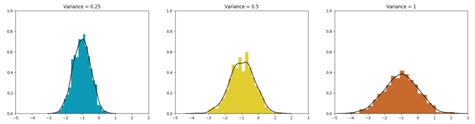
Amin Fadaee

Pattern Recognition HW1

Python 3.5, Libraries: Numpy, Matplotlib, Seaborn, Pandas, Scipy, Math, Random, Documentation: HTML, Bootstrap 4

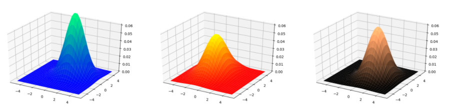
Problem 1

As can be seen by the charts, all of the samples are distributed around μ as expected and the more we increase the variance, the wider the plot will become. Also we can notice that each plot has the standard deviation roughly equal to the square root of the corresponding variance. As we increase the number of samples, the plots will resemble their distribution more and more. *(Here we used 1000 samples).*



Problem 2

Here is the chart illustrating each of the distributions:



Problem 3

a. $E(X) = 1.0$

b. $E(X^2) = 1.5$

c. $var(X) = E(X^2) - E(X)^2 = 0.5$

d. $E(g(x)) = 5.0$

Problem 4

a.

$$\rho(X_1+X_2, X_2+X_3) = \frac{cov(X_1+X_2, X_2+X_3)}{\sigma(X_1+X_2)\times\sigma(X_2+X_3)}$$

$$= \frac{E((X_1+X_2)(X_2+X_3)) - E(X_1+X_2)E(X_2+X_3)}{\sigma(X_1+X_2)\times\sigma(X_2+X_3)}$$

$$= \frac{E(X_1X_2)+E(X_1X_3)+E(X_2X_2)+E(X_2X_3)}{\sigma(X_1+X_2)\times\sigma(X_2+X_3)}$$

$$= \frac{E(X_1)E(X_2)+E(X_1)E(X_3)+1-E(X_2)^2+E(X_2)E(X_3)}{\sigma(X_1+X_2)\times\sigma(X_2+X_3)}$$

$$= \frac{0+0+1+0}{\sigma(X_1+X_2)\times\sigma(X_2+X_3)}$$

$$= \frac{1}{sqrt(var(X_1)+var(X_2)+2cov(X_1,X_2))\times sqrt(var(X_2)+var(X_3)+2cov(X_2,X_3))}$$

$$= \frac{1}{sqrt(1+1+0) * sqrt(1+1+0)}$$

$$= \frac{1}{2}$$

b.

$$\rho(X_1+X_2, X_3+X_4) = \frac{cov(X_1+X_2, X_3+X_4)}{\sigma(X_1+X_2)\times\sigma(X_3+X_4)}$$

$$= \frac{E((X_1+X_2)(X_3+X_4)) - E(X_1+X_2)E(X_3+X_4)}{\sigma(X_1+X_2)\times\sigma(X_3+X_4)}$$

$$= \frac{E(X_1X_3)+E(X_1X_4)+E(X_2X_3)+E(X_2X_4)}{\sigma(X_1+X_2)\times\sigma(X_3+X_4)}$$

$$= \frac{E(X_1)E(X_3)+E(X_1)E(X_4)+E(X_2)E(X_3)+E(X_2)E(X_4)}{\sigma(X_1+X_2)\times\sigma(X_3+X_4)}$$

$$= \frac{0+0+0+0}{\sigma(X_1+X_2)\times\sigma(X_3+X_4)}$$

$$= 0$$

Problem 5

$$A = \Phi\Lambda\Phi^{-1}$$

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} \varphi_1 & \varphi_2 \\ \varphi_3 & \varphi_4 \end{bmatrix} \times \begin{bmatrix} \lambda & 0 \\ 0 & \mu \end{bmatrix} \times \frac{1}{\varphi_1\varphi_4 - \varphi_2\varphi_3}\begin{bmatrix} \varphi_4 & -\varphi_2 \\ -\varphi_3 & \varphi_1 \end{bmatrix}$$

$$A_1 = \frac{\varphi_1\varphi_4\lambda - \varphi_2\varphi_3\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3}, A_2 = \frac{-\varphi_1\varphi_2\lambda + \varphi_1\varphi_2\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3}$$

$$A_3 = \frac{\varphi_3\varphi_4\lambda - \varphi_3\varphi_4\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3}, A_4 = \frac{-\varphi_3\varphi_2\lambda + \varphi_1\varphi_4\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3}$$

$$tr(A) = A_1 + A_3 = \frac{\lambda(\varphi_1\varphi_4 - \varphi_2\varphi_3) + \mu(\varphi_1\varphi_4 - \varphi_2\varphi_3)}{\varphi_1\varphi_4 - \varphi_2\varphi_3} = \lambda + \mu$$

