机器学习实验报告 (一)

2018/11/11

目录

1 题目描述与分析

1.1 第一问——马氏距离

题目描述: 求测试点到各类别均值间的 *mahalanobis* 距离

马氏距离, 是由印度统计学家马哈拉诺比斯提出的，表示数据的协方差距离。它是一种有效的计算两个未知样本集的相似度的方法，它考虑到各种特性之间的联系，且尺度无关的。马氏距离也可以定义为两个服从同一分布并且其协方差矩阵为 Σ 的随机变量之间的差异程度。很明显可以看出，马氏距离越小，差异程度越小。这里给出公式定义:

$$D_M(\mathbf{x}) = \sqrt{(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$

其中: $\mathbf{x}$ 是一个 d 维列向量，即测试点;

$\boldsymbol{\mu}$ 为类别的均值向量;

$\boldsymbol{\Sigma}$ 为类别的协方差矩阵

因此求解第一问只需将相应测试点和相应类别的均值向量及协方差矩阵带入即可。

1.2 第二问——等概率分类

问题描述: 求解等概率情况下，对各测试点进行分类

此问题有关分类器中的多类情况。贝叶斯分类器是各种分类器中分类错误概率最小或者在预先给定代价的情况下平均风险最小的分类器，它的设计方法是一种最基本的统计分类方法，其分类原理是通过某对象的先验概率，利用贝叶斯公式计算出其后验概率，即该对象属于某一类的概率，选择具有最大后验概率的类作为该对象所属的类。

我们选择用判别函数的的方式来表述模式分类器，即: 对于所有 j≠i，有

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \tag{1}$$

则此分类器将这个特征向量 $\mathbf{x}$ 判为 ω_i。在具有一般风险的情况下，由于最大的判别函数与最小的条件风险相对应，我们让 $g_i(\mathbf{x}) = -R(\alpha_i|\mathbf{x})$, 又因为在最小误差概率情况下，$R(\alpha_i|\mathbf{x}) = 1 - P(\omega_i|\mathbf{x})$，我们可以进一步简化判别函数:$g_i(\mathbf{x}) = P(\omega_i|\mathbf{x})$。更一般的，我们只需要知道不同类别判别函数的大小关系，因此将所有判别函数乘上相同的正常数或加减相同常量、带入单调递增函数操作得到的分类结果没有变化，由此可以得到:

$$g_i(\mathbf{x}) = \ln p(\mathbf{x}|\omega_i) + \ln P(\omega_i) \tag{2}$$

题目已经假设分布是正态的，我们知道一般的 d 维多元正态密度的形式如下:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right] \tag{3}$$

当 $p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ 时，联系 (2)(3) 式可以得到如下判别函数:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\mathbf{\Sigma}_i| + \ln P(\omega_i) \tag{4}$$

该式子为一般情况下正态分布的判别函数。于是我们可以求出每种类别的协方差矩阵后观察 $\mathbf{\Sigma}_i$ 的关系，来对 (4) 式进行化简求值，若测试点在 i 个判别函数中取最大即归为 i 类。

