

Sebastian Raschka
last modified: 03/31/2014

Problem Category

- Statistical Pattern Recognition
 - Supervised Learning
 - Parametric Learning
 - Bayes Decision Theory
 - Multivariate data (2-dimensional)
 - 2-class problem
 - different variances
 - equal prior probabilities
 - Gaussian model (2 parameters)
 - with conditional Risk (1-0 loss functions)
-

Sections

- [Given information](#)
 - [Deriving the decision boundary](#)
 - [Classifying some random example data](#)
 - [Calculating the empirical error rate](#)
-

Given information:

[\[back to top\]](#)

model: continuous univariate normal (Gaussian) model for the class-conditional densities

$$p(\vec{x}|\omega_j) \sim N(\vec{\mu}|\Sigma)$$

$$p(\vec{x}|\omega_j) \sim \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu}) \right]$$

Prior probabilities:

$$P(\omega_1) = P(\omega_2) = 0.5$$

Loss functions:

where

$$\lambda(\alpha_i|\omega_j) = \lambda_{ij},$$

the loss occurred if $action_i$ is taken if the actual true class is ω_j (assuming that $action_i$ classifies sample as ω_i)

$$\lambda = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The samples are of 2-dimensional feature vectors:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Means of the sample distributions for 2-dimensional features:

$$\vec{\mu}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \vec{\mu}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Covariance matrices for the statistically independent and identically distributed ('i.i.d') features:

$$\Sigma_i = \begin{bmatrix} \sigma_{i1}^2 & \sigma_{i12}^2 \\ \sigma_{i21}^2 & \sigma_{i22}^2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Deriving the decision boundary

[\[back to top\]](#)

Bayes' Rule:

$$P(\omega_j | x) = \frac{p(x|\omega_j) * P(\omega_j)}{p(x)}$$

Risk Functions:

$$R(\alpha_1 | \vec{x}) = \lambda_{11}P(\omega_1 | \vec{x}) + \lambda_{12}P(\omega_2 | \vec{x})$$

$$R(\alpha_2 | \vec{x}) = \lambda_{21}P(\omega_1 | \vec{x}) + \lambda_{22}P(\omega_2 | \vec{x})$$

with 1-0 loss function:

$$R(\alpha_1 | \vec{x}) = P(\omega_2 | \vec{x}) = 1 - P(\omega_1 | \vec{x})$$

$$R(\alpha_2 | \vec{x}) = P(\omega_1 | \vec{x}) = 1 - P(\omega_2 | \vec{x})$$

Discriminant Functions:

The goal is to maximize the discriminant function, which we define as the posterior probability here to perform a **minimum-error classification** (Bayes classifier).

$$g_1(\vec{x}) = P(\omega_1 | \vec{x}), \quad g_2(\vec{x}) = P(\omega_2 | \vec{x})$$

$$\Rightarrow g_1(\vec{x}) = P(\vec{x} | \omega_1) \cdot P(\omega_1) \quad | \ln$$

$$g_2(\vec{x}) = P(\vec{x} | \omega_2) \cdot P(\omega_2) \quad | \ln$$

We can drop the prior probabilities (since we have equal priors in this case):

$$\Rightarrow g_1(\vec{x}) = \ln(P(\vec{x} | \omega_1))$$

$$g_2(\vec{x}) = \ln(P(\vec{x} | \omega_2))$$

$$\Rightarrow g_1(\vec{x}) = \vec{x}^t \cdot \frac{1}{2} \Sigma_1^{-1} \vec{x} + \left(\Sigma_1^{-1} \vec{\mu}_1 \right)^t + \left(-\frac{1}{2} \vec{\mu}_1^t \Sigma_1^{-1} \vec{\mu}_1 - \frac{1}{2} \ln(|\Sigma_1|) \right)$$

$$g_2(\vec{x}) = \vec{x}^t \cdot \frac{1}{2} \Sigma_2^{-1} \vec{x} + \left(\Sigma_2^{-1} \vec{\mu}_2 \right)^t + \left(-\frac{1}{2} \vec{\mu}_2^t \Sigma_2^{-1} \vec{\mu}_2 - \frac{1}{2} \ln(|\Sigma_2|) \right)$$