$det(A) = A_1A_4 - A_2A_3 =$

$$\frac{\varphi_1\varphi_4\lambda - \varphi_2\varphi_3\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3} \times \frac{-\varphi_3\varphi_2\lambda + \varphi_1\varphi_4\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3} - \frac{-\varphi_1\varphi_2\lambda + \varphi_1\varphi_2\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3} \times \frac{\varphi_3\varphi_4\lambda - \varphi_3\varphi_4\mu}{\varphi_1\varphi_4 - \varphi_2\varphi_3} =$$

$$\frac{(\lambda\varphi_1\varphi_4 - \mu\varphi_2\varphi_3)(\mu\varphi_1\varphi_4 - \lambda\varphi_2\varphi_3)}{(\varphi_1\varphi_4 - \varphi_2\varphi_3)^2} - \frac{(\mu\varphi_1\varphi_2 - \lambda\varphi_1\varphi_2)(\lambda\varphi_3\varphi_4 - \mu\varphi_3\varphi_4)}{(\varphi_1\varphi_4 - \varphi_2\varphi_3)^2}$$

$$\frac{(\lambda\mu\varphi_2{}^2\varphi_3{}^2 - 2\lambda\mu\varphi_1\varphi_2\varphi_4\varphi_3 + \lambda\mu\varphi_1{}^2\varphi_4{}^2)}{(\varphi_1\varphi_4 - \varphi_2\varphi_3)^2} =$$

$$\frac{\lambda\mu(\varphi_2{}^2\varphi_3{}^2 - 2\varphi_1\varphi_2\varphi_4\varphi_3 + \varphi_1{}^2\varphi_4{}^2)}{(\varphi_1\varphi_4 - \varphi_2\varphi_3)^2} =$$

$$\frac{\lambda\mu(\varphi_1\varphi_4 - \varphi_2\varphi_3)^2}{(\varphi_1\varphi_4 - \varphi_2\varphi_3)^2} = \lambda\mu$$

Problem 6

a.

Finding Eigenvalues:

$|A - \lambda I| = 0$

$$\begin{bmatrix} 4-\lambda & 0 & 0 \\ 0 & 2-\lambda & 2 \\ 0 & 9 & -5-\lambda \end{bmatrix} = 0$$

$(4-\lambda)((2-\lambda)(-5-\lambda) - 18) - 0 + 0 = 0$

$(4-\lambda)(\lambda^2 + 3\lambda - 28) = 0$

$\lambda = 4, 4, -7$

Finding Eigenvectors ($\lambda = -7$):

$$\begin{bmatrix} 11 & 0 & 0 \\ 0 & 9 & 2 \\ 0 & 9 & 2 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$$

$11x = 0$

$9y + 2z = 0 \rightarrow z = -\frac{9}{2}y$

$$v = \begin{bmatrix} 0 \\ y \\ -92y \end{bmatrix} = y\begin{bmatrix} 0 \\ 1 \\ -4.5 \end{bmatrix} = c\begin{bmatrix} 0 \\ 1 \\ -4.5 \end{bmatrix}$$

finding c so that the vector is normalized

$sqrt(c^2 + (-4.5c)^2) = 1 \rightarrow c \approx 0.21693$

$$v = \begin{bmatrix} 0 \\ 0.21693 \\ -0.97618 \end{bmatrix}$$

Finding Eigenvectors ($\lambda = 4$):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 2 \\ 0 & 9 & -9 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$$

$$y = z \rightarrow v = \begin{bmatrix} x \\ y \\ y \end{bmatrix} = x\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + y\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$v_1 = c_1\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow[c_1 = 1]{} v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$v_2 = c_2\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \xrightarrow[c_2 = 0.7071]{} v_2 = \begin{bmatrix} 0 \\ 0.7071 \\ 0.7071 \end{bmatrix}\$$$

Python eig Function Solution:

```
Eigenvalues:
 [-7.  4.  4.]
Eigenvectors:
 [[ 0.          0.          1.        ]
 [ 0.21693046 -0.70710678  0.        ]
 [-0.97618706 -0.70710678  0.        ]]
```

As can be seen the results are almost identical, except for the second vector which is the negative of what we obtained but they are both correct.

b.

