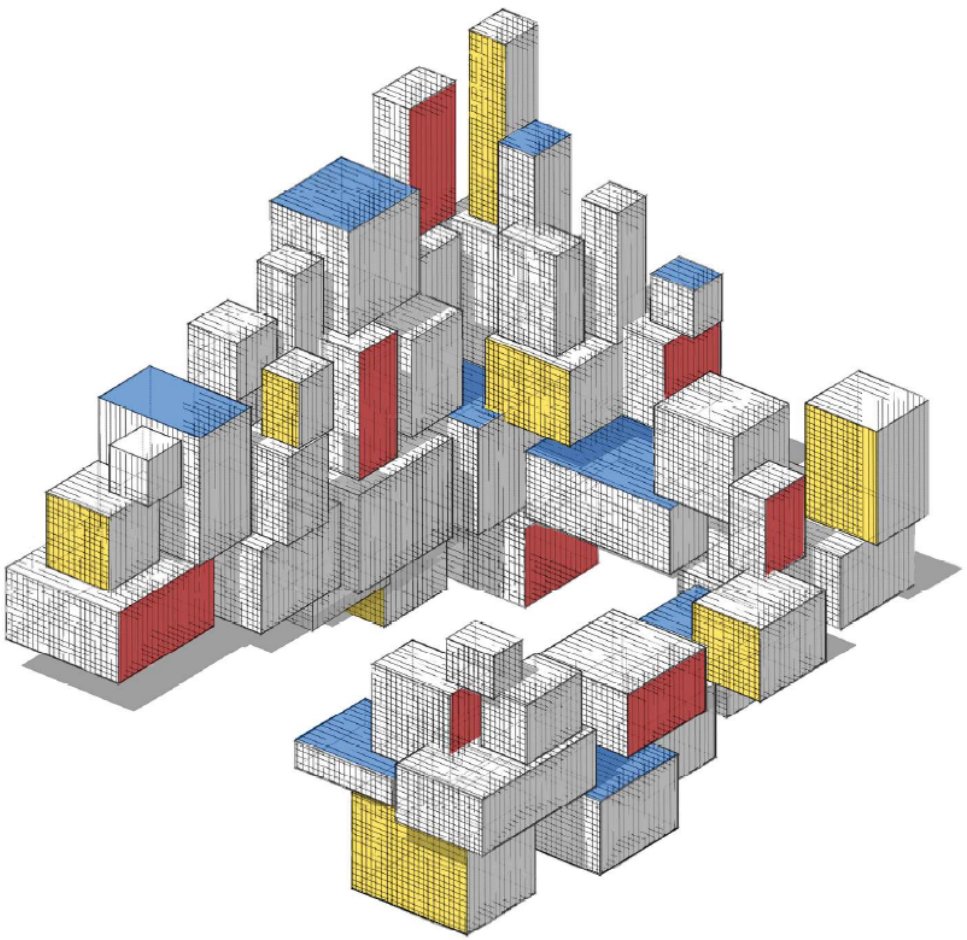
Linear and Quadratic Discriminant Analysis



Purpose

In this lecture, we consider:

- Linear discriminant analysis
- Quadratic discriminant analysis
- Classification after dimension reduction

Linear and Quadratic Discriminant Analysis

Recall the Bayesian model for (binary) classification in which the posterior density of the class label y is:

$$g(y \mid \boldsymbol{\theta}, \boldsymbol{x}) \propto \alpha_y \times g(\boldsymbol{x} \mid \boldsymbol{\theta}, y), \quad y \in \{0, 1\},$$

with the likelihood function of the feature vector $\boldsymbol{x} = [x_1, \ldots, x_p]^\top$:

$$g(\boldsymbol{x} \mid \boldsymbol{\theta}, y) = \frac{1}{\sqrt{(2\pi)^p \, |\boldsymbol{\Sigma}_y|}} \mathrm{e}^{-\frac{1}{2} (\boldsymbol{x}-\boldsymbol{\mu}_y)^\top \boldsymbol{\Sigma}_y^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_y)}, \quad \boldsymbol{x} \in \mathbb{R}^p, \quad y \in \{0, 1\},$$

where $\boldsymbol{\theta} = \{\alpha_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}_{j=0}^{c-1}$ collects all model parameters.

