Лекция 6 по дисциплине "Методы и средства распознавания и обработки данных"

Кафедра КБ-3 "Управление и моделирование систем" Института Комплексной Безопасности и Специального Приборостроения, МИРЭА, 2020г.

Дата проведения занятия: 21.03.2020.

Время: 18:20-21:20.

Место/Форма проведения занятия: Дистанционное обучение/Конференция **Skype.** Ссылка на видео-конференцию будет предоставлена непосредственно перед занятием по почте.

Цель: Реализация классификатора многомерных случайных величин с помощью полносвязной многослойной сети. Сравнение с базовым классификатором (байесовский классификатор).

Задача

Сформировать два набора случайных гауссовых двумерных векторов $\{\vec{x}_i\}^{(1)}$ и $\{\vec{x}_i\}^{(2)}$ с параметрами $\{\vec{\mu}_1, \Sigma_1\}$, $\{\vec{\mu}_2, \Sigma_2\}$.

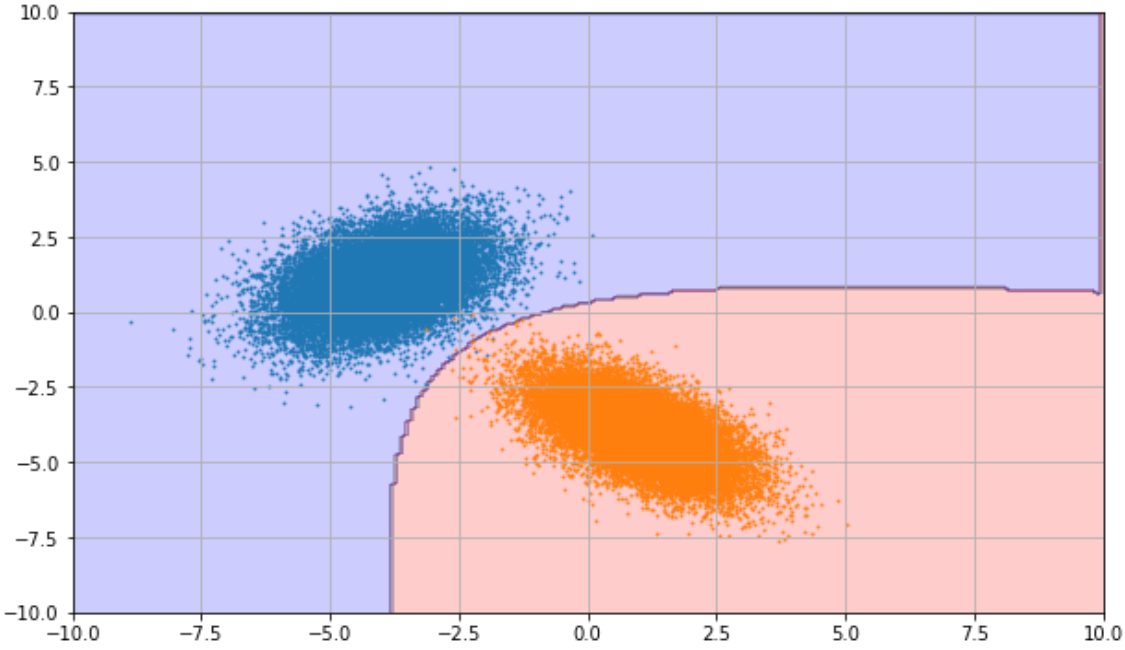
Реализовать байесовский классификатор для двух классов. Оценить вероятность ошибки классификации. Изобразить оба класса графически точками. Изобразить границу классификации.

Реализовать классификатор для двух классов на основе полносвязной многослойной сети средствами **Python/TensorFlow.** Оценить вероятность ошибки классификации. Изобразить оба класса графически точками. Изобразить границу классификации.

In [0]:

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

def plot_boundaries():
  G       = 200
  x       = np.linspace(-10, 10, G)
  y       = np.linspace(-10, 10, G)
  xv, yv  = np.meshgrid(x, y)
  F       = np.zeros((G,G))
  for k in range(0,G-1):
    for m in range(0,G-1):
      x = np.array([ [xv[k,m]], [yv[k,m]] ])
      phi1 = -x.T @ invSigma1 @ x + x.T @ invSigma1 @ mu1 + mu1.T @ invSigma1 @ x - mu1.
T @ invSigma1 @ mu1 - detSigma1
      phi2 = -x.T @ invSigma2 @ x + x.T @ invSigma2 @ mu2 + mu2.T @ invSigma2 @ x - mu2.
T @ invSigma2 @ mu2 - detSigma2
      F[k,m] = phi1 > phi2
  plt.contourf( xv, yv, F, colors=['red','black','blue','gray'], alpha=0.2 )


N         = 20000
mu1       = np.array([ [-4], [1] ])
mu2       = np.array([ [1], [-4] ])
sigma1    = np.array([ [1, 0.4], [0.4, 1] ])
sigma2    = np.array([ [1, -0.6], [-0.6, 1] ])
invSigma1 = np.linalg.inv(sigma1)
invSigma2 = np.linalg.inv(sigma2)
detSigma1 = np.log(np.linalg.det(sigma1))
detSigma2 = np.log(np.linalg.det(sigma2))

s1     = np.linalg.cholesky( sigma1 )
s2     = np.linalg.cholesky( sigma2 )


