实验要求

截止日期：11月18日

作业的提交格式参考之前的说明，提交到2120220594@nankai.edu.cn

基本要求

a) 采用分层采样的方式将数据集划分为训练集和测试集。 b) 给定编写一个朴素贝叶斯分类器，对测试集进行预测，计算分类准确率。

中级要求

使用测试集评估模型，得到混淆矩阵，精度，召回率，F值。

高级要求

在中级要求的基础上画出三类数据的ROC曲线，并求出AUC值。

分类原理——贝叶斯定理

另A和B表示两个事件

- 条件概率P(A|B)

  事件 A 在事件 B 已经发生条件下的发生概率

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

- 联合概率

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

贝叶斯规则

分类原理——贝叶斯定理

- 贝叶斯公式：给出了从先验概率计算后验概率的方法。

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

- $P(c|x)$：**后验概率**（在给定样本x的条件下，属于类别c的概率）。
- $P(x|c)$：**假设在c类下，观察到样本x的概率**。模式特征x的似然函数（特征x来自于类别c的可能性）。
- $P(c)$：样本为类别c的**先验概率**。在实际应用中，先验概率都是未知的，只能通过背景知识、训练数据等来估计这些概率。这也是贝叶斯方法的难处之一。
- $P(x)$：归一化的证据因子(比例因子)。

- 极大后验概率决策规则：

$$c_{MAP} = argmax_{c_k \in C} P(c_k|x) = argmax_{c_k \in C} \frac{P(x|c_k)P(c_k)}{P(x)} = argmax_{c_k \in C} P(x|c_k)P(c_k)$$

$P(x)$：与类别无关的常量

- 贝叶斯法则：假定数据遵循某种概率分布，通过对概率的分析推理以作出最优的决策。
- 最优决策意味着决策错误率最小的决策；
- 在未观测到模式之前，具有最大先验概率的决策就是最优决策；

$$\text{Decide } c_i \text{ if } P(c_i) > P(c_j), \forall i \neq j$$

分类原理——朴素贝叶斯理论

- 极大后验概率决策规则：

$$c_{MAP} = argmax_{c_k \in C} P(c_k|x) = argmax_{c_k \in C} \frac{P(x|c_k)P(c_k)}{P(x)} = argmax_{c_k \in C} P(x|c_k)P(c_k)$$

$P(x)$：与类别无关的常量

- 贝叶斯法则：假定数据遵循某种概率分布，通过对概率的分析推理以作出最优的决策。
- 最优决策意味着决策错误率最小的决策；
- 在未观测到模式之前，具有最大先验概率的决策就是最优决策；

$$\text{Decide } c_i \text{ if } P(c_i) > P(c_j), \forall i \neq j$$

NB是贝叶斯分类器较为实用的一种，为了减少计算量，朴素贝叶斯分类器假定：**在给定目标值时，各个属性之间相互独立**。那么，属性$x_1, x_2, \cdots, x_D$的联合概率等于每个单独属性概率的乘积。

$$c_{MAP} = argmax_{c_k \in C} P(x|c_k)P(c_k)$$

$$P(x|c_k) = \prod_{d=1}^{D} P(x_d|c_k)$$

$$c_{MAP} = argmax_{c_k \in C} \prod_{d=1}^{D} P(x_d|c_k)\, P(c_k)$$

优缺点

- 优点：计算简单、有稳定的分类效率
- 缺点：
  - 需要估计类别的先验概率，且先验概率很多时候取决于假设，假设的模型可以有很多种，因此在某些时候会由于假设的先验模型不准确导致预测效果不佳。（通过目标值在训练数据中的频率来估计）。
  - 假设属性之间相互独立，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。而在属性相关性较小时，朴素贝叶斯性能最为良好；

实例

分类任务：根据天气状况判断是否打网球

- 条件属性：
  - outlook {sunny, overcast, rainy}
  - temperature {hot, mild, cool}
  - humidity {high, normal}
  - windy {TRUE, FALSE}
- 决策属性：play {yes, no}