Quadratic Discriminant Function

According to the Bayes optimal decision rule, we classify $\boldsymbol{x}$ to come from class 0 if $\alpha_0\, g(\boldsymbol{x} \mid \boldsymbol{\theta}, 0) > \alpha_1\, g(\boldsymbol{x} \mid \boldsymbol{\theta}, 1)$ or, equivalently if,

$$\underbrace{\ln \alpha_0 - \frac{1}{2} \ln |\boldsymbol{\Sigma}_0| - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_0)}_{\delta_0(\boldsymbol{x})} > \underbrace{\ln \alpha_1 - \frac{1}{2} \ln |\boldsymbol{\Sigma}_1| - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}_1^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_1)}_{\delta_1(\boldsymbol{x})}.$$

The function $\delta_y(\boldsymbol{x})$ is called the quadratic discriminant function for class $y = 0, 1$.

A point $\boldsymbol{x}$ is classified to class y for which $\delta_y(\boldsymbol{x})$ is largest.

The function is quadratic in $\boldsymbol{x}$ and so the decision boundary $\{\boldsymbol{x} \in \mathbb{R}^p : \delta_0(\boldsymbol{x}) = \delta_1(\boldsymbol{x})\}$ is quadratic as well.

Linear Discriminant Function

An important simplification arises for the case where the assumption is made that $\boldsymbol{\Sigma}_0 = \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}$.

Now, the decision boundary is the set of $\boldsymbol{x}$ for which

$$\ln \alpha_0 - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_0) = \ln \alpha_1 - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_1).$$

Expanding the above expression shows that the quadratic term in $\boldsymbol{x}$ is eliminated, giving a *linear* decision boundary in $\boldsymbol{x}$:

$$\ln \alpha_0 - \frac{1}{2}\boldsymbol{\mu}_0 \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_0 + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_0 = \ln \alpha_1 - \frac{1}{2}\boldsymbol{\mu}_1 \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1.$$

The corresponding linear discriminant function for class y is

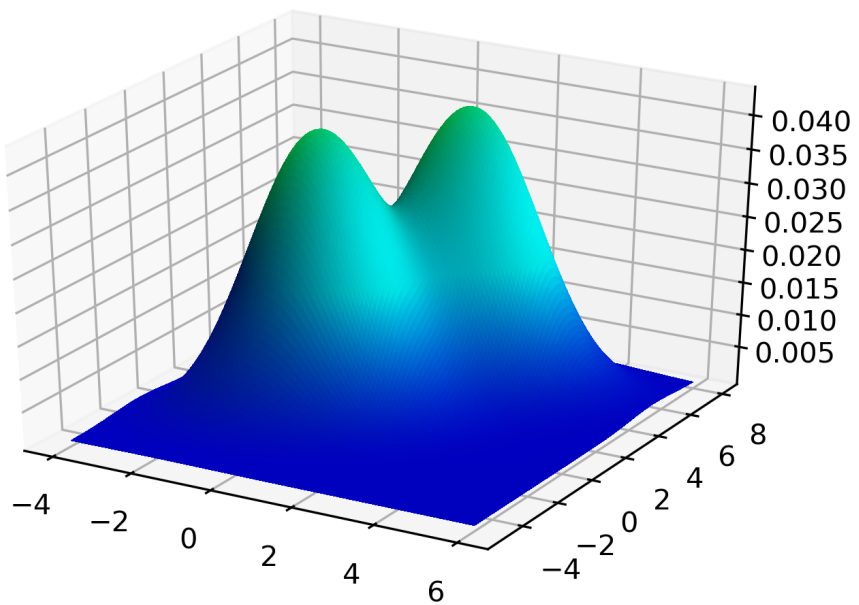
$$\delta_y(\boldsymbol{x}) = \ln \alpha_y - \frac{1}{2}\boldsymbol{\mu}_y^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_y + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_y, \quad \boldsymbol{x} \in \mathbb{R}^p.$$

Example: Linear Discriminant Analysis

Consider the case where $\alpha_0 = \alpha_1 = 1/2$ and

$$\boldsymbol{\Sigma} = \begin{bmatrix} 2 & 0.7 \\ 0.7 & 2 \end{bmatrix}, \quad \boldsymbol{\mu}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\mu}_1 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}.$$

The distribution of $\boldsymbol{X}$ is a mixture of two bivariate normal distributions, with pdf $g(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{2} g(\boldsymbol{x} \mid \boldsymbol{\theta}, y = 0) + \frac{1}{2} g(\boldsymbol{x} \mid \boldsymbol{\theta} y = 1)$.



