Técnicas Clássicas de Reconhecimento de Padrões (2020/01)

Exercício 04 - Classificador Bayesiano

Aluno: Ramon Gomes Durães de Oliveira (2019720188)

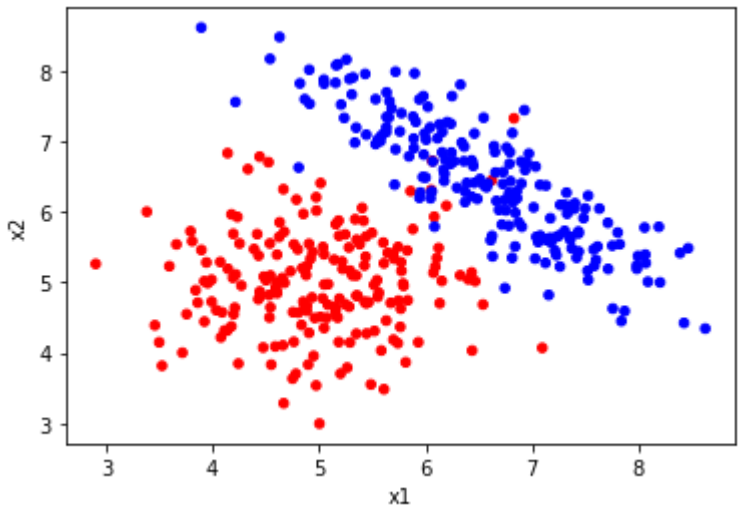
Neste exercício será implementado um classificador Bayesiano. Suas características e propriedades serão explorados em dados sintéticos e, em seguida, nos dados de leucemia (Golub et al 1999).

Gerando dados sintéticos

Abaixo, serão geradas duas classes sintéticas utilizando gaussianas multivariadas. Uma delas apresentará correlação entre os eixos.

In [85]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
%matplotlib inline

mean1 = np.array((5,5))
cov1 = np.array(((.5, 0.),
                 (0., .5)))
n1 = 200
y1 = 0
data1 = pd.DataFrame(np.random.multivariate_normal(mean1, cov1, n1), columns=[
'x1','x2'])
data1['y'] = y1

mean2 = np.array((6.5,6.5))
cov2 = np.array(((.8, -0.7),
                 (-0.7, .8)))
n2 = 200
y2 = 1
data2 = pd.DataFrame(np.random.multivariate_normal(mean2, cov2, n2), columns=[
'x1','x2'])
data2['y'] = y2

