贝叶斯估计

贝叶斯估计给出了从先验概率计算后验概率的基本发生，其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。并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，而本次实验的基本方法也来源于此：

数据准备

对wine.data进行数据载入，并进行对数据集与验证集的划分：

```
dataset=pd.read_csv('wine.data')
dataset=np.array(dataset)
np.random.shuffle(dataset)
train_dataset,test_dataset=dataset[:round(0.7*len(dataset))],dataset[round(0.7*l
en(dataset)):len(dataset)]
```

计算

在这一过程中，因为数据均为连续值，因此可以选择使用使用正态分布计算连续值的先验概率，这一过程因此不需要考虑平滑化问题。而是直接计算出训练集合中各个类别的基本统计信息后完成对基本信息的记录过程。

```
mean=[]
variable=[]
train_dataset=sorted(train_dataset,key=lambda x:x[0])
index1,index2=0,0
for i in range(len(train_dataset)):
    if(train_dataset[i][0])==1:
        index1=i+1
    elif(train_dataset[i][0])==2:
        index2=i+1

print(train_dataset[index1])
index_map=[0,index1,index2,len(train_dataset)]
train_dataset=pd.DataFrame(train_dataset)
print(train_dataset)
for index in range(0,3):
    for i in range(0,13):
        mean_x=np.mean(train_dataset[i][index_map[index]:index_map[index+1]])
        variable=np.var(train_dataset[i]
[index_map[index]:index_map[index+1]],ddof=1)
        record_mean[index][i-1]=mean_x
        record_var[index][i-1]=variable
print(record_mean)
print(record_var)
```

而后针对验证集合中样本的各个属性，进行对似然函数的计算：

```
def normal_arrtribution(x,mean,var):
    return stats.norm.pdf(x, mean, var)
def calculate_likelyhood(data,group):
    possible=[]
    for i in range(1,13):
        possible.append(normal_arrtribution(data[i],record_mean[group][i-
1],record_var[group][i-1]))
    likelyhood=1
    for iternum in range(0,len(possible)):
        likelyhood= likelyhood*possible[iternum]
    return likelyhood
```

最后针对验证集的结果，可以完成对统计信息的输出：

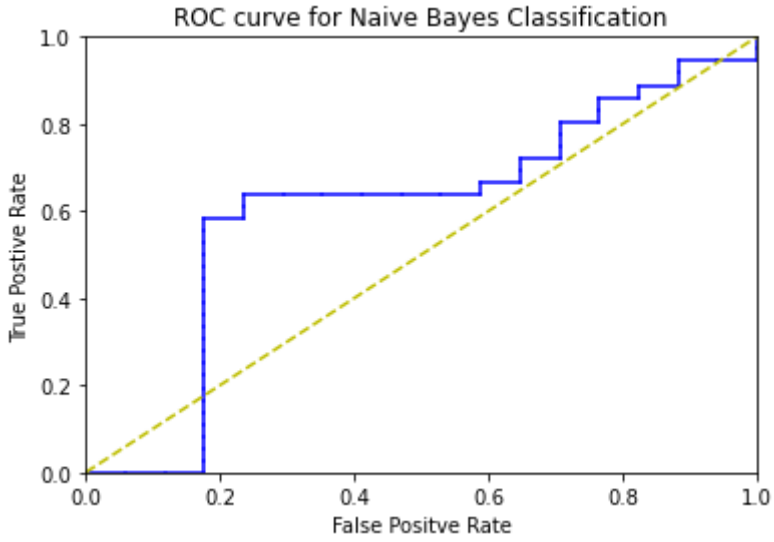
```
confusion_matrix=np.zeros([3,3])
def test_set_cul():
    for index in range(0,len(test_dataset)):
        caled=[]
        for iterable in range(0,3):
            caled_num=calculate_likelyhood(test_dataset[index],iterable)
            caled.append(caled_num)
        res=np.argmax(caled) #Hypotheized
        true_class=round(test_dataset[index][0])-1
        confusion_matrix[res][true_class]=confusion_matrix[res][true_class]+1
    print(confusion_matrix)
test_set_cul()
```

与此同时，完成对F-measure的计算：

```
def precision(type):
    return confusion_matrix[type][type]/sum(confusion_matrix[type,:])
def recall(type):
    return confusion_matrix[type][type]/sum(confusion_matrix[:,type])
def F_ratio(type):
    prec=precision(type)
    print('The precison of type {:n} is {:%}'.format(type, prec))
    reca=recall(type)
    print('The recall of type {:n} is {:%}'.format(type, reca))
    F=2/((1/prec)+(1/reca))
    print('The F-measure of type {:n} is {:%}'.format(type, F))
    print('\n')
for i in np.arange(3):
    F_ratio(i)
```

可以发现，在计算结果中，类别1的准确率较高但召回率较低，这也导致了其F-eature较低。而类别2的召回率较高却准确率较低，使得其F-measure较低，而类别3的准确率与召回率均处于较高水准。

而针对**高级要求**中提出的ROC曲线，需要根据不同的概率阈值对FPR与TPR进行处理，在本次实验中，在对不同次数的实验置信度得分进行排序后，分别计算TPR与FPR，得到了最终的结果，并画出AUC图如下：



总结

在本次实验中，使用朴素贝叶斯的方法，针对葡萄酒的分类问题进行了求解，完成了初级与中级与高级要求，计算出了该模型的混淆矩阵与AUC矩阵，并取得了较好的结果，但在实验过程中，会由于数据集或方法对独立性的一些假设限制，出现一些问题，这也是需要进一步解决与优化的。