To compute the matrix A given the eigens we can use the following formula: $A = \Phi\Lambda\Phi^{-1}$ and we have: $\Lambda = \begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}, \Phi = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ So: $\Phi^{-1} = \frac{1}{1}\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} A = \begin{bmatrix} 2 & 3 \\ 0 & 5 \end{bmatrix}$$

This result can be tested using the code we already wrote:

```
Eigenvalues:
 [ 2.  5.]
Eigenvectors:
 [[ 1.          0.70710678]
 [ 0.          0.70710678]]
```

As can be seen we obtained the premises of the problem (with a slight difference for the normalized eigenvector of $[1, 1]^T$.)

Problem 7 - Q1

a.

We consider two cases:

Case 1: $P(w_2 \mid x) \geq P(w_1 \mid x)$:

$$P(error \mid x) = P(w_1 \mid x) = 1 - P(w_2 \mid x) \rightarrow P(error) < \frac{1}{2} and 2P(w_2 \mid x) > 1$$

Therefore, $2P(w_2 \mid x)P(w_1 \mid x)$ is always greater than $P(w_1 \mid x)$, so:

$$\int 2P(w_2 \mid x)P(w_1 \mid x)p(x)dx \geq \int P(error \mid x)p(x)dx$$

Case 1: $P(w_1 \mid x) \geq P(w_2 \mid x)$:

Following the logic of the previous case we can obtain the same results

b.

For an $\alpha < 2$ we can't state the previous assertion because it **does not** hold for every x! e.g. if $P(w_1 \mid x) = P(w_2 \mid x) = \frac{1}{2}$ and $\alpha = 1.5$ then in case 1 of the previous problem:

$$\int 1.5P(w_2 \mid x)P(w_1 \mid x)p(x)dx = 1.5 \times \frac{1}{2}P(w_1 \mid x)p(x)dx \ngeq \int P(error \mid x)p(x)dx$$

c.

$P(w_1 \mid x)$ and $P(w_2 \mid x)$ are both smaller than 1, so it is obvious that:

$$\int P(w_2 \mid x)P(w_1 \mid x)p(x)dx < \int P(error \mid x)p(x)dx$$

and this will be true for all values of x.

d.

Problem a. for $\beta = \alpha = 2$ is a proof for this problem.

Problem 7 - Q2

a.

As stated in Eq.23 of the book:

$$R_{mm} = \lambda_{22} + (\lambda_{12} - \lambda_{22})\int_{R_1} p(x \mid w_2)dx = \lambda_{11} + (\lambda_{21} - \lambda_{11})\int_{R_2} p(x \mid w_1)dx$$

So it is really easy to obtain a simplified result using the zero-one loss function properties, more specifically:

- $\lambda_{ii} = 0$
- $\lambda_{ij} = 1, (i \neq j)$

So the simplified form would be:

$$R_{mm} = 0 + (1-0)\int_{R_1} p(x \mid w_2)dx = 0 + (1-0)\int_{R_2} p(x \mid w_1)dx$$

Therefore we would have:

$$\int_{R_1} p(x \mid w_2)dx = \int_{R_2} p(x \mid w_1)dx$$

Programming Assignment 1

The codes can be found in `ComputerProject1.py` by these names:

- `generate_random_sample(mu, cov, n)` :
  This function uses `multivariate_normal` function in `numpy` library and generated random samples drawn from a normal distribution specified by $N \sim (\mu, \Sigma)$
- `g(x, mu, cov, P)` :
  This function implements the following formula:
  $$g_i(x) = -\frac{1}{2}(x-\mu_i)^t\Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}ln2\pi - \frac{1}{2}ln\left|\Sigma_i\right| + lnP(\omega_i)$$
- `euclidean_distance(X, Y)` :
  This function simply computes the euclidean distance between vectors X, Y.
- `mahalanobis_distance(X, mu, cov)`
  This function implements the following formula:
  $$\sqrt{(x-\mu)^t\Sigma^{-1}(x-\mu)}$$

Programming Assignment 2,3,4

The table below summarizes the results:

