if we first make a fixed nonlinear transformation of the input variables using a vector of basis functions $\phi(x)$ as we did for

regression models. In what follows, we consider classification directly in the original input space $\mathbf{x}$ but this illustration can be generalizatble to a notation involving basis functions.

Discriminant Functions

A discriminant is a function that takes an input vector x and assigns it to one of K classes, denoted C_k.

Let's restrict our attention to **linear discriminants**, namely those for which the decision surfaces are hyperplanes. To simplify the discussion, we consider first the case of two classes and then investigate the extension to $K > 2$ classes.

Two Classes

The simplest representation of a linear discriminant function is obtained by taking a linear function of the input vector so that

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

where $\mathbf{w}$ is called a **weight vector**, and b is a **bias** (not to be confused with bias in the statistical sense). The negative of the bias is sometimes called a threshold. An input vector $\mathbf{x}$ is assigned to class C_1 if $y(\mathbf{x}) \geq 0$ and to class C_2 otherwise.

Looks pretty familiar, right? If you are on one side of the line, then you are in class 1. If you are on the other side of the line, then you are in class 2. So, the decision boundary is $y(\mathbf{x}) = 0$.

- The vector $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, and so $\mathbf{w}$ determines the orientation of the decision surface.
- Similarly, if $\mathbf{x}$ is a point on the decision surface, then $y(\mathbf{x}) = 0$, and so the normal distance from the origin to the decision surface is given by: $\frac{y(\overrightarrow{\mathbf{x}})}{\|\overrightarrow{\mathbf{w}}\|}$

In [3]:
```
from IPython.display import Image
Image('figures/Figure4.1.png', width=800)
# Source: Bishop textbook
```

Out[3]:

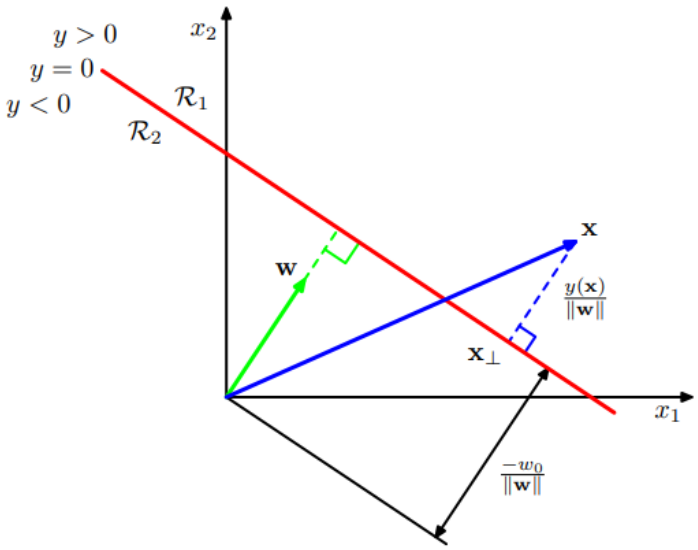


Figure 4.1 Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\mathbf{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\mathbf{x}$ from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$.

Multiple Classes

Considering the extension of linear discriminants to $K > 2$ classes. We might be tempted be to build a K-class discriminant by combining a number of two-class discriminant functions. However, this leads to some serious difficulties.

- Consider the use of $K - 1$ classifiers each of which solves a two-class problem of separating points in a particular class C_k from points not in that class. This is known as a **one-versus-all** classifier.
- An alternative is to introduce $K(K - 1)/2$ binary discriminant functions, one for every possible pair of classes. This is known as a **one-versus-one** classifier. Each point is then classified according to a majority vote amongst the discriminant functions. However, this too runs into the problem of ambiguous regions.

In [2]:
```
Image('figures/Figure4.2.png',width=700)
# Source: Bishop textbook
```

Out[2]:

