decision_boundaries_for_quadratic_discriminant_analysis

May 17, 2020

1 Decision Boundaries For Quadratic Discriminant Analysis

Assume that a classification is to be made. Given the parameters $\boldsymbol{\theta}$ and the input vector $\mathbf{x}$, the class of the output variable y is to be determined. The probability density function of y being from the class c is modeled using a discrete generative model as follows:

$$p\left(y=c|\mathbf{x},\boldsymbol{\theta}\right)=\frac{p\left(\mathbf{x}|y=c,\boldsymbol{\theta}\right)p\left(y=c|\boldsymbol{\theta}\right)}{p\left(\mathbf{x}|\boldsymbol{\theta}\right)}$$

If a multivariate Gaussian distribution is used to model the class conditional density of the input vector, i.e. $p\left(\mathbf{x}|y=c,\boldsymbol{\theta}\right)$, then this is called the quadratic discriminant analysis:

$$p\left(\mathbf{x}|y=c,\boldsymbol{\theta}\right)=\frac{1}{\left(2\pi\right)^{D/2}\left|\boldsymbol{\Sigma}_c\right|^{1/2}}\exp\left[-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)^T\boldsymbol{\Sigma}_c^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)\right]$$

The name "quadratic" comes from the fact that the decision boundaries separating classes are in general quadratic. Let the decision boundary between two classes be drawn to illustrate the quadratic boundary. First, the following equation is to be solved:

$$p\left(y=c_1|\mathbf{x},\boldsymbol{\theta}\right)=p\left(y=c_2|\mathbf{x},\boldsymbol{\theta}\right)\Rightarrow\frac{p\left(\mathbf{x}|y=c_1,\boldsymbol{\theta}\right)p\left(y=c_1|\boldsymbol{\theta}\right)}{p\left(\mathbf{x}|\boldsymbol{\theta}\right)}=\frac{p\left(\mathbf{x}|y=c_2,\boldsymbol{\theta}\right)p\left(y=c_2|\boldsymbol{\theta}\right)}{p\left(\mathbf{x}|\boldsymbol{\theta}\right)}\Rightarrow$$

$$p\left(\mathbf{x}|y=c_1,\boldsymbol{\theta}\right)p\left(y=c_1|\boldsymbol{\theta}\right)=p\left(\mathbf{x}|y=c_2,\boldsymbol{\theta}\right)p\left(y=c_2|\boldsymbol{\theta}\right)\Rightarrow$$

$$\pi_1\frac{1}{\left(2\pi\right)^{D/2}\left|\boldsymbol{\Sigma}_{c_1}\right|^{1/2}}\exp\left[-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)^T\boldsymbol{\Sigma}_{c_1}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)\right]=\pi_2\frac{1}{\left(2\pi\right)^{D/2}\left|\boldsymbol{\Sigma}_{c_2}\right|^{1/2}}\exp\left[-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)^T\boldsymbol{\Sigma}_{c_2}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)\right]$$

After taking the natural logarithm of both sides, the following simplified form is obtained:

$$0=\frac{1}{2}\left[\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)^T\boldsymbol{\Sigma}_{c_1}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)-\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)^T\boldsymbol{\Sigma}_{c_2}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)\right]+\frac{1}{2}\log\left(\frac{\left|\boldsymbol{\Sigma}_{c_1}\right|}{\left|\boldsymbol{\Sigma}_{c_2}\right|}\right)-\log\left(\frac{\pi_{c_1}}{\pi_{c_2}}\right)$$

Since $\mathbf{x}$, $\boldsymbol{\mu}_{c_1}$ and $\boldsymbol{\mu}_{c_2}$ are column vectors, $\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)^T=\mathbf{x}^T-\boldsymbol{\mu}_{c_1}^T$ and $\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)^T=\mathbf{x}^T-\boldsymbol{\mu}_{c_2}^T$. After these equalities are used in the above equation, the following final form is obtained:

