pattern_classification (/github/rasbt/pattern_classification/tree/master) / stat_pattern_class (/github/rasbt/pattern_classification/tree/master/stat_pattern_class) / supervised (/github/rasbt/pattern_classification/tree/master/stat_pattern_class/supervised) / parametric (/github/rasbt/pattern_classification/tree/master/stat_pattern_class/supervised/parametric) /

Sebastian Raschka
last modified: 04/03/2014

Problem Category

- Statistical Pattern Recognition
- Supervised Learning
- Parametric Learning
- Bayes Decision Theory
- Multivariate data (2-dimensional)
- 2-class problem
- equal variances
- equal prior probabilities
- Gaussian model (2 parameters)
- no conditional Risk (1-0 loss functions)

Sections

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:

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

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:

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

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

$$\Sigma_i = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 \end{bmatrix},\ \Sigma_1 = \Sigma_2 = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

Class conditional probabilities:

$$p(\vec{x} \mid \omega_1) \sim N\left(\vec{\mu_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = I\right)$$

$$p(\vec{x} \mid \omega_2) \sim N\left(\vec{\mu_2} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \Sigma = I\right)$$

Deriving the decision boundary

[back to top]

Bayes' Rule:

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

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):

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

$$\Rightarrow g_1(\vec{x}) = \frac{1}{2\sigma^2} \left[\vec{x}^{\,t} - 2\vec{\mu_1}^{\,t}\vec{x} + \vec{\mu_1}^{\,t} \right] \mu_1$$
$$= -\frac{1}{2} \left[\vec{x}^{\,t}\vec{x} - 2\ [0 \ \ 0]\ \vec{x} + [0 \ \ 0] \ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right]$$
$$= -\frac{1}{2}\vec{x}^{\,t}\vec{x}$$

$$\Rightarrow g_2(\vec{x}) = \frac{1}{2\sigma^2} \left[\vec{x}^{\,t} - 2\vec{\mu_2}^{\,t}\vec{x} + \vec{\mu_2}^{\,t} \right] \mu_2$$
$$= -\frac{1}{2} \left[\vec{x}^{\,t}\vec{x} - 2\ 2\ [1 \ \ 1]\ \vec{x} + [1 \ \ 1] \ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right]$$
$$= -\frac{1}{2} \ \left[\vec{x}^{\,t}\vec{x} - 2\ [1 \ \ 1]\ \vec{x} + 2 \right]$$

Decision Boundary

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

$$\Rightarrow -\frac{1}{2}\vec{x}^{\,t}\vec{x} = -\frac{1}{2} \ \left[\vec{x}^{\,t}\vec{x} - 2\ [1 \ \ 1]\ \vec{x} + 2 \right]$$

$$\Rightarrow -2[1 \ \ 1]\vec{x} + 2 = 0$$

$$\Rightarrow [-2 \ \ -2]\ \vec{x} + 2 = 0$$

$$\Rightarrow -2x_1 - 2x_2 + 2 = 0$$

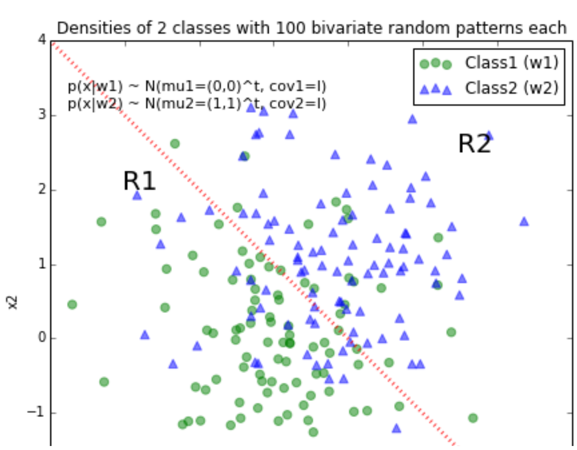
$$\Rightarrow -x_1 - x_2 + 1 = 0$$

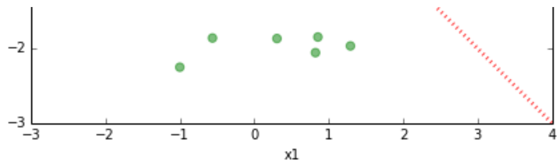
Classifying some random example data

[back to top]

In [16]:
```
%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 -x_1 + 1

# Generate 100 random patterns for class1
mu_vec1 = np.array([0,0])
cov_mat1 = np.array([[1,0],[0,1]])
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([1,1])
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=I)\np(x|w2) ~ N(mu2=(1,1)^t, cov2=I)'
plt.figtext(.15,.8, ftext, fontsize=11, ha='left')
plt.ylim([-3,4])
plt.xlim([-3,4])



# Plot decision boundary
x_1 = np.arange(-5, 5, 0.1)
bound = decision_boundary(x_1)
plt.annotate('R1', xy=(-2, 2), xytext=(-2, 2), size=20)
plt.annotate('R2', xy=(2.5, 2.5), xytext=(2.5, 2.5), size=20)
plt.plot(x_1, bound, color='r', alpha=0.8, linestyle=':', linewidth=3)

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

WARNING: pylab import has clobbered these variables: ['f']
`%matplotlib` prevents importing * from pylab and numpy
```





Calculating the Chernoff theoretical bounds for P(error)

[back to top]

$P(error) \leq p^{\beta}(\omega_1)\ p^{1-\beta}(\omega_2)\ e^{-(\beta(1-\beta))}$

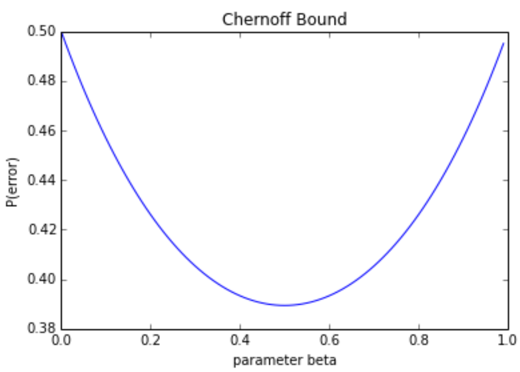
$\Rightarrow 0.5^{\beta} \cdot 0.5^{(1-\beta)}\ e^{-(\beta(1-\beta))}$

$\Rightarrow 0.5 \cdot e^{-\beta(1-\beta)}$

$min[P(\omega_1),\ P(\omega_2)] \leq 0.5\ e^{-(\beta(1-\beta))} \quad for\ P(\omega_1),\ P(\omega_2) \geq\ 0\ and\ 0\ \leq\ \beta\ \leq 1$

Plotting the Chernoff Bound for $0 \leq \beta \leq 1$

In [19]:
```
def chernoff_bound(beta):
    return 0.5 * np.exp(-beta * (1-beta))

betas = np.arange(0, 1, 0.01)
c_bound = chernoff_bound(betas)

plt.plot(betas, c_bound)
plt.title('Chernoff Bound')
plt.ylabel('P(error)')
plt.xlabel('parameter beta')

plt.show()
```



Finding the global minimum:

In [29]:
```
from scipy.optimize import minimize

x0 = [0.39] # initial guess (here: guessed based on the plot)
res = minimize(chernoff_bound, x0, method='Nelder-Mead')
print(res)
```

```
 success: True
     nit: 12
 message: 'Optimization terminated successfully.'
     fun: 0.38940039155946954
    nfev: 24
  status: 0
       x: array([ 0.49999219])
```

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 - x_2 + 1

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: 23.0%
```