**第3章 朴素贝叶斯**

**一、简述朴素贝叶斯的原理、实现的关键操作点（不超过300字，逻辑要通，要有小标题，要排版清晰，请不要网络大幅摘抄）**

1. **朴素贝叶斯原理**

朴素贝叶斯的过程主要分为两个阶段。第一阶段，对实验样本进行分类，分别计算不同条件下其概率。第二阶段，输入测试样本，计算不同的类条件改率，比较其概率大小，从而完成对测试样本的分类。

实现样本的分类，需要通过计算条件概率而得到，计算条件概率的方法称为贝叶斯准则。条件概率的计算方法：

1. **关键操作点**
2. 计算先验概率：统计训练集中每个类别的出现频率，作为先验概率。
3. 计算条件概率：对于每个特征，计算其在每个类别下的条件概率。
4. 特征独立性假设：假设特征之间相互独立，进行对于类条件概率的计算。
5. 后验概率计算：利用贝叶斯定理，结合先验概率和条件概率，计算样本属于每个类别的后验概率。
6. 分类决策：选择后验概率最大的类别作为预测结果。
7. **实现要点**
8. 数据预处理：对离散特征进行编码，对连续特征进行离散化或假设其服从某种分布（如高斯分布）。
9. 概率平滑处理：使用拉普拉斯平滑处理零概率问题，避免因未出现的特征导致概率为零，将离散的数据进行相应的平滑操作，对于概率的计算和。
10. 模型训练：统计训练数据中的概率分布，构建概率表。
11. 预测：对新样本，根据概率表计算其属于各类别的概率，选择最大概率对应的类别。

**二、利用朴素贝叶斯解决问题（预测或分类）。**

利用朴素贝叶斯对鸢尾花数据进行分类。

要求：

1）针对当前数据、模型，**详叙数据预处理、模型参数调整过程及效果比对**；

2）代码要列出，重点代码加注释说明，特别是自己调试过程中的自我理解；

3）运行结果要截图，结果要文字说明；

4）注意排版

**数据预处理部分：**

def open\_wwcsv():  
 readbook = xlrd.open\_workbook('./iris.xls') # 打开csv文件  
 sheet = readbook.sheet\_by\_index(0) # 获取表格  
 return sheet

**特征数据提取、并且对其进行标签化：**

def invalue(sheet,nrows) :  
 iris = []  
 y = []  
 # v1,v2,v3,v4 , name  
 for i in range(1, nrows):  
 v1 = sheet.cell(rowx=i, colx=1).value  
 v2 = sheet.cell(rowx=i, colx=2).value  
 v3 = sheet.cell(rowx=i, colx=3).value  
 v4 = sheet.cell(rowx=i, colx=4).value  
 name = sheet.cell(rowx=i, colx=5).value  
 iris.append([v1, v2, v3, v4,name])  
 y.append(name)  
 return iris, y

**数据划分为训练集和测试集**

def train\_test\_split\_custom(x, y, train\_size=0.8, random\_state=None):  
 if random\_state is not None: # 如果指定了随机种子，则设置随机种子以保证结果可重复。  
 np.random.seed(random\_state)  
 indices = np.arange(len(x)) # 创建一个从0到样本数-1的索引数组。  
 np.random.shuffle(indices) # 随机打乱索引顺序。  
 train\_size = int(len(x) \* train\_size) # 计算训练集样本数。  
 train\_indices = indices[:train\_size] # 获取训练集索引。  
 test\_indices = indices[train\_size:] # 获取测试集索引。  
 x\_train = x[train\_indices] # 根据索引提取训练集特征。  
 y\_train = y[train\_indices] # 根据索引提取训练集标签。  
 x\_test = x[test\_indices] # 根据索引提取测试集特征。  
 y\_test = y[test\_indices] # 根据索引提取测试集标签。  
 return x\_train, x\_test, y\_train, y\_test # 返回训练集和测试集。  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split\_custom(x, y, train\_size=0.8, random\_state=14) # 划分数据集。