1.3 第三问——不等概率分类

问题描述: 求解不同概率下，对各测试点进行分类

同第二问，将 $P(\omega_i)$ 代入相应的取值即可。

2 具体实现

2.1 库的导入及数据准备

```
1 import numpy as np #矩阵运算
2 import matplotlib.pyplot as plt #作图
3 from mpl_toolkits.mplot3d import Axes3D
4
5 list_point = [np.array([1,2,1]),np.array([5,3,2]),
```

```
                np.array([0,0,0]),np.array([1,0,0])]
lists1=np.array([[-5.01,-8.12,-3.68],[-5.43,-3.48,-3.54],
          [1.08,-5.52,1.66],[0.86,-3.78,-4.11],[-2.67,0.63,7.39],
          [4.94,3.29,2.08],[-2.51,2.09,-2.59],[-2.25,-2.13,-6.94],
          [5.56,2.86,-2.26],[1.03,-3.33,4.33]])
lists2=np.array([[-0.91,-0.18,-0.05],[1.30,-2.06,-3.53],
          [-7.75,-4.54,-0.95],[-5.47,0.50,3.92],[6.14,5.72,-4.85],
          [3.60,1.26,4.36],[5.37,-4.63,-3.65],[7.18,1.46,-6.66],
          [-7.39,1.17,6.30],[-7.50,-6.32,-0.31]])
lists3=np.array([[5.35,2.26,8.13],[5.12,3.22,-2.66],
          [-1.34,-5.31,-9.87],[4.48,3.42,5.19],[7.11,2.39,9.21],
          [7.17,4.33,-0.98],[5.75,3.97,6.65],[0.77,0.27,2.41],
          [0.90,-0.43,-8.71],[3.52,-0.36,6.43]])
list_group = [lists1,lists2,lists3]
```

2.2 初步观察

因为所给类别为三维，我们先将所给点在图中画出，做简单观察

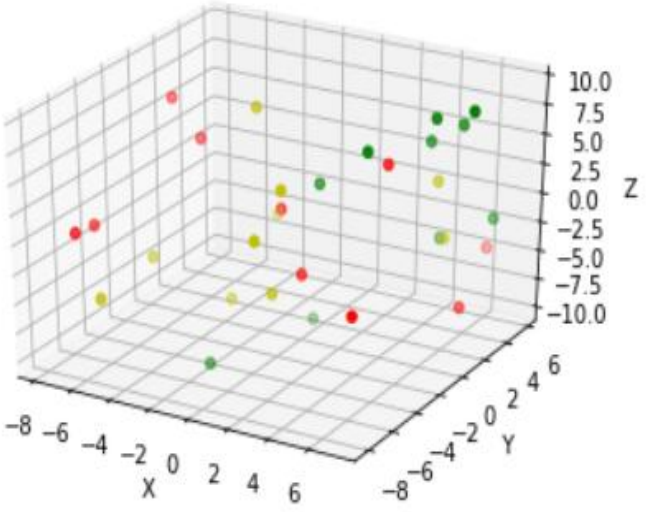

图 1 散点图

可以发现，仅凭观察很难得到一些有用的信息。

2.3 马氏距离的计算

在这里，我们定义了一个 $Mahalanovis_dis()$ 函数, 用于求解马氏距离。有两个参数，其中 $point$ 代表测试点的坐标，$points$ 代表类别。首先我们使用 $np.matrix()$ 函数将点转换成矩阵以便计算。为了求解均值向量，我们定义 $getMean()$ 函数。具体代码如下:

```
def getMean(points):
        #输入为矩阵 求列均值 10x3
    return np.mean(points,0)

def Mahalanovis_dis(point,points):
    m_points = np.matrix(points)#10x3
    m_pointsT = m_points.T#3x10
    D = np.cov(m_pointsT,bias=True)#3x3
    invD = np.linalg.inv(D)
    u=getMean(points)#1x3
    point=np.matrix(point)
    dis = point-u#1x3
    disT=dis.T#3x1
    temp = np.dot(dis,invD)
    result = np.sqrt(np.dot(temp,disT))
    return float(result)
```

随后编写调用函数，对问题一进行求解:

```
for i in list_point:
    count=0
    for j in list_group:
```

```
4           count=count+1
5           ans = Mahalanovis_dis(i,j)
6           print(i,'与集合',count,'的距离为',ans)
```

所得结果如下所示:

