شناسایی آماری الگو

تمرین های سری دو

فرهاد دلیرانی
۹۶۱۳۱۱۲۵

dalirani@aut.ac.ir

dalirani.1373@gmail.com

ساختار درختی پوشه‌ی ارسال شده:

report.pdf

./computerProject:

computerProject.py

./problem1:

problem1.py

./problem2:

problem2.py

./problem3:

problem3.py

./problem4:

problem4.py

./problem5:

problem5.py

./problem6:

problem6.py

./problem7:

problem7.py

تمام‌کدها با پایتون 3.6 نوشته شده‌اند.

همچنین از پکیج‌های زیر استفاده کرده ام:

- numpy
- matplotlib

البته برای راحتی در نصب پایتون 3.6 و پکیج های مربوط به دیتاساینس که numpy و matplotlib هم جزیی از آن پکیج‌ها هستند از Anaconda 5.0.0 استفاده کرده‌ام که همه‌ی موارد گفته شده را بدون دردسر و سختی نصب می کند. تنها کافی است آن را از https://www.anaconda.com/download دانلود کنید و Installer باقی کار را انجام می‌دهد.

زبان برنامه نویسی: پایتون 3.6
پکیج‌ها: پکیج‌های گفته شده را برای راحتی در نصب با Anaconda نصب کردم.
ورژن Anaconda من: Anaconda 5.0.0 For Linux Installer که البته همین ورژن برای سایر سیستم عامل‌ها هم موجود است.
محیط برنامه نویسی: pyCharm Community Edition
سیستم عامل: Linux/Gnome Fedora – البته همان طور که خودتان اطلاع دارید سیستم‌عامل مهم نیست، برای ارایه‌ی اطلاعات بیشتر به سیستم‌عامل اشاره کردم. همین طور کدهایم را در ویندوز تست کردم و مشکلی نداشتند.

سوال ۱:

در ابتدا کدها و توضیحات 4 بخش a,b,c و d این سوال را توضیح می‌دهم و در آخر سوال، نتایج و خروجی‌های هر چهار بخش را باهم ارایه می‌کنم.

بخش a)

این بخش از سوال خواسته است p(x| w1) و p(x| w2) را رسم کنیم برای این منظور کد کامنت‌گذاری شده زیر را نوشته ام که دو تابع داده شده را رسم می‌کند:

```python
#####################################################################
# Problem 1
#####################################################################

import matplotlib.pyplot as plt
import numpy as np


# p(x|w1) and p(x|w2):
# densities
def xw1(x):
    if x >=0 and x <= 1:
        return 2*x
    else:
        return 0


def xw2(x):
    if x >=0 and x <= 1:
        return 2-2*x
    else:
        return 0

```

```python

densities = [xw1, xw2]

#####################################################################
# Problem 1- a
#####################################################################
# Generate 200 point for drawing curves between (-1,2)
x = np.linspace(-1,2,num=200)

# Plot both densities
plt.plot(x, list(map(densities[0], x)), label='P(x|w1)', linestyle='-')
plt.plot(x, list(map(densities[1], x)), label='P(x|w2)', linestyle='-')
```

بخش b)

این بخش خواسته است مرز تصمیم را برای زمانی که $P(w1) = P(w2) = 0.5$ است را پیدا کنیم. به همین خاطر ابتدا تابع زیر را نوشته ام که بر اساس رابطه‌ی زیر

```
p(x|w1)*P(w1) - p(x|w2)*P(w2) = 0
```

```
p(x|w1) = 2x in (0,1)
p(x|w2) = 2-2x in (0,1)

x0 = P(w2)/( P(w1)+P(w2)
```

مقدار X0 = b که همان مرز تصمیم است را پیدا می‌کند:

```python
# Problem 1- b
##########################################################################
# By finding x according to P(w1) and P(w2) from below equation:
# p(x|w1)*P(w1) - p(x|w2)*P(w2) = 0
# we can easily find decision boundary.
# when p(x|w1) is 2x in (0,1) and
# p(x|w2) is 2-2x in (0,1) then
# decision boundary is x = P(w2)/( P(w1)+P(w2))
def calculate_decision_boundary(pw1, pw2):
    """
    This function finds decision boundary of p(x|w1) = 2x in (0,1) and
    p(x|w2) = 2-2x in (0,1)
    :param pw1: prior  P(w1)
    :param pw2: prior  P(w2)
    :return: return a number which indicate
            decision boundary(x = P(w2)/( P(w1)+P(w2))
    """
    return pw2/(pw1 + pw2)
```

و در ادامه تابع بالا را فراخوانده‌ام و آن را رسم کرده‌ام:

```python
# Generate 200 point for drawing decision boundary (-0.5,2.5)
y = np.linspace(-0.5, 2.5, num=200)

# Plot decision boundary when P(w1) = P(w2) = 0.5
plt.plot([calculate_decision_boundary(pw1=.5, pw2=0.5)]*200, y,
         label='Decision Boundary when P(w1)={},P(w2)={}'.format(0.5, 0.5)
         , linestyle='--')
```

بخش c)

این بخش خواسته است خطای classifier را حساب کنیم. که برابر می شود با مجموع مساحت دو مثلث سمت چپ و راست مرز تصمیم ضرب در prior هایشان که برای محاسبه‌ی آن تابع زیر را بر اساس روابط زیر نوشته ام:

```
p(x|w1) = 2x in (0,1)
p(x|w2) = 2-2x in (0,1)

x0 is decision boundary
Error = (P(w1)*(x0 * (2*x0)) / 2) + (P(w2) * ((1-x0) * (2-2*x0)) / 2)
```

برای محاسبه‌ی خطا کد زیر را نوشته‌ام که شامل تابع محاسبه‌ی خطا و فراخوانی آن برای محاسبه‌ی خطا است:

```python
######################################################################
# Problem 1- c
######################################################################
def error_of_classifier(pw1, pw2, x0):
    """
        This function finds errors of classifier when
        p(x|w1) = 2x in (0,1) and
        p(x|w2) = 2-2x in (0,1)
        :param pw1: prior  P(w1)
        :param pw2: prior  P(w2)
        :param x0: x0 is decision boundary
        :return: return a number which indicate
                error of classifier
    """
    return (pw1 * (x0 * (2*x0)) / 2) + (pw2 * ((1-x0) * (2-2*x0)) / 2)


# Calculate Error of classifier when P(w1) = P(w2) = 0.5
err = error_of_classifier(pw1=0.5, pw2=0.5, x0=calculate_decision_boundary(pw1=0.5, pw2=0.5))
print("Error of classifier when P(w1)={}, P(w2)={} is {}".format(
    0.5, 0.5, err))
```

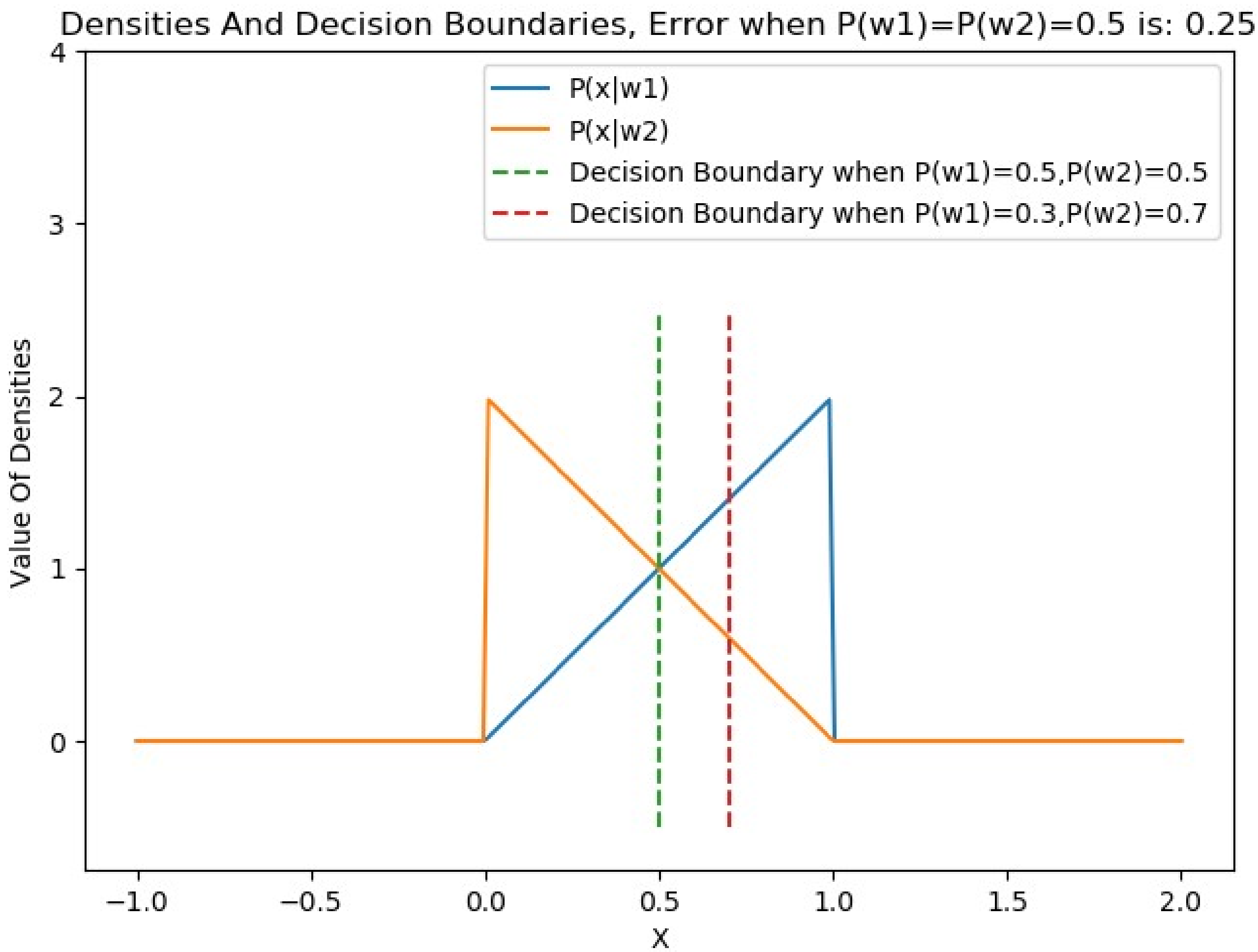
بخش d)

این بخش دقیقا مانند بخش b است با این تفاوت که $P(w1) = 0.3$ و $P(w2) = 0.7$ است و به همین خاطر تابع محاسبه‌ی مرز تصمیم بخش دو را با مقدار آرگومان‌های جدید فراخوانده‌ام و مرز تصمیم را رسم کرده ام.

```python
######################################################################
# Problem 1- d
######################################################################
# Plot decision boundary when P(w1) = 0.3, P(w2) = 0.7
plt.plot([calculate_decision_boundary(pw1=.3, pw2=0.7)]*200, y,
         label='Decision Boundary when P(w1)={},P(w2)={}'.format(0.3, 0.7)
         , linestyle='--')


plt.title("Densities And Decision Boundaries, Error when P(w1)=P(w2)=0.5 is: {}".format(err))
plt.xlabel('X')
plt.ylabel('Value Of Densities')
plt.ylim((-0.75,4))
plt.legend()
plt.show()
```

همین طور که در تصویر زیر مشاهده می شود مرز تصمیم به سمت راست حرکت کرده است تا اثر افزایش P(w2) را بر روی خطا را کم تر کند و خطای کل را مینیمم کند.

در تصویر زیر نتایج قسمت a , b , c و d را مشاهده می کنید:



خطای classifier در قسمت بالای شکل نمایش داده شده است که برابر 0.25 است.

سوال ۲:

قبل از پاسخگویی به قسمت‌های مختلف سوال تـابع discriminant_function را شرح می‌دهم که در این سوال و سری تمرین‌های یک از آن استفاده کرده‌ام.