colors = {0: 'red', 1: 'blue'}
data = pd.concat([data1, data2])
data.plot.scatter(x='x1', y='x2', c=data['y'].replace(colors))
```

Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x1deb9475e48>



Classificador Bayesiano

Assume normalidade para as classes

Média e Covariância Amostral por classe:

In [2]:
```
def estimate_means_and_covariances(data):
    classes = data.y.unique()
    result = {'covariances': list(),
              'means': list()}
    for c in classes:
        d = data[data['y']==c].loc[:,data.columns != 'y']
        cov = np.cov(d.T)
        mean = np.mean(d).values
        result['covariances'].append(cov)
        result['means'].append(mean)
    return result
```

In [3]:
```
stats = estimate_means_and_covariances(data)
stats
```

Out[3]:
```
{'covariances': [array([[ 0.59159635, -0.05219555],
         [-0.05219555,  0.41875764]]), array([[ 0.72562221, -0.59224901],
         [-0.59224901,  0.66566701]])],
 'means': [array([5.0275359 , 5.00023153]), array([6.52075218, 6.47378106])]}
```

Nota-se que a covariância amostral (estimada acima) é próxima à covariância utilizada para gerar os dados.

Função de Densidade de Probabilidade Multivariada

Regra de Bayes:

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

Em que $P(C_i)$ é uma contagem, $P(x)$ é um normalizador (não necessário aqui) e $P(x|C_i)$ é o "core" do classificador em si.

Implementando a função de densidade de probabilidade multivariada:

$$f(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^k * det(\Sigma)}} * e^{-\frac{1}{2}*((x-\mu)^T \cdot inv(\Sigma) \cdot (x-\mu))}$$

In [4]:
```
def multivariate_pdf(x, mean, cov):
    size = len(x)
    x = np.array(x).reshape((1, size))
    mean = np.array(mean).reshape((1, size))
    cov = np.array(cov).reshape((size,size))
    if x.shape == mean.shape and (size, size) == cov.shape:
        det = np.linalg.det(cov)
        term1 = 1.0 / ( np.power((2*np.pi),(float(size)/2)) * np.power(det, 0.
5))
        term2 = x-mean
        term3 = np.linalg.pinv(cov)
        term4 = np.power(np.e,(-0.5 * np.matmul(np.matmul(term2,  term3), term
2.T)))
        return term1 * term4
```

Avaliando a função no primeiro ponto, utilizando as médias e matrizes de covariância relativas a cada uma das classes, espera-se que o valor retornado pela classe 1 seja maior que o valor retornado pela classe 2:

In [5]:
```
multivariate_pdf(data.loc[:,data.columns!='y'].iloc[0],
                 stats['means'][0],  # mu1
                 stats['covariances'][0]) # Sigma1
```

Out[5]:
```
array([[0.06634934]])
```

In [6]:
```
multivariate_pdf(data.loc[:,data.columns!='y'].iloc[0],
                 stats['means'][1], # mu2
                 stats['covariances'][1]) # Sigma 2
```

Out[6]:
```
array([[1.06822561e-12]])
```

De fato este é o caso!

Visualização da Fronteira de Decisão e das Superfícies de Probabilidade

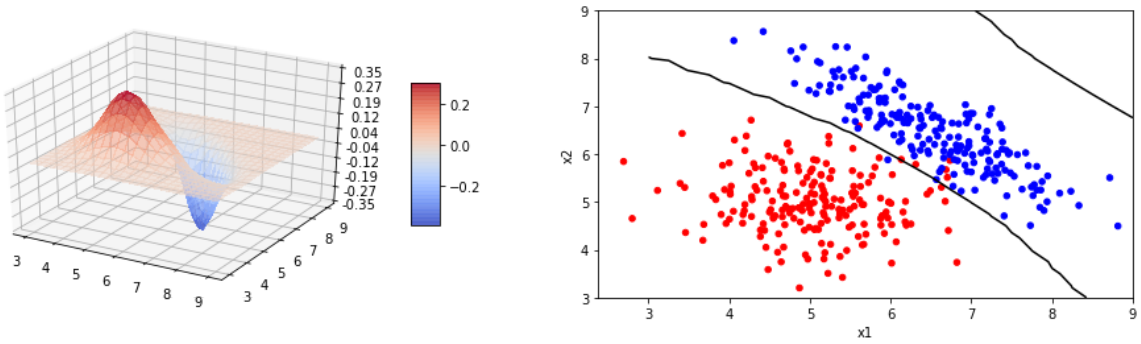
Calculando as superfícies de densidade de probabilidade para cada classe:

In [7]:
```
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from matplotlib import cm

X = np.arange(3, 9.25, 0.25)
Y = np.arange(3, 9.25, 0.25)
X, Y = np.meshgrid(X, Y)

Z1 = np.zeros(X.shape)
Z2 = np.zeros(X.shape)

for i in np.arange(X.shape[0]):
    for j in np.arange(X.shape[1]):
        Z1[i, j] = multivariate_pdf([X[i,j], Y[i,j]],
                    stats['means'][0],  # mu1
                    stats['covariances'][0]) # Sigma1
        Z2[i, j] = multivariate_pdf([X[i,j], Y[i,j]],
                    stats['means'][1],  # mu1
                    stats['covariances'][1]) # Sigma1
```

Abaixo, a superfície exibida à esquerda é a PDF resultante da subtração da PDF da classe 1 pela da classe 2. Isso é feito apenas para melhorar a visualização, já que não fariam sentido probabilidades negativas. À direita podemos ver a curva de separação resultante em duas dimensões $\frac{P(x|C_1)}{P(x|C_2)} = 1$.

In [8]:
```
fig = plt.figure(figsize=plt.figaspect(0.25))
ax = fig.add_subplot(1, 2, 1, projection='3d')
surf1 = ax.plot_surface(X, Y, Z1-Z2, cmap=cm.coolwarm, alpha = 0.5,
                        linewidth=0, antialiased=False)

#surf2 = ax.plot_surface(X, Y, Z2, cmap=cm.cool, alpha = 0.5,
#                        linewidth=0, antialiased=False)

