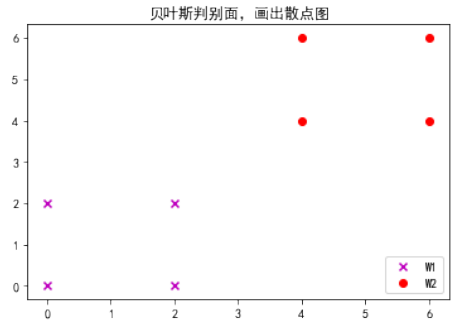
2017-10-1 田茂 201728018629013 - 设以下模式类别具有正态概率密度函数： ω1：{(0 0)T, (2 0)T, (2 2)T, (0 2)T} ω2：{(4 4)T, (6 4)T, (6 6)T, (4 6)T} （1）设P(ω1)= P(ω2)=1/2，求这两类模式之间的贝叶斯判别界面的方程式。 （2）绘出判别界面。 - 编写两类正态分布模式的贝叶斯分类程序。

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

matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['font.family']='sans-serif'
fig1=plt.figure(1)
W1=np.asarray([[0,0],[2,0],[2,2],[0,2]])
W2=np.asarray([[4,4],[6,4],[6,6],[4,6]])
#注意只有转换为numpy类型的array之后才能使用下面的切片方法
#print W1[:,0]
#print W1[:,1]
p1=plt.scatter(W1[:,1],W1[:,0],marker='x',color='m',label='W1')
p2=plt.scatter(W2[:,1],W2[:,0],marker='o',color='r',label='W2')
plt.legend(loc='lower right')
plt.title(u'贝叶斯判别面，画出散点图')
plt.show()
```




```
#求均值
m1=np.mean(W1,axis=0)
m2=np.mean(W2,axis=0)
print u"W1的均值:",m1
print u"W2的均值:",m2
```

```
W1的均值: [ 1.  1.]
W2的均值: [ 5.  5.]
```

```
#求协方差
c1=np.cov(W1.T)
c2=np.cov(W2.T)
print u"W1的协方差矩阵:\n",c1
print u"W2的协方差矩阵:\n",c2
```

```
W1的协方差矩阵:
[[ 1.33333333  0.        ]
 [ 0.          1.33333333]]
W2的协方差矩阵:
[[ 1.33333333  0.        ]
 [ 0.          1.33333333]]
```

W1和W2的协方差矩阵相等

又：P(W1)=P(W2)=1/2,则判别截面为：

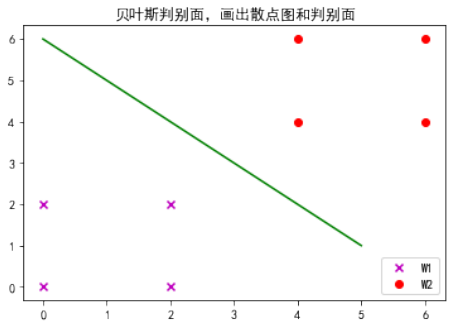
```
c_1=c1**(-1)
print (m1-m2).T*c_1
print (-0.5*m1.T*c_1*m1+0.5*m2.T*c_1*m2)
```

```
[[ -3. -inf]
 [-inf  -3.]]
[[  9.  nan]
 [ nan   9.]]


D:\anaconda\lib\site-packages\ipykernel\__main__.py:1: RuntimeWarning: divide by zero encountered in reciprocal
  if __name__ == '__main__':
D:\anaconda\lib\site-packages\ipykernel\__main__.py:3: RuntimeWarning: invalid value encountered in add
  app.launch_new_instance()
```

结果:d1(x)-d2(x)=-3x1-3x2+18=0

```
fg2=plt.figure(2)
p1=plt.scatter(W1[:,1],W1[:,0],marker='x',color='m',label='W1')
p2=plt.scatter(W2[:,1],W2[:,0],marker='o',color='r',label='W2')
plt.legend(loc='lower right')
plt.title(u'贝叶斯判别面，画出散点图和判别面')
x=np.linspace(0,5,1000)
y=-x+6
plt.plot(x,y,color='green')
plt.show()
```



使用sklearn训练并预测

```
#使用numpy的concatenate函数将array拼接在一起
W=np.concatenate([W1,W2],axis=0)
print u"训练数据:\n",W
label=np.array([0,0,0,0,1,1,1,1])
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB().fit(W,label)
print u"预测结果:",clf.predict([[1,1],[5,5]])
```

```
训练数据:
[[0 0]
 [2 0]
 [2 2]
 [0 2]
 [4 4]
 [6 4]
 [6 6]
 [4 6]]
预测结果: [0 1]
```

编写两类正态分布模式的贝叶斯分类程序。

```
#-*-coding:utf-8 -*-
#作者：田茂
#邮箱：tianmao1994@163.com
#编写时间：2017年10月1日
import numpy as np
class GaussianNB:
    """
```

```
    两类正态分布模式的贝叶斯分类程序
    class_count=2
    输入:numpy型的array
    原理:由输入的数据得到判别面方程，保存判别面参数，对新的数据进行预测
    """
#     def __init__():
#         pass
    def fit(self,X,y):
        """
        输入numpy型的array，就是输入先验，由X和y两组数据构成，其中y是标签
        """
        #样本总数
        count=X.shape[0]
        #target的取值，两分类
        self.classes=np.unique(y)
        #将数据拆分成为两份，W1，W2
        W1=[]
        W2=[]
        for i in range(count):
            if y[i]==self.classes[0]:
                W1.append(X[i])
            else:
                W2.append(X[i])
        W1=np.asarray(W1)
        W2=np.asarray(W2)

        W1_count=W1.shape[0]
        W2_count=W2.shape[0]

        self.P_W1=float(W1_count)/count
        self.P_W2=float(W2_count)/count

        #计算均值
        self.m1=np.mean(W1,axis=0)
        self.m2=np.mean(W2,axis=0)
        #计算协方差
        self.c1=np.cov(W1.T)
        self.c2=np.cov(W2.T)
        #得到判别面

    def predict(self,X):
        """
        输入时numpy型的array，对多对新样本进行predict，返回一个numpy型的array
        """
        classes=self.classes
        P_W1=self.P_W1
        P_W2=self.P_W2
        m1=self.m1
        m2=self.m2
        c1=self.c1
        c2=self.c2
        count=X.shape[0]
        Y=[]
        for i in range(count):
            #d1和d2是判别面，比较二者的大小，对新加入的样本进行预测
            d1=np.log(P_W1)-0.5*np.log(np.linalg.det(c1))-0.5*((X[i]-m1).T)*(c1**(-1))*(X[i]-m1)
            d2=np.log(P_W2)-0.5*np.log(np.linalg.det(c2))-0.5*((X[i]-m2).T)*(c2**(-1))*(X[i]-m2)
            if np.linalg.det(d1)>np.linalg.det(d2):
                Y.append(classes[0])
            else:
                Y.append(classes[1])
        return Y


X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
Y = np.array([1, 1, 1, 2, 2, 2])
clf=GaussianNB()
clf.fit(X,Y)
test=np.asarray([[3,3],[-3,-3],[2,2],[5,5],[-2,-7]])
clf.predict(test)


[2, 1, 2, 2, 1]
```