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
6	rainy	cool	normal	TRUE	no
8	sunny	mild	high	FALSE	no
14	rainy	mild	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
7	overcast	cool	normal	TRUE	yes
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes

给定< outlook=rainy, temperature=hot, humidity=high, wind=TRUE>, 判断能否打球？

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
6	rainy	cool	normal	TRUE	no
8	sunny	mild	high	FALSE	no
14	rainy	mild	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
7	overcast	cool	normal	TRUE	yes
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes

计算$P(c_k)$

$P(play=no)=\frac{5}{14}$, $P(play=yes)=\frac{9}{14}$

计算$P(x_d|c_k)$

outlook:$P(rainy|no)=\frac{2}{5}, P(rainy|yes)=\frac{3}{9}$

temperature:$P(hot|no)=\frac{2}{5}, P(hot|yes)=\frac{2}{9}$

humidity:$P(high|no)=\frac{4}{5}, P(high|yes)=\frac{3}{9}$

windy: $P(TRUE|no)=\frac{3}{5}, P(TRUE|yes)=\frac{3}{9}$

计算$P(x|c_k)P(c_k)$

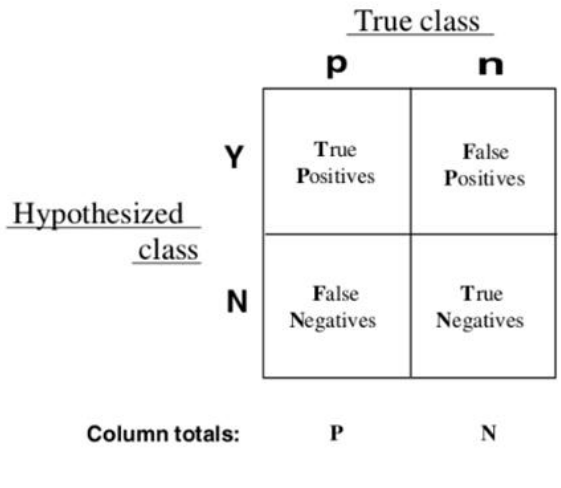
$P(\text{rainy, hot, high, TRUE}|no)P(no)=P(\text{rainy}|no)P(\text{hot}|no)P(high|no)P(\text{TRUE}|no)P(no)$

$=\frac{2}{5}*\frac{2}{5}*\frac{4}{5}*\frac{3}{5}*\frac{5}{14}=\frac{240}{8750}=0.0274$

$P(\text{rainy, hot, high, TRUE}|yes)P(yes)=P(\text{rainy}|yes)P(\text{hot}|yes)P(high|yes)P(\text{TRUE}|yes)P(yes)$

$=\frac{3}{9}*\frac{3}{9}*\frac{2}{9}*\frac{3}{9}*\frac{9}{14}=\frac{486}{91854}=0.00529<0.0274$

混淆矩阵



$\text{fp rate}=\frac{FP}{N}$ 假正率/假报警率

$\text{tp rate}=\frac{TP}{P}$ 真正率/命中率

$\text{precision}=\frac{TP}{TP+FP}$ 精度

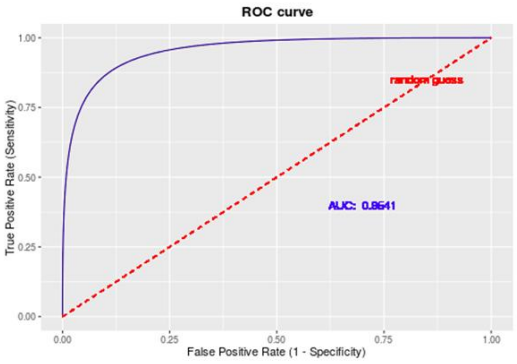
$\text{recall}=\frac{TP}{P}$ 召回率

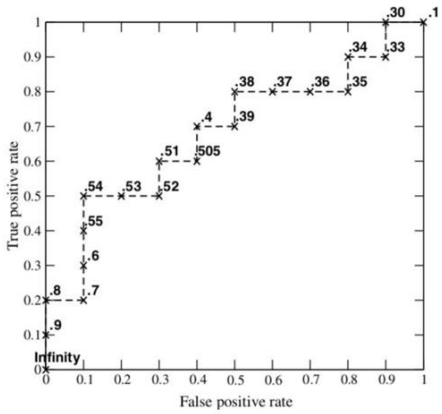
$\text{accuracy}=\frac{TP+TN}{P+N}$ 准确率

$\text{F-measure}=\frac{2}{1/\text{precision}+1/\text{recall}}$ F值

Fig. 1. Confusion matrix and common performance metrics calculated from it.

