Brenden Hein

CSE 802

Dr. Ross

Homework 3

1. $\sum = \begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \sigma_d^2 \end{bmatrix}$

$$p(x) = \frac{1}{\sqrt{2\pi}^d |\Sigma|^{\frac{1}{2}}} e^{[-\frac{1}{2}*(x-\mu)^t*\Sigma^{-1}*(x-\mu)]}$$

$$\frac{1}{\sqrt{2\pi}^d \left\| \begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \sigma_d^2 \end{bmatrix} \right\|^{\frac{1}{2}}} = \frac{1}{\sqrt{2\pi}^d * (\sigma_1^2 * \sigma_2^2 * \dots \sigma_d^2)^{\frac{1}{2}}} = \frac{1}{\sqrt{2\pi}^d * (\sigma_1 * \sigma_2 * \dots \sigma_d)} = \frac{1}{\prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}\sigma_i}}$$

$$e^{[-\frac{1}{2}*(x_i-\mu_i)^t*\Sigma^{-1}*(x_i-\mu_i)]}$$

$$= e^{-\frac{1}{2}*[x_1-\mu_1 \quad x_2-\mu_2 \quad \dots \quad x_d-\mu_d] \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2^2} & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \frac{1}{\sigma_d^2} \end{bmatrix} \begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \\ \dots \\ x_d-\mu_d \end{bmatrix}}$$

$$e^{[-\frac{1}{2}*(x_1-\mu_1)^t*\Sigma^{-1}*(x_1-\mu_1)]} = e^{-\frac{1}{2}*\begin{bmatrix} \frac{x_1-\mu_1}{\sigma_1^2} & \frac{x_2-\mu_2}{\sigma_2^2} & \dots & \frac{x_d-\mu_d}{\sigma_d^2} \end{bmatrix} \begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \\ \dots \\ x_d-\mu_d \end{bmatrix}}$$

$$e^{[-\frac{1}{2}*(x_1-\mu_1)^t*\Sigma^{-1}*(x_1-\mu_1)]} = e^{-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i-\mu_i)(x_i-\mu_i)}{\sigma_i^2}}$$

$$\boldsymbol{p(x) = \frac{1}{\prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}\sigma_i}} e^{-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i-\mu_i)^2}{\sigma_i^2}}}$$

2. $P_{original}(x) \sim N(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}20 & 10\\10 & 30\end{bmatrix})$

   a. $eigenvector = \begin{bmatrix}-0.8507 & -0.5257\\0.5257 & -0.8507\end{bmatrix}$

   $eigenvalues = [13.8197 \quad 36.1803]$

   $A_w = \begin{bmatrix}-0.8507 & -0.5257\\0.5257 & -0.8507\end{bmatrix}\begin{bmatrix}13.8197 & 0\\0 & 36.1803\end{bmatrix}^{-\frac{1}{2}}$

   $A_w = \begin{bmatrix}-0.8507 & -0.5257\\0.5257 & -0.8507\end{bmatrix}\begin{bmatrix}.07236 & 0\\0 & .02764\end{bmatrix}^{\frac{1}{2}}$

   $A_w = \begin{bmatrix}-0.8507 & -0.5257\\0.5257 & -0.8507\end{bmatrix}\begin{bmatrix}0.2690 & 0\\0 & 0.1663\end{bmatrix}$

   $A_w = \begin{bmatrix}-0.2288 & -0.0874\\0.1414 & 0.1414\end{bmatrix}$

   b. $A_w^t = \begin{bmatrix}-0.2288 & 0.1414\\-0.0874 & -0.1414\end{bmatrix}$

   $P_{transform}(x) \sim N(A^t\begin{bmatrix}0\\0\end{bmatrix}, A^t\begin{bmatrix}20 & 10\\10 & 30\end{bmatrix}A)$

   $N(\begin{bmatrix}-0.2288 & 0.1414\\-0.0874 & -0.1414\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}-0.2288 & 0.1414\\-0.0874 & -0.1414\end{bmatrix}\begin{bmatrix}20 & 10\\10 & 30\end{bmatrix}\begin{bmatrix}-0.2288 & -0.0874\\0.1414 & 0.1414\end{bmatrix})$

   $P_{transform}(x) \sim N(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix})$

   c. A plot of 10000 bivariate random patterns…

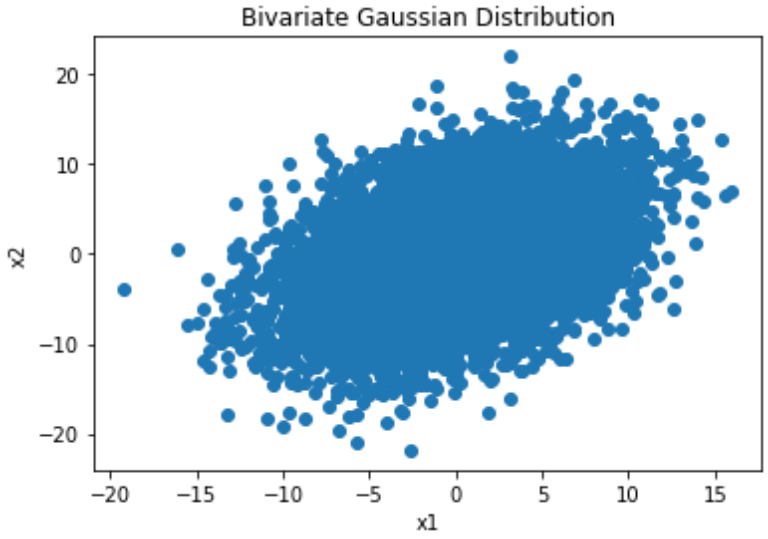

Fig. 1. 10,000 randomly generated bivariate points given a mean and covariance.

d. A plot of 10000 whitened bivariate random patterns

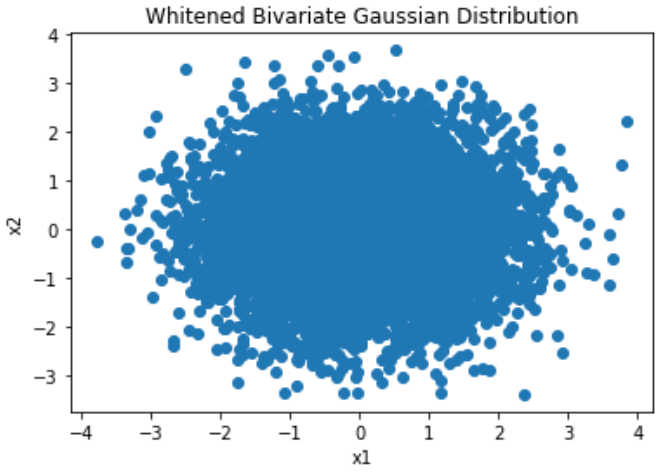

Fig. 2. 10,000 randomly generated whitened bivariate points given a mean and covariance.

e. In pattern 2c, the variables are positively correlated. This can be seen by the plots upwards tilt. However, in 2d, after applying the whitening transformation, the variables have become decorrelated (as there is no slant to the plot) and the distribution is more circular.