$$0=\frac{1}{2}\mathbf{x}^T\left(\boldsymbol{\Sigma}_{c_1}^{-1}-\boldsymbol{\Sigma}_{c_2}^{-1}\right)\mathbf{x}-\mathbf{x}^T\left(\boldsymbol{\Sigma}_{c_1}^{-1}\boldsymbol{\mu}_{c_1}-\boldsymbol{\Sigma}_{c_2}^{-1}\boldsymbol{\mu}_{c_2}\right)+\frac{1}{2}\left(\boldsymbol{\mu}_{c_1}^T\boldsymbol{\Sigma}_{c_1}^{-1}\boldsymbol{\mu}_{c_1}-\boldsymbol{\mu}_{c_2}^T\boldsymbol{\Sigma}_{c_2}^{-1}\boldsymbol{\mu}_{c_2}\right)+\frac{1}{2}\log\left(\frac{\left|\boldsymbol{\Sigma}_{c_1}\right|}{\left|\boldsymbol{\Sigma}_{c_2}\right|}\right)-\log\left(\frac{\pi_{c_1}}{\pi_{c_2}}\right)$$

This is a quadratic form. Hence, the decision boundary between the two classes is quadratic. Now, let this boundary be drawn for $D=2$, i.e. for the case when the random vectors are of dimension 2. If $A\triangleq\boldsymbol{\Sigma}_{c_1}^{-1}-\boldsymbol{\Sigma}_{c_2}^{-1}$, $B\triangleq\boldsymbol{\Sigma}_{c_1}^{-1}\boldsymbol{\mu}_{c_1}-\boldsymbol{\Sigma}_{c_2}^{-1}\boldsymbol{\mu}_{c_2}$ and $\mathbf{x}=\left[x_1\ x_2\right]^T$, then:

$$0=\frac{1}{2}a_{22}x_2^2+\left[\frac{1}{2}\left(a_{12}+a_{21}\right)x_1-b_{21}\right]x_2+\left[\frac{1}{2}a_{11}x_1^2-b_{11}x_1+\frac{1}{2}\left(\boldsymbol{\mu}_{c_1}^T\boldsymbol{\Sigma}_{c_1}^{-1}\boldsymbol{\mu}_{c_1}-\boldsymbol{\mu}_{c_2}^T\boldsymbol{\Sigma}_{c_2}^{-1}\boldsymbol{\mu}_{c_2}\right)+\frac{1}{2}\log\left(\frac{\left|\boldsymbol{\Sigma}_{c_1}\right|}{\left|\boldsymbol{\Sigma}_{c_2}\right|}\right)-\log\left(\right.$$

gives element x_2 given element x_1 of $\mathbf{x}$.

The covariance matrices of the distributions are to be formed. They must be positive definite. Let the distribution's eigenvectors be v_1 and v_2. Let the corresponding eigenvalues be λ_1 and λ_2 respectively. Then:

$$\mathbf{\Sigma}_c = \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} v_1 & v_2 \end{bmatrix}^{-1}$$

[1]:
```
import numpy as np
v1_t = np.array([1, 3])
v1_t = v1_t / np.linalg.norm(v1_t) # transpose of the first eigenvector
v2_t = np.array([3, -1]) # transpose of the second eigenvector
v2_t = v2_t / np.linalg.norm(v2_t)
eigenvector_matrix = np.array([v1_t, v2_t]).transpose() # eigenvector matrix
lambda1 = 1 # first eigenvalue
lambda2 = 2 # second eigenvalue
eigenvalue_matrix = np.array([[lambda1, 0],[0, lambda2]])
print("the eigenvector matrix of the first distribution:")
print(eigenvector_matrix)
print("the eigenvalue matrix of the first distribution")
print(eigenvalue_matrix)
cov_1 = np.matmul(np.matmul(eigenvector_matrix, eigenvalue_matrix), np.linalg.
 ↪inv(eigenvector_matrix))
print("the covariance matrix of the first distribution")
print(cov_1)
mean_1_t = np.array([1.0, 1.0]) # the mean vector of the first distribution
print("transpose of the mean vector of the first distribution")
print(mean_1_t)
```