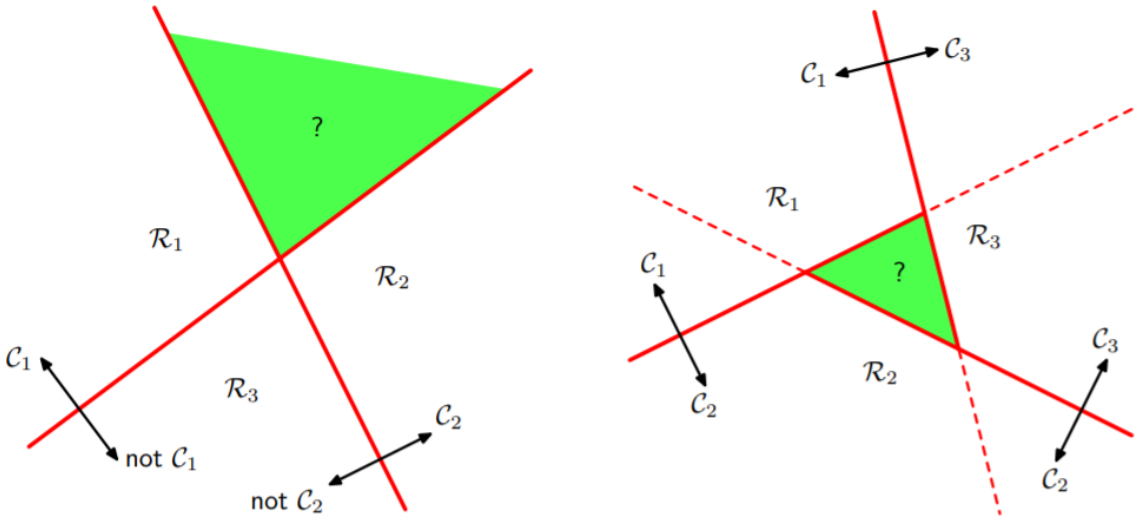


Figure 4.2 Attempting to construct a K class discriminant from a set of two class discriminants leads to ambiguous regions, shown in green. On the left is an example involving the use of two discriminants designed to distinguish points in class $\mathcal{C}_k$ from points not in class $\mathcal{C}_k$. On the right is an example involving three discriminant functions each of which is used to separate a pair of classes $\mathcal{C}_k$ and $\mathcal{C}_j$.

We can avoid these difficulties by considering a **single K-class discriminant** comprising K linear functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k$$

and then assigning a point $\mathbf{x}$ to class C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$. The decision boundary between class C_k and class C_j is therefore given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ and hence corresponds to a $(D-1)$-dimensional hyperplane defined by

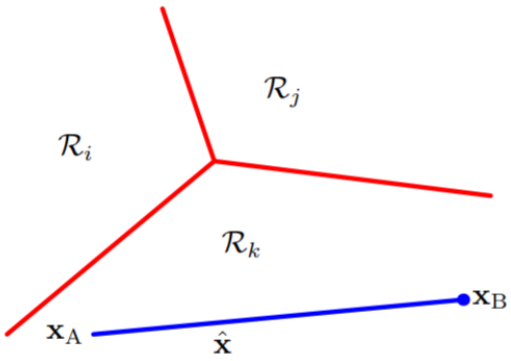
$$(\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} + (b_k - bj) = 0$$

This has the same form as the decision boundary for the two-class case.

In [3]:
```
Image('figures/Figure4.3.png',width=800)
# Source: Bishop textbook
```

Out[3]:

Figure 4.3 Illustration of the decision regions for a multiclass linear discriminant, with the decision boundaries shown in red. If two points $\mathbf{x}_A$ and $\mathbf{x}_B$ both lie inside the same decision region $\mathcal{R}_k$, then any point $\hat{\mathbf{x}}$ that lies on the line connecting these two points must also lie in $\mathcal{R}_k$, and hence the decision region must be singly connected and convex.



The decision regions of such a discriminant are always **singly connected** and **convex**. To see this, consider two points x_A and x_B both of which lie inside decision region R_k. Any point $\hat{\mathbf{x}}$ that lies on the line connecting x_A and x_B can be expressed in the form

$$\hat{\mathbf{x}} = \alpha x_A + (1-\alpha) x_B$$

where $0 \leq \alpha \leq 1$. From the linearity of the discriminant functions, it follows that

$$y_k(\hat{\mathbf{x}}) = \alpha y_k(x_A) + (1 - \alpha) y_k(x_B)$$

Because both x_A and x_B lie inside R_k, it follows that $y_k(x_A) > y_j(x_A)$, and $y_k(x_B) > y_j(x_B)$, for all $j \neq k$, and hence $y_k(\hat{\mathbf{x}}) > y_j(\hat{\mathbf{x}})$, and so $\hat{\mathbf{x}}$ also lies inside R_k. Thus R_k is singly connected and convex.