3. Given the 3 distributions…
   a. A plot of the 3 distributions' means can be seen below in figure 3.

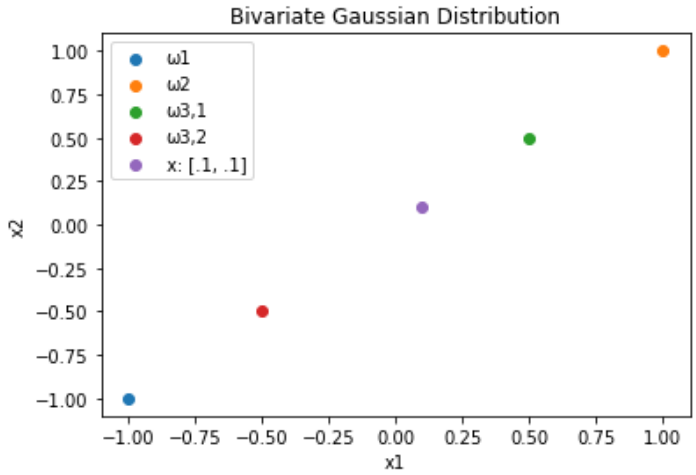

Fig. 3. A plot of [.1, .1], and the three means of the 3 classes

b. $p(x|\omega1) = 0.04746$
$p(x|\omega2) = 0.07080$
$p(x|\omega3) = 0.12333$

As the problem assumes 0-1 loss and equal priors, the bayes decision is based solely on $p(x|\omega j)$. As $p(x|\omega3)$ is the highest at the point [.1, .1] (closest to the mean as they all have the same spread of the identity matrix), it will be assigned to **ω3.**

4. If we have 2 distributions with the same covariance, but different means, and the covariance matrices are not assumed to be the identity matrix, then if the priors are equal, the decision boundary will pass halfway between the 2 means. However, the decision boundary is not guaranteed to be perpendicular to the line connecting the two means. If the decision boundary does not lie between the 2 means, then that implies that the difference between P(ω1) and P(ω2) is significant. When P(ωi) = P(ωj), the point of intersection can be seen below

$$x_0 = \frac{1}{2}(\mu_i + \mu_j) - \frac{\ln(1)}{(\mu_i - \mu_j)^t * \Sigma^{-1} * (\mu_i - \mu_j)}(\mu_i - \mu_j) = \boldsymbol{\frac{1}{2}(\mu_i + \mu_j)}$$

However, when the difference between P(ωi) and P(ωj) is large…

$$\frac{\ln\left(\frac{P(\omega i)}{P(\omega j)}\right)}{(\mu_i - \mu_j)^t * \Sigma^{-1} * (\mu_i - \mu_j)}(\mu_i - \mu_j)\ becomes\ significantly\ positive\ or\ negative, shifting\ x_0$$

$If\ the\ shift\ is\ large\ enough, the\ decision\ boundary\ doesnt\ fall\ between\ the\ 2\ means.$

5. Given two bivariate class conditional pdfs with equal priors…
   a. Case 1 holds, where $\Sigma = \sigma^2 I$, and with equal priors, the line will be perpendicular to the line between the two means, placed in the center of that line.

$$0 = g_1(x) - g_2(x)$$
$$0 = \frac{[4 \quad 4]}{1^2}x + \frac{-1}{2 * 1^2}[4 \quad 4]\begin{bmatrix}4\\4\end{bmatrix} + \ln\left(\frac{1}{2}\right) - \left(\frac{[0 \quad 0]}{1^2}x + \frac{-1}{2 * 1^2}[0 \quad 0]\begin{bmatrix}0\\0\end{bmatrix} + \ln\left(\frac{1}{2}\right)\right)$$
$$0 = \frac{[4 \quad 4]}{1^2}x + \frac{-1}{2 * 1^2}[4 \quad 4]\begin{bmatrix}4\\4\end{bmatrix}$$
$$0 = [4 \quad 4]x + \frac{-1}{2}(16 + 16)$$
$$0 = [4 \quad 4]\begin{bmatrix}x_1\\x_2\end{bmatrix} - 16$$
$$16 = 4x_1 + 4x_2 \implies \boldsymbol{4 = x_1 + x_2}$$

   b. For a given 2 pdfs…

$$P(error|x) = \min\{p(\omega 1|x), p(\omega 2|x)\} = \min\left\{\frac{1}{2}p(x|\omega 1), \frac{1}{2}p(x|\omega 2)\right\}$$
$$P(error) = \int \min\left\{\frac{1}{2}p(x|\omega 1), \frac{1}{2}p(x|\omega 2)\right\}p(x)dx$$
$$\min\left\{\frac{1}{2}p(x|\omega 1), \frac{1}{2}p(x|\omega 2)\right\} \leq \frac{1}{2}^{\beta} p(x|\omega 1)^{\beta} \frac{1}{2}^{1-\beta} p(x|\omega 2)^{1-\beta}$$
$$P(error) \leq \frac{1}{2}\int p(x|\omega 1)^{\beta} p(x|\omega 2)^{1-\beta} p(x)dx$$

   The Bhattacharyya theoretical bound….

$$k(1/2) = \frac{1}{8}[-4 \quad -4]\left(\begin{bmatrix}\frac{1}{2} & 0\\0 & \frac{1}{2}\end{bmatrix} + \begin{bmatrix}\frac{1}{2} & 0\\0 & \frac{1}{2}\end{bmatrix}\right)^{-1}\begin{bmatrix}-4\\-4\end{bmatrix} + \frac{1}{2}\ln\left(\frac{\left|\begin{bmatrix}\frac{1}{2} & 0\\0 & \frac{1}{2}\end{bmatrix} + \begin{bmatrix}\frac{1}{2} & 0\\0 & \frac{1}{2}\end{bmatrix}\right|}{\sqrt{\left|\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right|\left|\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right|}}\right)$$

$$k(1/2) = \frac{1}{8}\begin{bmatrix}-4 & -4\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-4\\ -4\end{bmatrix} + \frac{1}{2}\ln\left(\frac{\left|\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right|}{1}\right)$$

$$k(1/2) = \frac{1}{8}\begin{bmatrix}-4 & -4\end{bmatrix}\begin{bmatrix}-4\\ -4\end{bmatrix} = \frac{32}{8} = 4$$

$$\boldsymbol{Bhatacharyya = \frac{e^{-4}}{2} = \frac{.018316}{2} = .009158}$$

The Chernoff theoretical bound given equal priors…

$$k(\beta^*) = \frac{\beta^*(1-\beta^*)}{2}\begin{bmatrix}-4 & -4\end{bmatrix}\left(\beta^*\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} + (1-\beta^*)\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right)^{-1}\begin{bmatrix}-4\\ -4\end{bmatrix} + \frac{1}{2}\ln\left(\frac{\left|\beta^*\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} + (1-\beta^*)\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right|}{\left|\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right|^{\beta}\left|\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right|^{1-\beta}}\right)$$