تابع **discriminant_function**:

این تابع یک سری نقطه را می‌گیرد و همین طور مشخصات یک توزیع نرمال را می‌گیرد و بعد discriminant value را برای هر نقطه طبق رابطه‌ی زیر محاسبه می‌کند.در زیر کد کامنت گذاری شده‌ی زیر را مشاهده می‌کنید:

```python
def discriminant_function(mu, covariance, priorProbability, sampleVector):
    """
    This function gets an vector of samples and calculates
    value of discriminant function for each.

    :param mu: a vector of means (list, couple, ...)
    :param covariance: covariance matrix (list, couple, ...)
    :param sampleVector: vectors of samples (list, couple, ...)
    :param priorProbability: prior probability (list, couple, ...)
    :return: A vector of discriminant of each sample in the sample vector (np.array)
    """
    import numpy as np

    # Convert inputs to numpy.matrix()
    Mu = np.matrix(mu)
    Cov = np.matrix(covariance)
    inputVector = np.matrix(sampleVector)

    # Dimension of given normal distribution
    d = Cov.shape[0]

```

```python
    # Inverse of covariance matrix
    invCov = np.linalg.inv(Cov)

    # The part of discriminant function which is equal for different input samples
    discriminant_value_part2 = -(d/2)*np.log(2*np.pi)-\
                               (1/2)*np.log(np.linalg.det(Cov))+np.log(priorProbability)

    # Output
    outputVector = []

    # Calculating value of discriminant function for all input samples
    for i in range(inputVector.shape[0]):
        # Calculating value of discriminant function for input sample i
        sampleOutput = (-(1/2) * (inputVector[i] - Mu) * invCov * (inputVector[i] - Mu).transpose())+\
                       discriminant_value_part2
        outputVector.append(sampleOutput)

    #return an array of discriminant function
    return np.array(outputVector)
```

بخش a)

این بخش خواسته است دو تابع g1 و g2 را بنویسیم که برابراند با discriminant function توزیع نرمال p(x| w1) و p(x| w2) ، برای این منظور دو تابع زیر را نوشته ام که تابع discriminant_function بالا را با مشخصات توزیع نرمال مربوطه فرا می‌خواند. در تصویر زیر کد کامنت گذاری شده‌ی این دو تابع را مشاهده می‌کنید:

```python
######################################################################
# Problem 2 - a
######################################################################
def g1(sampleVector):
    """
    This function get a vector of sample and uses
    'discriminant_function' function to calculates
    value of discriminant function for p(x|w1) when
    P(w1) = 0.6, mean = (4, 16) and Sigma = 4*I
    :param sampleVector: a vector of sample
    :return: a vector that contains corresponding discriminant
            value for each input sample.
    """
    import numpy as np

    means = [4, 16]
    pw1 = 0.6
    sigma = np.identity(2) * 4

    return discriminant_function(mu=means, covariance=sigma,
                                 priorProbability=pw1, sampleVector=sampleVector)
```

```python
def g2(sampleVector):
    """
    This function get a vector of sample and uses
    'discriminant_function' function to calculates
    value of discriminant function for p(x|w2) when
    P(w1) = 0.4, mean = (16, 4) and Sigma = 4*I
    :param sampleVector: a vector of sample
    :return: a vector that contains corresponding discriminant
            value for each input sample.
    """
    import numpy as np

    means = [16, 4]
    pw2 = 0.4
    sigma = np.identity(2) * 4

    return discriminant_function(mu=means, covariance=sigma,
                                 priorProbability=pw2, sampleVector=sampleVector)
```

بخش b)

این بخش خواسته است مرز تصمیم را رسم کنیم که این کار را به دو روش انجام داده ام.

روش اول:

همان طور که در تصویر زیر مشاهده می‌کنید $g1(x)$ را برابر $g2(x)$ قرار داده‌ام و مرز تصمیم را به دست آورده‌ام:

$$g_1 = -\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_1| + \ln P(\omega_1)$$

$\mu_1 = \begin{bmatrix} 4 \\ 16 \end{bmatrix}$ $\quad P(\omega_1) = 0.6$

$\Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$

$d = 2$

$$g_1 = -\frac{1}{2}\left(x - \begin{bmatrix} 4 \\ 16 \end{bmatrix}\right)^T \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}\left(x - \begin{bmatrix} 4 \\ 16 \end{bmatrix}\right) - \frac{2}{2}\ln 2\pi - \frac{1}{2}\ln(16) + \ln 0.6$$

$$g_2 = -\frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_2| + \ln P(\omega_2)$$

$\mu_2 = \begin{bmatrix} 16 \\ 4 \end{bmatrix}$ $\quad P(\omega_2) = 0.4$

$\Sigma_2 = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$

$d = 2$

$$g_2 = -\frac{1}{2}\left(x - \begin{bmatrix} 16 \\ 4 \end{bmatrix}\right)^T \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}\left(x - \begin{bmatrix} 16 \\ 4 \end{bmatrix}\right) - \frac{2}{2}\ln 2\pi - \frac{1}{2}\ln(16) + \ln 0.4$$

$g_1 = g_2 \rightarrow$

$$-\frac{1}{2}\left(x - \begin{bmatrix} 4 \\ 16 \end{bmatrix}\right)^T \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}\left(x - \begin{bmatrix} 4 \\ 16 \end{bmatrix}\right) + \ln 0.6 = -\frac{1}{2}\left(x - \begin{bmatrix} 16 \\ 4 \end{bmatrix}\right)^T \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}\left(x - \begin{bmatrix} 16 \\ 4 \end{bmatrix}\right) + \ln 0.4$$

$$-\frac{1}{8}\left(x - \begin{bmatrix} 4 \\ 16 \end{bmatrix}\right)^T\left(x - \begin{bmatrix} 4 \\ 16 \end{bmatrix}\right) + \ln 0.6 = -\frac{1}{8}\left(x - \begin{bmatrix} 16 \\ 4 \end{bmatrix}\right)^T\left(x - \begin{bmatrix} 16 \\ 4 \end{bmatrix}\right) + \ln 0.4$$

$$-\frac{1}{8}\left((x_1-4)^2 + (x_2-16)^2\right) + \ln\frac{0.6}{0.4} + \frac{1}{8}\left((x_1-16)^2 + (x_2-4)^2\right) = 0$$

$$(x_1-4)^2 + (x_2-16)^2 - (x_1-16)^2 - (x_2-4)^2 - 8\ln\frac{3}{2} = 0$$

$$x_1^2 - 8x_1 + 16 + x_2^2 - 32x_2 + 256 - x_1^2 + 32x_1 - 256 - x_2^2 + 8x_2 - 16 - 8\ln\frac{3}{2} = 0$$

$$-8x_1 - 32x_2 + 32x_1 + 8x_2 - 8\ln\frac{3}{2} = 0$$

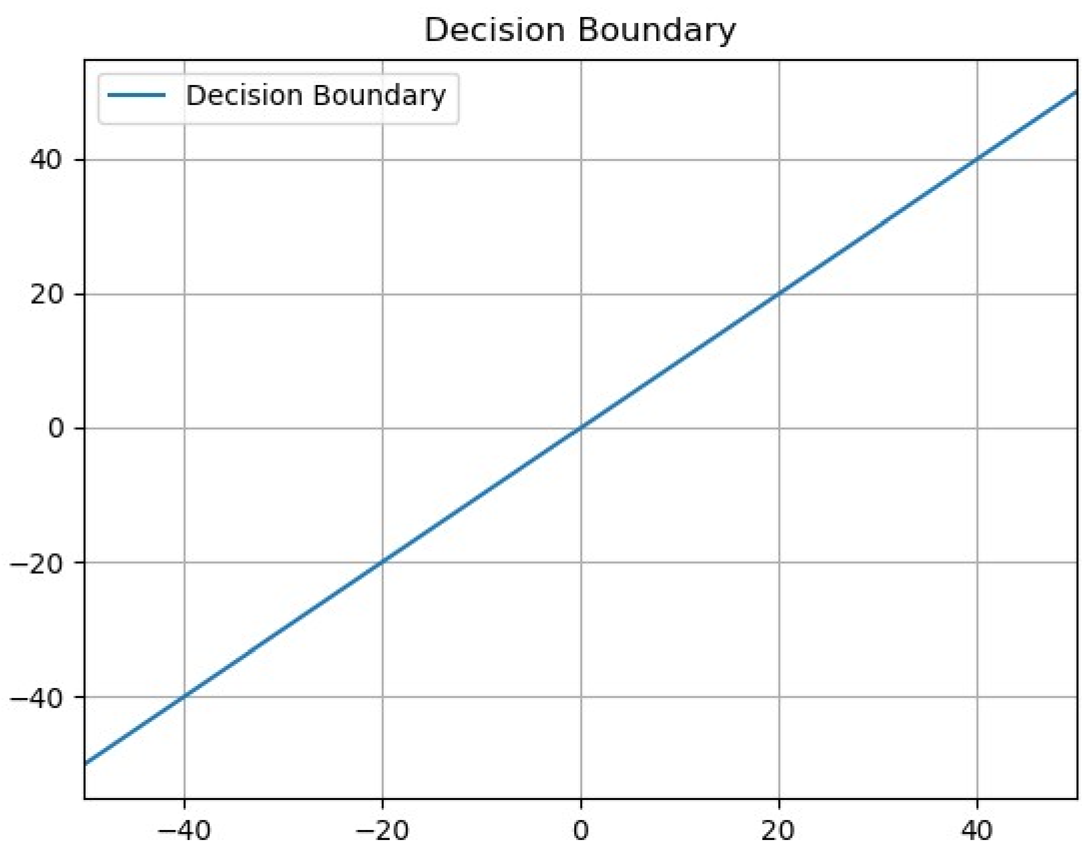
$$-x_1 - 4x_2 + 4x_1 + x_2 - \ln\frac{3}{2} = 0$$

$$\boxed{3x_1 - 3x_2 - \ln\frac{3}{2} = 0} \longrightarrow \boxed{x = \begin{bmatrix} x_2 + \frac{1}{3}\ln\frac{3}{2} \\ x_2 \end{bmatrix}} = \boxed{\begin{bmatrix} x_1 \\ x_1 - \frac{1}{3}\ln\frac{3}{2} \end{bmatrix}}$$

بعد از آنکه مرز تصمیم را به دست آوردم آن را با استفاده از کد زیر رسم کردم:

```python
########################################################################
# Problem 2 - Needed Function
########################################################################
import numpy as np
import matplotlib.pyplot as plt

# First manner: Finding equation of x according g1(x) = g2(x)
# which is equal to "x = [ x1,  x1-(1/3)*(Ln(3/2))]"
# feature 1 and 2 points on decision boundary
x1 = np.linspace(-50, 50, 200)
x2 = [x-(1/3)*(np.log(3/2)) for x in x1]

# Plot curve
plt.figure()
plt.plot(x1,x2, label="Decision Boundary")
plt.grid()
plt.legend()
plt.title("Decision Boundary")
plt.xlim(-50, 50)
```

که حاصل اجرای کد بالا تصویر زیر است (**x2 = x1 – 0.135155036**) :



در روش دوم تابع‌ای به اسم draw_decision_boundary نوشته‌ام که مقدار g1 و g2 را برای نقاط صفحه‌ای که می‌خواهیم رسم کنیم را حساب می‌کند و سپس بعد از آن نقطه‌هایی که مقدار g1 > g2 است را با رنگ متفاوتی از ناحیه‌هایی که در آن g1 <= g2 است را نشان می‌دهد. کد تابع را در تصویر زیر می‌بینید:

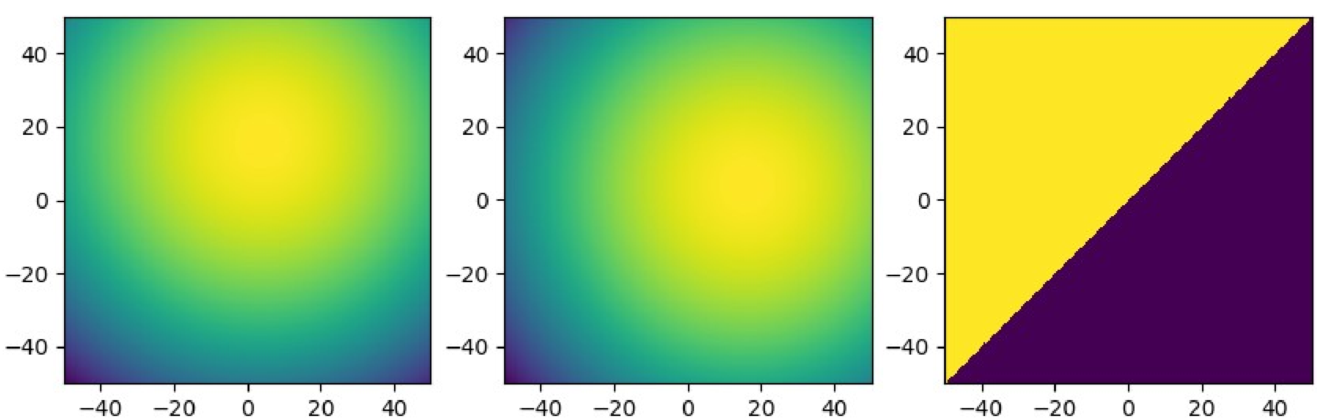
```python
# Second manner: Shows areas with g1(x) > g2(x) with
# different color that areas which g1(x) <= g2(x)
def draw_decision_boundary(plot_range):
    """
    This function draws discriminant function of to different
    normal distribution which are given in problem 2 and also,
    It draws decision boundary
    :param plot_range: range is a tuple (like: (-2,2)) that determines
            range of plot which decision boundary will be drawn on it
    :return: none
    """
    import pylab as pl
    import numpy as np

    # This determines how many points make the plot.
    # Number of points on plot: plot_resolution * plot_resolution
    plot_resolution = 200

    # Make points of plot
    X, Y = np.mgrid[plot_range[0]:plot_range[1]:plot_resolution*1j,
           plot_range[0]:plot_range[1]:plot_resolution*1j]

