UNIVERSIDADE CATÓLICA DE PELOTAS
Mestrado em Engenharia Eletrônica e Computação - PGEEC

LISTA DE EXERCÍCIOS 2

PEDRO A. TAVARES

DISCIPLINA: Reconhecimento de Padrões
PROFESSOR: Jose C. M. Bermudez

Pelotas, novembro de 2019

EXERCÍCIO 1

CÓDIGO EM PYTHON:

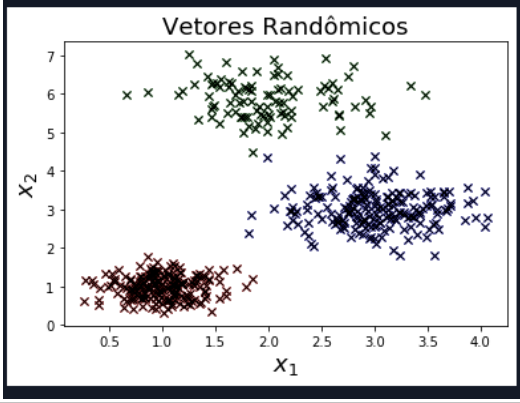
```
# -*- coding: utf-8 -*-
"""
Lista 2
Exercício 1
"""

import numpy as np
import matplotlib.pyplot as plt

from sklearn.mixture import GaussianMixture
from matplotlib.patches import Ellipse

"""
FUNÇÕES GERAIS
"""

## Função para gerar a eplipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
    #Plota a elipse
    ax = ax or plt.gca()

    #Converte a covariância nos eixos principais
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    #Gera a elipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

## Função que plota a elipse/círculo onde o EM classificou as PDFs.
## Quanto mais próximo da média, mais escura será a área.
def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)

    ax.axis('equal')
    w_factor = 0.2 / gmm.weights_.max()

    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)

"""
ITENS A) e B)
"""
print("ITENS A) e B):")

i = [[1, 0], [0, 1]]

mi1 = [1, 1]
mi2 = [3, 3]
mi3 = [2, 6]

s1=[[0.1, 0], [0, 0.1]]
s2=[[0.2, 0], [0, 0.2]]
s3=[[0.3, 0], [0, 0.3]]

p1=0.4
p2=0.4
p3=0.2

xp= np.zeros((2,500), dtype=float)

## Três distribuições Gaussianas:
x1 = (np.random.multivariate_normal(mi1, s1, 200).T);
```

```
x2 = (np.random.multivariate_normal(mi2, s2, 200).T);
x3 = (np.random.multivariate_normal(mi3, s3, 100).T);

## 500 amostras de treinamento:
xp[0,range(0,200)]=x1[0, :]
xp[1,range(0,200)]=x1[1, :]
xp[0,range(200,400)]=x2[0, :]
xp[1,range(200,400)]=x2[1, :]
xp[0,range(400,500)]=x3[0, :]
xp[1,range(400,500)]=x3[1, :]

## Geração dos gráficos para as 3 pdfs (vermelha, azul e verde respectivamente).
## O agrupamento de dados é plotado em preto para comparação.
## Percebe-se que as nuvens seguem os dados passados de média, covariância e
## probabilidade.
plt.figure()
plt.plot(x1[0,:], x1[1,:], 'x', color='red')
plt.plot(x2[0,:], x2[1,:], 'x', color='blue')
plt.plot(x3[0,:], x3[1,:], 'x', color='green')
plt.plot(xp[0,:], xp[1,:], 'x', color='black')
plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()

"""
ITEM C)
"""
print("")
print("ITEM C):")

## Função para determinar as estimativas utilizando o algoritmo EM
def MisturaGaussiana(x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose([x[0, :], x[1, :]]);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    plot_gmm(estimador, d);
    plt.show();
    return estimador.means_;

m_init1 = np.array([[0.,2.],[5.,2.],[5.,5.]])
prec_init1 = np.array([(1./0.15)*np.eye(2),(1./0.27)*np.eye(2),(1./0.4)*np.eye(2)])
w_init1 = np.array([1/3, 1/3, 1/3])

m_init2 = np.array([[1.6,1.4],[1.4,1.6],[1.3,1.5]])
prec_init2 = np.array([(1./0.2)*np.eye(2),(1./0.4)*np.eye(2),(1./0.3)*np.eye(2)])
w_init2 = np.array([0.2, 0.4, 0.4])

m_init3 = np.array([[1.6,1.4],[1.4,1.6]])
prec_init3 = np.array([(1./0.2)*np.eye(2),(1./0.4)*np.eye(2)])
w_init3 = np.array([1/2, 1/2])

a = MisturaGaussiana(xp, m_init1, prec_init1, w_init1, 3, 500);
plt.show();

f = MisturaGaussiana(xp, m_init2, prec_init2, w_init2, 3, 500);
plt.show();

g = MisturaGaussiana(xp, m_init3, prec_init3, w_init3, 2, 500);
plt.show();

"""
ITEM D)
"""
print("")
print("ITEM D):")

# As médias das distribuições são plotados em preto.
# As Figuras descrevem os casos 1, 2, e 3 respectivamente

plt.figure()
plt.scatter(x1[0,:], x1[1,:], c='y', alpha=0.5)
plt.scatter(x2[0,:], x2[1,:], c='m', alpha=0.5)
plt.scatter(x3[0,:], x3[1,:], c='c', alpha=0.5)
plt.scatter(a[:, 0], a[:, 1], c='k', alpha=1)
plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)
```

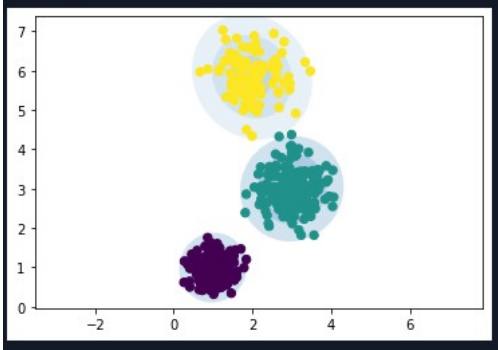
```
plt.show()
plt.figure()

plt.scatter(x1[0,:], x1[1,:], c='y', alpha=0.5)
plt.scatter(x2[0,:], x2[1,:], c='m', alpha=0.5)
plt.scatter(x3[0,:], x3[1,:], c='c', alpha=0.5)
plt.scatter(f[:, 0], f[:, 1], c='k', alpha=1)

plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()

plt.figure()

plt.scatter(x1[0,:], x1[1,:], c='y', alpha=0.5)
plt.scatter(x2[0,:], x2[1,:], c='m', alpha=0.5)
plt.scatter(x3[0,:], x3[1,:], c='c', alpha=0.5)
plt.scatter(g[:, 0], g[:, 1], c='k', alpha=1)

plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()
```