ROC曲线和AUC





AUC是与TPR和FPR密切相关的,而TPR和FPR分别从正例,负例的角度上去衡量了模型的分类能力(具有跟精准率和召回率一样的能在样本极端不平衡的情况下进行有效的衡量优势),因此在样本极端不平衡的情况下,AUC依然能够做出合理的评价.

初级要求

a) 采用分层采样的方式将数据集划分为训练集和测试集。 b) 给定编写一个朴素贝叶斯分类器，对测试集进行预测，计算分类准确率。

In [21]:

```
# -*- coding: UTF-8 -*-
# 在wine数据集中，这些数据包括了三种酒中13种不同成分的数量。
# 文件中，每行代表一种酒的样本，共有178个样本；
# 一共有14列，其中，第一个属性是类标识符，分别是1/2/3 来表示，代表葡萄酒的三个分类。
# 后面的13列为每个样本的对应属性的样本值。
# 其中第1类有59个样本，第2类有71个样本，第3类有48个样本。
# 一个整数label和13个属性值

import math
import numpy as np
import pandas as pd
import random
from random import sample

f = open('wine.data','r')
type_data = [[],[],[]]                          #按类分的所有数据
test_data = [[],[],[]]
train_data = [[],[],[]]
data_num = 0                                    #数据总数
test_len = []                                   #测试集里每一类的个数
means = [[],[],[]]                              #每一类的均值
std = [[],[],[]]                                #每一类的标准差
myline = '1'
while myline:
    myline = f.readline().split(',')
    if len(myline) != 14:
        break
    # 类别存为int，属性变为float
    for t in range(len(myline)):
        if t == 0:
            myline[t] = int(myline[t])
        else:
            myline[t] = float(myline[t])
    temp = myline.pop(0)
    type_data[temp - 1].append(myline)
test_len = [round(len(type_data[i]) / 4) for i in range(3)]
data_num = sum([len(type_data[i]) for i in range(3)])
```

In [22]:
```
# 分层抽样分割数据集，data为数据集，len为存测试集长度的list
def sample_split(data,len_data):
    test_data = [[],[],[]]
    train_data = [[],[],[]]
    for i in range(3):
        test_data[i]=sample(data[i],len_data[i])
        for temp in test_data[i]:
            data[i].remove(temp)
        train_data[i]=data[i]
    return test_data,train_data
```

In [23]:
```
# 计算概率
def prob(data,avg,sig):
    # print(data)
    sqrt_2pi=np.power(2*np.pi,0.5)
    coef=1/(sqrt_2pi*sig)
    powercoef=-1/(2*np.power(sig,2))
    mypow=powercoef*(np.power((data-avg),2))
    p=coef*(np.exp(mypow))
    for i in range(len(p)):
        p[i]=math.log(p[i])
    return sum(p)
```