```

```python
    # Concatenate X,Y
    points = np.c_[X.ravel(), Y.ravel()]

    # Discriminant function for normal distribution 1
    g1Value = g1(points)
    g1Value.shape = plot_resolution, plot_resolution

    # Discriminant function for normal distribution 2
    g2Value = g2(points)
    g2Value.shape = plot_resolution, plot_resolution

    # Creating a figure and three equal subplot in it
    fig, axes = pl.subplots(1, 3, figsize=(15, 5))
    ax1, ax2, ax3 = axes.ravel()
    for ax in axes.ravel():
        ax.set_aspect("equal")

    ax1.pcolormesh(X, Y, g1Value)
    ax2.pcolormesh(X, Y, g2Value)

    # Determining decision boundary
    ax3.pcolormesh(X, Y, g1Value > g2Value)
```

و سپس ا این تابع را برای بازه‌ی ۵۰- تا ۵۰ فراخوانده‌ام:

```python
draw_decision_boundary((-50, 50))
plt.show()
```

که خروجی آن را در تصویر زیر مشاهده می‌کنید:



سوال ۳:

این سوال یک سری داده‌ی دو بعدی را داده است که هر کدام از داده‌ها مربوط به کلاس یک است یا دو و در رابطه با این داده‌ها سوال‌هایی را مطرح کرده است. کد مربوط به این سوال در پوشه‌ی ”problem3“ و در فایل ”problem3.py“ قرار دارد.

در این سوال از تابع‌های زیر استفاده شده است که همان تابع هایی هستند که در سری تمرین‌های شماره‌ی یک آن ها را نوشتم و از آن‌ها استفاده کردم.

تابع covariance_matrix:

این تابع تعدادی sample را می گیرد و از طریق محاسباتی که در کامنت‌ها توضیح داده شده است ماتریس کواریانس سمپل ها را پیدا می‌کند. در تصویر زیر کد کامنت گذاری شده‌ی این تابع را می‌بینید:

```python
def covariance_matrix(samples):
    """
    This function gets some samples and return its covariance matrix
    :param samples: each col represents a variable,
     with observations in the row
    :return: Corresponding covariance matrix
    """
    import numpy as np
    samplesArray = np.asarray(samples)
    samplesArray = np.transpose(samplesArray)

    # Cov function calculates covariance it this way:
    # X_new(k) = X(k) - mu
    # B = [X_new(1),X_new(2),...,X_new(n)]
    # Cov = (1/(n-1))*B*transpose(B)
    return np.cov(samplesArray).tolist()
```

تابع mean_vector:

این تابع تعدادی sample را می‌گیرد و میانگین آن ها را محاسبه می‌کند و به عنوان خروجی یک لیست بر می‌گرداند که برابر است با میانگین سمپل‌ها، در زیر کد این تابع را مشاهده می‌کنید:

```python
def mean_vector(X):
    """

    :param X: each col represents a variable,
     with observations in the row
    :return: return a list that contains mean of each col
    """
    import numpy as np
    means = np.mean(X, axis=0)
    return means.tolist()
```

تابع mean_and_covariance:

این تابع سمپل‌ها را می‌گیرد و دو تابع بالا را فرامی خواند و میانگین و ماتریس کواریانس داده‌ها را در قالب یک دیکشنری که شامل کلیدهای means و covariance هست را برمی‌گرداند. در زیر کد کامنت‌گذاری شده‌ی این تابع را مشاهده می‌کنید:

```python
def mean_and_covariance(samples):
    """
    This function gets some samples and then it finds
    corresponding Covariance and mean of their normal distribution.
    :param samples: samples.each col represents a variable,
     with observations in the row
    :return: a dictionary with keys: {'means': , 'covariance'}
    """
    dic = {}
    dic['means'] = mean_vector(samples)
    dic['covariance'] = covariance_matrix(samples)
    return dic
```

تابع **discriminant_function:**

این تابع یک سری نقطه را می‌گیرد و همین طور مشخصات یک توزیع نرمال را می‌گیرد و بعد discriminant value را برای هر نقطه طبق رابطه‌ی زیر محاسبه می‌کند.

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i). \qquad (47)$$

در زیر کد کامنت گذاری شده‌ی زیر را مشاهده می‌کنید:

```python
def discriminant_function(mu, covariance, priorProbability, sampleVector):
    """
    This function gets an vector of samples and calculates
    value of discriminant function for each.

    :param mu: a vector of means (list, couple, ...)
    :param covariance: covariance matrix (list, couple, ...)
    :param sampleVector: vectors of samples (list, couple, ...)
    :param priorProbability: prior probability (list, couple, ...)
    :return: A vector of discriminant of each sample in the sample vector (np.array)
    """
    import numpy as np

    # Convert inputs to numpy.matrix()
    Mu = np.matrix(mu)
    Cov = np.matrix(covariance)
    inputVector = np.matrix(sampleVector)

    # Dimension of given normal distribution
    d = Cov.shape[0]

```

```python
    # Inverse of covariance matrix
    invCov = np.linalg.inv(Cov)

    # The part of discriminant function which is equal for different input samples
    discriminant_value_part2 = -(d/2)*np.log(2*np.pi)-\
                               (1/2)*np.log(np.linalg.det(Cov))+np.log(priorProbability)

    # Output
    outputVector = []

    # Calculating value of discriminant function for all input samples
    for i in range(inputVector.shape[0]):
        # Calculating value of discriminant function for input sample i
        sampleOutput = (-(1/2) * (inputVector[i] - Mu) * invCov * (inputVector[i] - Mu).transpose())+\
                       discriminant_value_part2
        outputVector.append(sampleOutput)

    #return an array of discriminant function
    return np.array(outputVector)
```

تابع draw_decision_boundary_2d_normal:

این تابع میانگین و ماتریس کواریانس و prior دو کلاس را می‌گیرد همچنین بازه‌ای را که می خواهد مرز تصمیم‌گیری را در آن رسم کند را به عنوان ورودی می‌گیرد. بعد در آن بازه مقدار discriminant value را برای هر دو توزیع حساب می کند و در آخر سه تصویر در بازه‌ی مشخص شده رسم می کند.

۱) مقدار discriminant value برای نقاط، توزیع اول

۲) مقدار discriminant value برای نقاط، توزیع دوم

۳) برای نقاطی که در توزیع اول مقدار discriminant value آن‌ها بیشتر از توزیع دوم است را با رنگی متفاوت می کشد.

در زیر کد کامنت گذاری شده‌ی این تابع را مشاهده می‌کنید:

```python
def draw_decision_boundary_2d_normal(mu1, cov1, prior1, mu2, cov2, prior2, plot_range):
    """
    This function draws discriminant function of to different
    normal distribution and also, It draws decision boundary
    :param mu1: means vector of distribution 1
    :param cov1: covariance matrix of distribution 1
    :param prior1: prior probability of distribution 1
    :param mu2: means vector of distribution 2
    :param cov2: covariance matrix of distribution 2
    :param prior2: prior probability of distribution 2
    :param plot_range: range is a tuple (like: (-2,2)) that determines
            range of plot which decision boundary will be drawn on it
    :return: none
    """
    import pylab as pl
    import numpy as np

    # This determines how many points make the plot.
    # Number of points on plot: plot_resolution * plot_resolution
    plot_resolution = 300

    # Make points of plot
```

```python
    X, Y = np.mgrid[plot_range[0]:plot_range[1]:plot_resolution*1j,
           plot_range[0]:plot_range[1]:plot_resolution*1j]

    # Concatenate X,Y
    points = np.c_[X.ravel(), Y.ravel()]

    # Inverse of matrix cov1, for preventing reparation of computation
    invC = np.linalg.inv(cov1)

    # Discriminant function for normal distribution 1
    g1 = discriminant_function(mu1, cov1, prior1, points)
    g1.shape = plot_resolution, plot_resolution

    # Inverse of matrix cov1, for preventing repartition of computation
    invC = np.linalg.inv(cov2)

    # Discriminant function for normal distribution 2
    g2 = discriminant_function(mu2, cov2, prior2, points)
    g2.shape = plot_resolution, plot_resolution

    # Creating a figure and three equal subplot in it
```

```python
    fig, axes = pl.subplots(1, 3, figsize=(15, 5))
    ax1, ax2, ax3 = axes.ravel()
    for ax in axes.ravel():
        ax.set_aspect("equal")

    ax1.pcolormesh(X, Y, g1)
    ax2.pcolormesh(X, Y, g2)

    # Determining decision boundary
    ax3.pcolormesh(X, Y, g1 > g2)

    pl.show()
```

اکنون که تابع‌های استفاده شده را شرح دادم به بخش‌های مختلف سوال می‌پردازم.

بخش A)

این بخش خواستار محاسبه‌ی prior probability هر کدام از کلاس‌ها است. برای این منظور کد زیر را نوشته ام که تعداد سمپل‌های هر کلاس نسبت به کل سمپل‌ها را به دست می‌آورد:

```python
#######################################################################
# Problem 3- a
#######################################################################
classesPrior = {}
numberOfAllSamples = 0
# Count number of all samples in all classes
for key in classes:
    numberOfAllSamples += len(classes[key])

# calculate and print prior probability of each class
for key in classes:
    classesPrior[key] = len(classes[key])/numberOfAllSamples
    print('Prior of w{} is :{}'.format(key, classesPrior[key]))
```

خروجی اجرای این قسمت از کد را در تصویر زیر مشاهده می‌کنید:

```
Prior of w1 is :0.5
Prior of w2 is :0.5
```

بخش b)

این بخش خواسته است که میانگین و ماتریس کواریانس هر کلاس را محاسبه کنیم، برای این منظور کد زیر را نوشته‌ام:

```python
#####################################################################
# Problem 3- b
#####################################################################
# Find mean and covariance of each class
classesMeanAndCov = {}
for key in classes:
    # This function return mean and cov matrix in a dictionary
    # data structure, {'means': , 'covariance': }
    classesMeanAndCov[key] = mean_and_covariance(classes[key])

# print mean an covariance for each class
for key in classesMeanAndCov:
    print('W{}:\nmeans is: {},\ncovariance matrix is: {}'.format(
        key, classesMeanAndCov[key]['means'], classesMeanAndCov[key]['covariance']))
```

خروجی اجرای این بخش از کد را در تصویر زیر مشاهده می کنید:

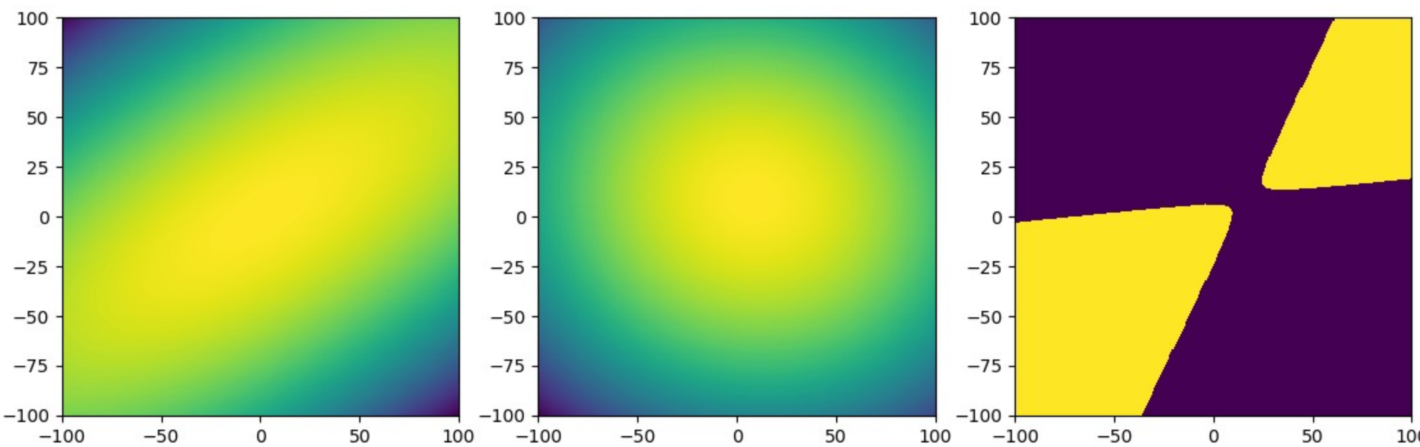
```
W1:
means is: [1.8333333333333333, 1.5],
covariance matrix is: [[2.166666666666667, 1.1], [1.1, 1.1]]
W2:
means is: [8.166666666666666, 9.333333333333334],
covariance matrix is: [[1.366666666666667, -0.06666666666666667], [-0.06666666666666667, 1.0666666666666669]]
```

بخش c)

این بخش خواستار رسم decision boundary شده است، به این منظور تابع رسم مرز تصمیم را که در بالا شرح دادم را در کد به شکل زیر فراخواندم:

```python
########################################################################
# Problem 3- C
########################################################################
# Draw decision boundary which separate class1 and class2
print("Drawing decision boundary takes several second, ...")
draw_decision_boundary_2d_normal(classesMeanAndCov[1]['means'],
                                 classesMeanAndCov[1]['covariance'],
                                 classesPrior[1],
                                 classesMeanAndCov[2]['means'],
                                 classesMeanAndCov[2]['covariance'],
                                 classesPrior[2],
                                 (-100,100)
                                 )
```

خروجی اجرای این قسمت از کد را در تصویر زیر مشاهده می‌کنید(بازه‌ی -100 تا 100) :



بخش d)

بله تحت تاثیر قرار می‌دهد. اگر به صورت zero-one loss نباشد مرز تصمیم حرکت می‌کند به طوری که مساحت قسمتی که وزن خطای کمتری دارد کمتر می‌شود و مساحت قسمتی‌که ضریب و وزن خطای بیشتر دارد بیشتر می‌شود.

سوال ۴:

بخش a, b, c)

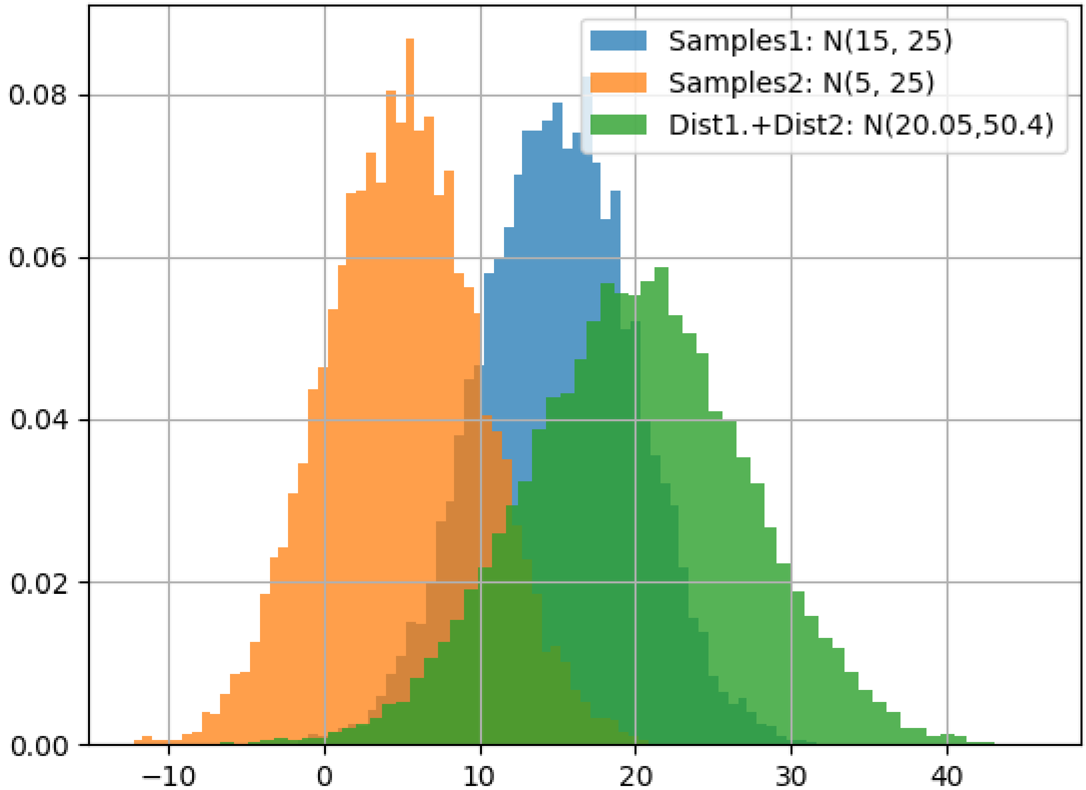
برای این سوال برنامه‌ای نوشتم که برای دو توزیع نرمال، برای هر کدام ۱۰۰۰۰ نمونه ایجاد می‌کند و هر نمونه از توزیع اول را با نمونه‌ی متناظر در توزیع دوم جمع می‌کند تا ۱۰۰۰۰ نمونه برای توزیع سوم ایجاد شود و بعد میانگین و واریانس توزیع سوم را حساب می کند و در نهایت Histogram هر سه توزیع را رسم می‌کنم و این کار را برای ۵ بار با داده‌های مختلف تکرار می‌کند. در زیر کد کامنت گذاری شده‌ی این برنامه را می‌بینید:

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

# Number of samples
n = 10000

# Different value of mean and standard deviation
# for two normal distributions
Means1 = [-30, 15, 2, 6, 0]
Means2 = [30, 5, 8, 12, 0]
Sigma1 = [3, 5, 8, 10, 1]
Sigma2 = [30, 5, 3, 2, 1]

# Do experiment with different values
for i in range(5):
    mu1 = Means1[i]
    sigma1 = Sigma1[i]

    mu2 = Means2[i]
    sigma2 = Sigma2[i]

    # Draw samples from distribution 1
```

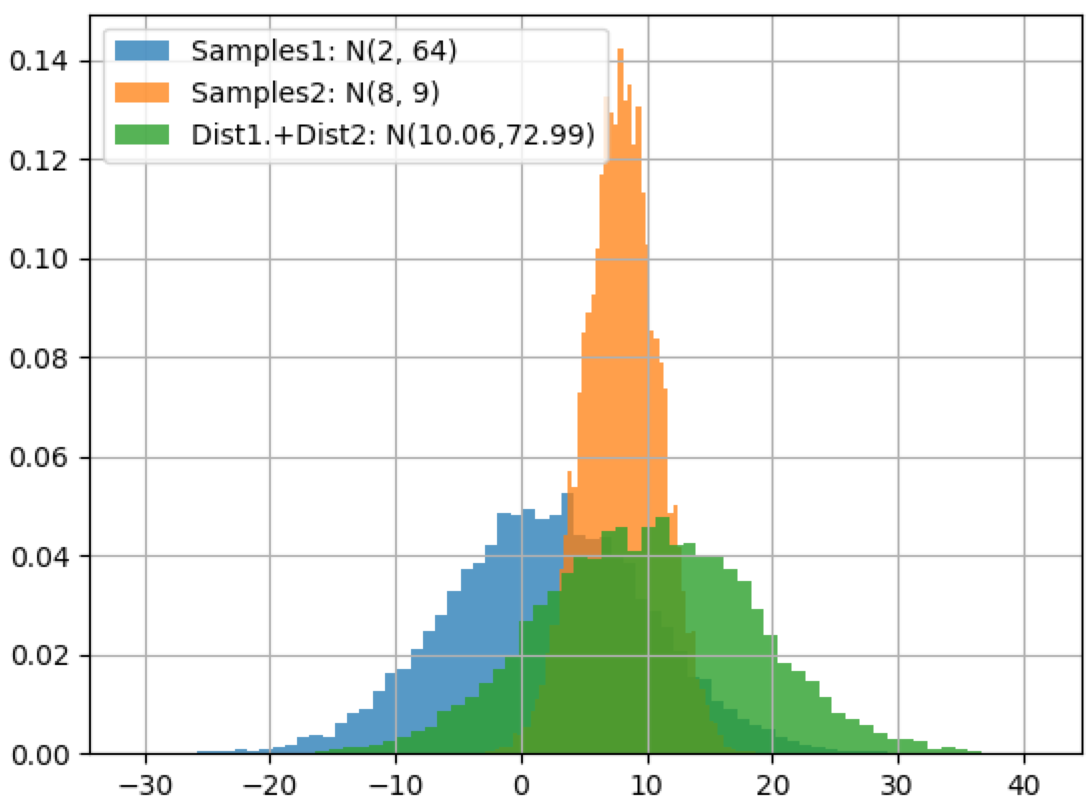
```python
    samples1 = np.random.normal(loc=mu1, scale=sigma1, size=n)

    # Draw samples from distribution 2
    samples2 = np.random.normal(loc=mu2, scale=sigma2, size=n)

    # Add each element of sample1 to corresponding element of sample2
    samples3 = samples1 + samples2

    # Mean of samples3
    mu3 = np.mean(samples3)

    # Var of samples3
    var3 = np.cov(samples3)

    plt.figure()
    # Plot histogram of sample form distribution N(mu1, sigma1^2)
    plt.hist(x=samples1, bins='auto', normed=True, alpha=0.75,
             label='Samples1: N({}, {})'.format(mu1, sigma1 ** 2))

    plt.hist(x=samples2, bins='auto', normed=True, alpha=0.75,
             label='Samples2: N({}, {})'.format(mu2, sigma2 ** 2))
```

```python

    # Plot histogram of sample form samples1+samples2
    plt.hist(x=samples3, bins='auto', normed=True, alpha=0.8,
             label='Dist1.+Dist2: N({},{})'.format(np.round(mu3, 2), np.round(var3, 2)))

    plt.grid()
    plt.legend()