Let:

$$\vec{W}_i = -\frac{1}{2} \Sigma_i^{-1}$$

$$\vec{w}_i = \left(\Sigma_i^{-1} \vec{\mu}_i \right)^t$$

$$\omega_{i0} = \left(-\frac{1}{2} \vec{\mu}_i^t \Sigma_i^{-1} \vec{\mu}_i - \frac{1}{2} \ln(|\Sigma_i|) \right)$$

$$\vec{W}_1 = \begin{bmatrix} (1/4) & 0 \\ 0 & (1/4) \end{bmatrix}$$

$$\vec{w}_1 = \begin{bmatrix} (1/4) & 0 \\ 0 & (1/4) \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\omega_{10} = -\frac{1}{2} \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} (1/4) & 0 \\ 0 & (1/4) \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \ln(2) = -\ln(2)$$

$$\vec{W}_2 = \begin{bmatrix} (-1/2) & 0 \\ 0 & (-1/2) \end{bmatrix}$$

$$\vec{w}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\omega_{20} = -\frac{1}{2} \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} (1/4) & 0 \\ 0 & (1/4) \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \frac{1}{2} \ln(1) = -2.5$$

$$\Rightarrow g_1(\vec{x}) = \vec{x}^t \begin{bmatrix} (1/4) & 0 \\ 0 & (1/4) \end{bmatrix} \vec{x} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}^t - \ln(2) = \vec{x}^t - \frac{1}{4} \vec{x} - \ln(2)$$

$$\Rightarrow g_2(\vec{x}) = \vec{x}^t \begin{bmatrix} (-1/2) & 0 \\ 0 & (-1/2) \end{bmatrix} \vec{x} + \begin{bmatrix} 1 \\ 2 \end{bmatrix}^t \vec{x} - 2.5 = \vec{x}^t - \frac{1}{2} \vec{x} + \begin{bmatrix} 1 & 2 \end{bmatrix} \vec{x} - 2.5$$

Decision Boundary

$$g_1(\vec{x}) = g_2(\vec{x})$$

$$\Rightarrow \vec{x}^t - \frac{1}{4} \vec{x} - \ln(2) = \vec{x}^t - \frac{1}{2} \vec{x} + \begin{bmatrix} 1 & 2 \end{bmatrix} \vec{x} - 2.5 \quad \Bigg| \cdot 4$$

$$\Rightarrow \vec{x}^t - \vec{x} - 4\ln(2) = \vec{x}^t - 2\vec{x} + 4 \left(\begin{bmatrix} 1 & 2 \end{bmatrix} \vec{x} \right) - 10$$

$$\Rightarrow \begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \left(- \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) - 4\ln(2) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \left(- \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} \right) + \begin{bmatrix} 4 & 8 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 10$$