In [24]:
```
def bayes_classificate(data_set,len_data,num):
    # print(type(data_set[0]))
    # print(len(data_set))
    # data_set在运算过程中会变

    # 存预测概率
    p_score=[[],[],[]]
    # 先验概率
    p=[]
    for i in range(3):
        p.append(len(data_set[i])/num)
    # print(p)
    test_data,train_data=sample_split(data_set,len_data)
    # 首先，分别计算训练集上三个类的均值和标准差
    mean=[]
    std=[]
    # 存储预测得到的值
    y_predict=[[],[],[]]
    # print(data)
    # print(len(data_set[0]))
    test_num = sum([len(test_data[i]) for i in range(3)])

    for i in range(3):
        train_data[i]=np.array(train_data[i])
        mean.append(train_data[i].mean(axis=0))
        std.append(np.std(train_data[i], axis=0))
        test_data[i]=np.array(test_data[i])
    # print(mean)
    # print(std)
    wrong_num = 0
    for i in range(3):
        for t in test_data[i]:                    #两层循环：从每一类取每一个测试样本
            my_type = []
            for j in range(3):
                #由于数据集中所有的属性都是连续值，连续值的似然估计可以按照高斯分布求
                # temp = ...
                # t每一维度算log再相加
```

```
                temp=prob(t,mean[j],std[j])+len(t)*math.log(p[j])
                # print(temp)
                my_type.append(temp)                                    #这里将所有score保存
                # 样本被预测为第j类的概率
                p_score[j].append(temp)
            pre_type = my_type.index(max(my_type))                      #取分值最大的为预测类别
            y_predict[i].append(pre_type)
            # print(pre_type)
            if pre_type != i:                                           #统计错误数
                wrong_num+=1
    error_rate=round(wrong_num/test_num,3)
    print("分类准确率：{:f}".format(1-error_rate))
    return error_rate,y_predict,p_score
```

In [25]:
```
error_rate,y_predict,p_score=bayes_classificate(type_data,test_len,data_num)
```

```
分类准确率：0.978000
```

中级要求

使用测试集评估模型，得到混淆矩阵，精度，召回率，F值。

In [26]:
```
# y_predict为3×n类型
def evaluate_model(y_predict):
    x=np.zeros((3,3))
    # confusion_matrix=np.mat(x)
    # test_num = sum([len(y_predict[i]) for i in range(3)])
    wrong_num=0
    precition=[]
    recall=[]
    for i in range(3):
        for pre in y_predict[i]:
            x[i][pre]+=1
            if(pre!=i):
                wrong_num+=1
    for i in range(x.shape[0]):
        precition.append(round(x[i][i]/sum([x[j][i] for j in range(x.shape[0])]),3))
        recall.append(round(x[i][i]/sum([x[i][j] for j in range(x.shape[1])]),3))
    P=np.mean(precition)
    R=np.mean(recall)
    F1=2*P*R/(P+R)
    confusion_matrix=np.mat(x)
    print("混淆矩阵为：",confusion_matrix)
    print("精度为 {:f},召回率为 {:f},F值为 {:f}".format(P,R,F1))
    return confusion_matrix,P,R,F1
```

In [27]:
```
evaluate_model(y_predict)
```

```
混淆矩阵为： [[15.  0.  0.]
 [ 0. 18.  0.]
 [ 0.  1. 11.]]
精度为 0.982333,召回率为 0.972333,F值为 0.977308
```

Out[27]:
```
(matrix([[15.,  0.,  0.],
         [ 0., 18.,  0.],
         [ 0.,  1., 11.]]),
 0.9823333333333334,
 0.9723333333333333,
 0.9773077535243292)
```