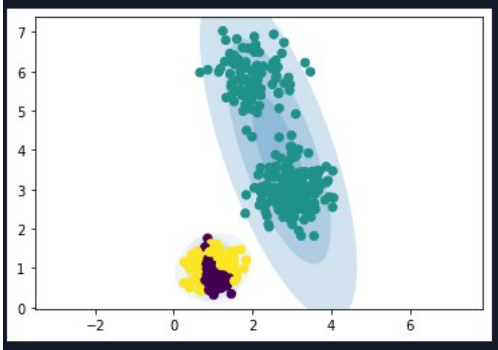
SAÍDA NO CONSOLE DO SPYDER:

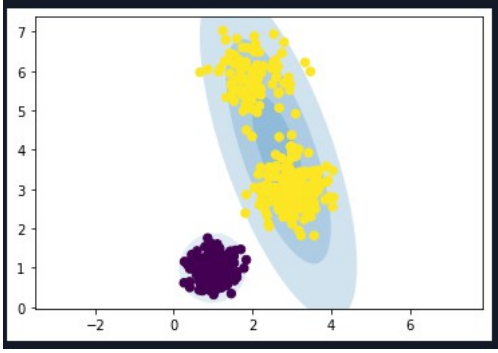
```
In [3]: runfile('C:/Users/Pedro/Desktop/Reconhecimento de Padrões/Lista2/exercicio1.py', wdir='C:/Users/Pedro/Desktop/
Reconhecimento de Padrões/Lista2')
ITENS A) e B):
```



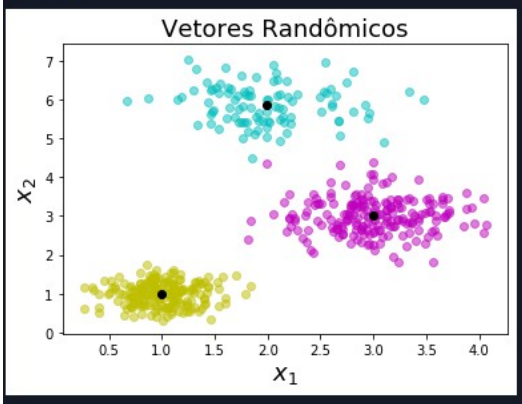
ITEM C):

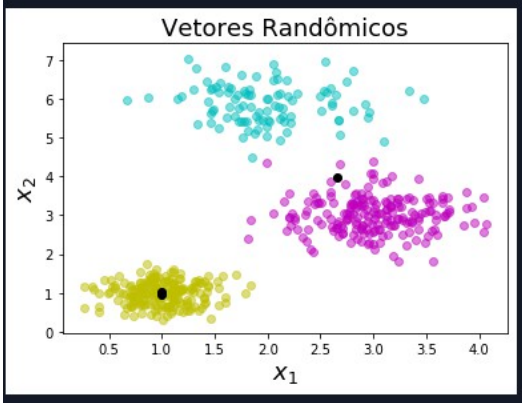


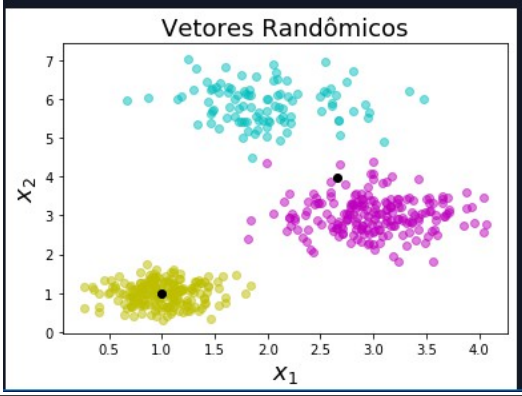




ITEM D):


Vetores Randômicos
x_2
x_1


Vetores Randômicos
x_2
x_1


Vetores Randômicos
x_2
x_1

EXERCÍCIO 2 (itens A ao D)

CÓDIGO EM PYTHON:

```
# -*- coding: utf-8 -*-
"""
Lista 2
Exercício 2
"""
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
import numpy as np

from sklearn.mixture import GaussianMixture
from matplotlib.patches import Ellipse
from scipy.stats import multivariate_normal
from sklearn.metrics import accuracy_score

## Função para gerar a eplipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
    #Plota a elipse
    ax = ax or plt.gca()

    #Converte a covariância nos eixos principais
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    #Gera a elipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

## Função que plota a elipse/círculo onde o EM classificou as PDFs.
## Quanto mais próximo da média, mais escura será a área.
def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)

    ax.axis('equal')
    w_factor = 0.2 / gmm.weights_.max()

    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)

"""
ITEM A)
"""
print("ITEM A):")

mi11 = [1.25, 1.25]
mi21 = [2.75, 2.75]
mi31 = [2, 6]

s11=[[0.1, 0], [0, 0.1]]
s21=[[0.2, 0], [0, 0.2]]
s31=[[0.3, 0], [0, 0.3]]

p11=0.4
p21=0.4
p31=0.2

mi12 = [1.25, 2.75]
mi22 = [2.75, 1.25]
mi32 = [4, 6]

s12=[[0.1, 0], [0, 0.1]]
s22=[[0.2, 0], [0, 0.2]]
s32=[[0.3, 0], [0, 0.3]]
```

```
p12=0.2
p22=0.3
p32=0.5

xp1= np.zeros((2,500), dtype=float)
xp2= np.zeros((2,500), dtype=float)
xp_total= np.zeros((2,1000), dtype=float)


#Geração de duas pdfs:
x11 = (np.random.multivariate_normal(mi11, s11, 200).T)
x21 = (np.random.multivariate_normal(mi21, s21, 200).T)
x31 = (np.random.multivariate_normal(mi31, s31, 100).T)
x12 = (np.random.multivariate_normal(mi12, s12, 100).T)
x22 = (np.random.multivariate_normal(mi22, s22, 150).T)
x32 = (np.random.multivariate_normal(mi32, s32, 250).T)

#Agrupamento dos primeiros 500 dados para cada classe:
xp1[0,range(0,200)]=x11[0, :]
xp1[1,range(0,200)]=x11[1, :]
xp1[0,range(200,400)]=x21[0, :]
xp1[1,range(200,400)]=x21[1, :]
xp1[0,range(400,500)]=x31[0, :]
xp1[1,range(400,500)]=x31[1, :]

xp2[0,range(0,100)]=x12[0, :]
xp2[1,range(0,100)]=x12[1, :]
xp2[0,range(100,250)]=x22[0, :]
xp2[1,range(100,250)]=x22[1, :]
xp2[0,range(250,500)]=x32[0, :]
xp2[1,range(250,500)]=x32[1, :]

#Vetor que contém os dois vetores provenientes da classe 1 e 2:
xp_total[0,range(0,500)]=xp1[0, :]
xp_total[1,range(0,500)]=xp1[1, :]
xp_total[0,range(500,1000)]=xp2[0, :]
xp_total[1,range(500,1000)]=xp2[1, :]

#Geração do gráfico das classes 1('x') e 2('o').
#Os dados não geram gaussianas. Mais à frente tentaremos gerar duas gaussianas
#uma para cada classe, para uma melhor classificação através do algoritmo.
plt.figure()
plt.plot(xp1[0,:], xp1[1,:], 'x', color='red')
plt.plot(xp2[0,:], xp2[1,:], '.', color='blue')

plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()

## Função para determinar as estimativas
def MisturaGaussiana(x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose(x);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    plot_gmm(estimador, d);
    plt.show();

#Função Estimador_PeQ, P e Q devido a função de densidade linear de Bayes.
#Nesta função apenas estima-se os dados com o EM, pega-se os valores de média, covariância e
#probabilidade, cria-se as 3 pdfs estimadas e então as somamos.
def Estimador_PeQ(x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose(x);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    media = estimador.means_;
    covariancia = estimador.covariances_;

    x11 = (multivariate_normal(media[0, :], covariancia[0,:]));
    x21 = (multivariate_normal(media[1, :], covariancia[1,:]));
    x31 = (multivariate_normal(media[2, :], covariancia[2,:]));

    pdf1 = x11.pdf(d);
    pdf2 = x21.pdf(d);
    pdf3 = x31.pdf(d);

    px = pdf1 + pdf2 + pdf3;
```