$$k(\beta^*) = \frac{\beta^*(1-\beta^*)}{2}\begin{bmatrix}-4 & -4\end{bmatrix}\left(\begin{bmatrix}\beta^* & 0\\ 0 & \beta^*\end{bmatrix} + \begin{bmatrix}1-\beta^* & 0\\ 0 & 1-\beta^*\end{bmatrix}\right)^{-1}\begin{bmatrix}-4\\ -4\end{bmatrix} + \frac{1}{2}\ln\left(\frac{\left|\begin{bmatrix}\beta^* & 0\\ 0 & \beta^*\end{bmatrix} + \begin{bmatrix}1-\beta^* & 0\\ 0 & 1-\beta^*\end{bmatrix}\right|}{1^{\beta^*}1^{1-\beta^*}}\right)$$

$$k(\beta^*) = \frac{\beta^*(1-\beta^*)}{2}\begin{bmatrix}-4 & -4\end{bmatrix}\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right)^{-1}\begin{bmatrix}-4\\ -4\end{bmatrix} + \frac{1}{2}\ln\left(\frac{\left|\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right|}{1^{\beta^*}1^{1-\beta^*}}\right)$$

$$k(\beta^*) = \frac{\beta^*(1-\beta^*)}{2}\begin{bmatrix}-4 & -4\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-4\\ -4\end{bmatrix} + \frac{1}{2}\ln\left(\frac{1}{1}\right)$$

$$k(\beta^*) = \frac{\beta^*(1-\beta^*)}{2}\begin{bmatrix}-4 & -4\end{bmatrix}\begin{bmatrix}-4\\ -4\end{bmatrix}$$

$$k(\beta^*) = \frac{\beta^*(1-\beta^*)}{2}[32] = 16\beta^*(1-\beta^*)$$

$$\boldsymbol{Chernoff = \frac{e^{-16\beta^* + 16\beta^{*2}}}{2} = \frac{.018316}{2} = .009158}$$

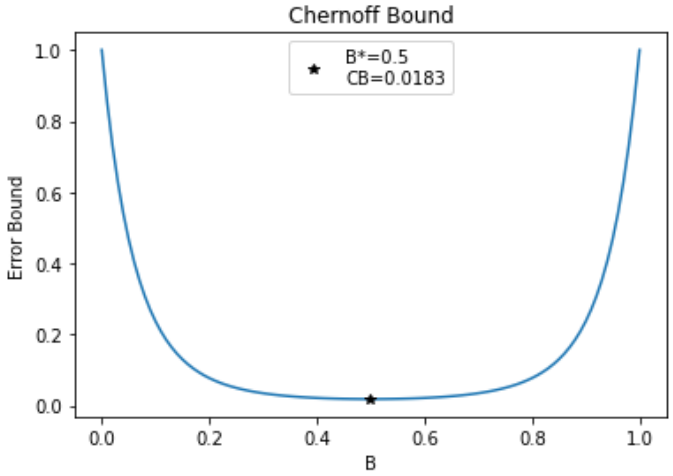

Fig. 4. A plot of the Chernoff bound as a function of B.

c. 25 test patterns for p(x|ω1) and p(x|ω2) can be seen below.

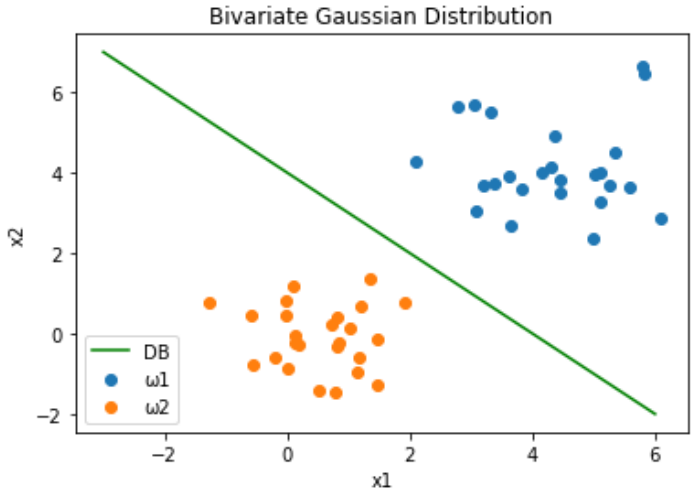

Fig. 5. 25 randomly generated samples from both ω1 and ω2 with the decision boundary

d. Confusion matrix

25	0
0	25

Empirical Error rate $= 0/50 =$ **0%**

e. The error rate cannot exceed the theoretical bounds on the probability of misclassification. By following Bayes decision rule for a two-class problem, the probability of error given x will be the minimum conditional probability for a class, ωj. You can minimize the probability of error for all x, by integrating over the minimum function. This integral leads to an upper bound on p(error), as the upper bound for a min{a,b} is a^B(b^(B-1)). As it is an upper bound, p(error) cannot surpass it. The demonstration of this with random samples of different sizes, n, can be seen below, where no matter the n, the error bound is not surpassed.

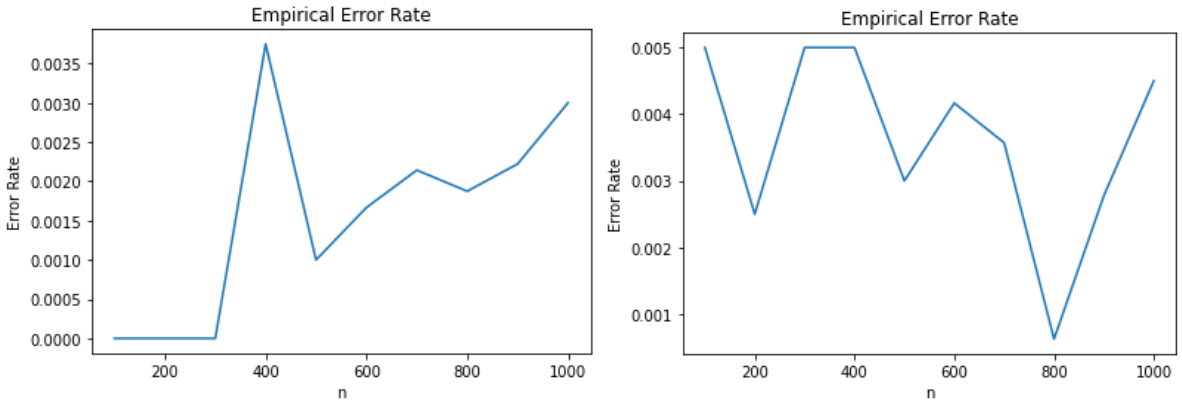

Fig. 6. Two distributions of error rates with n [100, 200, …. 1000]

6. Given a bunch of 1-dimensional…
   a. The MLE of theta

$$p(x|\theta) = 2\theta x e^{-\theta x^2}$$

$$\ln\left(p(x|\theta)\right) = \ln\left(2\theta x e^{-\theta x^2}\right)$$

$$\ln\big(p(x|\theta)\big) = \ln(2\theta x) + \ln(e^{-\theta x^2})$$

$$\ln\big(p(x|\theta)\big) = \ln(2) + \ln(\theta) + \ln(x) + -\theta x^2$$

$$\frac{d}{d\theta} = \frac{1}{\theta} - x^2 = 0$$

$$0 = \sum_{k=1}^{n} \frac{1}{\theta} - x^2 = \sum_{k=1}^{n} \frac{1}{\theta} - \sum_{k=1}^{n} x^2$$

$$0 = \frac{n}{\theta} - \sum_{k=1}^{n} x^2$$

$$\sum_{k=1}^{n} x^2 = \frac{n}{\theta} \;\Rightarrow\; \boldsymbol{\hat{\theta} = \frac{n}{\sum_{k=1}^{n} x^2}}$$

   b. A plot of the Rayleigh distribution given 1000 training samples with theta= 0.0195.