**计算均值和公差并且进行概率密度计算为类条件概率做准备**

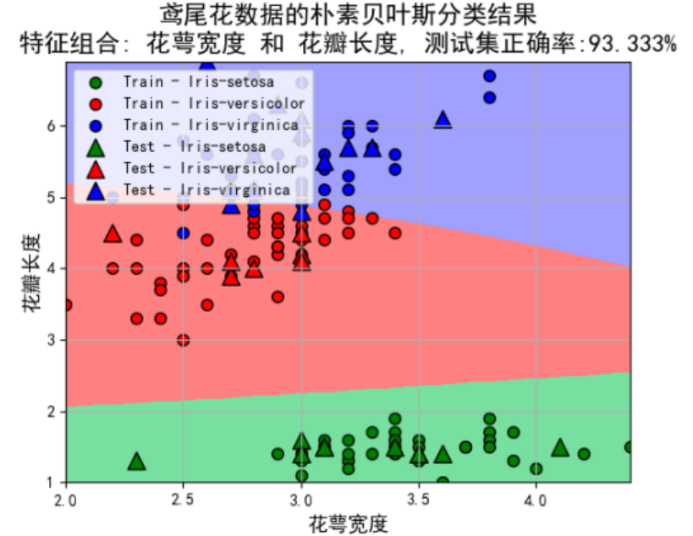
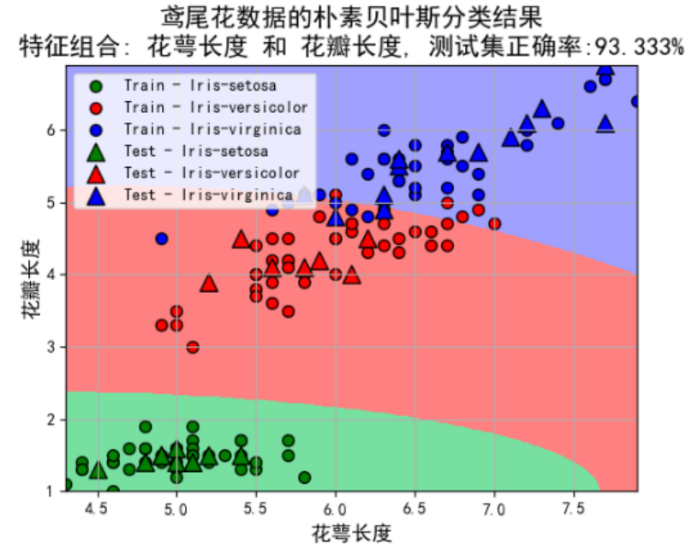
scaler = StandardScaler()  
x\_train = scaler.fit\_transform(x\_train)  
x\_test = scaler.transform(x\_test)

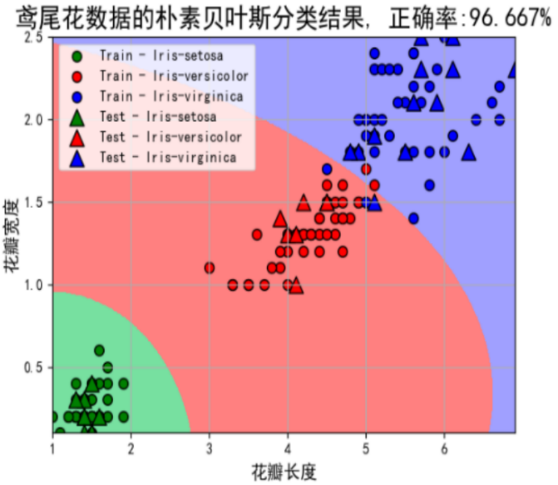
x1\_min1, x2\_min1 = x\_train.min(axis=0) # 训练集特征最小值。  
x1\_max1, x2\_max1 = x\_train.max(axis=0) # 训练集特征最大值。  
x1\_min2, x2\_min2 = x\_test.min(axis=0) # 测试集特征最小值。  
x1\_max2, x2\_max2 = x\_test.max(axis=0) # 测试集特征最大值。  
x1\_min = np.min((x1\_min1, x1\_min2)) # 综合最小值。  
x1\_max = np.max((x1\_max1, x1\_max2)) # 综合最大值。  
x2\_min = np.min((x2\_min1, x2\_min2)) # 综合最小值。  
x2\_max = np.max((x2\_max1, x2\_max2)) # 综合最大值。