LDAmixture.py

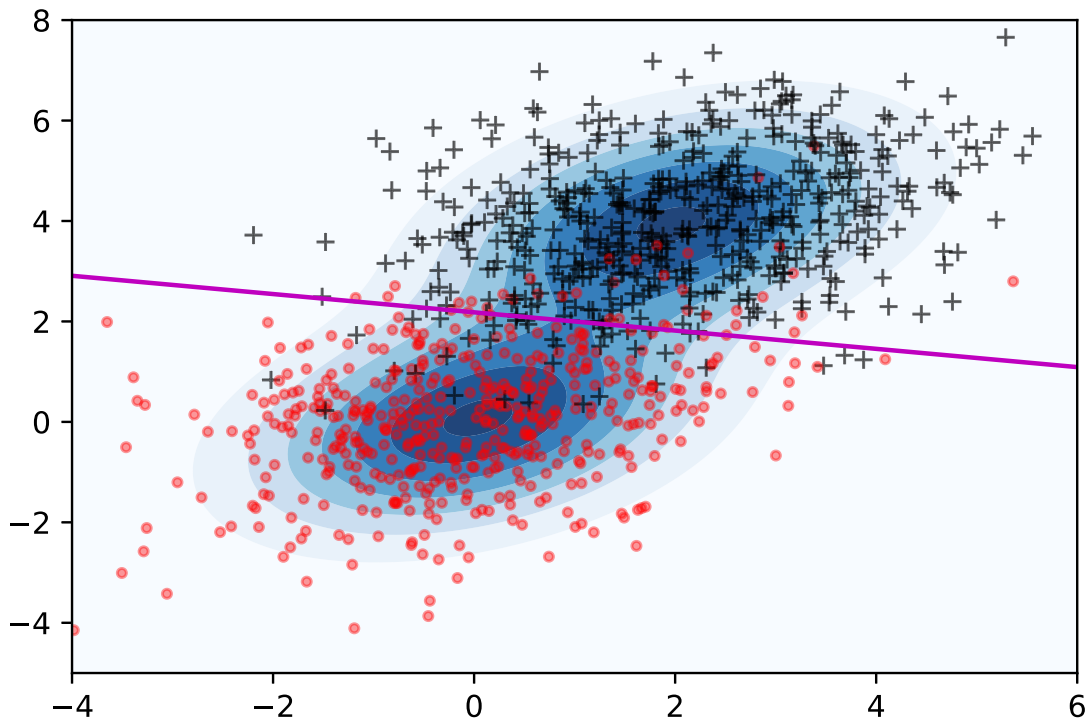
```
import numpy as np, matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LightSource

mu0, mu1 = np.array([0,0]), np.array([2,4])
Sigma = np.array([[2,0.7],[0.7, 2]])
x, y = np.mgrid[-4:6:150j,-5:8:150j]
mvn0 = multivariate_normal( mu0, Sigma )
mvn1 = multivariate_normal( mu1, Sigma )

xy = np.hstack((x.reshape(-1,1),y.reshape(-1,1)))
z = 0.5*mvn0.pdf(xy).reshape(x.shape) +  0.5*mvn1.pdf(xy).reshape(x.
     shape)

fig = plt.figure()
ax = fig.gca(projection='3d')
ls = LightSource(azdeg=180, altdeg=65)
cols = ls.shade(z, plt.cm.winter)
surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, linewidth=0,
                       antialiased=False, facecolors=cols)
plt.show()
```

Example: Linear Discriminant Analysis

The figure shows a contour plot of the mixture density, as well as 1000 simulated points from the mixture density.

To compute and display the linear decision boundary, let $[a_1, a_2]^\top = 2\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$ and $b = \boldsymbol{\mu}_0\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1$.

Then, the decision boundary can be written as $a_1x_1 + a_2x_2 + b = 0$.