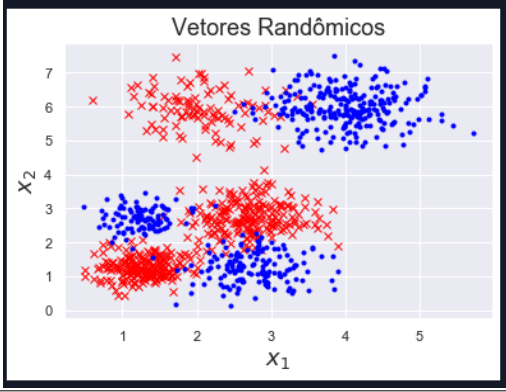
```
    return px;

"""
ITEM B)
"""
print("")
print("ITEM B):")
print("")
print("Treinamento EM:")
#Mesma função usada anteriormente(MisturaGaussiana),porém com dados diferentes
#iniciais de média, covariância e probabilidade, para estimação.
m_init1 = np.array([[0.,2.],[5.,2.],[5.,5.]])
prec_init1 = np.array([(1./0.15)*np.eye(2),(1./0.27)*np.eye(2),(1./0.4)*np.eye(2)])
w_init1 = np.array([1/3, 1/3, 1/3])

m_init2 = np.array([[5.,2.],[3.,4.],[2.,5.]])
prec_init2 = np.array([(1./0.15)*np.eye(2),(1./0.27)*np.eye(2),(1./0.35)*np.eye(2)])
w_init2 = np.array([1/3, 1/3, 1/3])

b = MisturaGaussiana(xp1, m_init1, prec_init1, w_init1, 3, 20);
f = MisturaGaussiana(xp2, m_init2, prec_init2, w_init2, 3, 20);

plt.show();

print("")
print("Bayes:")
#Geração de 3 pdfs estimadas a partir dos dados provenientes do treinamento
#com o algoritmo EM:
auxpx  = Estimador_PeQ(xp_total, m_init1, prec_init1, w_init1, 3, 1000);
auxpx1 = Estimador_PeQ(xp_total, m_init2, prec_init2, w_init2, 3, 1000);

teste = (auxpx/auxpx1)>1;
teste = teste * 1;

labels = np.ones ((1, 1000), dtype=int)
labels1= np.ones ((1, 500), dtype=int)
labels2= np.zeros((1, 500), dtype=int)
labels[0, range(0,500) ]=labels1
labels[0, range(500,1000)]=labels2
labels = np.transpose(labels);
labels1 = np.transpose(labels1);
labels2 = np.transpose(labels2);

#Conta-se os erros e usa-se a variável de bayes estimada para colorar
#de acordo com a classificação.
print("Acurácia b= ", accuracy_score(teste, labels))

prob = ((teste-labels));
count = 0;

for i in range(0, 1000):
    count = count + (teste[i]-labels[i,0])**2

probal= (count/1000);

print("Probabilidade de erro b = ", (probal*100))

plt.scatter(xp_total[0, :], xp_total[1, :], c=teste, cmap='RdBu', alpha=0.3)
plt.show();

"""
ITEM C)
"""
print("")
print("ITEM C):")

zp1= np.zeros((2,500), dtype=float)
zp2= np.zeros((2,500), dtype=float)
zp_total= np.zeros((2,1000), dtype=float)

z11 = (np.random.multivariate_normal(mi11, s11, 200).T)
z21 = (np.random.multivariate_normal(mi21, s21, 200).T)
z31 = (np.random.multivariate_normal(mi31, s31, 100).T)
z12 = (np.random.multivariate_normal(mi12, s12, 100).T)
z22 = (np.random.multivariate_normal(mi22, s22, 150).T)
z32 = (np.random.multivariate_normal(mi32, s32, 250).T)

zp1[0,range(0,200)]=z11[0, :]
zp1[1,range(0,200)]=z11[1, :]
zp1[0,range(200,400)]=z21[0, :]
```

```
zp1[1,range(200,400)]=z21[1, :]
zp1[0,range(400,500)]=z31[0, :]
zp1[1,range(400,500)]=z31[1, :]

zp2[0,range(0,100)]=z12[0, :]
zp2[1,range(0,100)]=z12[1, :]
zp2[0,range(100,250)]=z22[0, :]
zp2[1,range(100,250)]=z22[1, :]
zp2[0,range(250,500)]=z32[0, :]
zp2[1,range(250,500)]=z32[1, :]

zp_total[0,range(0,500)]=zp1[0, :]
zp_total[1,range(0,500)]=zp1[1, :]
zp_total[0,range(500,1000)]=zp2[0, :]
zp_total[1,range(500,1000)]=zp2[1, :]

plt.figure()

plt.plot(zp1[0,:], zp1[1,:], 'x', color='green')
plt.plot(zp2[0,:], zp2[1,:], '.', color='black')

plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()

"""
ITEM D)
"""
print("")
print("ITEM D):")

#Função para mostrar a classificação de Bayes
def Classifica(z, x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose(x);
    z = np.transpose(z);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    media = estimador.means_;
    covariancia = estimador.covariances_;

    x11 = (multivariate_normal(media[0, :], covariancia[0,:]));
    x21 = (multivariate_normal(media[1, :], covariancia[1,:]));
    x31 = (multivariate_normal(media[2, :], covariancia[2,:]));

    pdf1 = x11.pdf(z);
    pdf2 = x21.pdf(z);
    pdf3 = x31.pdf(z);

    px = pdf1 + pdf2 + pdf3;

    return px;

#O percentual de erro é maior devido aos dados não terem sido treinados com
#este vetor. No entanto este erro não é tão abrupto pois o vetor Z foi criado
#aleatoriamente, mas com os mesmos dados iniciais “reais” usados para a criação
#do vetor X:
auxpx  = Classifica(zp_total, xp_total, m_init1, prec_init1, w_init1, 3, 1000);
auxpx1 = Classifica(zp_total, xp_total, m_init2, prec_init2, w_init2, 3, 1000);

teste = (auxpx/auxpx1)>1;
teste = teste * 1;

labels = np.ones ((1, 1000), dtype=int)
labels1= np.ones ((1, 500), dtype=int)
labels2= np.zeros((1, 500), dtype=int)
labels[0, range(0,500) ]=labels1
labels[0, range(500,1000)]=labels2
labels = np.transpose(labels);
labels1 = np.transpose(labels1);
labels2 = np.transpose(labels2);

print("Acurácia b = ", accuracy_score(teste, labels))

prob = ((teste-labels));
count = 0;
for i in range(0, 1000):
    count = count + (teste[i]-labels[i,0])**2
```