[1 2 1] 与集合 *1* 的距离为 *1.0698729737115753*
[1 2 1] 与集合 *2* 的距离为 *0.9044653755297206*
[1 2 1] 与集合 *3* 的距离为 *2.8194414746233187*
[5 3 2] 与集合 *1* 的距离为 *1.6413677928169066*
[5 3 2] 与集合 *2* 的距离为 *1.8506499400391951*
[5 3 2] 与集合 *3* 的距离为 *0.6820073837310614*
[0 0 0] 与集合 *1* 的距离为 *0.5164648124154192*
[0 0 0] 与集合 *2* 的距离为 *0.2829526056849935*
[0 0 0] 与集合 *3* 的距离为 *2.3627499041029951*
[1 0 0] 与集合 *1* 的距离为 *0.5135926390697754*
[1 0 0] 与集合 *2* 的距离为 *0.4762752259695465*
[1 0 0] 与集合 *3* 的距离为 *1.541437911538147*

2.4 分类

首先我们定义了 $getCov()$ 函数来求解协方差矩阵，需要传入的参数 $points$ 为 $numpy.ndarray$ 类型，具体代码如下:

```
1 def getCov(points):
2     m_points = np.matrix(points)#10x3
3     m_pointsT = m_points.T#3x10
4     D = np.cov(m_pointsT,bias=True)#3x3
5     return D
```

在这我们编写调用函数求解三个类别各自的协方差矩阵:

```
count=0
for j in list_group:
    count=count+1
    ans = getCov(j)
    print("第",count,"类别协方差矩阵为")
    print(ans)
```

得到的结果为:

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 12.94246 & 6.92584 & 3.71009 \\ 6.92584 & 13.160809 & 3.516156 \\ 3.71009 & 3.516156 & 17.752084 \end{bmatrix} \tag{5}$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 33.146401 & 8.982834 & -14.730076 \\ 8.982834 & 11.851696 & 0.368146 \\ -14.730076 & 0.368146 & 16.579096 \end{bmatrix} \tag{6}$$

$$\boldsymbol{\Sigma}_3 = \begin{bmatrix} 7.474281 & 6.700452 & 11.83462 \\ 6.700452 & 7.704404 & 10.44775 \\ 11.83462 & 10.44775 & 42.55856 \end{bmatrix} \tag{7}$$

可以看到，此题属于 $\boldsymbol{\Sigma}_i =$ 任意的情况，对应上面 (4) 式，我们可以去掉常数项 $\frac{d}{2}\ln 2\pi$, 得到初步化简后的判别函数:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i) \tag{8}$$

更进一步，将二次型 $(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$ 展开，并带入化简可得如下式子:

$$g_i(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^t \mathbf{x} + omega_{i0} \tag{9}$$

其中:

$$\mathbf{W}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1} \tag{10}$$

$$\mathbf{w}_i = \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i \tag{11}$$

$$\omega_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^t\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(w_i) \tag{12}$$

上式 (9) 即为我么要求的判别函数,下面为具体代码实现。首先编写 $getLoss()$ 函数,用于计算判别函数。其中 $point$ 为测试点坐标,$points$ 为 $numpy.ndarray$ 类型的类别表示，px 为各自的先验概率。使用 $numpy$ 库辅助进行矩阵计算

```
def getLoss(point,points,px):
    point=np.matrix(point)#1X3
    #求协方差矩阵的逆
    m_points = np.matrix(points)#10x3
    m_pointsT = m_points.T#3x10
    D = np.cov(m_pointsT,bias=True)#3x3
    invD = np.linalg.inv(D)
    #求Wi
    Wi = -0.5*invD
    a0 = np.dot(np.dot(point,Wi),point.T)
    u=getMean(points)
    #求wi
    wi = np.dot(invD,u)
    a1 = np.dot(wi.T,point.T)
    #求wi0
    wi00 = -0.5*np.dot(np.dot(u.T,invD),u)
    wi01 = -0.5*np.log(np.linalg.det(D))
    wi02 = np.log(px)
    wi0 = wi00+wi01+wi02
    return float(a0+a1+wi0)
```