LDA.py

```
from LDAmixture import *
from numpy.random import rand
from numpy.linalg import inv

fig = plt.figure()
plt.contourf(x, y,z, cmap=plt.cm.Blues, alpha= 0.9,extend='both')
plt.ylim(-5.0,8.0)
plt.xlim(-4.0,6.0)
M = 1000
r = (rand(M,1) < 0.5)
for i in range(0,M):
    if r[i]:
       u = np.random.multivariate_normal(mu0,Sigma,1)
       plt.plot(u[0][0],u[0][1],'.r',alpha = 0.4)
    else:
       u = np.random.multivariate_normal(mu1,Sigma,1)
       plt.plot(u[0][0],u[0][1],'+k',alpha = 0.6)

a = 2*inv(Sigma) @ (mu1-mu0);
b = ( mu0.reshape(1,2) @ inv(Sigma) @ mu0.reshape(2,1)
    - mu1.reshape(1,2) @ inv(Sigma) @mu1.reshape(2,1) )
xx = np.linspace(-4,6,100)
yy = (-(a[0]*xx +b)/a[1])[0]
plt.plot(xx,yy,'m')
plt.show()
```

Example: Quadratic Discriminant Analysis

As in the previous example, we consider a mixture of two Gaussians, but now with different covariance matrices. The figure shows the quadratic decision boundary.

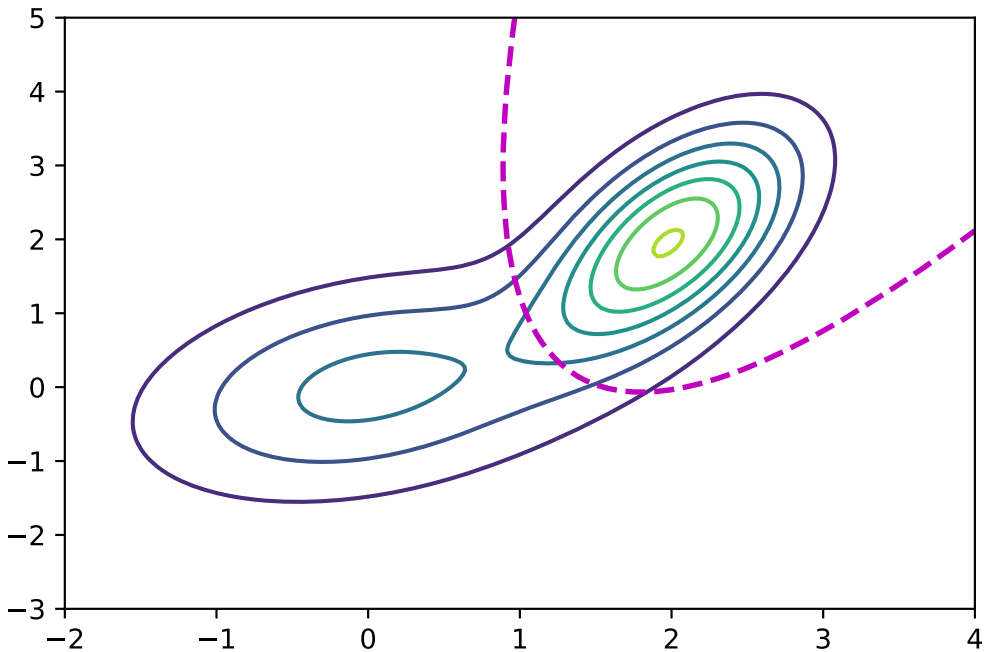


Figure: A quadratic decision boundary.

QDA.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

mu1 = np.array([0,0])
mu2 = np.array([2,2])
Sigma1 = np.array([[1,0.3],[0.3, 1]])
Sigma2 = np.array([[0.3,0.3],[0.3, 1]])
x, y = np.mgrid[-2:4:150j,-3:5:150j]
mvn1 = multivariate_normal( mu1, Sigma1 )
mvn2 = multivariate_normal( mu2, Sigma2 )

xy = np.hstack((x.reshape(-1,1),y.reshape(-1,1)))
z = ( 0.5*mvn1.pdf(xy).reshape(x.shape) +
      0.5*mvn2.pdf(xy).reshape(x.shape) )
plt.contour(x,y,z)

