朴素贝叶斯实验报告

信息安全 李涵 1711290

一、问题描述

现有一组数据是对意大利同一地区种植的，但来自三个不同品种的葡萄酒进行化学分析的结果，包括13个属性，拟采用一种分类方法，根据这些属性，对葡萄酒所属的品牌进行预测。

二、解决方法

1.解决思路

使用朴素贝叶斯分类算法进行对这三个品牌的葡萄酒进行分类，而各个属性的取值是连续的，假定他们都符合高斯分布，采用高斯贝叶斯分类器。贝叶斯方法把计算“具有某特征的条件下属于某类”的概率转换成需要计算“属于某类的条件下具有某特征”的概率。我们先预估一个"先验概率"，然后加入实验结果，看这个实验到底是增强还是削弱了"先验概率"，由此得到更接近事实的"后验概率"。

2.基本理论

朴素贝叶斯:

在概率论和统计学中，Bayes’ theorem（贝叶斯法则）根据事件的先验知识描述事件的概率。贝叶斯法则表达式如下所示:

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

P(A|B)： 在事件B下事件A发生的条件概率 P(B|A)： 在事件A下事件B发生的条件概率 P(A), P(B)： 独立事件A和独立事件B的边缘概率

贝叶斯定理的许多应用之一就是贝叶斯推断，一种特殊的统计推断方法，随着信息增加，贝叶斯定理可以用于更新假设的概率。在决策理论中，贝叶斯推断与主观概率密切相关，通常被称为“Bayesian probability(贝叶斯概率)”。

贝叶斯推断根据 prior probability(先验概率) 和统计模型导出的“likelihood function(似然函数)”的结果，再由贝叶斯定理计算 posterior probability(后验概率):

$$P(H|E)=\frac{P(E|H)P(H)}{P(E)}$$

P(H)：已知的先验概率 P(H|E)： 我们想求的后验概率，即在B事件发生后对于事件A概率的评估 P(E|H)： 在事件H下观测到E的概率 P(E)： marginal likelihood(边际似然)，对于所有的假设都是相同的，因此不参与决定不同假设的相对概率

P(E|H)/P(E)： likelihood function(可能性函数)，这是一个调整因子，通过不断的获取信息，可以使得预估概率更接近真实概率

若一个样本有n个特征，分别用

$$x_1, x_2, \ldots, x_n$$

表示，将其划分到类yk 的可能性为

$$P(y_k|x_1, x_2, \ldots, x_n) = P(y_k)\prod_{i=1}^{n} P(x_i|y_k)$$

根据上面的公式求得某个数据属于各个分类的可能性，可以得到最大可能性的分类结果。

高斯贝叶斯:

有些特征可能是连续型变量，比如数据集当中的特征值，这些特征可以转换成离散型的值，比如对数据进行舍入等，不过这些方式都不够细腻，高斯模型可以解决这个问题。

高斯朴素贝叶斯算法是一种特殊类型的NB算法，它特别用于当特征具有连续值时。同时假定所有特征都遵循高斯分布，即正态分布。

高斯模型（一维）假设特征的所有属于某个类别的观测值符合高斯分布，也就是：

$$P(x_i|y_k) = \frac{1}{\sqrt{2\pi\sigma_{y_k}}} e^{-\frac{(x_i - \mu_{y_k})^2}{2\sigma_{y_k}^2}}$$

混淆矩阵（confusion matrix），又称为可能性表格或是错误矩阵。它是一种特定的矩阵用来呈现算法性能的可视化效果，通常是监督学习（非监督学习，通常用匹配矩阵：matching matrix）。其每一列代表预测值，每一行代表的是实际的类别。这个名字来源于它可以非常容易的表明多个类别是否有混淆（也就是一个class被预测成另一个class）。

	实际上为正类	**实际上为负类**
被预测为正类	true positives(**TP** 正类判定为正类)	false positives(**FP** 负类判定为正类,"存伪")
被预测为负类	false negatives(**FN** 正类判定为负类,"去真")	true negatives(**TN** 负类判定为负类,)

通过这张表,我们可以很容易得到这几个值:

准确率：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比。accuracy = TP/(TP+FN+FP+TN)