编写调用函数 $determineType$，根据输入测试点，输出分类判别结果，并给出各个类别的判别函数值。

```
def determineType(point,list_group,p_list):
    k={}
    count=0
    for j in list_group:
        count=count+1
        ans = getLoss(point,j,p_list[count-1])
        print(point,'判为集合',count,'的判别函数值',ans)
        k[count] = ans
    sortlist=sorted(k.items(),key = lambda x:x[1],reverse = True)
    print(point,'点最有可能为w'+str(sortlist[0][0])+"类")
    print()
    return sortlist
```

第二问中，$P(w_i)$=1/3, 按如下调用函数:

```
# 等可能
for i in list_point:
    determineType(i,list_group,[1/3,1/3,1/3])
```

得到的结果为:

[1 2 1] 判为集合 *1* 的判别函数值 *-5.474314528872933*
[1 2 1] 判为集合 *2* 的判别函数值 *-5.420008090852984*
[1 2 1] 判为集合 *3* 的判别函数值 *-7.9285352274227225*
[1 2 1] 点最有可能为 *w2* 类
[5 3 2] 判为集合 *1* 的判别函数值 *-6.24904455458203*
[5 3 2] 判为集合 *2* 的判别函数值 *-6.723431883370463*
[5 3 2] 判为集合 *3* 的判别函数值 *-4.18647714874151*
[5 3 2] 点最有可能为 *w3* 类
[0 0 0] 判为集合 *1* 的判别函数值 *-5.035368390165355*
[0 0 0] 判为集合 *2* 的判别函数值 *-5.051010371618888*
[0 0 0] 判为集合 *3* 的判别函数值 *-6.7452036676790215*

[0 0 0] 点最有可能为 *w1* 类

[1 0 0] 判为集合 *1* 的判别函数值 *-5.033889138387036*

[1 0 0] 判为集合 *2* 的判别函数值 *-5.124398328523096*

[1 0 0] 判为集合 *3* 的判别函数值 *-5.141925530573207*

[1 0 0] 点最有可能为 *w1* 类

第三问中，$P(w_1)$=0.8,$P(w_2)$=0.1,$P(w_3)$=0.1, 按如下调用:

```
# 不等可能
for i in list_point:
    determineType(i,list_group,[0.8,0.1,0.1])
```

得到的结果为:

[1 2 1] 判为集合 *1* 的判别函数值 *-4.598845791519032*

[1 2 1] 判为集合 *2* 的判别函数值 *-6.62398089517892*

[1 2 1] 判为集合 *3* 的判别函数值 *-9.132508031748658*

[1 2 1] 点最有可能为 *w1* 类

[5 3 2] 判为集合 *1* 的判别函数值 *-5.373575817228129*

[5 3 2] 判为集合 *2* 的判别函数值 *-7.927404687696399*

[5 3 2] 判为集合 *3* 的判别函数值 *-5.390449953067446*

[5 3 2] 点最有可能为 *w1* 类

[0 0 0] 判为集合 *1* 的判别函数值 *-4.159899652811455*

[0 0 0] 判为集合 *2* 的判别函数值 *-6.254983175944824*

[0 0 0] 判为集合 *3* 的判别函数值 *-7.949176472004957*

[0 0 0] 点最有可能为 *w1* 类

[1 0 0] 判为集合 *1* 的判别函数值 *-4.158420401033136*

[1 0 0] 判为集合 *2* 的判别函数值 *-6.328371132849032*

[1 0 0] 判为集合 *3* 的判别函数值 *-6.345898334899143*

[1 0 0] 点最有可能为 *w1* 类因为 w_1) 的先验概率相对来说很大，这样的分类结果也比较合理。

3 总结与收获

通过这次实验，我对贝叶斯决策论有了更细致的理解，贝叶斯决策论的基本思想，也就是最小化总风险，总是选择那些能够最小化条件风险 R(a|x) 的行为，尤其是为了最小化分类问题中的误差概率，总是选择那些使后验概率 P(w|x) 最大的类别。