We will learn 3 methods to optimize the parameters of a linear discriminant function (classifier):

1. Least Squares for Classification
2. Fisher's Linear Discriminant
3. The Perceptron Algorithm
4. Support Vector Machines

Discriminant Functions

A discriminant is a function that takes an input vector x and assigns it to one of K classes, denoted C_k.

Let's restrict our attention to **linear discriminants**, namely those for which the decision surfaces are hyperplanes. To simplify the discussion, we consider first the case of two classes and then investigate the extension to $K > 2$ classes.

Two Classes

The simplest representation of a linear discriminant function is obtained by taking a linear function of the input vector so that

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

where $\mathbf{w}$ is called a **weight vector**, and b is a **bias** (not to be confused with bias in the statistical sense). The negative of the bias is sometimes called a threshold. An input vector $\mathbf{x}$ is assigned to class C_1 if $y(\mathbf{x}) \geq 0$ and to class C_2 otherwise.

Looks pretty familiar, right? If you are on one side of the line, then you are in class 1. If you are on the other side of the line, then you are in class 2. So, the decision boundary is $y(\mathbf{x}) = 0$.

- The vector $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, and so $\mathbf{w}$ determines the orientation of the decision surface.
- Similarly, if $\mathbf{x}$ is a point on the decision surface, then $y(\mathbf{x}) = 0$, and so the normal distance from the origin to the decision surface is given by: $\frac{y(\overrightarrow{\mathbf{x}})}{\|\overrightarrow{\mathbf{w}}\|}$

In [1]:
```
Image('figures/Figure4.1.png', width=800)
# Source: Bishop textbook
```

Out[1]:

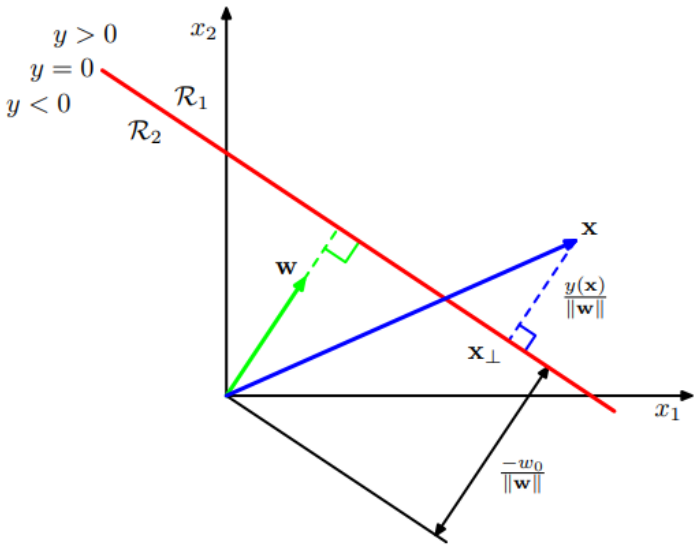


Figure 4.1 Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\mathbf{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\mathbf{x}$ from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$.

Fisher's Linear Discriminant Analysis (or LDA)

A very popular type of a linear discriminant is the **Fisher's Linear Discriminant**.

- Given two classes, we can compute the mean of each class:

$$\vec{\mathbf{m}}_1 = \frac{1}{N_1} \sum_{n \in C_1} \vec{\mathbf{x}}_\mathbf{n}$$
$$\vec{\mathbf{m}}_\mathbf{2} = \frac{1}{N_2} \sum_{n \in C_2} \vec{\mathbf{x}}_\mathbf{n}$$

We can maximize the separation of the means:

$$m_2 - m_1 = \vec{\mathbf{w}}^T (\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)$$

- $\vec{\mathbf{w}}^T \vec{\mathbf{x}}$ takes a D dimensional data point and projects it down to 1-D with a weight sum of the original features. We want to find a weighting that maximizes the separation of the class means.

- Not only do we want well separated means for each class, but we also want each class to be *compact* to minimize overlap between the classes.