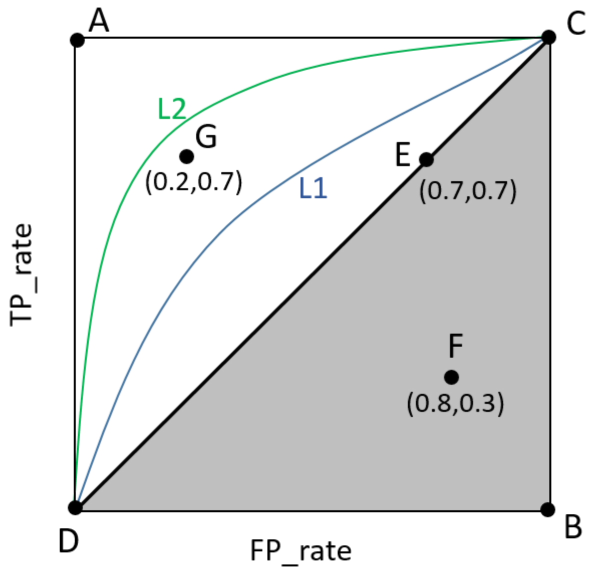
精确度：表示被分为正例的示例中实际为正例的比例。precision = TP/(TP+FP)

召回率：召回率是覆盖面的度量，度量有多个正例被分为正例。recall = TP/(TP+FN)

F值：P和R指标有时候会出现的矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure（又称为F-Score）。 F-Measure是Precision和Recall加权调和平均：

$$F = \frac{(\alpha^2 + 1)P * R}{\alpha^2(P + R)}$$

ROC（Receiver Operating Characteristic）曲线是以假正率（FP_rate）和假负率（TP_rate）为轴的曲线，ROC曲线下面的面积我们叫做AUC，如下图所示：



曲线与FP_rate轴围成的面积（记作AUC）越大，说明性能越好，即图上L2曲线对应的性能优于曲线L1对应的性能。即：曲线越靠近A点（左上方）性能越好，曲线越靠近B点（右下方）曲线性能越差。 位于C-D线上的点说明算法性能和random猜测是一样的–如C、D、E点。位于C-D之上（即曲线位于白色的三角形内）说明算法性能优于随机猜测–如G点，位于C-D之下（即曲线位于灰色的三角形内）说明算法性能差于随机猜测–如F点。 虽然ROC曲线相比较于Precision和Recall等衡量指标更加合理，但是其在高不平衡数据条件下的的表现仍然过于理想，不能够很好的展示实际情况。

如何画ROC曲线：根据每个测试样本属于正样本的概率值从大到小排序，接下来，我们从高到低，依次将“Score”值作为阈值threshold，当测试样本属于正样本的概率大于或等于这个threshold时，我们认为它为正样本，否则为负样本。

3.算法流程

（1）分别计算各个类别的均值和协方差矩阵

（2）计算各个类别本身的概率

（3）计算在各个类别当中，某特征向量出现的概率密度

（4）根据贝叶斯公式，计算在特征向量的条件下，各个类别的概率

（5）比较概率，得出分类结果

三、实验分析

1.实验数据

这些数据是对意大利同一地区种植的，但来自三个不同品种的葡萄酒进行化学分析的结果。分析确定了三种葡萄酒中每一种中的13种成分的数量（13个属性）。

这些属性是1) 酒精 2) 苹果酸 3) 灰 4) 灰的盐度 4) 捐款 5) 镁 6) 总酚类化合物 7) 黄酮类化合物 8) 非黄酮酚 9) 原花青素 10)颜色深度 11)色调 12)稀释葡萄酒的 OD280/OD315 13)脯氨酸

2.实验设计

根据特征矩阵，计算均值、协方差等相关系数

```
def cal_para(x):
    mean=np.zeros((1,col-1))
    for i in range(col-1):
        mean[0,i]=np.mean(x[:,i])
    cov=np.cov(x.T)
    return mean,cov
```

根据均值、协方差和特征向量，计算属于这个类别的概率密度

```
def cal_density(x,mean,cov,n=col-1):
    #求概率密度
    det_cov=np.linalg.det(cov)
    cov_=np.linalg.inv(cov)
    para=1/(pow((2*np.pi),n/2)*pow(det_cov,0.5))
    exponent=-0.5*(x-mean).dot(cov_).dot((x-mean).T)
    return para*pow(np.e,(exponent[0,0]))
```

