Lecture 27 - Fisher's Linear Discriminant Analysis (LDA) continued

Fisher's Linear Discriminant

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:*

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

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

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

where

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

and

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

- Ok, so let's optimize:

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

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

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

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

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

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

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

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

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

- Note that if the within-class covariance, S_W, is isotropic, so that S_W is proportional to the unit matrix, we find that $\vec{\mathbf{w}}$ 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:

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

- 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 $\vec{\mathbf{y}} = \vec{\mathbf{w}}^* \vec{\mathbf{x}} + w_0$

The *bias* term w_0 can be defined as:

$$w_0 = \frac{1}{2} \left(\frac{1}{N_1} \sum_{n \in C_1} \vec{x}_n + \frac{1}{N_2} \sum_{n \in C_2} \vec{x}_n \right) \vec{\mathbf{w}}^*$$

- 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')
```

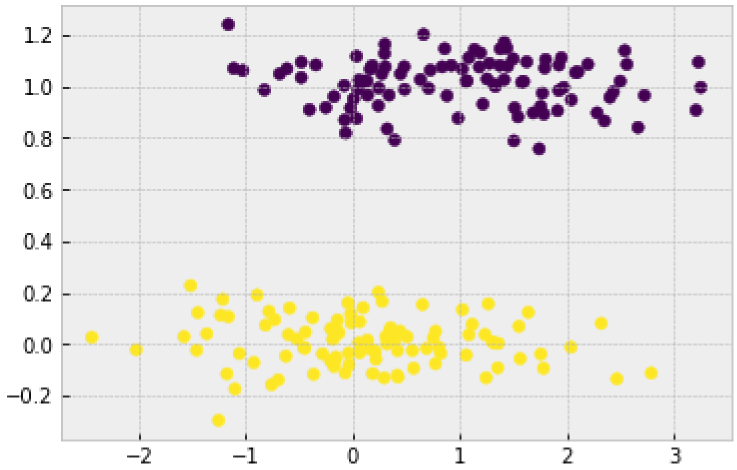
In [2]:

```
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.shape
    Sw2 = np.dstack([(data2[i,:]-mean2).T@(data2[i,:]-mean2) for i in range(data2.shape
    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();
```

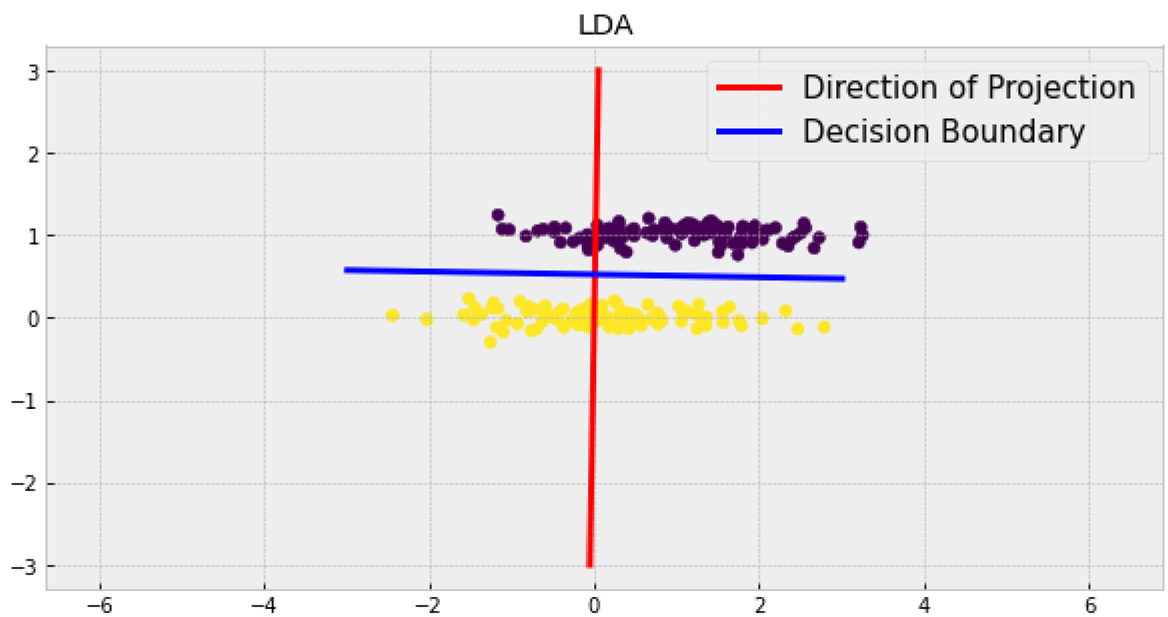


In [3]:
```
v, Y = fisherDiscriminant(X,labels)

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

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

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



Least Squares Classification as a Linear Discriminant Function

We could use a **least squares** error function to solve for $\vec{\mathbf{w}}$ and w_0 as we did in regression. But, there are some issues. *Can you think of any?*

- In regression, the prediction label will be a continuous number between $[-1, 1]$. So the predicted class label will be for example: -0.8, 0.4 or 0.01. To simplify, we can say, if the predicted class $y \geq 0$ than is class 1 otherwise is class 0.
- The problem that comes about is that, if we look at the distribution of our errors, in our estimation $\epsilon = t - y$ is not Gaussian.
- The errors samples are assumed independent, with a mean and a variance independent from each other.
- If we use regression, what we going end up with is an error distribution where the variance is dependent on the mean. This becomes a signal-dependent problem therefore regression is not a good approach to classification.