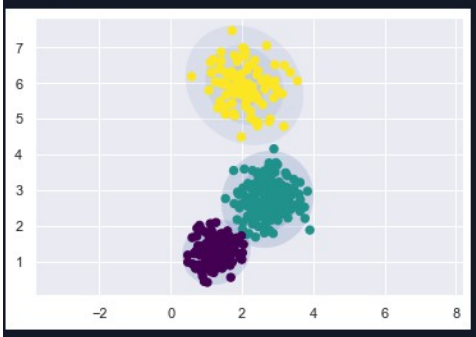
```
probal= (count/1000);

print("Probabilidade de erro b = ", (probal*100))

plt.scatter(zp_total[0, :], zp_total[1, :], c=teste, cmap='RdBu', alpha=0.3)

plt.show();
```

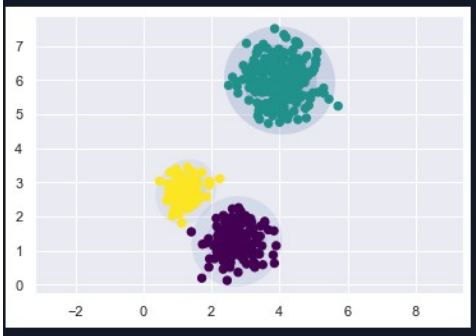
SAÍDA NO CONSOLE DO SPYDER:

```
In [13]: runfile('C:/Users/Pedro/Desktop/Reconhecimento de Padrões/Lista2/exercício2_ABCD.py', wdir='C:/Users/Pedro/Desktop/
Reconhecimento de Padrões/Lista2')
ITEM A):
```

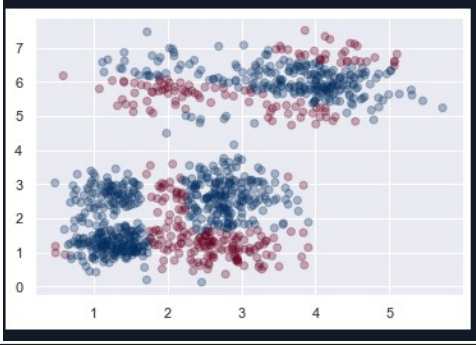


```
ITEM B):

Treinamento EM:
```

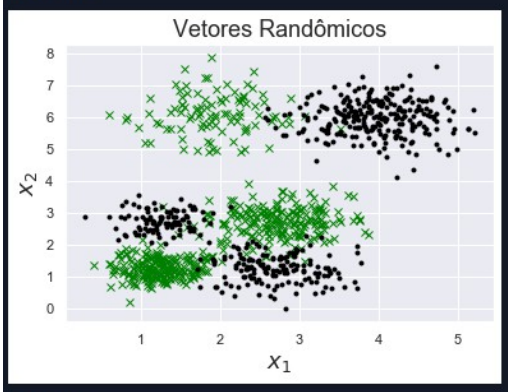




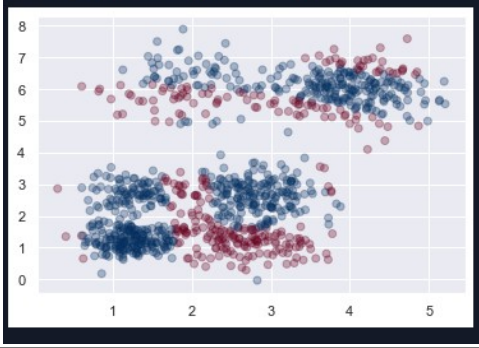

```
Bayes:
Acurácia b=  0.588
Probabilidade de erro b =  41.199999999999996
```



```
ITEM C):
```




```
ITEM D):
Acurácia b =  0.615
Probabilidade de erro b =  38.5
```



EXERCÍCIO 2 (item E)