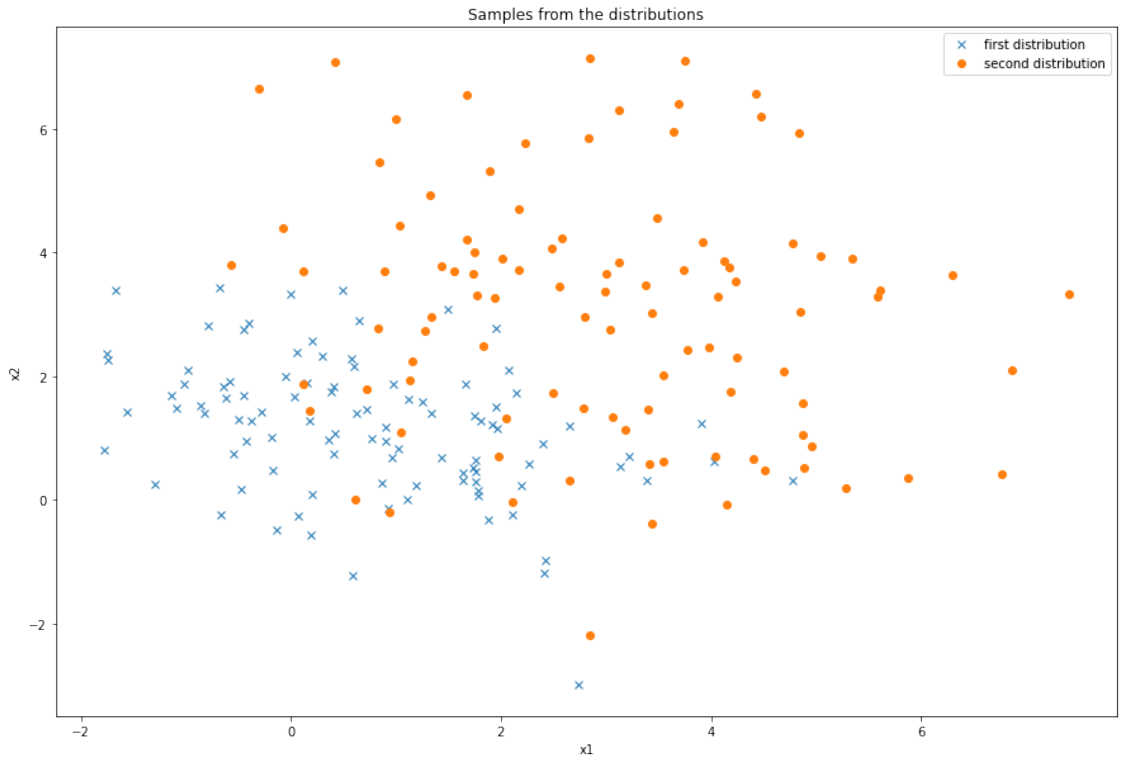
```
the eigenvector matrix of the first distribution:
[[ 0.31622777  0.9486833 ]
 [ 0.9486833  -0.31622777]]
the eigenvalue matrix of the first distribution
[[1 0]
 [0 2]]
the covariance matrix of the first distribution
[[ 1.9 -0.3]
 [-0.3  1.1]]
transpose of the mean vector of the first distribution
[1. 1.]
```

[2]:
```
v1_t = np.array([-1, 4])
v1_t = v1_t / np.linalg.norm(v1_t) # transpose of the first eigenvector
v2_t = np.array([4, 1])
v2_t = v2_t / np.linalg.norm(v2_t) # transpose of the second eigenvector
eigenvector_matrix = np.array([v1_t, v2_t]).transpose() # eigenvector matrix
lambda1 = 4 # first eigenvalue
lambda2 = 2 # second eigenvalue
```

```
eigenvalue_matrix = np.array([[lambda1, 0],[0, lambda2]])
print("the eigenvector matrix of the second distribution:")
print(eigenvector_matrix)
print("the eigenvalue matrix of the second distribution")
print(eigenvalue_matrix)
cov_2 = np.matmul(np.matmul(eigenvector_matrix, eigenvalue_matrix), np.linalg.
 ↪inv(eigenvector_matrix))
print("the covariance matrix of the second distribution")
print(cov_2)
mean_2_t = np.array([3.0, 3.0]) # the mean vector of the second distribution
print("transpose of the mean vector of the second distribution")
print(mean_2_t)
```