plt.show()
```

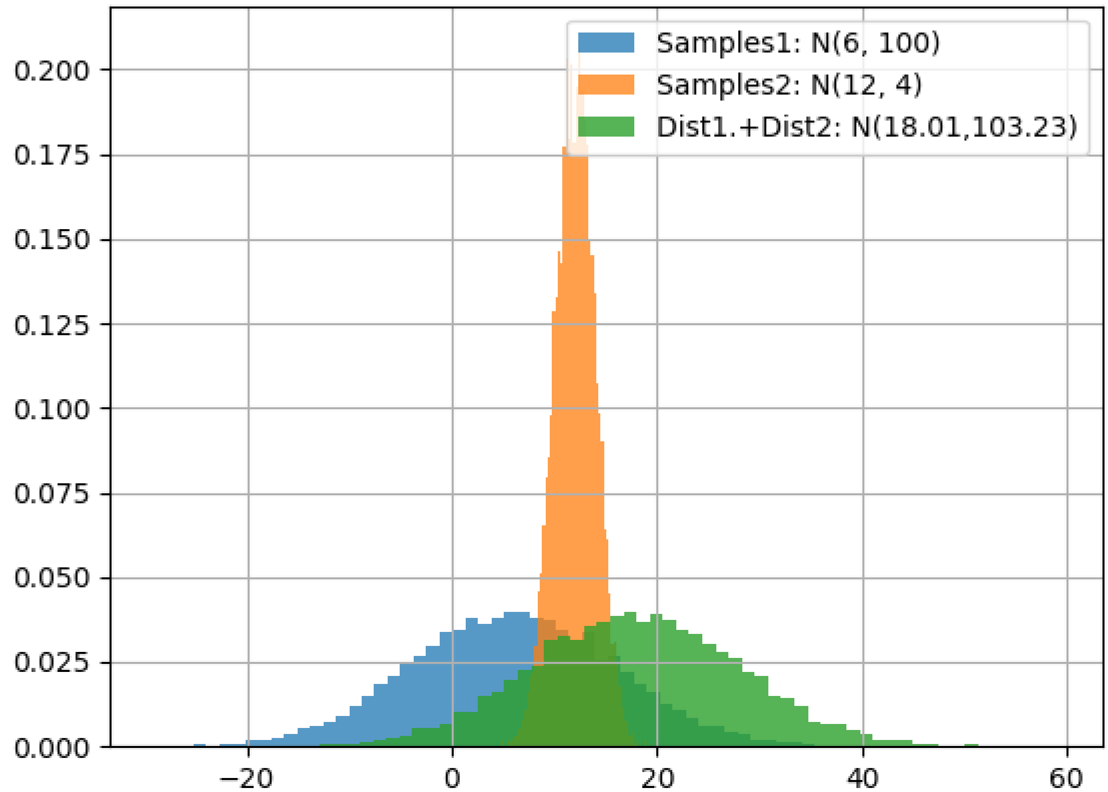
بعد از اجرای این کد ۵ شکل رسم می‌شود که هر شکل حاوی histogram دو توزیع نرمال و توزیع جدید حاصل از جمع اجزای آن دو است و همین طور در قسمت label ها میانگین و واریانس دو توزیع اول و میانگین و واریانس تخمینی توزیع سوم(رنگ سبز) نمایش داده می شود:

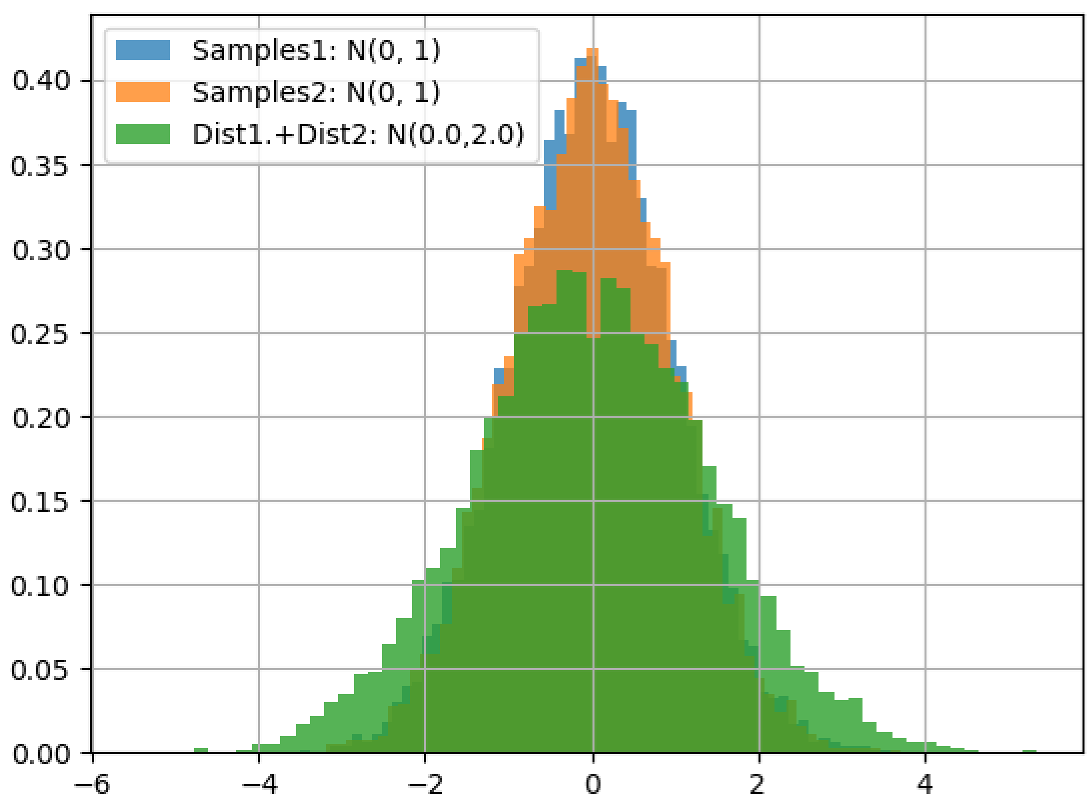


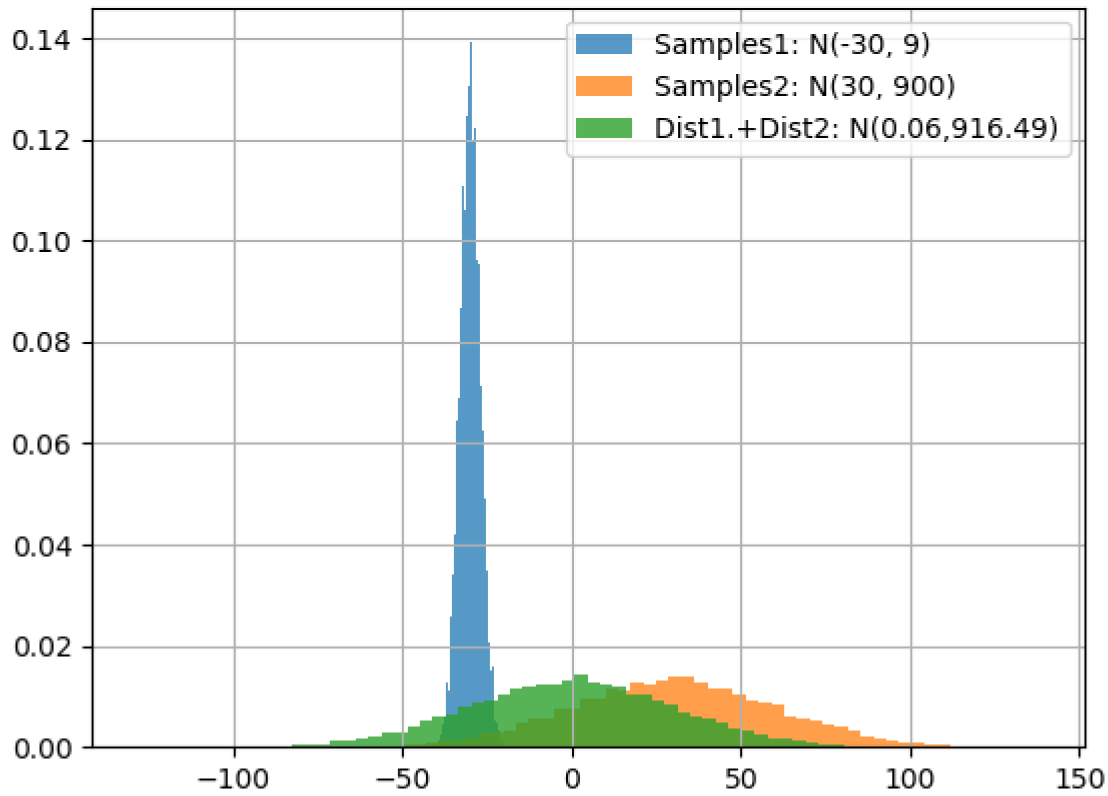
Samples1: N(15, 25)
Samples2: N(5, 25)
Dist1.+Dist2: N(20.05,50.4)
0.00
0.02
0.04
0.06
0.08
−10
0
10
20
30
40



Samples1: N(2, 64)
Samples2: N(8, 9)
Dist1.+Dist2: N(10.06,72.99)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
−30
−20
−10
0
10
20
30
40


Samples1: N(6, 100)
Samples2: N(12, 4)
Dist1.+Dist2: N(18.01,103.23)
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
−20
0
20
40
60


Samples1: N(0, 1)
Samples2: N(0, 1)
Dist1.+Dist2: N(0.0,2.0)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
−6
−4
−2
0
2
4



همین طور که مشاهده می‌کنید در هر پنج تصویر

mean3 ≈ mean1 + mean2

variance3 ≈ variance1 + variance2

میانگین و واریانس جدید**تقریبا** برابر با مجموع میانگین و واریانس دو توزیع نرمال اول است.

سوال ۵:

قبل از پاسخگویی به بخش های مختلف سوال ابتدا تابع‌هایی را که استفاده کرده‌ام را معرفی می‌کنم.
تابع‌های `mean_and_covariance`، `mean_vector`، `covariance_matrix` ، `discriminant_function` و `draw_decision_boundary_2d_normal` را در سوال سه استفاده کرده‌ام و آن‌ها را ان جا توضیح داده‌ام.

تابع `bhattacharyya_bound` شباهت دو توزیع نرمال را حساب می‌کند که باندی برای خطا است کد کامنت گذاری شده‌ی این تابع را در تصویر زیر مشاهده می‌کنید(این تابع را در سری تمرین‌های شماره‌ی یک استفاده کرده ام):

```python
def bhattacharyya_bound(normalDist1, normalDist2):
    """
    This measure shows similarities between two
    distribution and Also it's a error bound.
    :param normalDist1: is dictionary that contains
            these keys {'means','covariance', 'name','prior'}
    :param normalDist2: is dictionary that contains
            these keys {'means','covariance', 'name','prior'}
    :return: return a scalar
    """
    import numpy as np

    # Convert data to proper form
    mean1 = np.asmatrix(normalDist1['means'])
    mean2 = np.asmatrix(normalDist2['means'])
    cov1 = np.asmatrix(normalDist1['covriance'])
    cov2 = np.asmatrix(normalDist2['covriance'])

    # Average of two covariances
    covAve = (cov1 + cov2) / 2

    # Subtract of means
    meanSub = mean2 - mean1

    # Calculate the bound
    bound = (np.sqrt(normalDist1['prior'] * normalDist2['prior'])) * (
            np.exp(-1 * ((1/8) * (meanSub * np.linalg.inv(covAve) * np.transpose(meanSub)) +
                         (1/2) * np.log(
                             np.linalg.det(covAve)/np.sqrt(np.linalg.det(cov1) *
                                                           np.linalg.det(cov2))
                                        ))))

    #return bound
    return bound.tolist()[0][0]
```

بخش a)

این بخش دو توزیع نرمال و prior هایشان را داده است و مرز تصمیم را خواسته است که رسم کنیم. در تصویر زیر محاسبه‌ی مرز تصمیم را مشاهده می‌کنید:

$$g_1 = -\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_1| + \ln P(\omega_1)$$

$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\quad P(\omega_1) = 0.5 \quad$ Covariance Matrix $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad d = 2$

$$g_1 = -\frac{1}{2}\left(x - \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) - \frac{2}{2}\ln 2\pi - \frac{1}{2}\times 1 + \ln(0.5)$$

$$g_2 = -\frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_2| + \ln P(\omega_2)$$

$\mu_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\quad P(\omega_2) = 0.5 \quad$ Covariance Matrix $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad d = 2$

$$g_2 = -\frac{1}{2}\left(x - \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) - \frac{2}{2}\ln 2\pi - \frac{1}{2}\times 1 + \ln(0.5)$$

$$g_1(x) = g_2(x) \Rightarrow$$

$$-\cancel{\frac{1}{2}}(x)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}(x) = -\cancel{\frac{1}{2}}\left(x - \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) \Rightarrow$$

$$x_1^2 + x_2^2 = (x_1 - 1)^2 + (x_2 - 1)^2 \Rightarrow$$

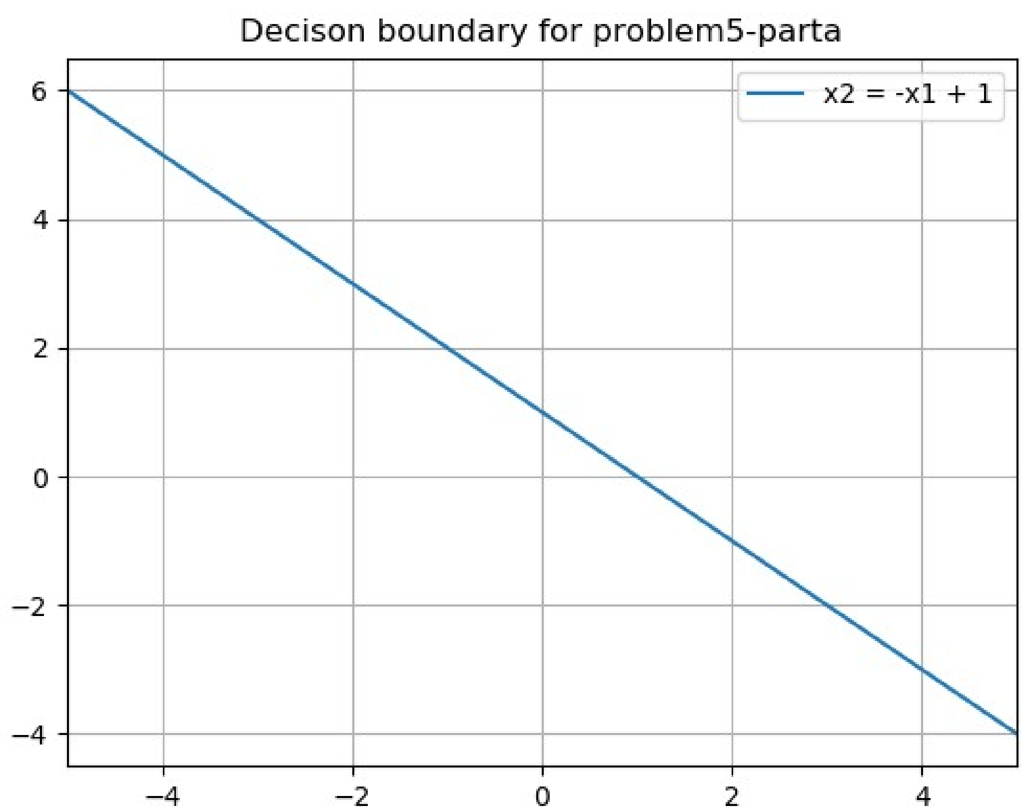
$$\cancel{x_1^2} + \cancel{x_2^2} = \cancel{x_1^2} - 2x_1 + 1 + \cancel{x_2^2} - 2x_2 + 1 \Rightarrow$$

$$\boxed{x_1 = -x_2 + 1} \Rightarrow x = \begin{bmatrix} x_1 \\ -x_1 + 1 \end{bmatrix}$$

که برای رسم این خط کد زیر را نوشته ام:

```python
###########################################################################
# Problem 5- a
###########################################################################
# Draw decision boundary
# As I've shown in report when we put g1(x) = g2(x)
# we gain x2 = -x1 + 1
import numpy as np
import matplotlib.pyplot as plt

# Generate 200 points for drawing decision boundary
x1 = np.linspace(-5, 5, 200)

# Calculate feature x2 according x2 = -x1 + 1
x2 = [(-1*x)+1 for x in x1]

plt.plot(x1, x2, label='x2 = -x1 + 1')
plt.legend()
plt.grid()
plt.xlim(-5, 5)
plt.title("Decison boundary for problem5-parta")
plt.show()
```

بعد از اجرای این قطعه از کد نمودار زیر ساخته می‌شود:



بخش b)

این بخش از سوال خواسته است Bhattacharyya error bound را حساب کنیم که به همین منظور تابع `bhattacharyya_bound` را به صورت زیر فراخوانده‌ایم:

```python
######################################################################
# Problem 5- b
######################################################################
normalDist1 = {'means': [0, 0], 'covriance': [[1, 0],[0, 1]], 'name': 'w1','prior': 0.5}
normalDist2 = {'means': [1, 1], 'covriance': [[1, 0],[0, 1]], 'name': 'w2','prior': 0.5}

# calculate bhattacharyya bound
bhb = bhattacharyya_bound(normalDist1, normalDist2)
print("Bhattacharyya Bound is: ", bhb)

```

حاصل اجرای کد بالا را در تصویر زیر مشاهده می‌کنید:

```
/home/bat/anaconda3/bin/python /home/bat/Dropbox/codes/pythonCo
Bhattacharyya Bound is:  0.38940039153570244
```

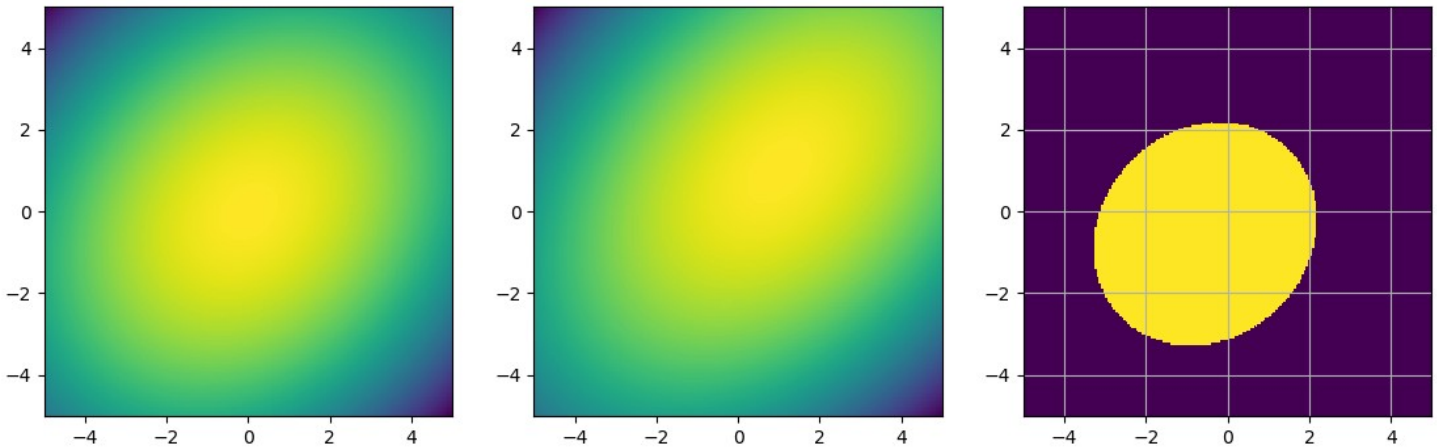
بخش C)

بخش d)

برای رسم decision boundary توزیع های نرمال داده شده تابع `draw_decision_boundary_2d_normal` فراخوانده‌ام که دو توزیع نرمال و prior آن ها را می‌گیرد همان طور بازه‌ای که باید مرز تصمیم را رسم کند را می‌گیرد و برای نقطه‌های درون بازه discriminant value را برای دو توزیع نرمال یک و دو حساب می‌کند و ناحیه‌های که g1 > g2 را با رنگ متفاوتی از ناحیه‌هایی که g1 <= g2 است را نشان می‌دهد و مرز تصمیم گیری برابر می‌شود با محل تلاقی این دو ناحیه. کد زیر را جهت فراخوانی تابع رسم مرز تصمیم گیری و حد Bhattacharyya نوشته‌ام:

```python
######################################################################
# Problem 5- d
######################################################################
print('wait, drawing decision boundary takes several second ...')
draw_decision_boundary_2d_normal(mu1=[0, 0], cov1=[[2, 0.5],[0.5, 2]], prior1=0.5,
                                 mu2=[1, 1], cov2=[[5, 2], [2, 5]], prior2=0.5,
                                 plot_range=(-5, 5))

normalDist1 = {'means': [0, 0], 'covriance': [[2, 0.5],[0.5, 2]], 'name': 'w1','prior': 0.5}
normalDist2 = {'means': [1, 1], 'covriance': [[5, 2],[2, 5]], 'name': 'w2','prior': 0.5}

# calculate bhattacharyya bound
bhb = bhattacharyya_bound(normalDist1, normalDist2)
print("5-d) Bhattacharyya Bound is: ", bhb)

```

بعد از اجرای کد بالا نتایج زیر به دست می آید:



و

```
wait, drawing decision boundary takes several second ...
5-d) Bhattacharyya Bound is:  0.4322519655344438
```

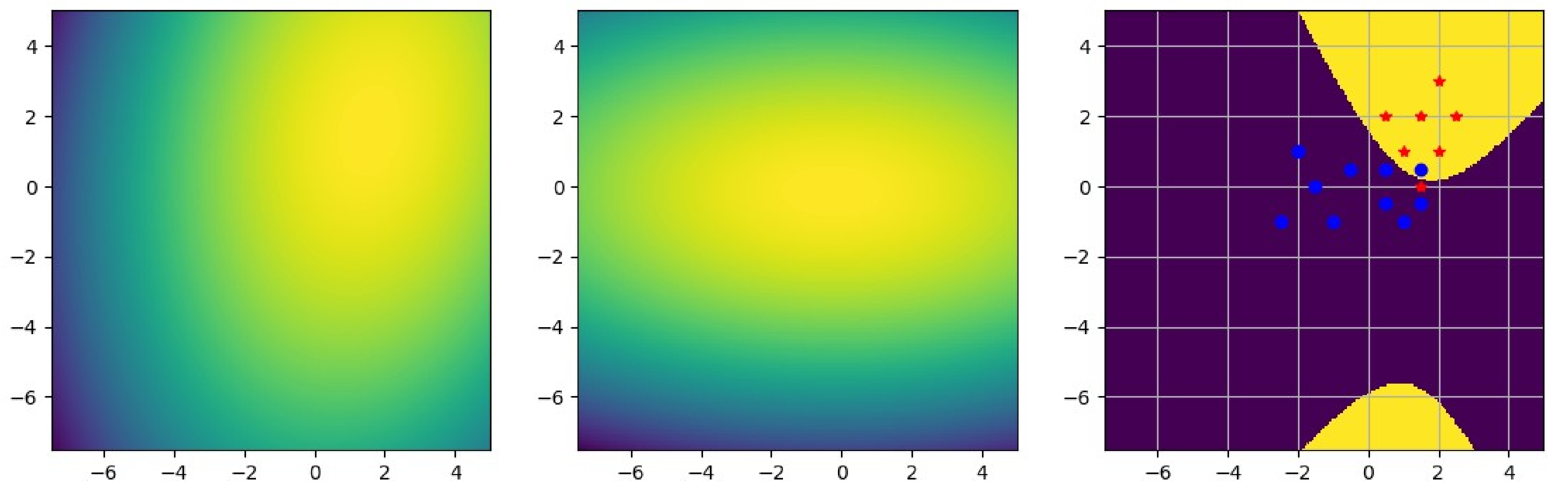
سوال ٦:

کدهای این سوال در پوشه‌ی problem6 و در فایل problem6.py قرار دارند. برای این سوال از تابع‌های زیر استفاده کرده‌ام:

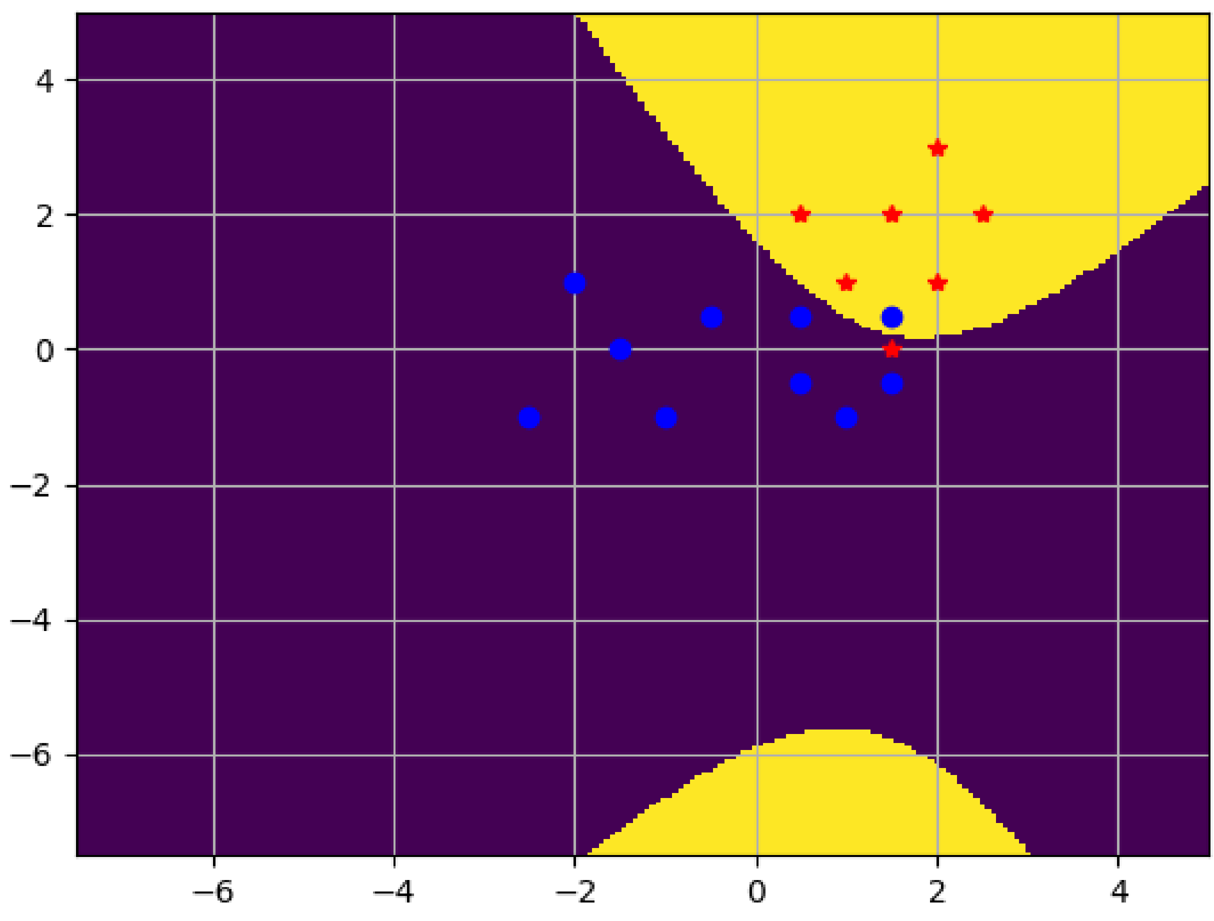
`discriminant_function`، `mean_and_covariance`، mean_vector، `covariance_matrix`، و `draw_decision_boundary_2d_normal` کرده ام که همه‌ی آن ها را در سوال‌های قبل استفاده کرده ام و همان جا آن‌ها را توضیح داده ام به همین دلیل از آوردن کد آن‌ها در این سوال خودداری می‌کنم. فقط در تابع `draw_decision_boundary_2d_normal` تغییر کوچکی ایجاد کردم و در آرگمان ها `sample` های کلاس `w1` و `w2` را می‌گیرم تا علاوه بر رسم مرز تصمیم `sample` ها را هم بر روی صفحه رسم کنم. در تصویر زیر فراخوانی این تابع‌ها را برای پیدا کردن میانگین و ماتریس کواریانس توزیع‌ها و پیدا کردن مرز تصمیم را مشاهده می‌کنید:

```python
#########################################################################
# Problem 6-
#########################################################################

# mean and covariance matrix w1:
normal_dist_w1 = mean_and_covariance(w1)

# mean and covariance matrix w2:
normal_dist_w2 = mean_and_covariance(w2)

# Find Prior probabilities of w1 and w2
prior1 = len(w1) / (len(w1)+len(w2))
prior2 = len(w2) / (len(w1)+len(w2))

print('Drawing takes several seconds, Wait ...')
# Draw decision boundary and samples
draw_decision_boundary_2d_normal(mu1=normal_dist_w1['means'], cov1=normal_dist_w1['covariance'], prior1= prior1,
                                 mu2=normal_dist_w2['means'], cov2=normal_dist_w2['covariance'], prior2= prior2,
                                 plot_range=(-7.5, 5),
                                 samplesw1= w1,
                                 samplesw2= w2
                                 )
```

نتیجه‌ی حاصل اجرای کد بالا را در تصویر زیر مشاهده می کنید. دو تصویر سمت چپ مقدار `discriminat values` کلاس `w1` و `w2` است در بازه‌ی مشخص شده برای رسم است و **تصویر سمت راست تصویر مرز تصمیم و `samples`** است.



تصویری بزرگ تر از مرز تصمیم و سمپل‌ها:



سوال ۷:

کدهای مربوط به این سوال در پوشه‌ی problem7 و در فایل problem7.py قرار دارند.

بخش a , b)

این دو بخش از سوال خواسته است که توزیع دو کلاس ضرب در prior هایشان را رسم کنیم و مرز تصمیم‌ها را مشخص کنیم. برای رسم موارد خواسته شده کد زیر را نوشته‌ام:

تابع c1 مقدار توزیع نرمال کلاس یک را در نقطه‌ی x را مشخص می‌کند.

```python
###########################################################################
# Problem 7 - a, b
###########################################################################


def c1(x):
    """
    Determine value of x in uniform distribution (2, 4)
    :param x: x is a given point
    :return: value of point x in U(2, 4)
    """
    if(x >= 2 and x <= 4):
        return 1/2
    else:
        return 0
```

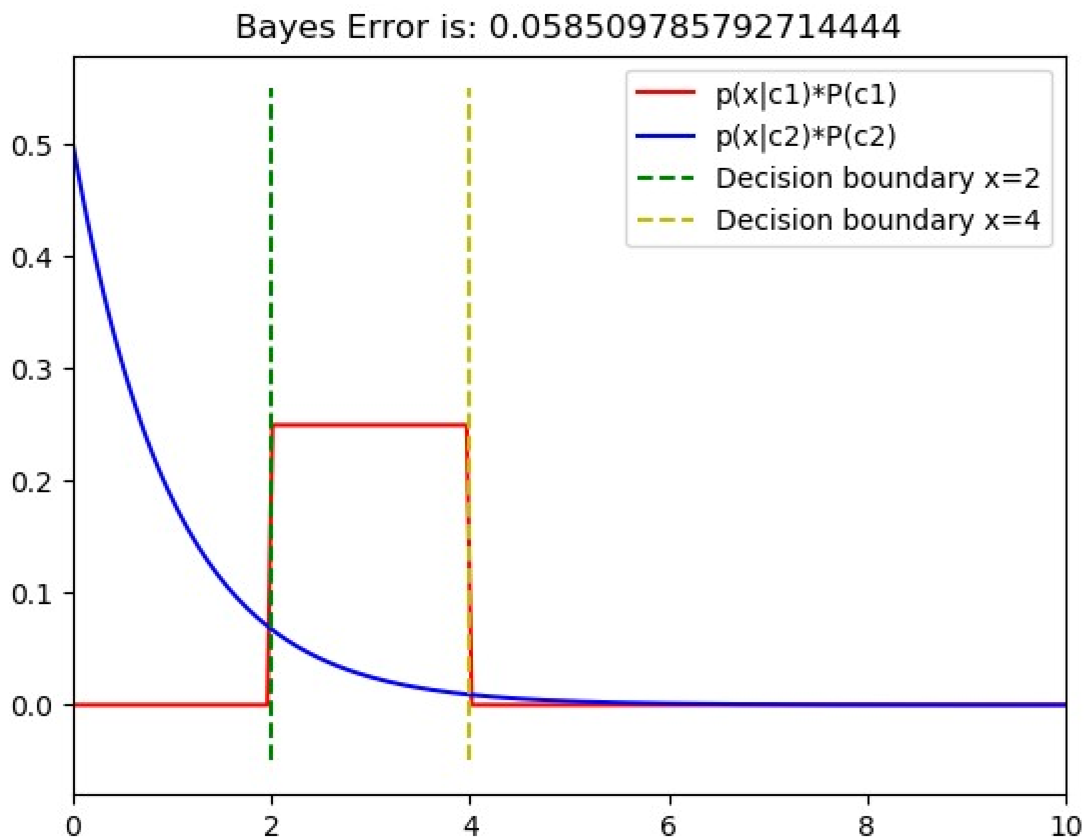
تابع c2 مقدار توزیع نرمال کلاس دو را در نقطه‌ی x را مشخص می‌کند.

```python
def c2(x):
    """
    Determine value of x in exponential distribution (lambda = 1)
    :param x: x is a given point
    :return: value of point x in exponential(lambda = 1)
    """
    import numpy as np
    return np.exp(-1 * x)
```

در ادامه در بازه‌ی 0 تا 10 دویست نقطه ایجاد کرده‌ام و در آن نقطه‌ها میزان توزیع هر دو کلاس ضرب در prior هایشان را حساب کرده‌ام و بعد آن ها را رسم کرده‌ام:

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

priorC1 = priorC2 = 0.5

# Generate 200 point for drawing curves between 0 & 10
x = np.linspace(0, 10, 200)

# Generate density of class1
density1 = [c1(point) * priorC1 for point in x]

# Generate density of class2
density2 = [c2(point) * priorC2 for point in x]

# Plot densities
plt.plot(x, density1, 'r-', label='p(x|c1)*P(c1)')
plt.plot(x, density2, 'b-', label='p(x|c2)*P(c2)')

# Plot decision boundaries
y = np.linspace(-0.05, 0.55, 100)
plt.plot([2]*100, y, 'g--', label='Decision boundary x=2')
plt.plot([4]*100, y, 'y--', label='Decision boundary x=4')

#plt.grid()
plt.legend()
plt.xlim(0, 10)
```

حاصل اجرای کد بالا را در تصویر زیر مشاهده می‌کنید:



همین طور که مشاهده می‌کنید دو خط جدا کننده داریم. ناحیه‌ی بین 2 و 4 برابر است با ناحیه‌ی کلاس یک و باقی محور x مربوط به کلاس دو می‌شود زیرا تابع دو در آن نقاط مقدار بیشتر دارد.

بخش C)

مقدار خطا برابر است با مساحت زیر خط e ^ (-x) ضرب در prior کلاس دو. ولی از آنجایی که انتگرال e^(-x) شکل مشخصی ندارد باید آن را تخمین بزنیم که برای این کار از **بسط تیلور** به صورت زیر استفاده کرده ام:

```
Taylor e^x: e^x = 1 + x + (x^2)/2! + (x^3)/3! + …    ===>
Taylor e^(-x): e^(-x) = 1 - x + (x^2)/2! - (x^3)/3! + …   ===>
∫ e^(-x) = Integral 1 - x + (x^2)/2! - (x^3)/3! + … ===>
which is equal to
∫ e^(-x) = [x - (x^2)/2! + (x^3)/3! - (x^4)/4! + ...] on interval[a,b]
```

که با توجه به رابطه‌ی بالا تابع زیر را نوشته‌ام که مقدار تقریبی خطا را حساب می‌کند:

```python
#######################################################################
def approximate_error(a, b, priorOfClassC2):
    """
    This function approximates Integral e^(-x)
    according to below relations
    taylor e^x: e^x = 1 + x + (x^2)/2! + (x^3)/3! + ...
    taylor e^(-x): e^(-x) = 1 - x + (x^2)/2! - (x^3)/3! + ...

    Integral e^(-x) = Integral 1 - x + (x^2)/2! - (x^3)/3! + ...
    which is equal to
    Integral e^(-x) = [x - (x^2)/2! + (x^3)/3! - (x^4)/4! + ...] on interval[a,b]

    :param a: start of integral interval
    :param b: end of integral interval
    :param priorOfClassC2: prior of class c2 for multipling it to area under
            e^(-x)
    :return: approximate integral
    """
    # Calculate Integral according to above formula
    factoral = 1
    valueOfIntegral = 0
    sign = 1
    for i in range(1, 21):
        valueOfIntegral += sign * (b**i - a**i) / factoral
        sign *= -1
        factoral *= i+1

    return valueOfIntegral * priorOfClassC2
```

که بعد از اجرای کد بالا میزان خطا هم در بالای نمودار نمایش داده می‌شود (خروجی قسمت قبل) و هم بر روی صفحه:

```python
plt.title('Bayes Error is: {}'.format(approximate_error(a=2, b=4, priorOfClassC2=0.5)) )
plt.show()


print("> Approximate Error is: ",
        approximate_error(a=2, b=4, priorOfClassC2=0.5))
```

خروجی اجرای قسمت بالا:

```
> Approximate Error is:  0.05850978579271444
```

:Computer Projects

بخش a)

تابع میانگین و کواریانس یک توزیع نرمال را می‌گیرد و n سمپل از آن بیرون می‌کشد:

```
########################################################################
# Computer exercise 7-a
########################################################################


def d_normal_distribution(mu, covMat, n):
    """
    D-dimensional normal distribution function,
       it draws samples from a D-dimensional normal distribution.
     :param
       (mu)mean vector,
      (covMat)covariance matrix,
      (n)number of samples that should be drawn.
     :return Output:
       each entry out[i,j,...,:] is an D-dimensional value drawn from normal distribution.
    An example of this function:
        d_normal_distribution(mu=[0,0],covMat=[[1, 0],[0, 100]], n=10)
    """
    import numpy as np
    # Use multivariate normal distribution of numpy package
    return np.random.multivariate_normal(mean=mu, cov=covMat, size=n, check_valid='ignore').tolist()
```