# Customize the z axis.
ax.set_zlim(-.35, .35)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf1, shrink=0.5, aspect=5)


ax1 = fig.add_subplot(1, 2, 2)
data.plot.scatter(x='x1', y='x2', c=data['y'].replace(colors), ax=(ax1))
ax1.contour(X,Y,(Z1/Z2), [1.], colors='k')
```

Out[8]: <matplotlib.contour.QuadContourSet at 0x1deb56d7da0>



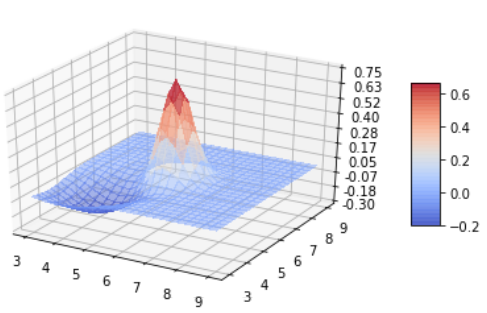
Repetindo o processo para um segundo conjunto de dados sintético:

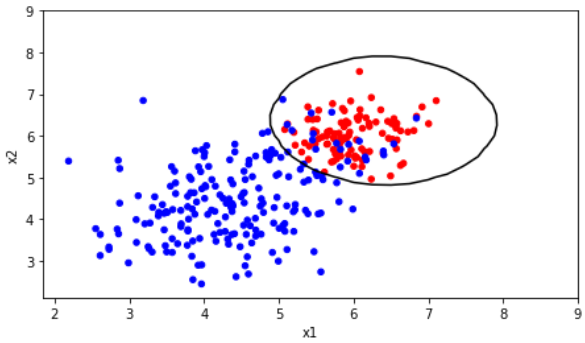
Para explorar as propriedades do classificador, todo o processo acima será repetido para novos dados sintéticos. Dessa vez, haverá desbalanceamento entre as classes e uma delas estará mais concentrada em uma região.

In [9]:
```
# Gerando os dados
mean1 = np.array((6,6))
cov1 = np.array(((.2, 0.),
                 (0., .2)))
n1 = 100
y1 = 0
data1 = pd.DataFrame(np.random.multivariate_normal(mean1, cov1, n1), columns=[
'x1','x2'])
data1['y'] = y1

mean2 = np.array((4.5,4.5))
cov2 = np.array(((.8, 0.3),
                 (0.3, .8)))
n2 = 200
y2 = 1
data2 = pd.DataFrame(np.random.multivariate_normal(mean2, cov2, n2), columns=[
'x1','x2'])
data2['y'] = y2

colors = {0: 'red', 1: 'blue'}
data = pd.concat([data1, data2])

stats = estimate_means_and_covariances(data)
stats

X = np.arange(3, 9.25, 0.25)
Y = np.arange(3, 9.25, 0.25)
X, Y = np.meshgrid(X, Y)

Z1 = np.zeros(X.shape)
Z2 = np.zeros(X.shape)

for i in np.arange(X.shape[0]):
    for j in np.arange(X.shape[1]):
        Z1[i, j] = multivariate_pdf([X[i,j], Y[i,j]],
                    stats['means'][0],  # mu1
                    stats['covariances'][0]) # Sigma1
        Z2[i, j] = multivariate_pdf([X[i,j], Y[i,j]],
                    stats['means'][1],  # mu1
                    stats['covariances'][1]) # Sigma1

fig = plt.figure(figsize=plt.figaspect(0.25))
ax = fig.add_subplot(1, 2, 1, projection='3d')
surf1 = ax.plot_surface(X, Y, Z1-Z2, cmap=cm.coolwarm, alpha = 0.5,
                        linewidth=0, antialiased=False)

#surf2 = ax.plot_surface(X, Y, Z2, cmap=cm.cool, alpha = 0.5,
#                        linewidth=0, antialiased=False)

# Customize the z axis.
ax.set_zlim(-.3, .75)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf1, shrink=0.5, aspect=5)
```

```
ax1 = fig.add_subplot(1, 2, 2)
data.plot.scatter(x='x1', y='x2', c=data['y'].replace(colors), ax=(ax1))
ax1.contour(X,Y,(Z1/Z2), [1.], colors='k')
```

Out[9]: <matplotlib.contour.QuadContourSet at 0x1deb57e4668>





Classificando os dados de Leucemia (Golub et al)

Pré-processamento de dados

Bem como no trabalho anterior, os dados serão limpados e o conjunto de genes mais relevante para a classificação será selecionado antes da classificação em si. As explicações detalhadas de cada processo estão no relatório anterior, de replicação do trabalho de Golub et al. Aqui elas foram removidas por economia de espaço.

Carregando e Limpeza os Dados

Dados de treinamento

In [10]:
```
class_df = pd.read_csv('./data/actual.csv')
train_df = pd.read_csv('./data/data_set_ALL_AML_train.csv')
test_df = pd.read_csv('./data/data_set_ALL_AML_independent.csv')