- Consider the *within class variance:*

$$
\begin{aligned}
s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2 &= \sum_{n \in C_k} (\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}_n - m_k)^2 \\
&= \overrightarrow{\mathbf{w}}^T \sum_{n \in C_k} (\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_\mathbf{k})(\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_\mathbf{k})^T \overrightarrow{\mathbf{w}}
\end{aligned}
$$

- So, we want to minimize within class variance and maximize between class separability. How about the following objective function:

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \\
&= \frac{\overrightarrow{\mathbf{w}}^T (\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)^T \overrightarrow{\mathbf{w}}}{\sum_{n \in C_1} (\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}_n - m_1)^2 + \sum_{n \in C_2} (\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}_n - m_2)^2} \\
&= \frac{\overrightarrow{\mathbf{w}}^T (\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)^T \overrightarrow{\mathbf{w}}}{\overrightarrow{\mathbf{w}}^T \left(\sum_{n \in C_1} (\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_1)^T + \sum_{n \in C_2} (\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_2)(\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_2)^T \right) \overrightarrow{\mathbf{w}}} \\
&= \frac{\overrightarrow{\mathbf{w}}^T \mathbf{S}_B \overrightarrow{\mathbf{w}}}{\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}}}
\end{aligned}
$$

where

$$
S_B = (\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)^T
$$

and

$$
S_W = \sum_{n \in C_1} (\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_1)^T + \sum_{n \in C_2} (\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_2)(\overrightarrow{\mathbf{x}}_n - \overrightarrow{\mathbf{m}}_2)^T
$$

- Ok, so let's optimize:

$$
\begin{aligned}
\frac{\partial J(\overrightarrow{\mathbf{w}})}{\partial \overrightarrow{\mathbf{w}}} &= \frac{2(\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}}) \mathbf{S}_B \overrightarrow{\mathbf{w}} - 2(\overrightarrow{\mathbf{w}}^T \mathbf{S}_B \overrightarrow{\mathbf{w}}) \mathbf{S}_W \overrightarrow{\mathbf{w}}}{(\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}})^2} = 0 \\
0 &= \frac{\mathbf{S}_B \overrightarrow{\mathbf{w}}}{(\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}})} - \frac{(\overrightarrow{\mathbf{w}}^T \mathbf{S}_B \overrightarrow{\mathbf{w}}) \mathbf{S}_W \overrightarrow{\mathbf{w}}}{(\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}})^2} \\
(\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}}) \mathbf{S}_B \overrightarrow{\mathbf{w}} &= (\overrightarrow{\mathbf{w}}^T \mathbf{S}_B \overrightarrow{\mathbf{w}}) \mathbf{S}_W \overrightarrow{\mathbf{w}} \\
\mathbf{S}_B \overrightarrow{\mathbf{w}} &= \frac{\overrightarrow{\mathbf{w}}^T \mathbf{S}_B \overrightarrow{\mathbf{w}}}{\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}}} \mathbf{S}_W \overrightarrow{\mathbf{w}} \\
\mathbf{S}_W^{-1} \mathbf{S}_B \overrightarrow{\mathbf{w}} &= \lambda \overrightarrow{\mathbf{w}}
\end{aligned}
$$

where the scalar $\lambda = \frac{\overrightarrow{\mathbf{w}}^T \mathbf{S}_B \overrightarrow{\mathbf{w}}}{\overrightarrow{\mathbf{w}}^T \mathbf{S}_W \overrightarrow{\mathbf{w}}}$

Does this look familiar?

This is the generalized eigenvalue problem!

- So the direction of projection correspond to the eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ with the largest eigenvalues.

The solution is easy when $S_w^{-1} = (\Sigma_1 + \Sigma_2)^{-1}$ exists.

In this case, if we use the definition of $S_B = (\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)^T$:

$$S_W^{-1} S_B \overrightarrow{\mathbf{w}} = \lambda \overrightarrow{\mathbf{w}}$$
$$S_W^{-1}(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)^T \overrightarrow{\mathbf{w}} = \lambda \overrightarrow{\mathbf{w}}$$

Noting that $\alpha = (\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)^T \overrightarrow{\mathbf{w}}$ is a constant, this can be written as:

$$S_W^{-1}(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1) = \frac{\lambda}{\alpha} \overrightarrow{\mathbf{w}}$$

- Since we don't care about the magnitude of $\overrightarrow{\mathbf{w}}$:

$$\overrightarrow{\mathbf{w}}^* = S_W^{-1}(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1) = (\Sigma_1 + \Sigma_2)^{-1}(\overrightarrow{\mathbf{m}}_2 - \overrightarrow{\mathbf{m}}_1)$$