先计算三个类别各自的概率，再分别计算在三个类别里特征向量的概率密度，最后根据贝叶斯公式，分别计算在这个特征向量下属于三个类别的概率

```
def classify(trainData, labels, features):

    row1=np.sum(labels==1)
    row2=np.sum(labels==2)
    row3=np.sum(labels==3)

    P_y = {}
    P_y[1]=row1/labels.shape[0]
    P_y[2]=row2/labels.shape[0]
    P_y[3]=row3/labels.shape[0]

    #对label==1,2,3分别求均值和协方差矩阵

    X1=trainData[0:row1,:]
    X2=trainData[row1:row1+row2,:]
    X3=trainData[row1+row2:row1+row2+row3,:]

    mean1,cov1=cal_para(X1)
    mean2,cov2=cal_para(X2)
    mean3,cov3=cal_para(X3)

    #条件概率
    P_xy={}
    P_xy[1]=cal_density(features,mean1,cov1)
    P_xy[2]=cal_density(features,mean2,cov2)
    P_xy[3]=cal_density(features,mean3,cov3)

    P={}
    P[1]=P_xy[1]*P_y[1]
    P[2]=P_xy[2]*P_y[2]
    P[3]=P_xy[3]*P_y[3]
```

```
    summ=P[1]+P[2]+P[3]
    P[1]=P[1]/summ
    P[2]=P[2]/summ
    P[3]=P[3]/summ

    pred=max(P, key=P.get)
    return pred,P[pred]  #概率最大值对应的类别，及得分
```

分层采样，进行验证

```
for i in range(10):
    curr1=range(int(i*row1/10),int((i+1)*row1/10))
    curr2=range(row1+int(i*row2/10),row1+int((i+1)*row2/10))
    curr3=range(row1+row2+int(i*row3/10),row1+row2+int((i+1)*row3/10))

    #curr=curr1+curr2+curr3

    X=A[:,1:14].copy()
    Y=A[:,0].copy()

    testX1=X[curr1,:].copy()
    testY1=Y[curr1,:].copy()

    testX2=X[curr2,:].copy()
    testY2=Y[curr2,:].copy()

    testX3=X[curr3,:].copy()
    testY3=Y[curr3,:].copy()

    X=np.delete(X,curr3,0)
    X=np.delete(X,curr2,0)
    X=np.delete(X,curr1,0)

    Y=np.delete(Y,curr3,0)
    Y=np.delete(Y,curr2,0)
    Y=np.delete(Y,curr1,0)


    for j in curr1:
        _pred,_score=classify(X,Y,A[j,1:14])

        if _pred==1:
            correct=correct+1

        reality.append(1)
        pred.append(_pred)
        score.append(_score)
        confusion[0,int(_pred)-1]=confusion[0,int(_pred)-1]+1

        print(1 ,"--",_pred)
```

```
    for j in curr2:
        _pred,_score=classify(X,Y,A[j,1:14])

        if _pred==2:
            correct=correct+1

        reality.append(2)
        pred.append(_pred)
        score.append(_score)
        confusion[1,int(_pred)-1]=confusion[1,int(_pred)-1]+1

        print(2 ,"--",_pred)

    for j in curr3:
        _pred,_score=classify(X,Y,A[j,1:14])

        if _pred==3:
            correct=correct+1

        reality.append(3)
        pred.append(_pred)
        score.append(_score)
        confusion[2,int(_pred)-1]=confusion[2,int(_pred)-1]+1

        print(3 ,"--",_pred)
```

计算准确率、精确率、召回率、和F值

```
accuracy=correct/row
precision={}
precision[1]=confusion[0,0]/np.sum(confusion[0,:])
precision[2]=confusion[1,1]/np.sum(confusion[1,:])
precision[3]=confusion[2,2]/np.sum(confusion[2,:])
recall={}
recall[1]=confusion[0,0]/np.sum(confusion[:,0])
recall[2]=confusion[1,1]/np.sum(confusion[:,1])
recall[3]=confusion[2,2]/np.sum(confusion[:,2])
F={}
F[1]=((pow(alpha,2)+1)*precision[1]*recall[1])/(pow(alpha,2)*(precision[1]+recall[1]))
F[2]=((pow(alpha,2)+1)*precision[2]*recall[2])/(pow(alpha,2)*(precision[2]+recall[2]))
F[3]=((pow(alpha,2)+1)*precision[3]*recall[3])/(pow(alpha,2)*(precision[3]+recall[3]))
```