**根据不同的比例和特征值给出不同的分类标准和分类状态**

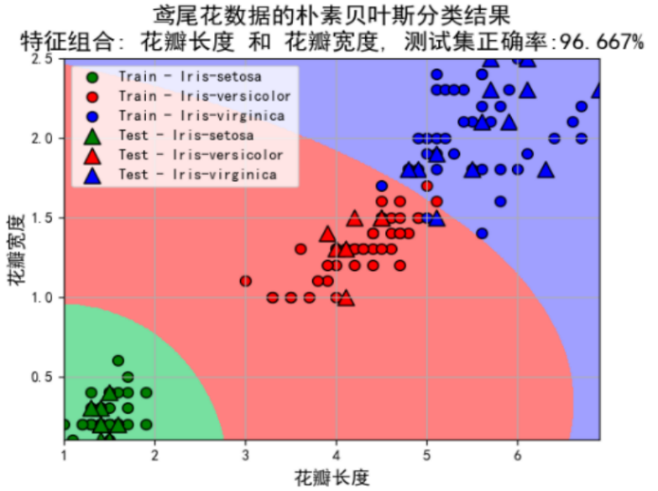
# 尝试不同的特征组合  
features\_combinations = [  
 [0, 1], # 花萼长度和花萼宽度  
 [0, 2], # 花萼长度和花瓣长度  
 [0, 3], # 花萼长度和花瓣宽度  
 [1, 2], # 花萼宽度和花瓣长度  
 [1, 3], # 花萼宽度和花瓣宽度  
 [2, 3] # 花瓣长度和花瓣宽度（默认组合）  
]  
  
best\_accuracy = 0  
best\_features = None  
  
for features in features\_combinations:  
 x = data[features].values # 提取当前特征组合。  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split\_custom(x, y, train\_size=0.8, random\_state=14)  
  
 # 训练模型  
 clf = GaussianNaiveBayes()  
 clf.fit(x\_train, y\_train)  
  
 # 计算准确率  
 y\_test\_hat = clf.predict(x\_test)  
 accuracy = accuracy\_score\_custom(y\_test, y\_test\_hat)  
  
 print(f"Feature combination: {features}, Test Accuracy: {测试集准确度:.2f}%")  
  
 if accuracy > best\_accuracy:  
 best\_accuracy = accuracy  
 best\_features = features  
  
print(f"Best feature combination: {最佳特征组合}, Best Accuracy: {最佳测试集准确率:.2f}%")

train\_sizes = [0.6, 0.7, 0.8, 0.9] # 不同的训练集比例。  
best\_accuracy = 0  
best\_train\_size = None  
  
for train\_size in train\_sizes:  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split\_custom(x, y, train\_size=train\_size, random\_state=14)  
  
 # 标准化数据  
 scaler = StandardScaler()  
 x\_train = scaler.fit\_transform(x\_train)  
 x\_test = scaler.transform(x\_test)  
  
 # 训练模型  
 clf = GaussianNaiveBayes()  
 clf.fit(x\_train, y\_train)  
  
 # 计算准确率  
 y\_test\_hat = clf.predict(x\_test)  
 accuracy = accuracy\_score\_custom(y\_test, y\_test\_hat)  
  
 print(f"Train size: {train\_size}, Test Accuracy: {accuracy:.2f}%")  
  
 if accuracy > best\_accuracy:  
 best\_accuracy = accuracy  
 best\_train\_size = train\_size  
  
print(f"Best train size: {best\_train\_size}, Best Accuracy: {best\_accuracy:.2f}%")