Make sure $\overrightarrow{\mathbf{w}}^*$ is a unit vector by taking: $\overrightarrow{\mathbf{w}}^* \leftarrow \frac{\overrightarrow{\mathbf{w}}^*}{\|\overrightarrow{\mathbf{w}}^*\|}$

- Note that if the within-class covariance, S_W, is isotropic, so that S_W is proportional to the unit matrix, we find that $\overrightarrow{\mathbf{w}}$ is proportional to the difference of the class means.
- This result is known as *Fisher's linear discriminant*, although strictly it is not a discriminant but rather a specific choice of direction for projection of the data down to one dimension. However, the projected data can subsequently be used to construct a discriminant, by choosing a threshold y_0 so that we classify a new point as belonging to C_1 if $y(x) \geq y_0$ and classify it as belonging to C_2 otherwise.

Also, note that:

- For a classification problem with Gaussian classes of equal covariance $\Sigma_i = \Sigma$, the boundary is the plane of normal:

$$\overrightarrow{\mathbf{w}} = \Sigma^{-1}(\overrightarrow{\mathbf{m}}_i - \overrightarrow{\mathbf{m}}_j)$$

- If $\Sigma_2 = \Sigma_1$, this is also the LDA solution.

This gives two different **interpretations** of LDA:

- It is optimal *if and only if* the classes are Gaussian and have equal covariance.
- A classifier on the LDA features, is equivalent to the boundary after the approximation of the data by two Gaussians with equal covariance.

The final discriminant decision boundary is $\overrightarrow{\mathbf{y}} = \overrightarrow{\mathbf{w}}^* \overrightarrow{\mathbf{x}} + w_0$

The *bias* term w_0 can be defined as:

$$w_0 = \left(\frac{1}{N_1} \sum_{n \in C_1} \overrightarrow{x}_n + \frac{1}{N_2} \sum_{n \in C_2} \overrightarrow{x}_n \right) \overrightarrow{\mathbf{w}}^*$$

- An extension to multi-class problems has a similar derivation.

Limitations of LDA:

1. LDA produces at most $C - 1$ feature projections, where C is the number of classes.
2. If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features.
3. LDA is a parametric method (it assumes unimodal Gaussian likelihoods).
4. If the distributions are significantly non-Gaussian, the LDA projections may not preserve complex structure in the data needed for classification.
5. LDA will also fail if discriminatory information is not in the mean but in the variance of the data.

A popular variant of LDA are the **Multi-Layer Perceptrons** (or MLPs).

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

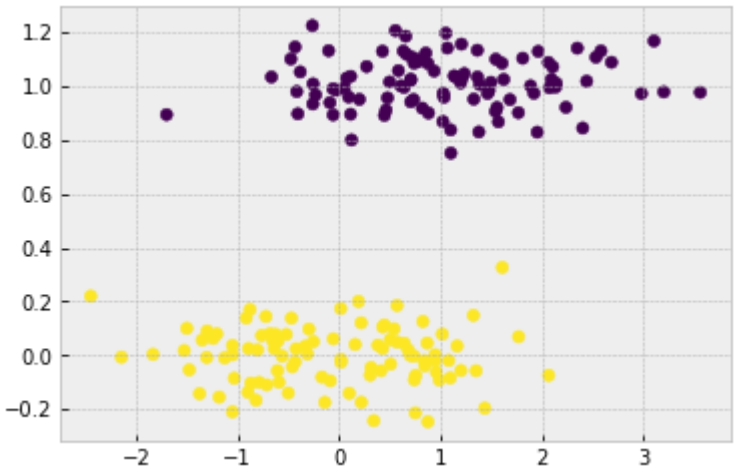
In [3]:
```
def fisherDiscriminant(data,t):
    data1 = data[t==0,:]
    data2 = data[t==1,:]
    mean1 = np.atleast_2d(np.mean(data1,0))
    mean2 = np.atleast_2d(np.mean(data2,0))
    Sw1 = np.dstack([(data1[i,:]-mean1).T@(data1[i,:]-mean1) for i in range(data1.shap
    Sw2 = np.dstack([(data2[i,:]-mean2).T@(data2[i,:]-mean2) for i in range(data2.shap
    Sw = np.sum(Sw1,2) + np.sum(Sw2,2)
    w = np.linalg.inv(Sw)@(mean2 - mean1).T
    w = w/np.linalg.norm(w)
    data_t = data@w
    return w, data_t

def discriminant(data, labels, v):
    v_perp = np.array([v[1], -v[0]])
    b = ((np.mean(data[labels==0,:],axis=0)+np.mean(data[labels==1,:],axis=0))/2)@v
```

```
    lambda_vec = np.linspace(-3,3,len(data))
    v_line = lambda_vec * v
    decision_boundary = b * v + lambda_vec * v_perp
    return v_line, decision_boundary

# Generate Synthetic Data
N1 = 100 #number of points for class1
N2 = 100 #number of points for class0
covM = [1,0.01]*np.eye(2) # covariance matrix
data = np.random.multivariate_normal([0,0], covM, N1) #generate points  for class 1
X = np.vstack((data, np.random.multivariate_normal([1,1], covM, N2))) #generate points
labels = np.hstack((np.ones(N1),np.zeros(N2)))

plt.scatter(X[:,0],X[:,1],c=labels); plt.show();
```