z1 = ( 0.5*mvn1.pdf(xy).reshape(x.shape) -
       0.5*mvn2.pdf(xy).reshape(x.shape))
plt.contour(x,y,z1, levels=[0],linestyles ='dashed',
            linewidths = 2, colors = 'm')
plt.show()
```

Estimating the Parameters

In practice, the true parameter $\boldsymbol{\theta} = \{\alpha_j, \Sigma_j, \boldsymbol{\mu}_j\}_{j=1}^{c}$ is not known and must be estimated from the training data — for example, by minimizing the cross-entropy training loss with respect to $\boldsymbol{\theta}$:

$$\frac{1}{n}\sum_{i=1}^{n} \text{Loss}(f(\boldsymbol{x}_i, y_i), g(\boldsymbol{x}_i, y_i \,|\, \boldsymbol{\theta})) = -\frac{1}{n}\sum_{i=1}^{n} \ln g(\boldsymbol{x}_i, y_i \,|\, \boldsymbol{\theta}),$$

where

$$\ln g(\boldsymbol{x}, y \,|\, \boldsymbol{\theta}) = \ln \alpha_y - \frac{1}{2}\ln|\boldsymbol{\Sigma}_y| - \frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_y)^\top \boldsymbol{\Sigma}_y^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_y) - \frac{p}{2}\ln(2\pi).$$

The corresponding estimates of the model parameters are:

$$\widehat{\alpha}_y = \frac{n_y}{n}, \quad \widehat{\boldsymbol{\mu}}_y = \frac{1}{n_y}\sum_{i:y_i=y} \boldsymbol{x}_i, \quad \widehat{\boldsymbol{\Sigma}}_y = \frac{1}{n_y}\sum_{i:y_i=y} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_y)(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_y)^\top$$

for $y = 0, \ldots, c-1$, where $n_y := \sum_{i=1}^{n} \mathbb{I}\{y_i = y\}$. For the case where $\boldsymbol{\Sigma}_y = \boldsymbol{\Sigma}$ for all y, we have $\widehat{\boldsymbol{\Sigma}} = \sum_y \widehat{\alpha}_y \widehat{\boldsymbol{\Sigma}}_y$.

Extentions

When $c > 2$ classes are involved, the classification procedure carries through in exactly the same way, leading to the same quadratic and linear discriminant functions as before.

The space $\mathbb{R}^p$ now is partitioned into c regions, determined by the linear or quadratic boundaries determined by each pair of Gaussians.

For the linear discriminant case (that is, when $\mathbf{\Sigma}_y = \mathbf{\Sigma}$ for all y), it is convenient to first transform the data as follows:

Let $\mathbf{B}$ be an invertible matrix such that $\mathbf{\Sigma} = \mathbf{B}\mathbf{B}^\top$, obtained, for example, via the Cholesky method. We linearly transform each data point $\boldsymbol{x}$ to $\boldsymbol{x}' := \mathbf{B}^{-1}\boldsymbol{x}$ and each mean $\boldsymbol{\mu}_y$ to $\boldsymbol{\mu}'_y := \mathbf{B}^{-1}\boldsymbol{\mu}_y$, $y = 0, \ldots, c-1$.

Sphering the Data

Let the random vector $\boldsymbol{X}$ be distributed according to the mixture pdf