CÓDIGO EM PYTHON:

```python
# -*- coding: utf-8 -*-
"""
Lista 2
Exercício 2 (item E)

Foi criado um novo conjunto de vetores, que estão mais separados entre si,
fazendo com que o o algoritmo EM (Expectation-Maximization) consiga obter
melhores resultados de estimação.

Tem-se os novos dados (vetor Z) criados com treinamento a partir do vetor X.
O erro de classificação, apesar de ser maior do que do vetor X pois a rede não
foi treinada com o vetor Z, não é significativamente maior. Isso porque, apesar
dos dados serem randômicos, possuem os mesmos dados iniciais “reais” usados
para a criação do vetor X.

Os itens A a D foram plotados novamente para comparação.
"""
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
import numpy as np

from sklearn.mixture import GaussianMixture
from matplotlib.patches import Ellipse
from scipy.stats import multivariate_normal
from sklearn.metrics import accuracy_score

## Função para gerar a eplipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
    #Plota a elipse
    ax = ax or plt.gca()

    #Converte a covariância nos eixos principais
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
```

```
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    #Gera a elipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

## Função que plota a elipse/círculo onde o EM classificou as PDFs.
## Quanto mais próximo da média, mais escura será a área.
def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)

    ax.axis('equal')
    w_factor = 0.2 / gmm.weights_.max()

    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)

"""
ITEM A)
"""
print("ITEM A):")

mi11 = [1.25, 1.25]
mi21 = [1.75, 1.75]
mi31 = [2, 2]

s11=[[0.1, 0], [0, 0.1]]
s21=[[0.2, 0], [0, 0.2]]
s31=[[0.3, 0], [0, 0.3]]

p11=0.4
p21=0.4
p31=0.2

mi12 = [5.0, 2.75]
mi22 = [4.75, 1.25]
mi32 = [6, 2]

s12=[[0.1, 0], [0, 0.1]]
s22=[[0.2, 0], [0, 0.2]]
s32=[[0.3, 0], [0, 0.3]]

p12=0.2
p22=0.3
p32=0.5

xp1= np.zeros((2,500), dtype=float)
xp2= np.zeros((2,500), dtype=float)
xp_total= np.zeros((2,1000), dtype=float)


#Geração de duas pdfs:
x11 = (np.random.multivariate_normal(mi11, s11, 200).T)
x21 = (np.random.multivariate_normal(mi21, s21, 200).T)
x31 = (np.random.multivariate_normal(mi31, s31, 100).T)
x12 = (np.random.multivariate_normal(mi12, s12, 100).T)
x22 = (np.random.multivariate_normal(mi22, s22, 150).T)
x32 = (np.random.multivariate_normal(mi32, s32, 250).T)

#Agrupamento dos primeiros 500 dados para cada classe:
xp1[0,range(0,200)]=x11[0, :]
xp1[1,range(0,200)]=x11[1, :]
xp1[0,range(200,400)]=x21[0, :]
xp1[1,range(200,400)]=x21[1, :]
xp1[0,range(400,500)]=x31[0, :]
xp1[1,range(400,500)]=x31[1, :]

xp2[0,range(0,100)]=x12[0, :]
xp2[1,range(0,100)]=x12[1, :]
xp2[0,range(100,250)]=x22[0, :]
xp2[1,range(100,250)]=x22[1, :]
```

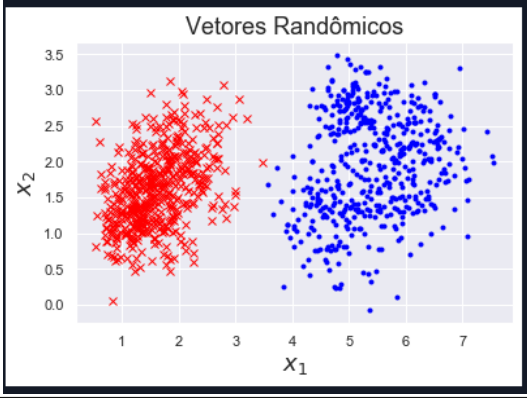
```
xp2[0,range(250,500)]=x32[0, :]
xp2[1,range(250,500)]=x32[1, :]

#Vetor que contém os dois vetores provenientes da classe 1 e 2:
xp_total[0,range(0,500)]=xp1[0, :]
xp_total[1,range(0,500)]=xp1[1, :]
xp_total[0,range(500,1000)]=xp2[0, :]
xp_total[1,range(500,1000)]=xp2[1, :]

#Geração do gráfico das classes 1('x') e 2('o').
#Os dados não geram gaussianas. Mais à frente tentaremos gerar duas gaussianas
#uma para cada classe, para uma melhor classificação através do algoritmo.
plt.figure()
plt.plot(xp1[0,:], xp1[1,:], 'x', color='red')
plt.plot(xp2[0,:], xp2[1,:], '.', color='blue')

plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()

## Função para determinar as estimativas
def MisturaGaussiana(x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose(x);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    plot_gmm(estimador, d);
    plt.show();

#Função Estimador_PeQ, P e Q devido a função de densidade linear de Bayes.
#Nesta função apenas estima-se os dados com o EM, pega-se os valores de média, covariância e
#probabilidade, cria-se as 3 pdfs estimadas e então as somamos.
def Estimador_PeQ(x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose(x);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    media = estimador.means_;
    covariancia = estimador.covariances_;

    x11 = (multivariate_normal(media[0, :], covariancia[0,:]));
    x21 = (multivariate_normal(media[1, :], covariancia[1,:]));
    x31 = (multivariate_normal(media[2, :], covariancia[2,:]));

    pdf1 = x11.pdf(d);
    pdf2 = x21.pdf(d);
    pdf3 = x31.pdf(d);

    px = pdf1 + pdf2 + pdf3;
    return px;

"""
ITEM B)
"""
print("")
print("ITEM B):")
print("")
print("Treinamento EM:")
#Mesma função usada anteriormente(MisturaGaussiana),porém com dados diferentes
#iniciais de média, covariância e probabilidade, para estimação.
m_init1 = np.array([[0.,2.],[5.,2.],[5.,5.]])
prec_init1 = np.array([(1./0.15)*np.eye(2),(1./0.27)*np.eye(2),(1./0.4)*np.eye(2)])
w_init1 = np.array([1/3, 1/3, 1/3])

m_init2 = np.array([[5.,2.],[3.,4.],[2.,5.]])
prec_init2 = np.array([(1./0.15)*np.eye(2),(1./0.27)*np.eye(2),(1./0.35)*np.eye(2)])
w_init2 = np.array([1/3, 1/3, 1/3])

b = MisturaGaussiana(xp1, m_init1, prec_init1, w_init1, 3, 20);
f = MisturaGaussiana(xp2, m_init2, prec_init2, w_init2, 3, 20);

plt.show();

print("")
print("Bayes:")
#Geração de 3 pdfs estimadas a partir dos dados provenientes do treinamento
#com o algoritmo EM:
auxpx  = Estimador_PeQ(xp_total, m_init1, prec_init1, w_init1, 3, 1000);
```

```
auxpx1 = Estimador_PeQ(xp_total, m_init2, prec_init2, w_init2, 3, 1000);

teste = (auxpx/auxpx1)<1;
teste = teste * 1;

labels = np.ones ((1, 1000), dtype=int)
labels1= np.ones ((1, 500), dtype=int)
labels2= np.zeros((1, 500), dtype=int)
labels[0, range(0,500) ]=labels1
labels[0, range(500,1000)]=labels2
labels = np.transpose(labels);
labels1 = np.transpose(labels1);
labels2 = np.transpose(labels2);

#Conta-se os erros e usa-se a variável de bayes estimada para colorar
#de acordo com a classificação.
print("Acurácia b= ", accuracy_score(teste, labels))

prob = ((teste-labels));
count = 0;

for i in range(0, 1000):
    count = count + (teste[i]-labels[i,0])**2

probal= (count/1000);

print("Probabilidade de erro b = ", (probal*100))

plt.scatter(xp_total[0, :], xp_total[1, :], c=teste, cmap='RdBu', alpha=0.3)
plt.show();

"""
ITEM C)
"""
print("")
print("ITEM C):")

zp1= np.zeros((2,500), dtype=float)
zp2= np.zeros((2,500), dtype=float)
zp_total= np.zeros((2,1000), dtype=float)

z11 = (np.random.multivariate_normal(mi11, s11, 200).T)
z21 = (np.random.multivariate_normal(mi21, s21, 200).T)
z31 = (np.random.multivariate_normal(mi31, s31, 100).T)
z12 = (np.random.multivariate_normal(mi12, s12, 100).T)
z22 = (np.random.multivariate_normal(mi22, s22, 150).T)
z32 = (np.random.multivariate_normal(mi32, s32, 250).T)

zp1[0,range(0,200)]=z11[0, :]
zp1[1,range(0,200)]=z11[1, :]
zp1[0,range(200,400)]=z21[0, :]
zp1[1,range(200,400)]=z21[1, :]
zp1[0,range(400,500)]=z31[0, :]
zp1[1,range(400,500)]=z31[1, :]

zp2[0,range(0,100)]=z12[0, :]
zp2[1,range(0,100)]=z12[1, :]
zp2[0,range(100,250)]=z22[0, :]
zp2[1,range(100,250)]=z22[1, :]
zp2[0,range(250,500)]=z32[0, :]
zp2[1,range(250,500)]=z32[1, :]

zp_total[0,range(0,500)]=zp1[0, :]
zp_total[1,range(0,500)]=zp1[1, :]
zp_total[0,range(500,1000)]=zp2[0, :]
zp_total[1,range(500,1000)]=zp2[1, :]

plt.figure()

plt.plot(zp1[0,:], zp1[1,:], 'x', color='green')
plt.plot(zp2[0,:], zp2[1,:], '.', color='black')

plt.title('Vetores Randômicos', fontsize=18)
plt.xlabel(r'$x_1$', fontsize=18)
plt.ylabel(r'$x_2$', fontsize=18)

plt.show()
```