$$\Rightarrow -x_1^2 - x_2^2 - 4\ln(2) = -2x_1^2 - 2x_2^2 + 4x_1 + 8x_2 - 10$$

$$\Rightarrow x_1^2 + x_2^2 - 4x_1 - 8x_2 - 4\ln(2) + 10 = 0$$

Classifying some random example data

[\[back to top\]](#)

```
In [1]: %pylab inline

import numpy as np
from matplotlib import pyplot as plt

def decision_boundary(x_1):
    """ Calculates the x_2 value for plotting the decision boundary. """
    return 4 - np.sqrt(-x_1**2 + 4*x_1 + 6 + np.log(16))

def decision_rule(x_vec)
x_1^2 + x_2^2 - 4x_1 - 8x_2 - 4ln(2) + 10 = 0

# Generate 100 random patterns for class1
mu_vec1 = np.array([0,0])
cov_mat1 = np.array([[2,0],[0,2]])
x1_samples = np.random.multivariate_normal(mu_vec1, cov_mat1, 100)
mu_vec1 = mu_vec1.reshape(1,2).T # to 1-col vector

# Generate 100 random patterns for class2
mu_vec2 = np.array([1,2])
cov_mat2 = np.array([[1,0],[0,1]])
x2_samples = np.random.multivariate_normal(mu_vec2, cov_mat2, 100)
mu_vec2 = mu_vec2.reshape(1,2).T # to 1-col vector

# Scatter plot
f, ax = plt.subplots(figsize=(7, 7))
ax.scatter(x1_samples[:,0], x1_samples[:,1], marker='o', color='green', s=40, alpha=0.5)
ax.scatter(x2_samples[:,0], x2_samples[:,1], marker='^', color='blue', s=40, alpha=0.5)
plt.legend(['Class1 (w1)', 'Class2 (w2)'], loc='upper right')
plt.title('Densities of 2 classes with 100 bivariate random patterns each')
plt.ylabel('x2')
plt.xlabel('x1')
ftext = 'p(x|w1) ~ N(mu1=(0,0)^t, cov1=2*I) \ np(x|w2) ~ N(mu2=(1,2)^t, cov2=I)'
plt.figtext(.15,.8, ftext, fontsize=11, ha='left')
```

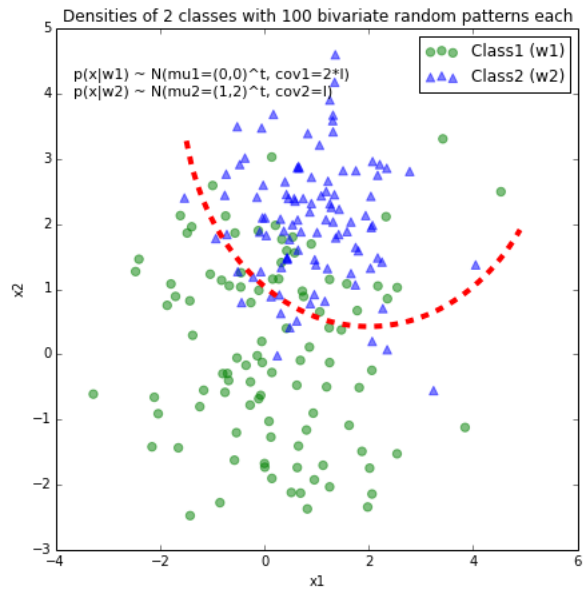
```
# Plot decision boundary
x_1 = np.arange(-5, 5, 0.1)
bound = decision_boundary(x_1)
plt.plot(x_1, bound, 'r--', lw=4)

x_vec = np.linspace(*ax.get_xlim())
x_1 = np.arange(0, 100, 0.05)

plt.show()
```

Populating the interactive namespace from numpy and matplotlib

-c:8: RuntimeWarning: invalid value encountered in sqrt



Calculating the empirical error rate

[\[back to top\]](#)

```
In [17]: def decision_rule(x_vec):
          """ Returns value for the decision rule of 2-d row vectors """
          x_1 = x_vec[0]
          x_2 = x_vec[1]
          return x_1**2 + x_2**2 - 4*x_1 - 8*x_2 - 4*np.log(2) + 10

          w1_as_w2, w2_as_w1 = 0, 0

          for x in x1_samples:
              if decision_rule(x) < 0:
                  w1_as_w2 += 1
          for x in x2_samples:
              if decision_rule(x) > 0:
                  w2_as_w1 += 1

          emp_err = (w1_as_w2 + w2_as_w1) / float(len(x1_samples) + len(x2_samples))

          print('Empirical Error: {}'.format(emp_err * 100))

          Empirical Error: 19.5%
```