valid_columns = [col for col in train_df.columns if "call" not in col]
train_df = train_df[valid_columns]
train_df = train_df.T
train_df = train_df.drop(['Gene Description','Gene Accession Number'],axis=0)
train_df.index = pd.to_numeric(train_df.index)
train_df.sort_index(inplace=True)
class_dict = {'AML':0,'ALL':1}
train_df['class'] = class_df[:38]['cancer'].replace(class_dict).values
train_df.head()
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	...	7120	7121	7122	7123	7124
1	-214	-153	-58	88	-295	-558	199	-176	252	206	...	511	-125	389	-37	793
2	-139	-73	-1	283	-264	-400	-330	-168	101	74	...	837	-36	442	-17	782
3	-76	-49	-307	309	-376	-650	33	-367	206	-215	...	1199	33	168	52	1138
4	-135	-114	265	12	-419	-585	158	-253	49	31	...	835	218	174	-110	627
5	-106	-125	-76	168	-230	-284	4	-122	70	252	...	649	57	504	-26	250

5 rows × 7130 columns

Dados de teste:

In [11]:
```
valid_columns = [col for col in test_df.columns if "call" not in col]
test_df = test_df[valid_columns]
test_df = test_df.T
test_df = test_df.drop(['Gene Description','Gene Accession Number'],axis=0)
test_df.index = pd.to_numeric(test_df.index)
test_df.sort_index(inplace=True)
test_df['class'] = class_df[38:]['cancer'].replace(class_dict).values
test_df.head()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	7120	7121	7122	7123	7124
39	-342	-200	41	328	-224	-427	-656	-292	137	-144	...	1023	67	214	-135	1074
40	-87	-248	262	295	-226	-493	367	-452	194	162	...	529	-295	352	-67	67
41	-62	-23	-7	142	-233	-284	-167	-97	-12	-70	...	383	46	104	15	245
42	22	-153	17	276	-211	-250	55	-141	0	500	...	399	16	558	24	893
43	86	-36	-141	252	-201	-384	-420	-197	-60	-468	...	91	-84	615	-52	1235

5 rows × 7130 columns

Seleção de Genes Relevantes

In [12]:
```
mu1 = train_df[train_df['class']==0].iloc[:,:-1].mean()
sigma1 = train_df[train_df['class']==0].iloc[:,:-1].std()
mu2 = train_df[train_df['class']==1].iloc[:,:-1].mean()
sigma2 = train_df[train_df['class']==1].iloc[:,:-1].std()
Pgc = (mu1 - mu2) / (sigma1 + sigma2)
abs_Pgc = np.abs(Pgc)

selected_genes = abs_Pgc>.91
selected_genes = selected_genes.index[selected_genes.values]
train_df[selected_genes].shape
```

Out[12]:
```
(38, 50)
```

Nota-se que agora o conjunto de dados tem 38 observações mas apenas 50 genes.

Classificação por Bayes

Estimando as Médias e Matrizes de Covariância

In [99]:
```
from sklearn.preprocessing import StandardScaler

training_data = train_df[selected_genes]
training_data = training_data.astype(float)
training_data['y'] = train_df['class']

stats = estimate_means_and_covariances(training_data)
```

Classificação

Para classificar os dados sem erros numéricos, utilizou-se o tipo de dados "Decimal" do Python, devido à sua precisão aumentada. Além disso, assumiu-se independência na distribuição.

In [96]:
```
from decimal import Decimal

testing_data = test_df[selected_genes]
testing_data = testing_data.astype(float)
testing_data['y'] = test_df['class']

n_test = testing_data.shape[0]
PxC1 = np.zeros((n_test, 1))
PxC2 = np.zeros((n_test, 1))

for i in np.arange(n_test):
    x = testing_data.loc[:,testing_data.columns != 'y'].iloc[i,:].values
    PxC1[i] = multivariate_pdf(x,
                               stats['means'][0],  # mu1
                               stats['covariances'][0]) # Sigma1
    PxC2[i] = multivariate_pdf(x.astype(Decimal),
                               stats['means'][1],  # mu1
                               np.diag(np.diag(stats['covariances'][1]))) # S
igma1
predicted_classes = np.ravel(1*np.array(PxC1 > PxC2))
predicted_classes
```

Out[96]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

Calculando a matriz de confusão para avaliar os resultados da classificação:

In [97]:
```
confusion_matrix(testing_data.y, predicted_classes)
```

Out[97]:
```
array([[12,  2],
       [ 0, 20]], dtype=int64)
```

Nota-se que, das 34 observações do conjunto de teste, 32 foram classificadas corretamente!

A performance do classificador foi satisfatória e semelhante à obtida no exercício anterior (replicando os resultados de Golub et al).