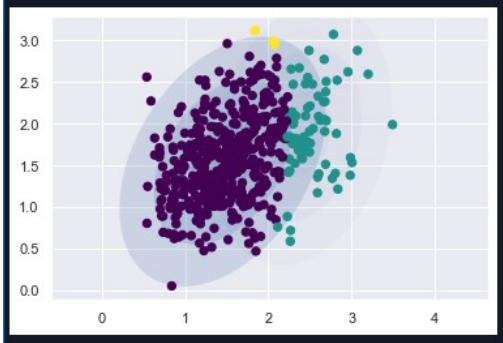
```
"""
ITEM D)
"""
print("")
print("ITEM D):")

#Função para mostrar a classificação de Bayes
def Classifica(z, x, m_init, prec_init, w_init, componentes, iteracoes):
    d = np.transpose(x);
    z = np.transpose(z);
    estimador = GaussianMixture(n_components=componentes, covariance_type='full',
max_iter=iteracoes, n_init=1, weights_init=w_init, means_init=m_init, precisions_init=prec_init)
    estimador.fit(d)
    media = estimador.means_;
    covariancia = estimador.covariances_;

    x11 = (multivariate_normal(media[0, :], covariancia[0,:]));
    x21 = (multivariate_normal(media[1, :], covariancia[1,:]));
    x31 = (multivariate_normal(media[2, :], covariancia[2,:]));

    pdf1 = x11.pdf(z);
    pdf2 = x21.pdf(z);
    pdf3 = x31.pdf(z);

    px = pdf1 + pdf2 + pdf3;

    return px;

#O percentual de erro é maior devido aos dados não terem sido treinados com
#este vetor. No entanto este erro não é tão abrupto pois o vetor Z foi criado
#aleatoriamente, mas com os mesmos dados iniciais “reais” usados para a criação
#do vetor X:
auxpx  = Classifica(zp_total, xp_total, m_init1, prec_init1, w_init1, 3, 1000);
auxpx1 = Classifica(zp_total, xp_total, m_init2, prec_init2, w_init2, 3, 1000);

teste = (auxpx/auxpx1)<1;
teste = teste * 1;

labels = np.ones ((1, 1000), dtype=int)
labels1= np.ones ((1, 500), dtype=int)
labels2= np.zeros((1, 500), dtype=int)
labels[0, range(0,500) ]=labels1
labels[0, range(500,1000)]=labels2
labels = np.transpose(labels);
labels1 = np.transpose(labels1);
labels2 = np.transpose(labels2);

print("Acurácia b = ", accuracy_score(teste, labels))

prob = ((teste-labels));
count = 0;

for i in range(0, 1000):
    count = count + (teste[i]-labels[i,0])**2

probal= (count/1000);

print("Probabilidade de erro b = ", (probal*100))

plt.scatter(zp_total[0, :], zp_total[1, :], c=teste, cmap='RdBu', alpha=0.3)

plt.show();
```

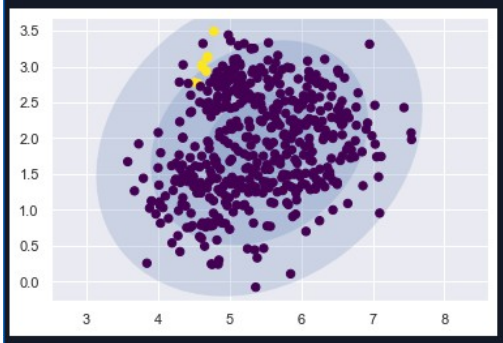
SAÍDA NO CONSOLE DO SPYDER:

```
In [14]: runfile('C:/Users/Pedro/Desktop/Reconhecimento de Padrões/Lista2/exercício2_E.py', wdir='C:/Users/Pedro/Desktop/
Reconhecimento de Padrões/Lista2')
ITEM A):
```