بخش b)

همین طور که در تصویر زیر مشاهده می‌کنید مرز تصمیم گیری برابر است با $x0 = 0$:

$$g_1 = -\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_1| + \ln(P(\omega_1))$$

$\mu_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $P(\omega_1) = 0.5$ Covariance matrix $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $d = 2$

$$g_1 = -\frac{1}{2}\left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) - \frac{2}{2}\ln 2\pi - \frac{1}{2} * 1 + \ln(0.5)$$

$$g_2 = -\frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma_2| + \ln P(\omega_2)$$

$\mu_2 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ $P(\omega_2) = 0.5$ Covariance matrix $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $d = 2$

$$g_2 = -\frac{1}{2}\left(x - \begin{bmatrix} -1 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} -1 \\ 0 \end{bmatrix}\right) - \frac{2}{2}\ln 2\pi - \frac{1}{2} * 1 + \ln(0.5)$$

$$g_1(x) = g_2(x) \Longrightarrow$$

$$-\cancel{\frac{1}{2}}\left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = -\cancel{\frac{1}{2}}\left(x - \begin{bmatrix} -1 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\left(x - \begin{bmatrix} -1 \\ 0 \end{bmatrix}\right)$$

$$(x_1-1)^2 + \cancel{(x_2-0)^2} = (x_1+1)^2 + \cancel{(x_2-0)^2}$$

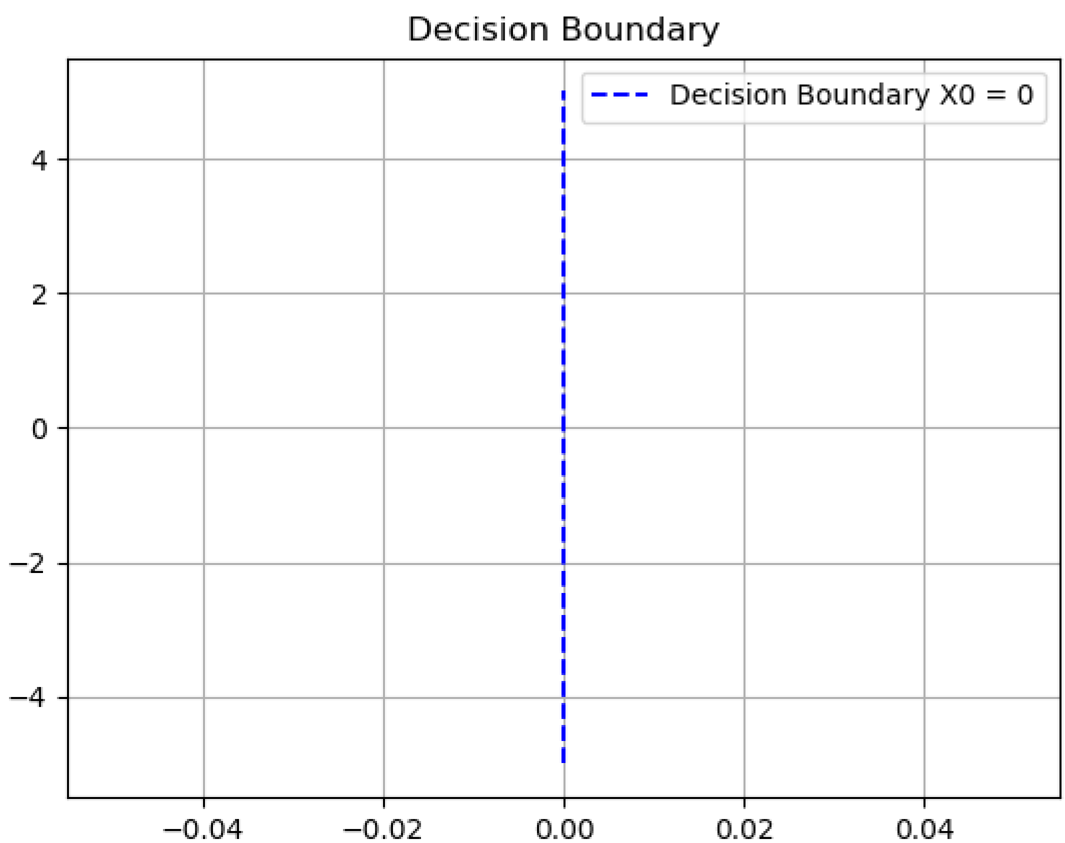
$$\rightarrow \cancel{x_1^2} - 2x_1 \cancel{+1} = \cancel{x_1^2} + 2x_1 \cancel{+1}$$

$$\rightarrow \boxed{x_1 = 0} \longrightarrow x = \begin{bmatrix} 0 \\ x_2 \end{bmatrix}$$

برای رسم مرز تصمیم کد زیر را نوشته ام:

```python
########################################################################
# Computer exercise 7-b
########################################################################

# Distributions of classes
xw1 = {'means': [1, 0], 'covriance': [[1, 0], [0, 1]], 'prior': 0.5, 'name': 'w1'}
xw2 = {'means': [-1, 0], 'covriance': [[1, 0], [0, 1]], 'prior': 0.5, 'name': 'w2'}

# As I mathematically have shown in report, decision boundary is x0 = 0.
# Plot decision boundary
y = np.linspace(-5, 5, 100)
plt.plot([0]*100 , y, 'b--', label='Decision Boundary X0 = 0')
plt.title("Decision Boundary")
plt.grid()
plt.legend()
plt.show()
```

خروجی اجرای کد بالا را در تصویر زیر مشاهده می‌کنید:



بخش c و dو e)

در این بخش به ازای تعداد داده‌های 100، 200، 300، ... و 1000 از هر دو توزیع نرمال داده تولید می‌کنیم و با استفاده از مرز تصمیم گیر آن‌ها را برچسب می‌دهیم و میزان خطا را اندازه می‌گیریم و در نهایت خطاها را رو نمودار هم با باند Bhttacharyya Bound نشان می‌دهیم.

به همین منظور کد زیر را نوشته ام(تابع Bhttacharyya را در تمرین های قبل توضیح داده ام):

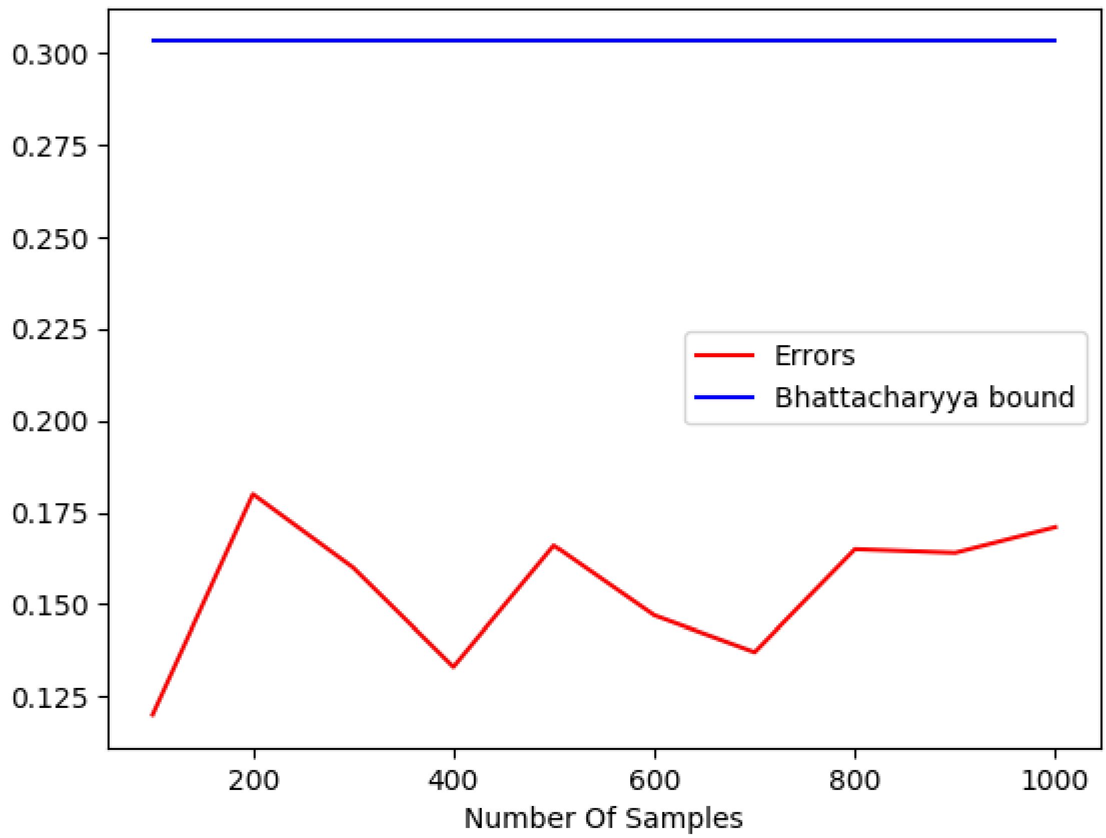
```
########################################################################
# Computer exercise 7- c, d
########################################################################
def bhattacharyya_bound(normalDist1, normalDist2):
    """
    This measure shows similarities between two
    distribution.
    :param normalDist1: is dictionary that contains
            these keys {'means','covariance', 'name','prior'}
    :param normalDist2: is dictionary that contains
            these keys {'means','covariance', 'name','prior'}
    :return: return a scalar
    """
    import numpy as np

    # Convert data to proper form
    mean1 = np.asmatrix(normalDist1['means'])
    mean2 = np.asmatrix(normalDist2['means'])
    cov1 = np.asmatrix(normalDist1['covriance'])
    cov2 = np.asmatrix(normalDist2['covriance'])

    # Average of two covariances
    covAve = (cov1 + cov2) / 2

    # Subtract of means
```

```python
    meanSub = mean2 - mean1

    # Calculate the bound
    bound = (np.sqrt(normalDist1['prior'] * normalDist2['prior'])) * (
            np.exp(-1 * ((1/8) * (meanSub * np.linalg.inv(covAve) * np.transpose(meanSub)) +
                          (1/2) * np.log(
                              np.linalg.det(covAve)/np.sqrt(np.linalg.det(cov1) *
                                                            np.linalg.det(cov2))
                                     ))))
    #return bound
    return bound.tolist()[0][0]


# Print Bhattacharyya bound
bhb = bhattacharyya_bound(xw1, xw2)
print('Bhattacharyya bound for w1 & w2:{:f}'.format(bhb))

# For plotting Errors by increasing number of samples
errors = []

# Generate samples for both distribution and classify them
# and calculate empirical error of them
# Do it for different number of samples 100, 200, 300, ..., 1000
startNumOfSamples = 50
```

```python
startNumOfSamples = 50
endNumOfSamples = 501
stepOfSample = 50
for numOfSamples in np.arange(startNumOfSamples, endNumOfSamples, stepOfSample):
    # Draw samples from distribution 1
    w1 = d_normal_distribution(mu=xw1['means'], covMat=xw1['covriance'],
                               n=numOfSamples)

    # Draw samples from distribution 2
    w2 = d_normal_distribution(mu=xw2['means'], covMat=xw2['covriance'],
                               n=numOfSamples)

    error = 0
    for samples in w1:
        # Decision Boundary: x0 = 0
        if(samples[0] < 0):
            error += 1

    for samples in w2:
        # Decision Boundary: x0 = 0
        if (samples[0] > 0):
            error += 1

    print("Number of sample {}, Error is {}".format(numOfSamples*2, round(error/(len(w1)+len(w2)),3)))
```

```python

    # Add error to the list of error
    errors.append(round(error/(len(w1)+len(w2)),3))

plt.plot(np.arange(startNumOfSamples*2,
                   endNumOfSamples*2-1,
                   stepOfSample*2).tolist(), errors, 'r-', label="Errors")

plt.plot(np.arange(startNumOfSamples*2,
                   endNumOfSamples*2-1,
                   stepOfSample*2).tolist(), [bhb]*len(errors),'b-', label="Bhattacharyya bound")
plt.legend()
plt.xlabel("Number Of Samples")
plt.show()
```

بعد از اجرای کد بالا تصویر خروجی زیر ایجاد می‌شود:



همین طور که مشاهده می‌شود مقدار خطا از باند خطا کمتر است و از آن بیشتر نمی شود.