高级要求

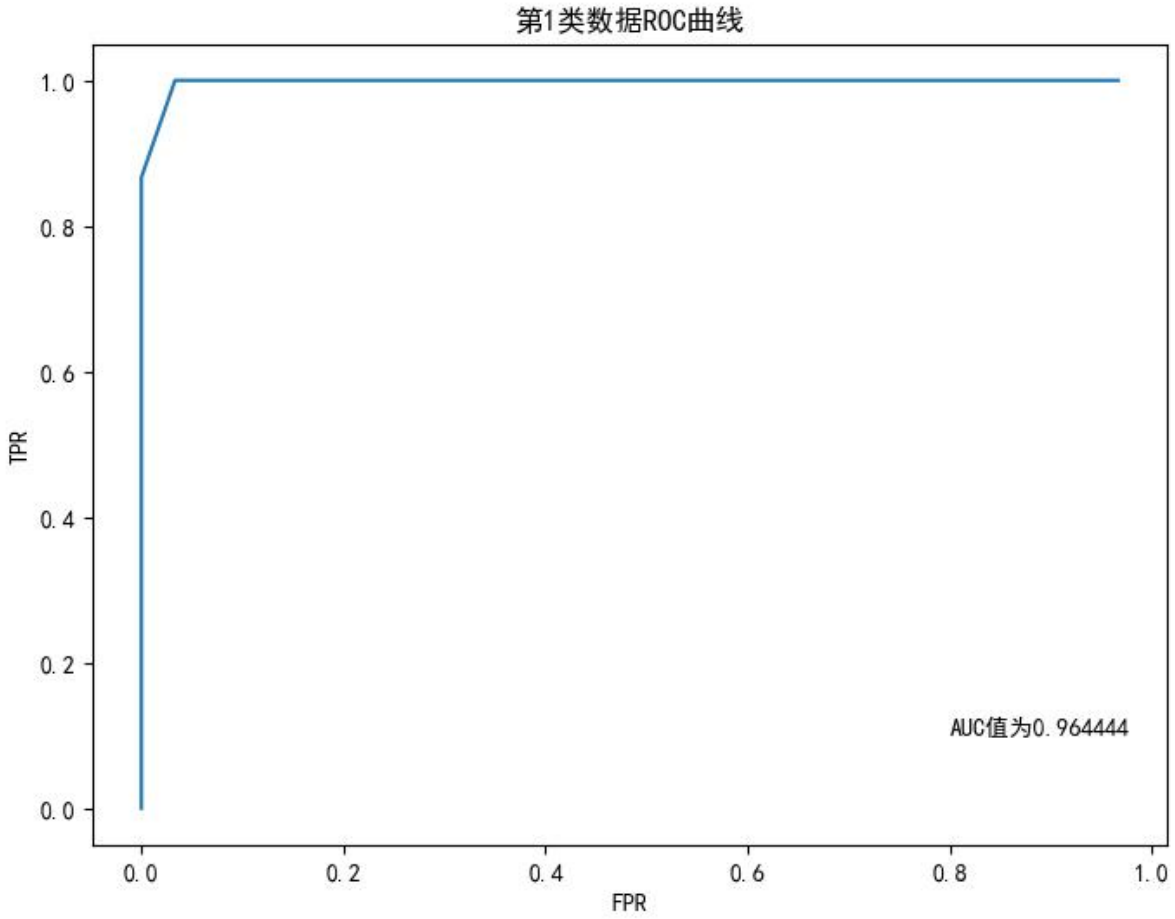
在中级要求的基础上画出三类数据的ROC曲线，并求出AUC值

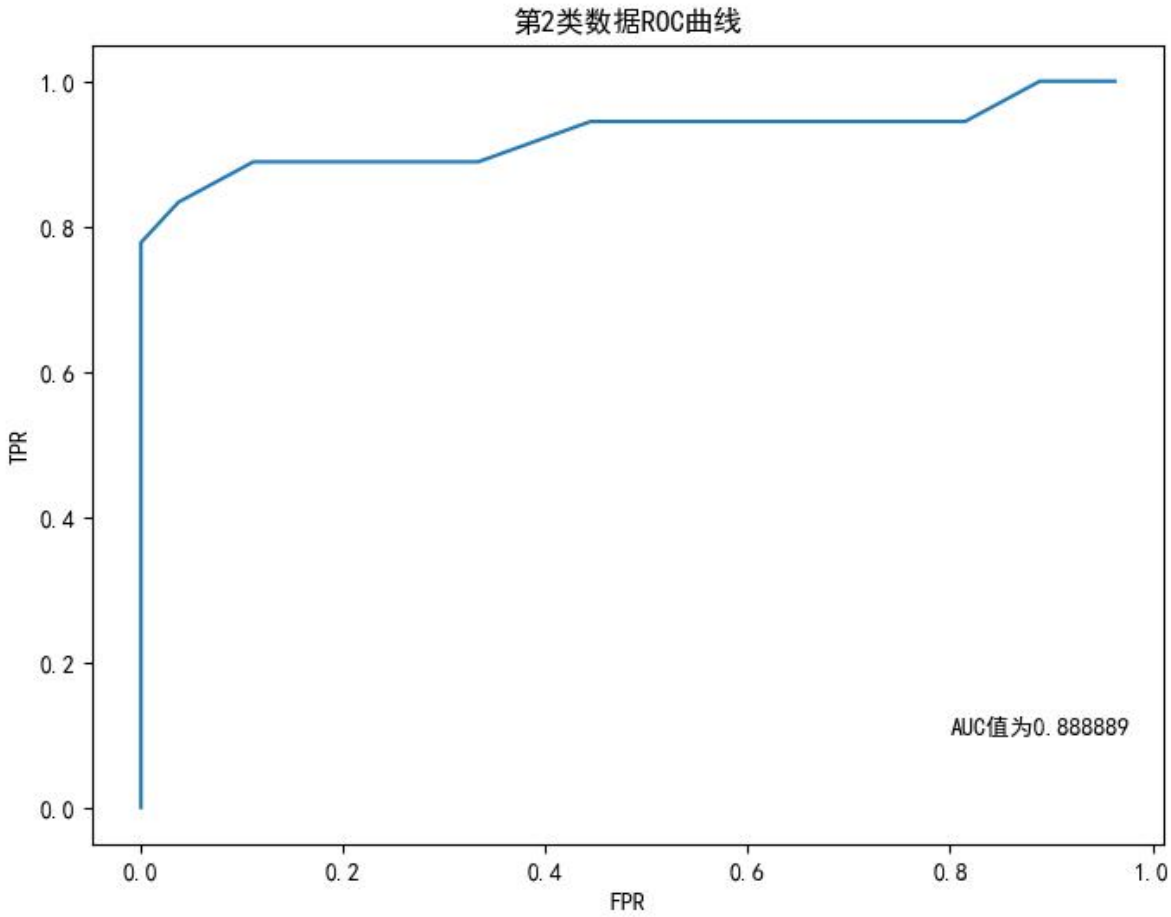
In [28]:
```
# 把p_score做归一化
def normalization(score):
    for i in range(3):
        max_v=max(score[i])
        min_v=min(score[i])
        distance=max_v-min_v
        # print(max_v)
        for j in range(len(score[i])):
            score[i][j]=(score[i][j]-min_v)/distance
    return score
```