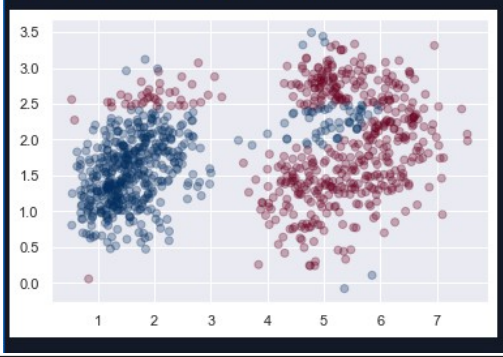



```
ITEM B):

Treinamento EM:
```

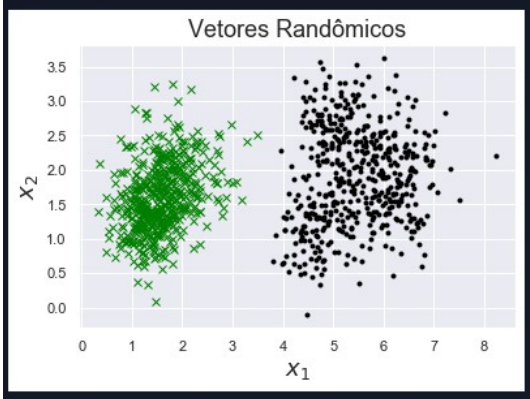




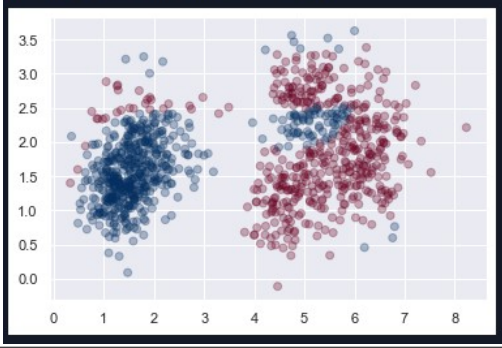
```
Bayes:
Acurácia b=  0.9
Probabilidade de erro b =  10.0
```



```
ITEM C):
```




```
ITEM D):
Acurácia b =  0.893
Probabilidade de erro b =  10.7
```