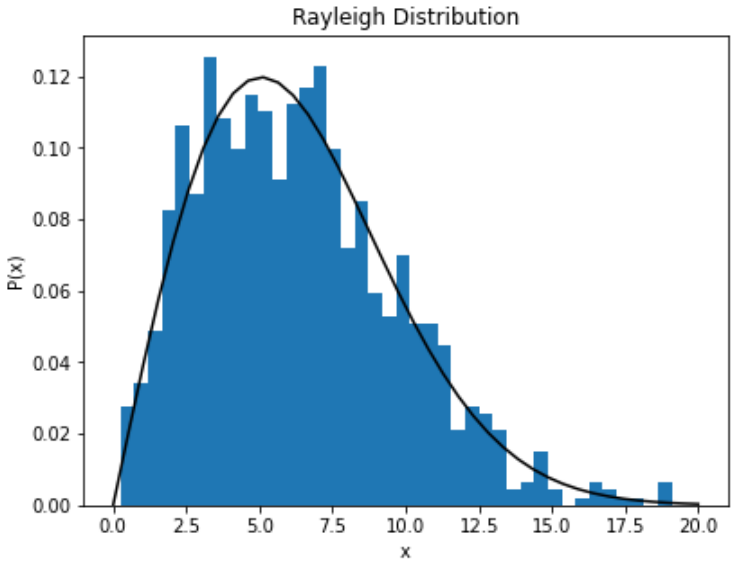

Fig. 7. A plot of the Rayleigh distribution given 1000 training samples.

c. A normal distribution with parameter estimates: mean = 6.307, covariance = 11.54.

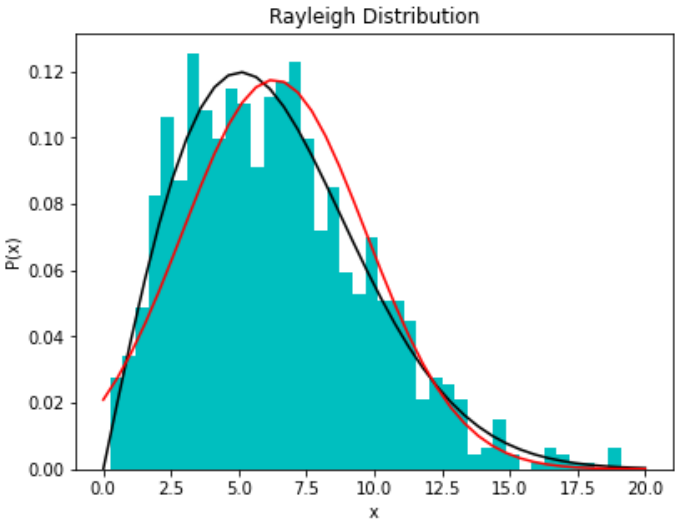

Fig. 8. A plot of the Rayleigh and the Gaussian distributions with the 1000 training samples

d. The Rayleigh distribution is the better fit for the data. The pdf more tightly hugs the values of the histogram, leading to a better, more accurate distribution.

7. After generating 50 random bivariate Gaussian patterns give a class ([0,0], I) and ([5,5],I)

a. $\mu_{MLE_1} = [0.175 \quad -0.0364]$

$\mu_{MLE_2} = [5.1142 \quad 4.9942]$

$\Sigma_{MLE_1} = \begin{bmatrix} 0.8046 & 0.1564 \\ 0.1564 & 1.0536 \end{bmatrix}$

$$\Sigma_{MLE_2} = \begin{bmatrix} 0.8677 & 0.0666 \\ 0.0666 & 1.1817 \end{bmatrix}$$

b. $$0 = \frac{1}{\sqrt{\left|\begin{bmatrix} 0.8046 & 0.1564 \\ 0.1564 & 1.0536 \end{bmatrix}\right|}} e^{\left[-\frac{1}{2}*[x_1-0.175 \;\; x_2+0.0364]*\begin{bmatrix} 0.8046 & 0.1564 \\ 0.1564 & 1.0536 \end{bmatrix}^{-1}*\left(\begin{bmatrix} x_1-0.175 \\ x_2+0.0364 \end{bmatrix}\right)\right]} -$$

$$\frac{1}{\sqrt{\left|\begin{bmatrix} 0.8677 & 0.0666 \\ 0.0666 & 1.1817 \end{bmatrix}\right|}} e^{\left[-\frac{1}{2}*[x_1-5.1142 \;\; x_2-4.9942]*\begin{bmatrix} 0.8677 & 0.0666 \\ 0.0666 & 1.1817 \end{bmatrix}^{-1}*\left(\begin{bmatrix} x_1-5.1142 \\ x_2-4.9942 \end{bmatrix}\right)\right]}$$

$$\frac{1}{1.01040} e^{\left[-\frac{1}{2}*[x_1-5.1142 \;\; x_2-4.9942]*\begin{bmatrix} 1.15747 & -0.0652 \\ -0.0652 & 0.84991 \end{bmatrix}*\left(\begin{bmatrix} x_1-5.1142 \\ x_2-4.9942 \end{bmatrix}\right)\right]}$$

$$= \frac{1}{0.90733} e^{\left[-\frac{1}{2}*[x_1-0.175 \;\; x_2+0.0364]*\begin{bmatrix} 1.27978 & -0.18997 \\ -0.18997 & 0.97732 \end{bmatrix}*\left(\begin{bmatrix} x_1-0.175 \\ x_2+0.0364 \end{bmatrix}\right)\right]}$$

$$\ln(0.89799) = \ln\left(\frac{e^{\left[-\frac{1}{2}*[x_1-0.175 \;\; x_2+0.0364]*\begin{bmatrix} 1.27978 & -0.18997 \\ -0.18997 & 0.97732 \end{bmatrix}*\left(\begin{bmatrix} x_1-0.175 \\ x_2+0.0364 \end{bmatrix}\right)\right]}}{e^{\left[-\frac{1}{2}*[x_1-5.1142 \;\; x_2-4.9942]*\begin{bmatrix} 1.15747 & -0.0652 \\ -0.0652 & 0.84991 \end{bmatrix}*\left(\begin{bmatrix} x_1-5.1142 \\ x_2-4.9942 \end{bmatrix}\right)\right]}}\right)$$