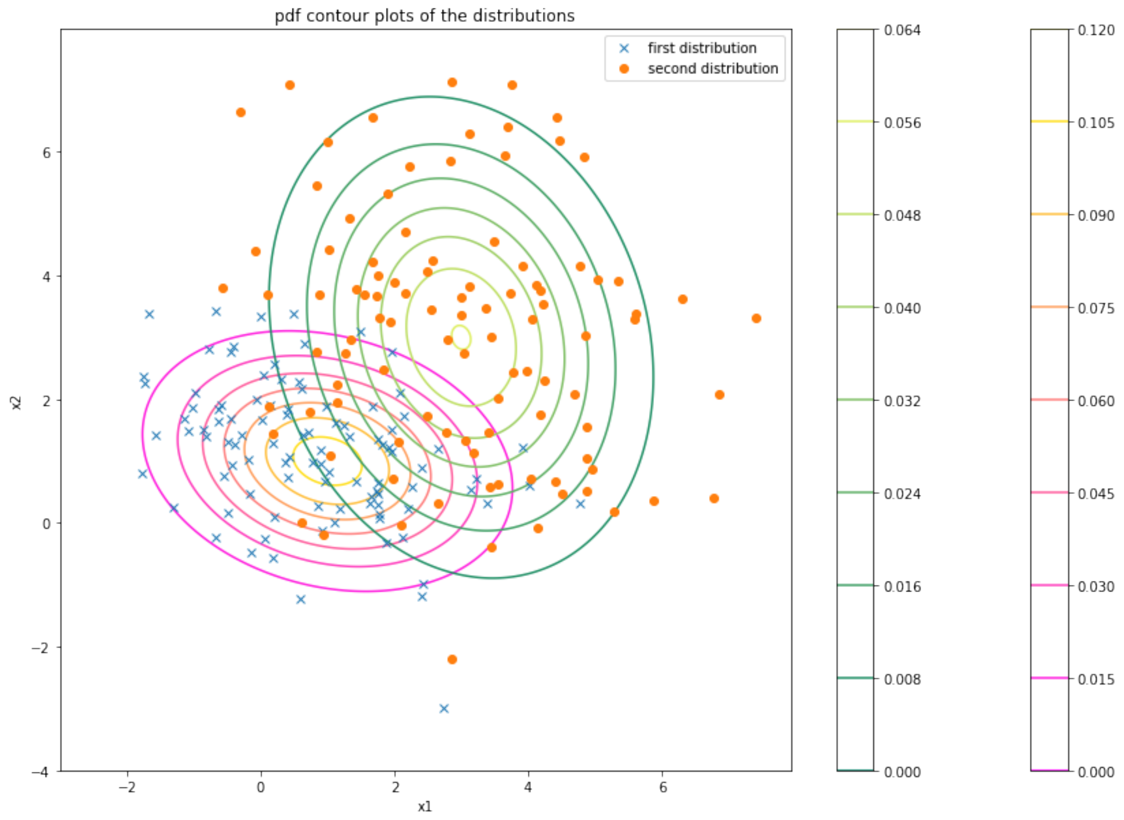
```
the eigenvector matrix of the second distribution:
[[-0.24253563  0.9701425 ]
 [ 0.9701425   0.24253563]]
the eigenvalue matrix of the second distribution
[[4 0]
 [0 2]]
the covariance matrix of the second distribution
[[ 2.11764706 -0.47058824]
 [-0.47058824  3.88235294]]
transpose of the mean vector of the second distribution
[3. 3.]
```

[3]:
```
import matplotlib.pyplot as plt

# samples from the first distribution
x1_s, y1_s = np.random.default_rng().multivariate_normal(mean_1_t, cov_1, 100).T

# samples from the second distribution
x2_s, y2_s = np.random.default_rng().multivariate_normal(mean_2_t, cov_2, 100).T

fig, ax = plt.subplots(figsize=(15, 10))
_ = ax.plot(x1_s, y1_s, 'x', label='first distribution')
_ = ax.plot(x2_s, y2_s, 'o', label='second distribution')
handles, labels = ax.get_legend_handles_labels()
_ = ax.legend(handles, labels)
_ = ax.set_title('Samples from the distributions')
_ = plt.xlabel('x1')
_ = plt.ylabel('x2')
```