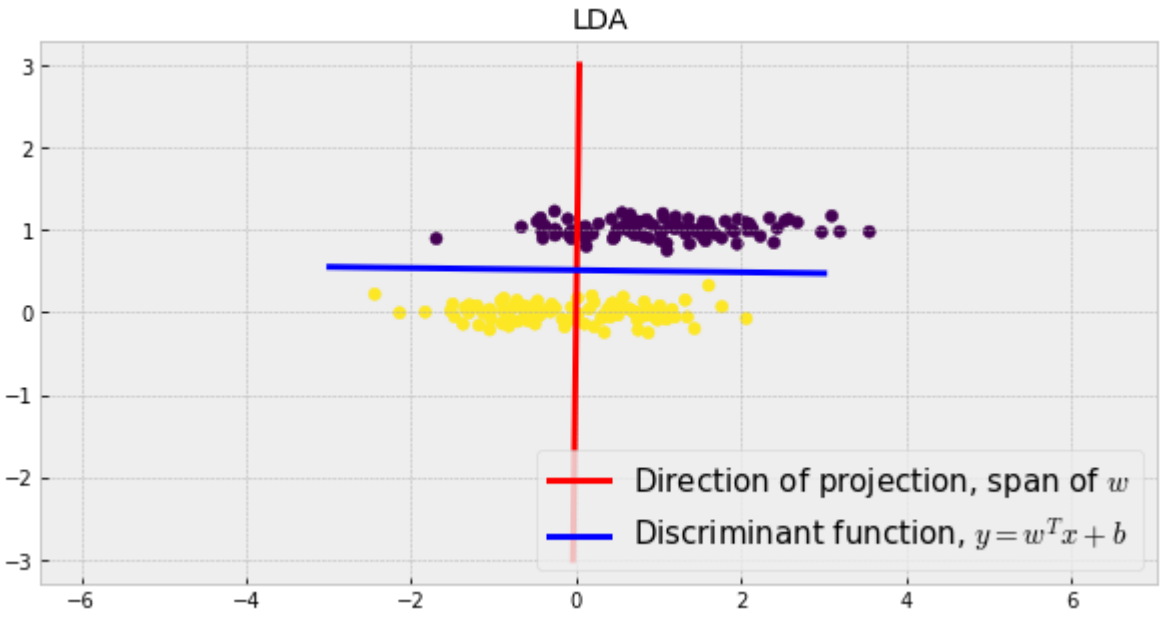
In [16]:

```
w, Y = fisherDiscriminant(X,labels)

plt.figure(figsize=(10,5))
plt.scatter(X[:,0],X[:,1],c=labels)

v_line, decision_boundary = discriminant(X, labels, w);

plt.plot(v_line[0], v_line[1], 'red', linewidth=3, label='Direction of projection, spa
plt.plot(decision_boundary[0,:], decision_boundary[1,:],'blue',linewidth=3, label='Dis
plt.title('LDA'); plt.axis('equal'); plt.legend(loc='lower right',fontsize=15);
```



In [17]:
```
w
```

Out[17]:
```
array([[-0.01291884],
       [-0.99991655]])
```

Fisher's LDA for Multiple Classes

We now consider the generalization of the Fisher discriminant to $K > 2$ classes, and we shall assume that the dimensionality D of the input space is greater than the number K of classes. Next, we introduce $M > 1$ linear "features" $\mathbf{y}_k = \mathbf{w}_k^T \mathbf{x}$, where $k = 1, \ldots, M$ and $M \leq D$.