$$-.1076 = -\frac{1}{2} * [x_1 - 0.175 \;\; x_2 + 0.0364] * \begin{bmatrix} 1.27978 & -0.18997 \\ -0.18997 & 0.97732 \end{bmatrix} * \left(\begin{bmatrix} x_1 - 0.175 \\ x_2 + 0.0364 \end{bmatrix}\right) + \frac{1}{2}$$
$$* [x_1 - 5.1142 \;\; x_2 - 4.9942] * \begin{bmatrix} 1.15747 & -0.0652 \\ -0.0652 & 0.84991 \end{bmatrix} * \left(\begin{bmatrix} x_1-5.1142 \\ x_2 - 4.9942 \end{bmatrix}\right))$$

$$-.21519 = [x_1 - 5.1142 \;\; x_2 - 4.9942] * \begin{bmatrix} 1.15747 & -0.0652 \\ -0.0652 & 0.84991 \end{bmatrix} * \left(\begin{bmatrix} x_1-5.1142 \\ x_2 - 4.9942 \end{bmatrix}\right))$$
$$-[x_1 - 0.175 \;\; x_2 + 0.0364] * \begin{bmatrix} 1.27978 & -0.18997 \\ -0.18997 & 0.97732 \end{bmatrix} * \left(\begin{bmatrix} x_1 - 0.175 \\ x_2 + 0.0364 \end{bmatrix}\right)$$

$$\mathbf{-0.2152 = -0.1223x_1^2 + 0.2495x_1x_2 - 10.7537x_1 + 48.104 \;\; - 0.127x_2^2 - 7.81769x_2}$$

Error Rate: 0%

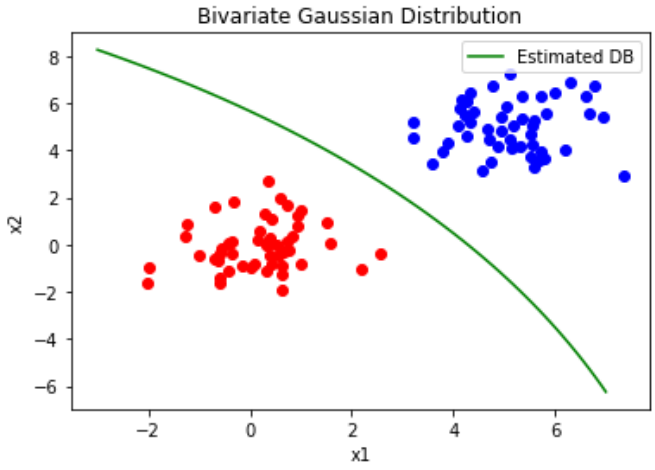

Fig. 9. A plot of 50 gaussian points with an estimated decision boundary using MLE.

c. $$0 = \frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}} e^{\left[-\frac{1}{2}*(x-[0\;0])^t*\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}*(x-[0\;0])\right]} - \frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}} e^{\left[-\frac{1}{2}*(x-[5\;5])^t*\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1}*(x-[5\;5])\right]}$$

$$0 = e^{\left[-\frac{1}{2}*[x_1 \;\; x_2]*\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}*\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)\right]} - e^{\left[-\frac{1}{2}*[x_1-5 \;\; x_2-5]*\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}*\left(\begin{bmatrix} x_1-5 \\ x_2-5 \end{bmatrix}\right)\right]}$$

$$0 = e^{\left[-\frac{1}{2}*[x_1 \;\; x_2]*\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right]} - e^{\left[-\frac{1}{2}*[x_1-5 \;\; x_2-5]*\begin{bmatrix} x_1-5 \\ x_2-5 \end{bmatrix}\right]}$$

$$0 = e^{\left[-\frac{1}{2}*(x_1^2+x_2^2)\right]} - e^{\left[-\frac{1}{2}*((x_1-5)^2 \;+\; (x_2-5)^2\right]}$$

$\ln\left(e^{\left[-\frac{1}{2}*((x_1-5)^2\ +\ (x_2-5)^2\right]}\right) = \ln\left(e^{\left[-\frac{1}{2}*(x_1^2+\ x_2^2)\right)}\right)$

$((x_1-5)^2\ +\ (x_2-5)^2 = (x_1^2+\ x_2^2)$

$x_1^2 - 10x_1 + 25 + x_2^2 - 10x_2 + 25 = (x_1^2+\ x_2^2)$

$-10x_1 - 10x_2 + 50 = 0$

$\boldsymbol{x_1 + x_2 = 5}$

Error rate: 0%

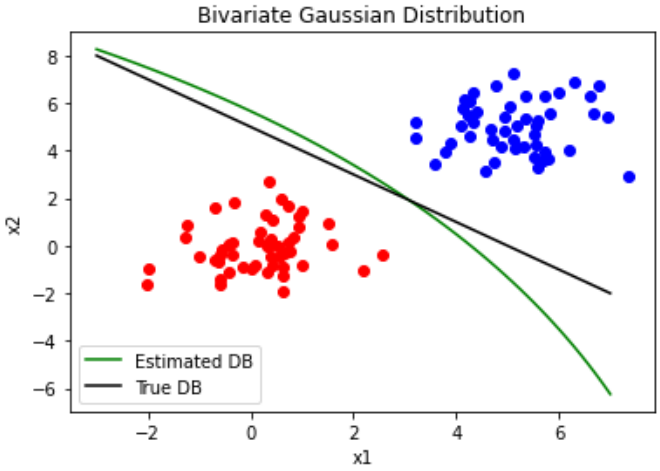

Fig. 10. A plot of 50 gaussian points with a true and estimated decision boundary.

d. As the number of points increases, the estimated parameters approach their true values. In the case of this problem, the means approach [0 ,0] and [5, 5], while the covariance matrix for both classes approach the identity matrix [[1, 0], [0, 1]]. The estimated decision boundary becomes more like the true decision boundary as n increases, as well (by approaching a straight line). In terms of the error rate, as n increases, the error rate does as well. As more samples are taken, there are more opportunities for a value to fall within the wrong region, which can be seen in the next 2 figures, compared to the figure where n=50, and the error rate was 0.

Error rate for estimated bound: 0%

Error rate for true bound: 0%

New Values:

$\mu_{MLE_1} = [-.0367 \quad -0.0088]$

$\mu_{MLE_2} = [5.015 \quad 5.0775]$

$\Sigma_{MLE_1} = \begin{bmatrix} 0.9361 & 0.078 \\ 0.078 & 1.02982 \end{bmatrix}$

$\Sigma_{MLE_2} = \begin{bmatrix} 0.9577 & 0.0375 \\ 0.0375 & 0.9562 \end{bmatrix}$

Estimated Decision Boundary:

$\mathbf{0.0217x_1^2\text{-}10.057x_1\text{-}0.0809x_1x_2\text{-}10.110x_2\text{-}0.0737x_2^2 = \text{-}50.6018}$