EXERCÍCIO 3

CÓDIGO EM PYTHON:

```python
# -*- coding: utf-8 -*-
"""
Lista 2
Exercício 3
"""

import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from scipy.stats import multivariate_normal


"""
ITEM A)
"""
print("ITEM A)\n")

m1 = [0,0,0,0,0];
m2 = [1,1,1,1,1];

s1 = [[0.8 , 0.2 , 0.1 , 0.05, 0.01],
      [0.2 , 0.7 , 0.1 , 0.03, 0.02],
      [0.1 , 0.1 , 0.8 , 0.02, 0.01],
      [0.05, 0.03, 0.02, 0.9 , 0.01],
      [0.01, 0.02, 0.01, 0.01, 0.8]];

s2 = [[0.9 , 0.1 , 0.05, 0.02, 0.01],
      [0.1 , 0.8 , 0.1 , 0.02, 0.02],
      [0.05, 0.1 , 0.7 , 0.02, 0.01],
      [0.02, 0.02, 0.02, 0.6 , 0.02],
      [0.01, 0.02, 0.01, 0.02, 0.7]];

#Geração de dados randômicos Gaussianos a partir dos dados iniciais de
#dimensão 5 (X1 com 50 e X2 com 1000 dados).
z11 = (np.random.multivariate_normal(m1, s1, 500).T);
z21 = (np.random.multivariate_normal(m2, s2, 500).T);
```

```
z1 = np.ones ((5, 1000), dtype=int)
z1[0,range(0,500)]=z11[0, :]
z1[1,range(0,500)]=z11[1, :]
z1[2,range(0,500)]=z11[2, :]
z1[3,range(0,500)]=z11[3, :]
z1[4,range(0,500)]=z11[4, :]
z1[0,range(500,1000)]=z21[0, :]
z1[1,range(500,1000)]=z21[1, :]
z1[2,range(500,1000)]=z21[2, :]
z1[3,range(500,1000)]=z21[3, :]
z1[4,range(500,1000)]=z21[4, :]

z1_test  = np.transpose(z1);

#Concatenação destes 25 e 500 dados de cada classe em um vetor de 50
#e 1000 unidades chamado de z e z1 respectivamente. Depois usou-se uma variável
#'labels' para classificação dos dados:

labels = np.ones ((1, 1000), dtype=int)
labels1= np.ones ((1, 500), dtype=int)
labels2= np.zeros((1, 500), dtype=int)
labels[0, range(0,500) ]=labels1
labels[0, range(500,1000)]=labels2
labels_z1 = np.transpose(labels);

z1 = (np.random.multivariate_normal(m1, s1, 25).T);
z2 = (np.random.multivariate_normal(m2, s2, 25).T);

z = np.ones ((5, 50), dtype=int)
z[0,range(0,25)]=z1[0, :]
z[1,range(0,25)]=z1[1, :]
z[2,range(0,25)]=z1[2, :]
z[3,range(0,25)]=z1[3, :]
z[4,range(0,25)]=z1[4, :]
z[0,range(25,50)]=z2[0, :]
z[1,range(25,50)]=z2[1, :]
z[2,range(25,50)]=z2[2, :]
z[3,range(25,50)]=z2[3, :]
z[4,range(25,50)]=z2[4, :]

z  = np.transpose(z);
z1 = np.transpose(z1);
z2 = np.transpose(z2);

labels = np.ones ((1, 50), dtype=int)
labels1= np.ones ((1, 25), dtype=int)
labels2= np.zeros((1, 25), dtype=int)
labels[0, range(0,25) ]=labels1
labels[0, range(25,50)]=labels2

labels = np.transpose(labels);
labels1 = np.transpose(labels1);
labels2 = np.transpose(labels2);

#Classificadores ingênuos, para os 50 e 1000 dados:
gnb1 = GaussianNB()
model1 = gnb1.fit(z1, labels1)
prob1  = gnb1.class_prior_;
media1 = gnb1.theta_;
media1 = np.transpose(media1)
cov1   = gnb1.sigma_;
cov1 = np.transpose(cov1)

gnb2 = GaussianNB()
model2 = gnb1.fit(z2, labels2)
prob2  = gnb1.class_prior_;
media2 = gnb1.theta_;
media2 = np.transpose(media2)
cov2   = gnb1.sigma_;
cov2 = np.transpose(cov2)

gnb = GaussianNB()
model = gnb.fit(z, labels)
prob  = gnb.class_prior_;
media = gnb.theta_;
media = np.transpose(media)
cov   = gnb1.sigma_;
cov = np.transpose(cov)
```

```
pred  = gnb.predict(z1_test)

print("Acurácia A = ", accuracy_score(pred, labels_z1))

prob = ((pred-labels_z1))**2;
count = 0;

for i in range(0, 1000):
    count = count + (pred[i]-labels_z1[i,0])**2

probal= (count/1000);

print("Count A = ",(count))
print("Probabilidade de erro A = ", (probal*100),"\n")


"""
ITEM B)
"""
print("ITEM B)\n")

#Novo vetor para ser testado na classificação.
#O classificador NB apresenta uma significativa melhora à medida que os dados
#de treinamento aumentam.
ecov1 = np.array([(cov1)*np.eye(5, 5)])
ez1 = (np.random.multivariate_normal(media1[:, 0], ecov1[0, :, :], 500).T);
ecov2 = np.array([(cov2)*np.eye(5, 5)])
ez2 = (np.random.multivariate_normal(media2[:, 0], ecov2[0, :, :], 500).T);
ez = np.ones ((5, 1000), dtype=int)

ez[0,range(0,500)]=ez1[0, :]
ez[1,range(0,500)]=ez1[1, :]
ez[2,range(0,500)]=ez1[2, :]
ez[3,range(0,500)]=ez1[3, :]
ez[4,range(0,500)]=ez1[4, :]
ez[0,range(500,1000)]=ez2[0, :]
ez[1,range(500,1000)]=ez2[1, :]
ez[2,range(500,1000)]=ez2[2, :]
ez[3,range(500,1000)]=ez2[3, :]
ez[4,range(500,1000)]=ez2[4, :]
ez = np.transpose(ez);

labels = np.ones ((1, 1000), dtype=int)
labels1= np.ones ((1, 500), dtype=int)
labels2= np.zeros((1, 500), dtype=int)
labels[0, range(0,500) ]=labels1
labels[0, range(500,1000)]=labels2

labels = np.transpose(labels);
labels1 = np.transpose(labels1);
labels2 = np.transpose(labels2);

gnb = GaussianNB()
model = gnb.fit(ez, labels)
pred  = gnb.predict(z1_test)

print("Acurácia B = ", accuracy_score(pred, labels_z1))

prob = ((pred-labels_z1));
count = 0;

for i in range(0, 1000):
    count = count + (pred[i]-labels_z1[i,0])**2

probal= (count/1000);

print("Count B = ",(count))
print("Probabilidade de erro B = ", (probal*100),"\n")

"""
ITEM C)
"""
print("ITEM C)\n")

def FuncErro(w1, w2, iteracoes):
    count1 = 0;
    count2 = 0;
    for i in range(0, iteracoes):
        count1 = count1 + w1[i];
        count2 = count2 + w2[i];
```

```
    probal= ((count1+count2)/(1000))*100;
    return probal

z1 = np.transpose(z1);
z2 = np.transpose(z2);

#Exemplo para N=20:
#Onde para 20 e 25 amostras o erro se situa em 0,3% e 0,6 %. Este erro não é
#tão elevado pois os dados foram treinados a partir das mesmas 25 amostras usadas
#com o algoritmo Naive Bayes.
#Agora quando temos 1000 dados para serem classificados por um conjunto treinado
#com 25, a probabilidade de erro aumenta significantemente para 14,5%.
mean1 = np.mean(z1);
mean1 = (mean1)*np.ones ((1, 5), dtype=int)
mean2 = np.mean(z2);
mean2 = (mean2)*np.ones ((1, 5), dtype=int)

cov1  = np.cov(z1);
cov2  = np.cov(z2);

a1 = (np.random.multivariate_normal(mean1[0, :], cov1, 20).T);
b1 = (np.random.multivariate_normal(mean2[0, :], cov2, 20).T);
a1=np.transpose(a1);
b1=np.transpose(b1);

gaussiana1 =  multivariate_normal(mean1[0, :], cov1)

x_11 = gaussiana1.pdf(a1)
x_12 = gaussiana1.pdf(b1)

gaussiana2 =  multivariate_normal(mean2[0, :], cov2)

x_21 = gaussiana2.pdf(a1)
x_22 = gaussiana2.pdf(b1)

aux1 = (x_11 > x_12);
aux1 = aux1*1;
aux2 = (x_21 < x_22);
aux2 = aux2*1;

teste1 = (1-aux1)
teste2 = (1-aux2)

prob1 = FuncErro(teste1, teste2, 20);

print("Probabilidade de erro de 20 amostras", prob1)

a2 = (np.random.multivariate_normal(mean1[0, :], cov1, 25).T);
b2 = (np.random.multivariate_normal(mean2[0, :], cov2, 25).T);

a2=np.transpose(a2);
b2=np.transpose(b2);

gaussiana1 =  multivariate_normal(mean1[0, :], cov1)

x_11 = gaussiana1.pdf(a2)
x_12 = gaussiana1.pdf(b2)

gaussiana2 =  multivariate_normal(mean2[0, :], cov2)

x_21 = gaussiana2.pdf(a2)
x_22 = gaussiana2.pdf(b2)

aux1 = (x_11 > x_12);
aux1 = aux1*1;

aux2 = (x_21 < x_22);
aux2 = aux2*1;

teste1 = (1-aux1)
teste2 = (1-aux2)

prob1 = FuncErro(teste1, teste2, 25);

print("Probabilidade de erro de 25 amostras", prob1)

a3 = (np.random.multivariate_normal(mean1[0, :], cov1, 500).T);
b3 = (np.random.multivariate_normal(mean2[0, :], cov2, 500).T);
a3=np.transpose(a3);
b3=np.transpose(b3);
```

```
gaussiana1 =  multivariate_normal(mean1[0, :], cov1)

x_11 = gaussiana1.pdf(a3)
x_12 = gaussiana1.pdf(b3)

gaussiana2 =  multivariate_normal(mean2[0, :], cov2)

x_21 = gaussiana2.pdf(a3)
x_22 = gaussiana2.pdf(b3)

aux1 = (x_11 > x_12);
aux1 = aux1*1;
aux2 = (x_21 < x_22);
aux2 = aux2*1;

teste1 = (1-aux1)
teste2 = (1-aux2)

prob1 = FuncErro(teste1, teste2, 500);

print("Probabilidade de erro de 500 amostras", prob1,"\n")
```

SAÍDA NO CONSOLE DO SPYDER:

```
In [23]: runfile('C:/Users/Pedro/Desktop/Reconhecimento de Padrões/Lista2/exercício3.py', wdir='C:/Users/Pedro/Desktop/
Reconhecimento de Padrões/Lista2')
ITEM A)

Acurácia A =  0.816
Count A =  184
Probabilidade de erro A =  18.4

ITEM B)

Acurácia B =  0.835
Count B =  165
Probabilidade de erro B =  16.5

ITEM C)

Probabilidade de erro de 20 amostras 0.7000000000000001
Probabilidade de erro de 25 amostras 1.3
Probabilidade de erro de 500 amostras 15.9
```