These feature values can conveniently be grouped together to form a vector $\mathbf{y}$. Similarly, the weight vectors $\{\mathbf{w}_k\}$ can be considered to be the columns of a matrix $\mathbf{W}$, so that

$$\mathbf{y} = \mathbf{W}^T \mathbf{x}$$

Note that again we are not including any bias parameters in the definition of $\mathbf{y}$. The generalization of the within-class covariance matrix to the case of K classes is as follows:

$$S_W = \sum_{k=1}^{K} S_k$$

where

$$S_k = \sum_{n \in C_k} (x_n - \vec{\mathbf{m}}_k)^T (x_n - \vec{\mathbf{m}}_k)^T$$

$$\vec{\mathbf{m}}_k = \frac{1}{N_k} \sum_{n \in C_k} x_n$$

and N_k is the number of patterns in class C_k. In order to find a generalization of the between-class covariance matrix, let's consider first the total covariance matrix:

$$S_T = \sum_{n=1}^{N} (x_n - \overrightarrow{\mathbf{m}})(x_n - \overrightarrow{\mathbf{m}})^T$$

where $\overrightarrow{\mathbf{m}}$ is the mean of the total data set

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^{N} x_n = \frac{1}{N} \sum_{k=1}^{K} N_k \overrightarrow{\mathbf{m}}_k$$

and $N = \sum_{k=1}^{K} N_k$ is the total number of data points.

The total covariance matrix can be decomposed into the sum of the **within-class covariance scatter matrix** plus an additional matrix S_B, which we identify as a measure of the **between-class covariance**:

$$S_T = S_W + S_B$$

where

$$S_B = \sum_{k=1}^{K} N_k (\overrightarrow{\mathbf{m}}_k - \overrightarrow{\mathbf{m}})(\overrightarrow{\mathbf{m}}_k - \overrightarrow{\mathbf{m}})^T$$

These covariance matrices have been defined in the original x-input space. We can now define similar matrices in the projected M-dimensional y-space:

$$s_W = \sum_{k=1}^{K} \sum_{n \in C_k} (\mathbf{y}_n - \mu_k)(\mathbf{y}_n - \mu_k)^T$$

and

$$s_B = \sum_{k=1}^{K} N_k (\mu_k - \mu)(\mu_k - \mu)^T$$

where

$$\mu_k = \frac{1}{N_k} \sum_{n \in C_k} y_n$$

$$\mu = \frac{1}{N} \sum_{k=1}^{K} N_k \mu_k$$

Once again, we wish to construct a scalar that is large when the between-class covariance is large and when the within-class covariance is small.

$$J(\mathbf{W}) = \frac{s_B}{s_W}$$

This criterion can then be rewritten as an explicit function of the projection matrix $\mathbf{W}$ in the form

$$J(\mathbf{w}) = \frac{\mathbf{W}S_B\mathbf{W}^T}{\mathbf{W}S_W\mathbf{W}^T}$$

Maximization of such criteria is straightforward, though somewhat involved. The weight values are determined by those eigenvectors of $S_W^{-1}S_B$ that correspond to the M largest eigenvalues.

There is one important result that is common to all such criteria, which is worth emphasizing. We first note that SB is composed of the sum of K matrices, each of which is an outer product of two vectors and therefore of rank 1. In addition, only $(K-1)$ of these matrices are independent. Thus, S_B has rank at most equal to $(K-1)$ and so there are at most $(K-1)$ nonzero eigenvalues.

This shows that the projection onto the $(K-1)$-dimensional subspace spanned by the eigenvectors of S_B does not alter the value of $J(\mathbf{w})$, and so we are therefore unable to find more than $(K-1)$ linear "features".

Limitations of LDA:

1. LDA produces at most $K-1$ feature projections, where K is the number of classes.
2. If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features.
3. LDA is a parametric method (it assumes unimodal Gaussian likelihoods).
4. If the distributions are significantly non-Gaussian, the LDA projections may not preserve complex structure in the data needed for classification.
5. LDA will also fail if discriminatory information is not in the mean but in the variance of the data.

A popular variant of LDA are the **Multi-Layer Perceptrons** (or MLPs).