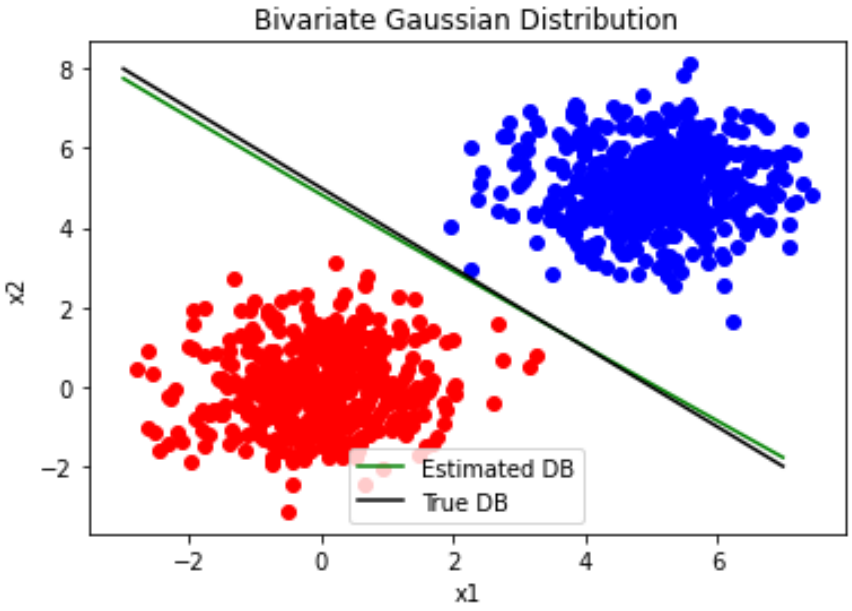

Fig. 11. A plot of 500 gaussian points with a true and estimated decision boundary.

Error rate for estimated bound: .017%
Error rate for true bound: 0.017%

Values:

$$\mu_{MLE_1} = [-0.0048 \quad -0.01]$$

$$\mu_{MLE_2} = [4.9985 \quad 5.0125]$$

$$\Sigma_{MLE_1} = \begin{bmatrix} 1.0028 & -0.0015 \\ -0.0015 & 1.0001 \end{bmatrix}$$

$$\Sigma_{MLE_2} = \begin{bmatrix} 1.0070 & -.0005 \\ -.0005 & .99711 \end{bmatrix}$$

Estimated Decision Boundary:

$$\mathbf{0.00417}x_1^2 \mathbf{- 10.071}x_1 \mathbf{+0.00197}x_1x_2 \mathbf{-10.011}x_2 \mathbf{-0.003}x_2^2 = \mathbf{-50.1876}$$

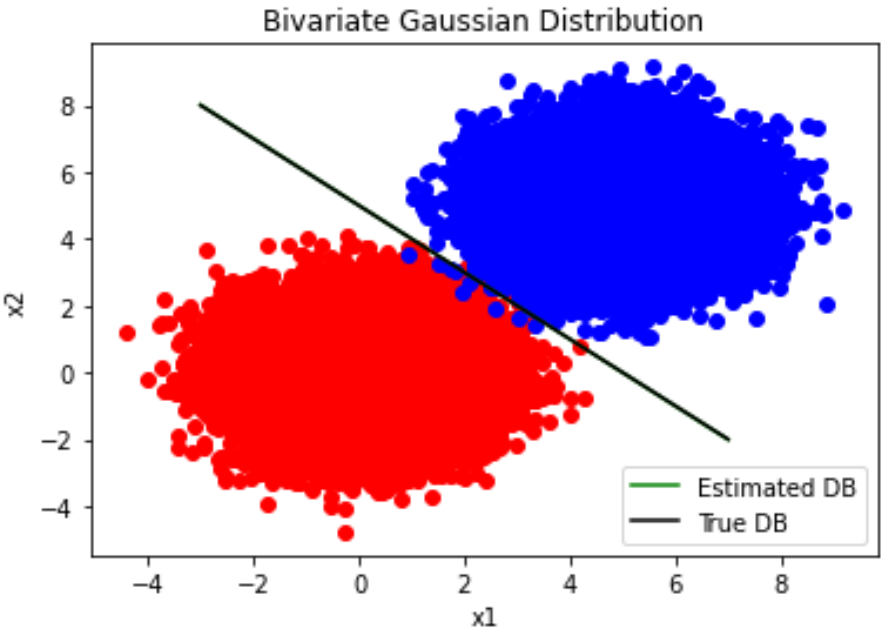

Fig. 12. A plot of 50000 gaussian points with a true and estimated decision boundary.

8. Given the iris set of data…
   a. $\mu 1 = [5.028 \quad 3.48, \quad 1.46 \quad 0.248]$
   $\mu 2 = [6.012 \quad 2.776 \quad 4.312 \quad 1.344]$
   $\mu 3 = [6.576 \quad 2.928 \quad 5.64 \quad 2.044]$
   $$\sigma 1 = \begin{bmatrix} .1604 & .1181 & .0241 & .0194 \\ .1181 & .1358 & .0063 & .0223 \\ .0241 & .0063 & .0392 & .0066 \\ .0194 & .0223 & .0066 & 0.0109 \end{bmatrix}$$
   $$\sigma 2 = \begin{bmatrix} .3003 & .1095 & .1865 & .052 \\ .1095 & .1244 & .0886 & .0465 \\ .1865 & .0886 & .1969 & .0640 \\ .052 & .0465 & .0640 & .0426 \end{bmatrix}$$
   $$\sigma 3 = \begin{bmatrix} .5244 & .1215 & .4323 & .0619 \\ .1215 & .1304 & .0988 & .0604 \\ .4323 & .0988 & .4175 & .0673 \\ .0619 & .0604 & .0673 & .0651 \end{bmatrix}$$
   b. The classifier program can be seen in the appendix.
   c. The confusion matrix

	1	**2**	**3**
1	25	0	0
2	0	24	1
3	0	1	24

   The empirical error rate on this test set: **2.67%**

Appendix

- Q2

```
import numpy as np
import matplotlib.pyplot as plt

def rand_multinorm(mean, cov, aw):
    """Plots the normal bivariate distrubution and the whitened form"""
    vals = np.random.multivariate_normal(mean, cov, 10000)
    x1 = [v[0] for v in vals]
    x2 = [v[1] for v in vals]

    fig, ax = plt.subplots()
    ax.scatter(x1, x2)
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title("Bivariate Gaussian Distribution")

    # Gets the whitened points
    w1, w2 = [], []
    for i in range(len(x1)):
        white = np.dot(aw, [[x1[i]], [x2[i]]])
        w1.append(white[0])
        w2.append(white[1])

    fig, ax1 = plt.subplots()
    ax1.scatter(w1, w2)
    ax1.set_xlabel("x1")
    ax1.set_ylabel("x2")
    ax1.set_title("Whitened Bivariate Gaussian Distribution")


def whiten(mean, cov, mean_t):
    vals, vec = np.linalg.eig(cov)

    # Creates matrix of values
    lam = []
    for i in range(len(vals)):
        row = [0] * len(vals)
        row[i] = vals[i]
        lam.append(row)

    lam = np.linalg.inv(lam)
    lam = np.sqrt(lam)
    white = np.dot(vec, lam)
    white_t = np.transpose(white)

    print("Whitening:\n", white)
    print()
    print("Whitening Transposed:\n", white_t)
    print()
    mean_w = np.dot(white_t, mean)
    print("Whitened Mean:\n", mean_w)
    print()
    dot = np.dot(white_t, cov)
    dot = np.dot(dot, white)
    print("Whitened Covariance:\n", dot)

    rand_multinorm(mean_t, cov, white_t)


def main():
    mean = [[0], [0]]
    mean_t = [0, 0]
    cov = [[20, 10], [10, 30]]

    whiten(mean, cov, mean_t)

if __name__ == "__main__":
    main()
```