#print(np.shape(mu1))
vmu1    = np.tile(mu1,(1,N))
#print(np.shape(mu1))
```

```
x1      = s1 @ np.random.randn(2,N) + vmu1

vmu2    = np.tile(mu2,(1,N))
x2      = s2 @ np.random.randn(2,N) + vmu2

# Bayesian classifier
phi   = np.zeros((2,N))
rand_x = np.random.rand(2,N)*20 -10
for k in range(0,N-1):
  phi[0,k] = -rand_x[:,k].T @ invSigma1 @ rand_x[:,k] + rand_x[:,k].T @ invSigma1 @ mu1
+ mu1.T @ invSigma1 @ rand_x[:,k] - mu1.T @ invSigma1 @ mu1 - detSigma1
  phi[1,k] = -rand_x[:,k].T @ invSigma2 @ rand_x[:,k] + rand_x[:,k].T @ invSigma2 @ mu2
+ mu2.T @ invSigma2 @ rand_x[:,k] - mu2.T @ invSigma2 @ mu2 - detSigma2


fig = plt.figure(figsize=[10,6])

plot_boundaries()

plt.scatter( x1[0], x1[1], s=1, alpha=0.85  )
plt.scatter( x2[0], x2[1], s=1, alpha=0.85 )

plt.grid(True)
```



In [3]:

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)

N         = 20000
mu1       = np.array([ [-3], [1] ])
mu2       = np.array([ [1], [-4] ])
sigma1    = np.array([ [1, 0.6], [0.6, 1] ])
sigma2    = np.array([ [1, -0.7], [-0.7, 1] ])
invSigma1 = np.linalg.inv(sigma1)
invSigma2 = np.linalg.inv(sigma2)
detSigma1 = np.log(np.linalg.det(sigma1))
detSigma2 = np.log(np.linalg.det(sigma2))

s1      = np.linalg.cholesky( sigma1 )
s2      = np.linalg.cholesky( sigma2 )
```

```
#print(np.shape(mu1))
vmu1         = np.tile(mu1,(1,N))
#print(np.shape(mu1))
tx1           = s1 @ np.random.randn(2,N) + vmu1

vmu2         = np.tile(mu2,(1,N))
tx2           = s2 @ np.random.randn(2,N) + vmu2

train_data   = np.column_stack((tx1,tx2)).transpose()
train_labels = np.column_stack((np.zeros((1,N)),np.ones((1,N)))).transpose()
print(np.shape(train_data))
print(np.shape(train_labels))

model = keras.Sequential([
    keras.layers.Dense(25, activation='linear'),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(2, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_data, train_labels, epochs=5)
```

```
1.15.0
(40000, 2)
(40000, 1)
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/resource_
variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.r
esource_variable_ops) with constraint is deprecated and will be removed in a future versi
on.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Train on 40000 samples
Epoch 1/5
40000/40000 [==============================] - 2s 55us/sample - loss: 0.0166 - acc: 0.996
1
Epoch 2/5
40000/40000 [==============================] - 2s 49us/sample - loss: 0.0018 - acc: 0.999
4
Epoch 3/5
40000/40000 [==============================] - 2s 51us/sample - loss: 0.0021 - acc: 0.999
3
Epoch 4/5
40000/40000 [==============================] - 2s 48us/sample - loss: 0.0021 - acc: 0.999
5
Epoch 5/5
40000/40000 [==============================] - 2s 49us/sample - loss: 0.0020 - acc: 0.999
4
```

Out[3]:

```
<tensorflow.python.keras.callbacks.History at 0x7fca8b16eb38>
```

In [5]:

```
N         = 20000


#print(np.shape(mu1))
vmu1         = np.tile(mu1,(1,N))
#print(np.shape(mu1))
x1           = s1 @ np.random.randn(2,N) + vmu1

vmu2         = np.tile(mu2,(1,N))
x2           = s2 @ np.random.randn(2,N) + vmu2

test_data   = np.column_stack((x1,x2)).transpose()
test_labels = np.column_stack((np.zeros((1,N)),np.ones((1,N)))).transpose()
```

```
print(np.shape(test_data))
print(np.shape(test_labels))

test_loss, test_acc = model.evaluate(test_data,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)

def plot_boundaries(model):
  G       = 200
  x       = np.linspace(-10, 10, G)
  y       = np.linspace(-10, 10, G)
  xv, yv  = np.meshgrid(x, y)
  plotData= np.column_stack((xv.flatten(),yv.flatten()))
  predictions = model.predict(plotData)
  F       = predictions[:,0]>predictions[:,1]
  print(np.shape(F))
  F       = F.reshape(G,G,order='C').copy()
  plt.contourf( xv, yv, F, colors=['red','black','blue','gray'], alpha=0.2 )

plt.figure(figsize=[10,6])

plot_boundaries(model)

plt.scatter( x1[0], x1[1], s=1, alpha=0.75  )
plt.scatter( x2[0], x2[1], s=1, alpha=0.75 )

plt.scatter( tx1[0], tx1[1], s=15, alpha=1  )
plt.scatter( tx2[0], tx2[1], s=25, alpha=1 )

plt.grid(True)
```

```
(40000, 2)
(40000, 1)
40000/40000 - 1s - loss: 0.0021 - acc: 0.9994

Test accuracy: 0.9994
(40000,)
```