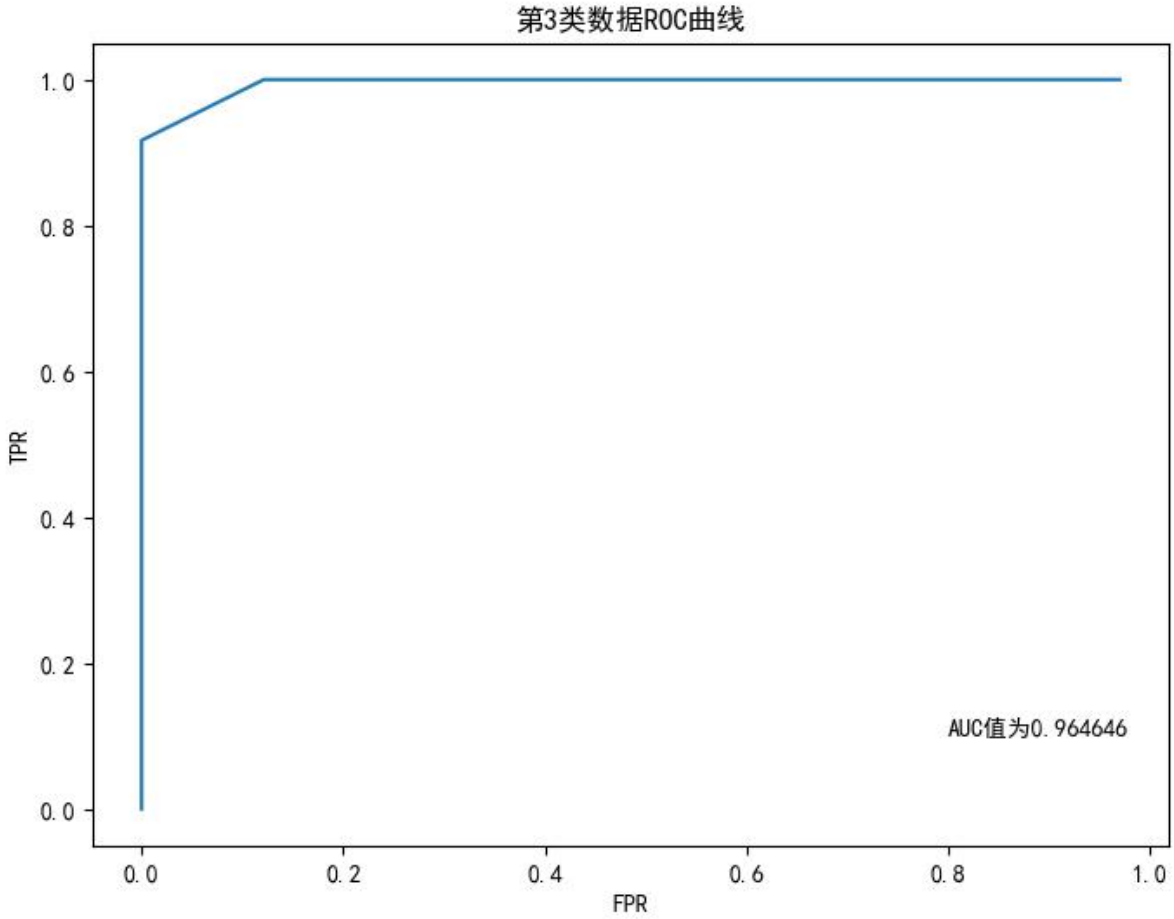
In [29]:
```
# p_score 是预测概率，levels是阈值，data_len存每类数据的数目，返回点对
def ROC_AUC(p_score,levels,data_len):
    p_score=normalization(p_score)
    x=[[],[],[]]
    y=[[],[],[]]

    for i in range(3):
        true_num=data_len[i]
        false_num=sum(data_len)-true_num
        start=0
        end=start+data_len[i]
        if i!=0:
            start=sum(data_len[j] for j in range(i))
            end=start+data_len[i]
        # true_p=[]
        false_p=[]
        true_p=p_score[i][start:end]
        for j in range(sum(data_len)):
            if j not in range(start,end):
                false_p.append(p_score[i][j])


        for level in levels:
            # print("true_p")
            tempy=sum(value>level for value in true_p)
            tempx=sum(value>level for value in false_p)
            # print("tempx",tempx)
            # print("tempy",tempy)
            tempx/=false_num
            tempy/=true_num
            x[i].append(tempx)
            y[i].append(tempy)
    return x,y
```

In [30]:
```
import matplotlib.pyplot as plt
```

```
levels=np.linspace(1,0,20)
x,y=ROC_AUC(p_score,levels,test_len)
# 绘制图形
for i in range(3):
    # 解决中文显示问题
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False

    plt.figure(figsize=(8,6))  # 定义图的大小
    plt.xlabel("FPR")      # X轴标签
    plt.ylabel("TPR")         # Y轴坐标标签
    plt.title("第{:d}类数据ROC曲线".format(i+1))       #  曲线图的标题
    plt.plot(x[i],y[i])              # 绘制曲线图
    plt.text(0.8,0.1,"AUC值为{:f}".format(np.trapz(y[i], x[i], dx = 0.05)))
    #在ipython的交互环境中需要这句话才能显示出来
    plt.show()
# 计算AUC值，也即曲线积分
```




第2类数据ROC曲线
1.0
0.8
0.6
0.4
0.2
0.0
TPR
0.0
0.2
0.4
0.6
0.8
1.0
FPR
AUC值为0.888889


第3类数据ROC曲线
1.0
0.8
0.6
0.4
0.2
0.0
TPR
0.0
0.2
0.4
0.6
0.8
1.0
FPR
AUC值为0.964646