[4]:
```
from scipy.stats import multivariate_normal

x_min = -3.0
x_max = 7.0
y_min = -4.0
y_max = 8.0
x, y = np.mgrid[x_min:x_max:.01, y_min:y_max:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y

rv1 = multivariate_normal(mean_1_t, cov_1)
rv2 = multivariate_normal(mean_2_t, cov_2)

cmap1 = plt.cm.get_cmap("spring")
cmap2 = plt.cm.get_cmap("summer")

fig, ax = plt.subplots(figsize=(15, 10))
contour_plot_1 = ax.contour(x, y, rv1.pdf(pos), cmap=cmap1)
contour_plot_2 = ax.contour(x, y, rv2.pdf(pos), cmap=cmap2)
_ = fig.colorbar(contour_plot_1, ax=ax)
_ = fig.colorbar(contour_plot_2, ax=ax)

_ = ax.plot(x1_s, y1_s, 'x', label='first distribution')
```

```
_ = ax.plot(x2_s, y2_s, 'o', label='second distribution')
handles, labels = ax.get_legend_handles_labels()
_ = ax.legend(handles, labels)

_ = ax.set_title('pdf contour plots of the distributions')
_ = plt.xlabel('x1')
_ = plt.ylabel('x2')
```



The equation to be solved for determining the decision boundary and the related variables are repeated. If $A \triangleq \mathbf{\Sigma}_{c_1}^{-1} - \mathbf{\Sigma}_{c_2}^{-1}$, $B \triangleq \mathbf{\Sigma}_{c_1}^{-1}\boldsymbol{\mu}_{c_1} - \mathbf{\Sigma}_{c_2}^{-1}\boldsymbol{\mu}_{c_2}$ and $\mathbf{x} = [x_1\ x_2]^T$, then:

$$0 = \frac{1}{2}a_{22}x_2^2 + \left[\frac{1}{2}(a_{12}+a_{21})\,x_1 - b_{21}\right]x_2 + \left[\frac{1}{2}a_{11}x_1^2 - b_{11}x_1 + \frac{1}{2}\left(\boldsymbol{\mu}_{c_1}^T\mathbf{\Sigma}_{c_1}^{-1}\boldsymbol{\mu}_{c_1} - \boldsymbol{\mu}_{c_2}^T\mathbf{\Sigma}_{c_2}^{-1}\boldsymbol{\mu}_{c_2}\right) + \frac{1}{2}\log\left(\frac{|\mathbf{\Sigma}_{c_1}|}{|\mathbf{\Sigma}_{c_2}|}\right) - \log\Bigg($$

gives element x_2 given element x_1 of $\mathbf{x}$.