绘制ROC曲线，并计算AUC值

```
#ROC曲线绘制

import matplotlib.pyplot as plt

def draw_roc(reality,pred,score,row,color):
    temp=np.zeros((row,3))
```

```
    temp[:,0]=np.array(reality)
    temp[:,1]=np.array(pred)
    temp[:,2]=np.array(score)

    sorted_temp=temp[np.lexsort(-temp.T)]

    curr_x=0
    curr_y=0

    node_x=[]
    node_y=[]

    for i in range(row):
        if sorted_temp[i,0]!=sorted_temp[i,1] :
            curr_x=curr_x+1
        else:
            curr_y=curr_y+1

        node_x.append(curr_x)
        node_y.append(curr_y)

    node_x=np.array(node_x)/curr_x
    node_y=np.array(node_y)/curr_y

    plt.plot(node_x,node_y,c=color)

    return np.trapz(node_y,node_x)


auc1=draw_roc(reality1,pred1,score1,row1,'b')
auc2=draw_roc(reality2,pred2,score2,row2,'g')
auc3=draw_roc(reality3,pred3,score3,row3,'r')

plt.show()
```

3.实验结果

混淆矩阵:

$$\begin{bmatrix} 56 & 3 & 0 \\ 1 & 70 & 0 \\ 0 & 1 & 47 \end{bmatrix}$$

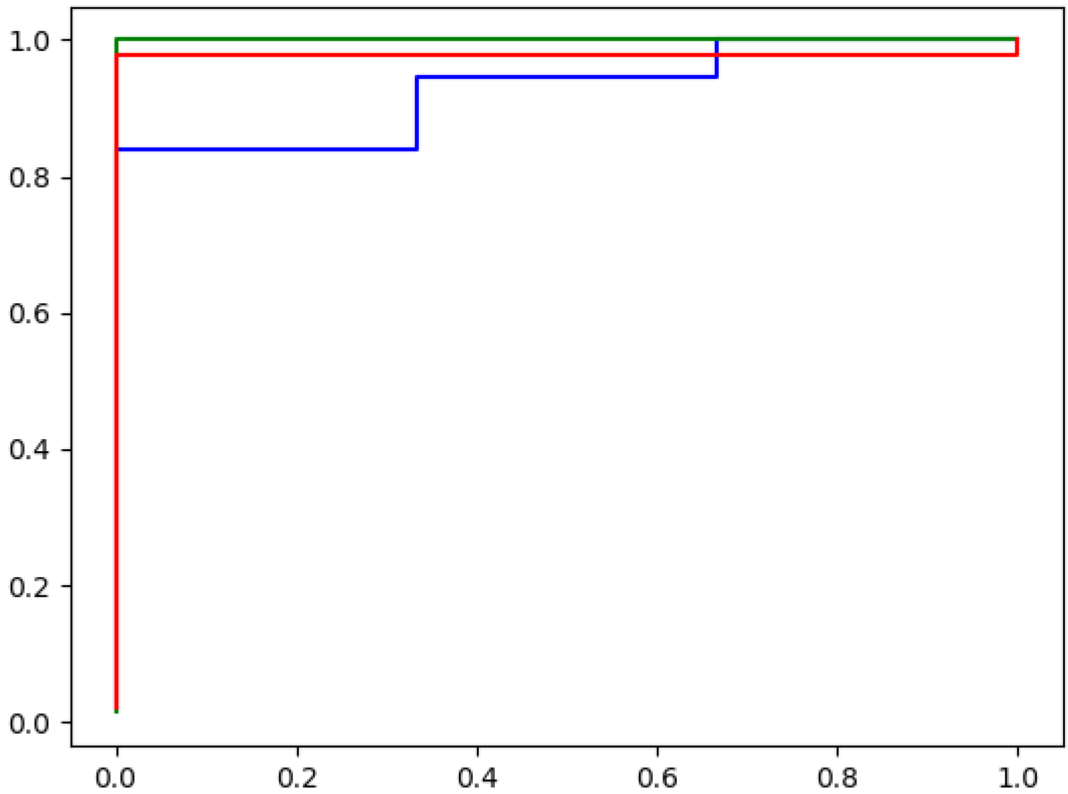
准确率：accuracy = 0.9719101123595506（留一法验证的准确率可达0.9943820224719101）

精确度：precision = {1: 0.9491525423728814, 2: 0.9859154929577465, 3: 0.9791666666666666}

召回率：recall = {1: 0.9824561403508771, 2: 0.9459459459459459, 3: 1.0}

F值：F = {1: 0.9655172413793103, 2: 0.9655172413793103, 3: 0.9894736842105264}

ROC曲线:



auc1 = 0.9285714285714286； auc2 = 1.0； auc3 = 0.9787234042553191