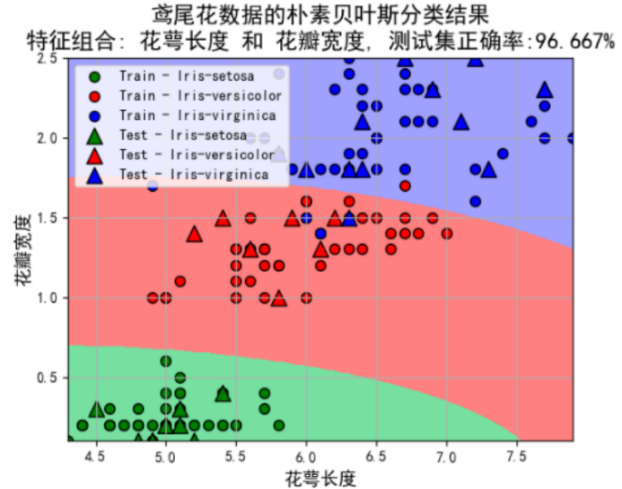




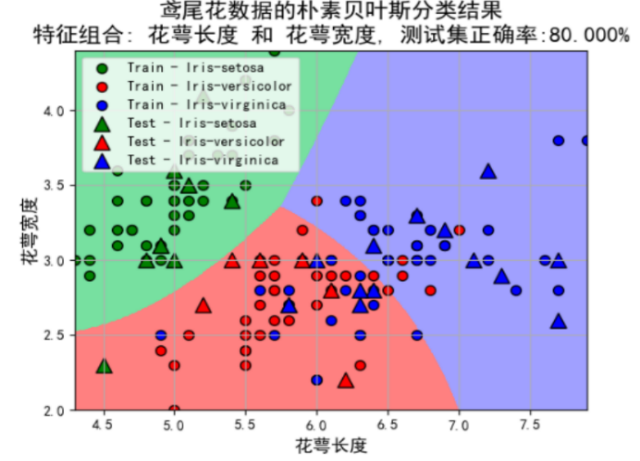
图表 1 未进行参数调整时分类结果



图表 2 花瓣长度和花瓣宽度分类结果

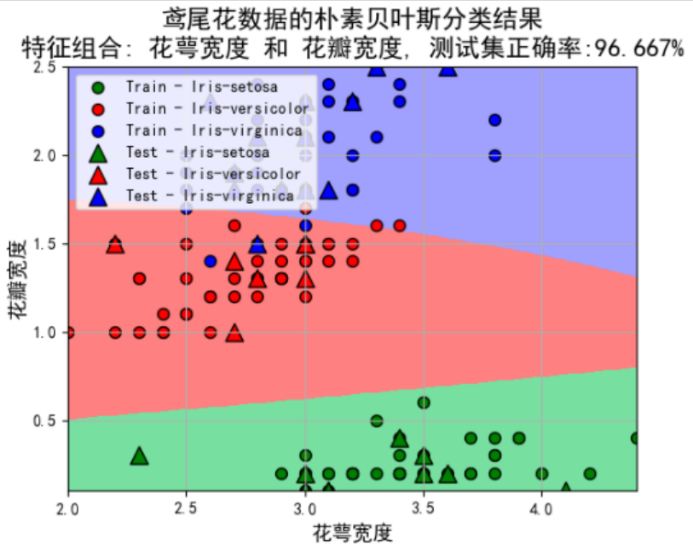


图表 3 花萼长度和花瓣宽度分类结果



图表 4 花萼长度和花萼宽度分类结果

图表 7 花萼宽度和花瓣宽度分类结果



图表 5 花萼长度和花瓣长度分类结果

图表6花萼宽度和花瓣长度分类结果

**三、[附加题].利用朴素贝叶斯解决自己遇到的问题。**

[解题要求同题2]