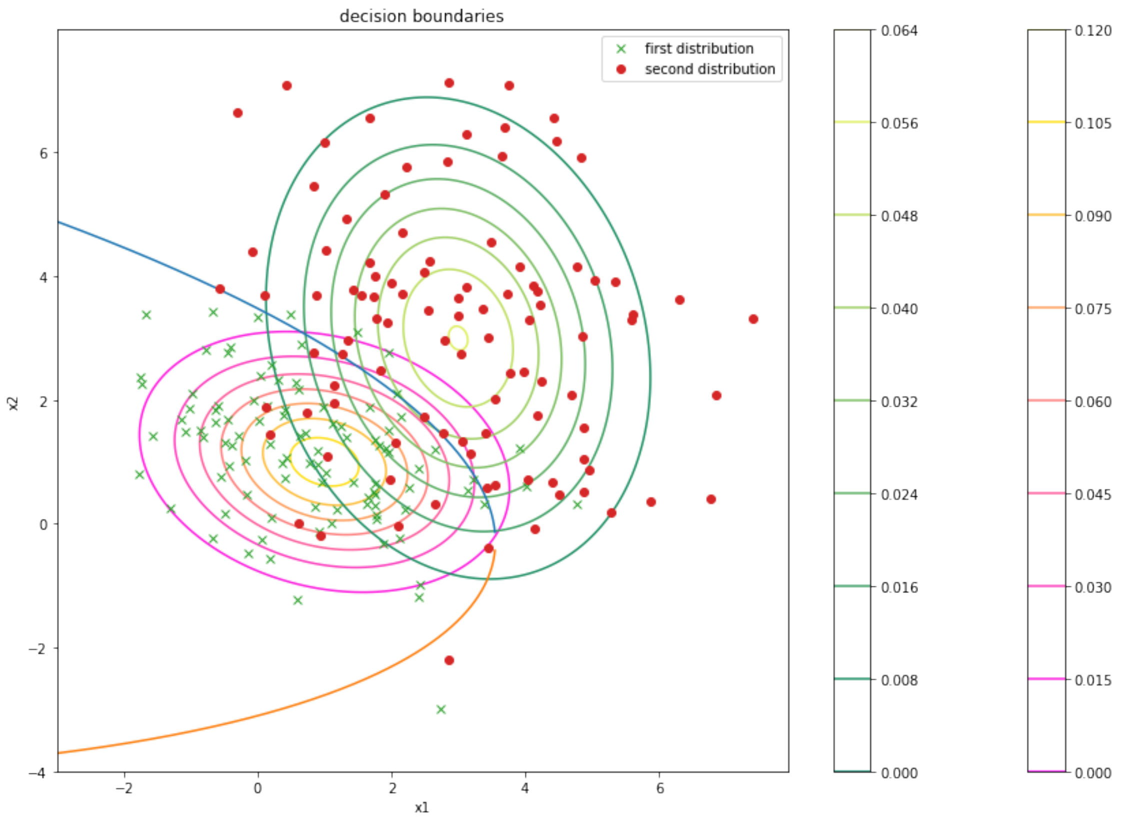
[5]:
```
A = np.linalg.inv(cov_1)-np.linalg.inv(cov_2)
a11 = A[0, 0]
a12 = A[0, 1]
a21 = A[1, 0]
a22 = A[1, 1]

cov1_inv = np.linalg.inv(cov_1)
```

```
cov2_inv = np.linalg.inv(cov_2)
B = np.matmul(cov1_inv, mean_1_t)-np.matmul(cov2_inv, mean_2_t)
b11 = B[0]
b21 = B[1]

pi_c1 = 0.5
pi_c2 = 0.5

x1 = np.arange(x_min, x_max, step=0.01)

a_term = 0.5*a22

b_term = 0.5*(a12+a21)*x1-b21

c_term = (0.5*a11*x1*x1)- \
         (b11*x1)+ \
         0.5*(np.matmul(mean_1_t, np.matmul(cov1_inv, mean_1_t))-
              np.matmul(mean_2_t, np.matmul(cov2_inv, mean_2_t)))+ \
         0.5*(np.log(np.linalg.det(cov_1))-np.log(np.linalg.det(cov_2)))- \
         (np.log(pi_c1)-np.log(pi_c2))

delta = b_term*b_term-4*a_term*c_term

x2_1 = (-b_term+np.sqrt(delta))/(2*a_term)
x2_2 = (-b_term-np.sqrt(delta))/(2*a_term)
```

```
/home/saffetgokcen/.local/share/virtualenvs/Gaussian_Models-2sbrQxYv/lib/python3
.6/site-packages/ipykernel_launcher.py:31: RuntimeWarning: invalid value
encountered in sqrt
/home/saffetgokcen/.local/share/virtualenvs/Gaussian_Models-2sbrQxYv/lib/python3
.6/site-packages/ipykernel_launcher.py:32: RuntimeWarning: invalid value
encountered in sqrt
```

[6]:
```
fig, ax = plt.subplots(figsize=(15.0, 10.0))
_ = ax.plot(x1, x2_1)
_ = ax.plot(x1, x2_2)
contour_plot_1 = ax.contour(x, y, rv1.pdf(pos), cmap=cmap1)
contour_plot_2 = ax.contour(x, y, rv2.pdf(pos), cmap=cmap2)
_ = fig.colorbar(contour_plot_1, ax=ax)
_ = fig.colorbar(contour_plot_2, ax=ax)

_ = ax.plot(x1_s, y1_s, 'x', label='first distribution')
_ = ax.plot(x2_s, y2_s, 'o', label='second distribution')
handles, labels = ax.get_legend_handles_labels()
_ = ax.legend(handles, labels)

_ = ax.set_title('decision boundaries')
```

```
_ = plt.xlabel('x1')
_ = plt.ylabel('x2')
```



2 References

Machine Learning A Probabilistic Perspective, Kevin P. Murphy