$$g_{\boldsymbol{X}}(\boldsymbol{x} \mid \boldsymbol{\theta}) := \sum_y \alpha_y \, \frac{1}{\sqrt{(2\pi)^p \, |\boldsymbol{\Sigma}|}} \, \mathrm{e}^{-\frac{1}{2} \, (\boldsymbol{x}-\boldsymbol{\mu}_y)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_y)}.$$

Then, by the linear transformation theorem, $\boldsymbol{X}' = \mathbf{B}^{-1}\boldsymbol{X}$ has density

$$g_{\boldsymbol{X}'}(\boldsymbol{x}' \mid \boldsymbol{\theta}) = \frac{g_{\boldsymbol{X}}(\boldsymbol{x} \mid \boldsymbol{\theta})}{|\mathbf{B}^{-1}|} = \sum_{y=0}^{c-1} \frac{\alpha_y}{\sqrt{(2\pi)^p}} \, \mathrm{e}^{-\frac{1}{2} \, (\boldsymbol{x}-\boldsymbol{\mu}_y)^\top (\mathbf{B}\mathbf{B}^\top)^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_y)}$$

$$= \sum_{y=0}^{c-1} \frac{\alpha_y}{\sqrt{(2\pi)^p}} \, \mathrm{e}^{-\frac{1}{2} \, (\boldsymbol{x}'-\boldsymbol{\mu}'_y)^\top (\boldsymbol{x}'-\boldsymbol{\mu}'_y)} = \sum_{y=0}^{c-1} \frac{\alpha_y}{\sqrt{(2\pi)^p}} \, \mathrm{e}^{-\frac{1}{2} \, \|\boldsymbol{x}'-\boldsymbol{\mu}'_y\|^2}.$$

This is the pdf of a mixture of $\mathcal{N}(\boldsymbol{\mu}_y, \mathbf{I}_p)$ distributions.

The transformation is said to sphere the data, as contours of each mixture component are perfect spheres.

Classification after Dimension Reduction

Classification of the sphered data is now easier: classify $\boldsymbol{x}$ as

$$\widehat{y} := \underset{y}{\operatorname{argmin}}\{ \|\boldsymbol{x}' - \boldsymbol{\mu}'_y\|^2 - 2\ln \alpha_y \}.$$

Note that this rule only depends on the prior probabilities $\{\alpha_y\}$ and the distance from $\boldsymbol{x}'$ to the transformed means $\{\boldsymbol{\mu}'_y\}$.

This procedure can lead to a significant dimensionality reduction of the data. Namely, the data can be projected onto the space spanned by the differences between the mean vectors $\{\boldsymbol{\mu}'_y\}$.

When there are c classes, this is a $(c-1)$-dimensional space, as opposed to the p-dimensional space of the original data.

Example: Dimension Reduction

Consider an equal mixture of three 3-dimensional Gaussian distributions with identical covariance matrices. After sphering the data, the covariance matrices are all equal to the identity matrix.

Suppose the mean vectors of the sphered data are $\boldsymbol{\mu}_1 = [2, 1, -3]^\top$, $\boldsymbol{\mu}_2 = [1, -4, 0]^\top$, and $\boldsymbol{\mu}_3 = [2, 4, 6]^\top$.

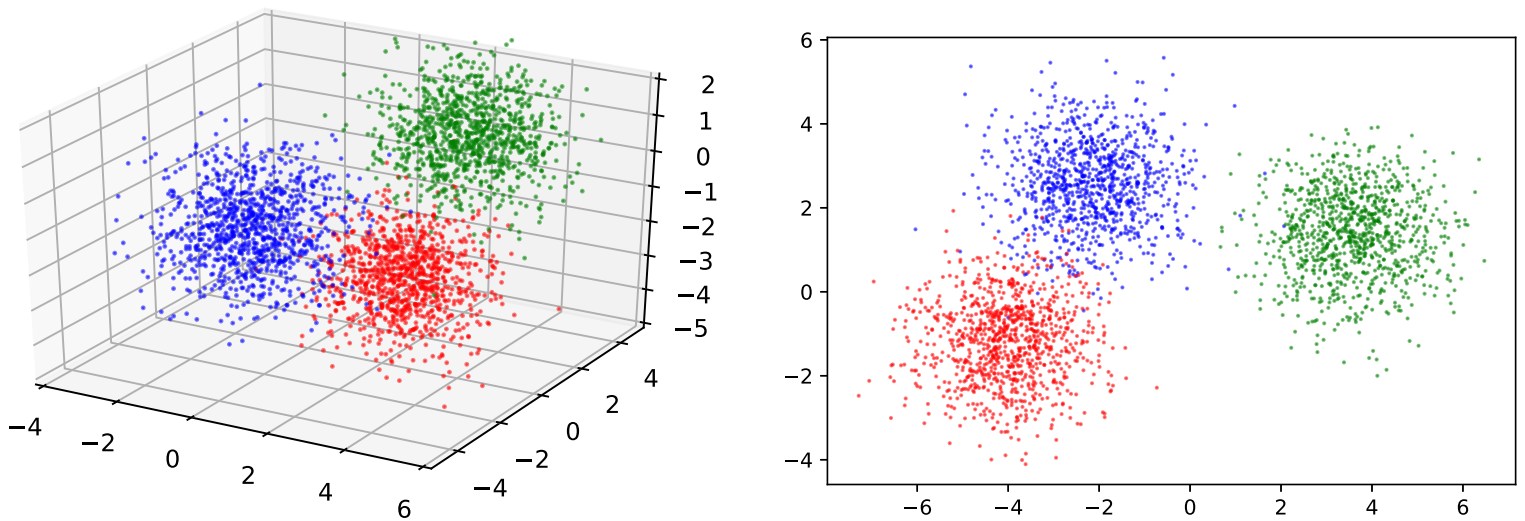


Figure: Left: original data. Right: projected data.

The data are stored in three 1000×3 matrices $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$.

datared.py