- Q3

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sp


def main():
    mu1, cov1 = [-1, -1], [[1, 0], [0, 1]]
    mu2, cov2 = [1, 1], [[1, 0], [0, 1]]
    mu3, cov3 = [.5, .5], [[1, 0], [0, 1]]
    mu4, cov4 = [-.5, -.5], [[1, 0], [0, 1]]

    fig, ax = plt.subplots()
    ax.scatter(mu1[0], mu1[1], label="\u03C91")
    ax.scatter(mu2[0], mu2[1], label="\u03C92")
    ax.scatter(mu3[0], mu3[1], label="\u03C93,1")
    ax.scatter(mu4[0], mu4[1], label="\u03C93,2")
    ax.scatter(.1, .1, label="x: [.1, .1]")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title("Bivariate Gaussian Distribution")
    ax.legend()

    # Assigns a point
    prob = []
    v = [.1, .1]
    prob.append(sp.multivariate_normal.pdf(v, mu1, cov1))
    prob.append(sp.multivariate_normal.pdf(v, mu2, cov2))
    prob.append(.5 * sp.multivariate_normal.pdf(v, mu3, cov3) + \
                .5 * sp.multivariate_normal.pdf(v, mu4, cov4))
    print("Densities:", prob)
    print("Max:", max(prob), "(\u03C9" + str(prob.index(max(prob)) + 1) + ")")


if __name__ == "__main__":
    main()
```

- Q7

```
import numpy as np
import matplotlib.pyplot as plt
import pprint as pp
import math


def chernoff():
    B = [i / 100 for i in range(0, 101, 1)]
    y = []
    for b in B:
        y_B = math.e ** (-16 * b + 16 * b ** 2)
        y.append(y_B)

    bs = y.index(min(y)) / 100
    print("Error:", min(y), "B*:", bs)
    plt.xlabel("B")
    plt.ylabel("Error Bound")
    plt.title("Chernoff Bound")
    plt.plot(B, y)
    s = "B*=" + str(bs) + "\nCB=" + str(round(min(y), 4))
    plt.plot(bs, min(y), "*k", label=s)
    plt.legend()


def get_points(mean, cov, n):
    vals1 = np.random.multivariate_normal(mean, cov, n)
    x1 = [v[0] for v in vals1]
    x2 = [v[1] for v in vals1]
    return x1, x2
```

```
def error_rate(x11, x12, x21, x22, pr=False):
    # Creates the confusion matrix
    co1, inc1, co2, inc2 = 0, 0, 0, 0
    for i in range(len(x11)):
        v = x11[i] + x12[i]
        if v > 4:  # It's class1
            co1 += 1
        else:
            inc1 += 1
    for i in range(len(x22)):
        v = x21[i] + x22[i]
        if v <= 4:  # Its class2
            co2 += 1
        else:
            inc2 += 1
    if pr:
        print(co1, inc1)
        print(co2, inc2)

    # Calculates error rate
    return (inc1 + inc2) / (co1 + inc1 + inc2 + co2)


def main():
    chernoff()

    x11, x12 = get_points([4, 4], [[1, 0], [0, 1]], 25)
    x21, x22 = get_points([0, 0], [[1, 0], [0, 1]], 25)

    fig, ax = plt.subplots()
    ax.scatter(x11, x12, label="\u03C91")
    ax.scatter(x21, x22, label="\u03C92")

    x = np.linspace(-3, 6, 90)
    db = [4 - p for p in x]

    ax.plot(x, db, "g", label="DB")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.legend()
    ax.set_title("Bivariate Gaussian Distribution")

    eer = error_rate(x11, x12, x21, x22, True)
    print("\nError:", eer, "(n=25)")

    # Checks the empirical error rate
    for j in range(3):
        error_n, n = [], []
        for i in range(100, 1100, 100):
            x11, x12 = get_points([4, 4], [[1, 0], [0, 1]], i)
            x21, x22 = get_points([0, 0], [[1, 0], [0, 1]], i)
            eer = error_rate(x11, x12, x21, x22)
            error_n.append(eer)
            n.append(i)
            print("Error:", eer, "(n=" + str(i) + ")")

        # plots the error rate as EER(n)
        fig1, ax1 = plt.subplots()
        ax1.plot(n, error_n)
        ax1.set_xlabel("n")
        ax1.set_ylabel("Error Rate")
        ax1.set_title("Empirical Error Rate")


if __name__ == "__main__":
    main()
```

- Q6

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


def gaussian(data):
    mu = sum(data) / len(data)
    cov = np.cov(data)
    xs = np.linspace(0, 20, 40)
    ys = stats.norm.pdf(xs, mu, cov ** .5)
    plt.plot(xs, ys, color="r")


def main():
    fp = open("data6.txt", "r")
    data = [float(line.strip()) for line in fp]

    # Plots histogram
    ran = 2 * m.ceil(max(data)) - m.floor(min(data))
    plt.hist(data, ran, density=True, color="c")
    plt.xlabel("x")
    plt.ylabel("P(x)")
    plt.title("Rayleigh Distribution")

    theta = len(data) / sum([x * x for x in data])
    xs = np.linspace(0, 20, 40)
    ys = []
    for x in xs:
        y = 2 * theta * x * m.e ** (-theta * x * x)
        ys.append(y)
    plt.plot(xs, ys, color="k")

    gaussian(data)


if __name__ == "__main__":
    main()
```

- Q7 – Generating Patterns

```
import numpy as np
n = int(input("How many samples:"))
mu1, cov1 = [0,0], [[1, 0], [0, 1]]
mu2, cov2 = [5,5], [[1, 0], [0, 1]]
w1 = np.random.multivariate_normal(mu1, cov1, n)
w2 = np.random.multivariate_normal(mu2, cov2, n)
fp1 = open("gen1_" + str(n) + ".txt", "w")
fp2 = open("gen2_" + str(n) + ".txt", "w")
for x1, x2 in w1:
    fp1.write(str(x1)+","+str(x2)+"\n")
for x1, x2 in w2:
    fp2.write(str(x1)+","+str(x2)+"\n")
fp1.close()
fp2.close()
```