Classes	Features	Accuracy	Error	Bound
w_1, w_2	x_1	%70	%30	%47.4
w_1, w_2	x_1, x_2	%55	%45	%46.0
w_1, w_2	x_1, x_2, x_3	%85	%15	%41.1
w_1, w_3	x_1	%75	%25	%39.9
w_1, w_3	x_1, x_2	%80	%20	%32.9
w_1, w_3	x_1, x_2, x_3	%85	%15	%29.9
w_2, w_3	x_1	%70	%30	%39.4
w_2, w_3	x_1, x_2	%80	%20	%30.3
w_2, w_3	x_1, x_2, x_3	%85	%15	%22.4

As can be seen in the table, as the number of features increase the empirical error may or may not change (In w_1, w_2 case it did)! However we should not forget that if we increase the number of features beyond a certain point, we always decrease the training error (which leads to overfitting and loss of generality).

Programming Assignment 2,3,4 Continued

The code for these problem can be found in `ComputerProject2345.py` .
First the data are stored in `pandas.DataFrames` for easier manipulation, after that the mean and the sample covariance matrices are obtained to represent the estimated parameters of each class distributions. In the next line the following functions are implemented and used:

- `g(x, mu, cov, P)`
- `bhattacharyya_bound(mu1, mu2, cov1, cov2, P1, P2)` :
  This function implements the following formula to obtain the Bhattacharyya error bound:
  $$P(error) \leq \sqrt{P(w_1)P(w_2)}e^{-k\left(\frac{1}{2}\right)}$$
  $$k\left(\frac{1}{2}\right) = \frac{1}{8}(\mu_2-\mu_1)^t\left[\frac{\Sigma_1+\Sigma_2}{2}\right]^{-1}(\mu_2-\mu_1) + \frac{1}{2}ln\frac{\left|\frac{\Sigma_1+\Sigma_2}{2}\right|}{\sqrt{|\Sigma_1||\Sigma_2|}}$$
- `dichotomizer(data, labels, mu1, mu2, cov1, cov2, P1, P2)` :
  This function uses the $g(x) = g_1(x) - g_2(x) > 0$ to classify x, if $x > 0$ it belongs to class 1 and otherwise it belongs to class 2. Also the accuracy, error and bound is computed in this function.

Programming Assignment 5

a. Here are the Distances:

- Distance of **(1, 2, 1)** from **mu1** is: **1.015**
- Distance of **(1, 2, 1)** from **mu2** is: **0.8581**
- Distance of **(1, 2, 1)** from **mu3** is: **2.6748**
- Distance of **(5, 3, 2)** from **mu1** is: **1.5571**
- Distance of **(5, 3, 2)** from **mu2** is: **1.7557**
- Distance of **(5, 3, 2)** from **mu3** is: **0.647**
- Distance of **(0, 0, 0)** from **mu1** is: **0.49**
- Distance of **(0, 0, 0)** from **mu2** is: **0.2684**
- Distance of **(0, 0, 0)** from **mu3** is: **2.2415**
- Distance of **(1, 0, 0)** from **mu1** is: **0.4872**
- Distance of **(1, 0, 0)** from **mu2** is: **0.4518**
- Distance of **(1, 0, 0)** from **mu3** is: **1.4623**

b. And the classification of the 4 points based on $P(w_1) = P(w_2) = P(w_3) = \frac{1}{3}$

- **(1,2,1)**: Class **2**
- **(5,3,2)**: Class **3**
- **(0,0,0)**: Class **1**
- **(1,0,0)**: Class **1**

c. And the classification of the 4 points based on $P(w_1) = 0.8, P(w_2) = P(w_3) = 0.1$

- **(1,2,1)**: Class **1**
- **(5,3,2)**: Class **1**
- **(0,0,0)**: Class **1**
- **(1,0,0)**: Class **1**

The codes to derive these results are also in `ComputerProject2345.py` and the following functions are used:

- `mahalanobis_distance(X, mu, cov)`
- `trichotomizer(data, mu1, mu2, mu3, cov1, cov2, cov3, P1, P2, P3)` :
  This function classifies x into one of 3 categories using the following formula:
  $$Class(x) = Argmax(g_i(x), i = 1, 2, 3) + 1$$

Programming Assignment 6

As can be seen in the diagrams below the distribution of averages of iids are indeed a normal distribution and the more we increase the sample size, the more we will get the perfect shape of the corresponding normal distribution (The code to obtain the diagrams is in `ComputerProject6.py`).