```
import numpy as np
from numpy.random import randn
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

n=1000
mu1 = np.array([2,1,-3])
mu2 = np.array([1,-4,0])
mu3 = np.array([2,4,0])
X1 = randn(n,3) + mu1
X2 = randn(n,3) + mu2
X3 = randn(n,3) + mu3
fig = plt.figure()
ax = fig.gca(projection='3d',)
ax.plot(X1[:,0],X1[:,1],X1[:,2],'r.',alpha=0.5,markersize=2)
ax.plot(X2[:,0],X2[:,1],X2[:,2],'b.',alpha=0.5,markersize=2)
ax.plot(X3[:,0],X3[:,1],X3[:,2],'g.',alpha=0.5,markersize=2)
ax.set_xlim3d(-4,6)
ax.set_ylim3d(-5,5)
ax.set_zlim3d(-5,2)
plt.show()
```

Example: Dimension Reduction

Since we have equal mixtures, we classify each data point $\boldsymbol{x}$ according to the closest distance to $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, or $\boldsymbol{\mu}_3$.

We can achieve a reduction in the dimensionality of the data by *projecting* the data onto the two-dimensional affine space spanned by the $\{\boldsymbol{\mu}_i\}$; that is, all vectors are of the form

$$\boldsymbol{\mu}_1 + \beta_1(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) + \beta_2(\boldsymbol{\mu}_3 - \boldsymbol{\mu}_1), \quad \beta_1, \beta_2 \in \mathbb{R}.$$

In fact, one may just as well project the data onto the subspace spanned by the vectors $\boldsymbol{\mu}_{21} = \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_{31} = \boldsymbol{\mu}_3 - \boldsymbol{\mu}_1$.

Let $\mathbf{W} = [\boldsymbol{\mu}_{21}, \boldsymbol{\mu}_{31}]$ be the 3×2 matrix whose columns are $\boldsymbol{\mu}_{21}$ and $\boldsymbol{\mu}_{31}$.

The orthogonal projection matrix onto the subspace $\mathcal{W}$ spanned by the columns of $\mathbf{W}$ is (by the projection theorem):

$$\mathbf{P} = \mathbf{W}\mathbf{W}^{+} = \mathbf{W}(\mathbf{W}^{\top}\mathbf{W})^{-1}\mathbf{W}^{\top}.$$

Example: Dimension Reduction

Let $\mathbf{UDV}^\top$ be the singular value decomposition of $\mathbf{W}$. Then $\mathbf{P}$ can also be written as

$$\mathbf{P} = \mathbf{UD}(\mathbf{D}^\top\mathbf{D})^{-1}\mathbf{D}^\top\mathbf{U}^\top.$$

Note that $\mathbf{D}$ has dimension 3×2, so is not square.

The first two columns of $\mathbf{U}$, say $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, form an orthonormal basis of the subspace $\mathcal{W}$.

What we want to do is rotate this subspace to the $x - y$ plane, mapping $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ to $[1, 0, 0]^\top$ and $[0, 1, 0]^\top$, respectively.

This is achieved via the rotation matrix $\mathbf{U}^{-1} = \mathbf{U}^\top$, giving the skewed projection matrix

$$\mathbf{R} = \mathbf{U}^\top\mathbf{P} = \mathbf{D}(\mathbf{D}^\top\mathbf{D})^{-1}\mathbf{D}^\top\mathbf{U}^\top,$$

whose 3rd row only contains zeros.

Applying $\mathbf{R}$ to all the data points, and ignoring the 3rd component of the projected points, gives the right panel of the previous figure.

dataproj.py

```
from datared import *
from numpy.linalg import svd, pinv
mu21 = (mu2 - mu1).reshape(3,1)
mu31 = (mu3 - mu1).reshape(3,1)
W = np.hstack((mu21, mu31))
U,_,_ = svd(W)  # we only need U
P = W @ pinv(W)
R = U.T @ P

RX1 = (R @ X1.T).T
RX2 = (R @ X2.T).T
RX3 = (R @ X3.T).T
plt.plot(RX1[:,0],RX1[:,1],'b.',alpha=0.5,markersize=2)
plt.plot(RX2[:,0],RX2[:,1],'g.',alpha=0.5,markersize=2)
plt.plot(RX3[:,0],RX3[:,1],'r.',alpha=0.5,markersize=2)
plt.show()
```