- Q7 – Solve

```
import numpy as np
import matplotlib.pyplot as plt
import math as m
from scipy import stats
from sympy import *


def mle_mean(data):
    s = np.sum(data, axis=0)
    s = [s[0] / len(data), s[1] / len(data)]
    return s


def mle_cov(data):
    mu = mle_mean(data)
    pro = []
    for x in data:
        diff = np.subtract(x, mu)  # x-u
        p = np.outer(diff, diff)  # (x-u)(x-u)^t
        pro.append(p)
    sig = sum(pro)  # sum (x-u)(x-u)^t
    return sig / len(pro)


def main():
    mu1, cov1 = [0, 0], [[1, 0], [0, 1]]
    mu2, cov2 = [5, 5], [[1, 0], [0, 1]]

    # opens the files to get the data
    ns = ["50", "500", "50000"]
    for n in ns:
        fp1 = open("gen1_" + n + ".txt")
        fp2 = open("gen2_" + n + ".txt")
        w1, w2 = [], []
        for line in fp1:
            line = [float(x) for x in line.strip().split(",")]
            w1.append(line)
        for line in fp2:
            line = [float(x) for x in line.strip().split(",")]
            w2.append(line)
        w1, w2 = np.array(w1), np.array(w2)

        mu = mle_mean(w1)
        cov = mle_cov(w1)
        print("Mean:", np.array([round(mu[0], 4), round(mu[1], 4)]))
        print("Covariance:\n", cov)

        mu = mle_mean(w2)
        cov = mle_cov(w2)
        print("Mean:", np.array([round(mu[0], 4), round(mu[1], 4)]))
        print("Covariance:\n", cov)

        # Gets the class points
        w11 = [x[0] for x in w1]
        w12 = [x[1] for x in w1]
        w21 = [x[0] for x in w2]
        w22 = [x[1] for x in w2]
        xt = np.linspace(-3, 7, 100)
        yt = [5 - x for x in xt]
        yb = []
        for x in xt:
            x2 = symbols('x2')
            if n == "50":
                y = solve(.2152 - .1223 * x ** 2 + .2495 * x * x2 - 10.7537 * x + \
                          48.104 - .127 * x2 ** 2 - 7.81769 * x2)
                yb.append(y[1])
            elif n == "500":
                y = solve(50.6018 + 0.0217 * x ** 2 - 0.0809 * x * x2 - 10.057 * x - \
                          0.0737 * x2 ** 2 - 10.110 * x2)
```

```
                yb.append(y[1])
            elif n == "50000":
                y = solve(50.18766 + 0.00417 * x ** 2 + 0.00197 * x * x2 - 10.071 * x \
                          - 0.00301 * x2 ** 2 - 10.011 * x2)
                yb.append(y[1])

        # Plots the estimated boundary
        fig, ax = plt.subplots()
        ax.scatter(w11, w12, color="r")
        ax.scatter(w21, w22, color="b")
        ax.plot(xt, yb, color="g", label="Estimated DB")
        ax.set_xlabel("x1")
        ax.set_ylabel("x2")
        ax.set_title("Bivariate Gaussian Distribution")
        ax.plot(xt, yt, color="k", label="True DB")
        ax.legend()

        # Finds the error rate
        err_t, err_b = 0, 0
        for x in w1:
            x, x2 = x[0], x[1]
            if n == "50":
                if (-.1223 * x ** 2 + .2495 * x * x2 - 10.7537 * x - \
                    .127 * x2 ** 2 - 7.81769 * x2) <= -48.104 - .2152:
                    err_b += 1
            elif n == "500":
                if (0.0217 * x ** 2 - 0.0809 * x * x2 - 10.057 * x - \
                    0.0737 * x2 ** 2 - 10.110 * x2) <= -50.6018:
                    err_b += 1
            elif n == "50000":
                if (0.00417 * x ** 2 + 0.00197 * x * x2 - 10.071 * x - \
                    0.00301 * x2 ** 2 - 10.011 * x2) <= -50.1876:
                    err_b += 1
            if x + x2 >= 5:
                err_t += 1

        for x in w2:
            x, x2 = x[0], x[1]
            if n == "50":
                if (-.1223 * x ** 2 + .2495 * x * x2 - 10.7537 * x - \
                    .127 * x2 ** 2 - 7.81769 * x2) > -48.104 - .2152:
                    err_b += 1
            elif n == "500":
                if (0.0217 * x ** 2 - 0.0809 * x * x2 - 10.057 * x - \
                    0.0737 * x2 ** 2 - 10.110 * x2) > -50.6018:
                    err_b += 1
            elif n == "50000":
                if (0.00417 * x ** 2 + 0.00197 * x * x2 - 10.071 * x - \
                    0.00301 * x2 ** 2 - 10.011 * x2) > -50.1876:
                    err_b += 1
            if x + x2 < 5:
                err_t += 1

        print("Error True:", err_t / (2 * int(n)))
        print("Error Estimate:", err_b / (2 * int(n)), "\n")


if __name__ == "__main__":
    main()
```

- Q8

```
import numpy as np
import scipy.stats as sp
import pandas as pd


def confusion(predicted, actual):
    actual = pd.Series(actual, name="actual")
    predicted = pd.Series(predicted, name="predicted")
    confuse = pd.crosstab(actual, predicted)
    print(confuse)

    # finds error rate
    err = (confuse[1][2] + confuse[1][3] + confuse[2][1] + confuse[2][3] + \
           confuse[3][1] + confuse[3][2]) / len(actual)
    print("Error Rate:", round(err, 4))


def classify(pattern, mus, covs):
    max_p, max_i = -1, -1
    pattern = np.array(pattern)

    for i in range(len(covs)):
        # Calculates the condiitonal probability (equal priors and 0-1 loss)
        mu, cov = mus[i], covs[i]
        density = sp.multivariate_normal.pdf(pattern, mu, cov)
        if density > max_p:
            max_p = density
            max_i = i
    print("Class:", max_i + 1)
    return max_i + 1


def find_mean(iris):
    pat = [iris[1][:25], iris[2][:25], iris[3][:25]]
    means = []
    for i, p in enumerate(pat):
        m = np.mean(p, axis=0)
        means.append(m)
        print("Means:", m)
    return means[0], means[1], means[2]


def find_var(iris):
    pat = [iris[1][:25], iris[2][:25], iris[3][:25]]
    for i in range(len(pat)):
        pat[i] = np.array(pat[i]).T

    covs = []
    for i, p in enumerate(pat):
        cov = np.cov(p)
        covs.append(cov)
        print("Covariance:\n", cov, "\n")
    return covs[0], covs[1], covs[2]


def main():
    fp = open("iris.txt", "r")
    lines = []
    for line in fp:
        line = line.strip().split()
        for i in range(len(line)):
            if i == 4:
                line[i] = int(line[i])
            else:
                line[i] = float(line[i])
        if len(line) == 5:
            lines.append(line)
    line = line[:-1]
```

```
    # Turns into a dict
    iris = {}
    for el in lines:
        if el[4] not in iris:
            iris[el[4]] = [el[:-1]]
        else:
            iris[el[4]].append(el[:-1])

    mus = find_mean(iris)
    covs = find_var(iris)

    # Classifies the remaining patterns
    pat_test = [iris[1][25:], iris[2][25:], iris[3][25:]]
    actual = [1] * 25 + [2] * 25 + [3] * 25
    predicted = []
    for i, patts in enumerate(pat_test):
        for p in patts:
            res = classify(p, mus, covs)
            predicted.append(res)
    confusion(predicted, actual)


if __name__ == "__main__":
    main()
```