目录

**[（1）3.5 3](#_Toc32096)**

[1.代码： 3](#_Toc26592)

[2.运行结果： 8](#_Toc24845)

**[（2）4.10 9](#_Toc20270)**

[1. 代码： 9](#_Toc11456)

[2. 运行结果： 19](#_Toc15703)

**[（3）5.10 19](#_Toc27677)**

[1. 代码： 19](#_Toc26842)

[2. 运行结果： 21](#_Toc5990)

**[（4）6.8 21](#_Toc7595)**

[1. 代码： 21](#_Toc5263)

[2. 运行结果： 24](#_Toc5238)

**[（5）7.3 24](#_Toc8699)**

[1. 代码： 24](#_Toc24814)

[2. 运行结果： 33](#_Toc32767)

**[（6）8.5 33](#_Toc6078)**

[1. 代码： 33](#_Toc6047)

[2. 运行结果： 38](#_Toc4305)

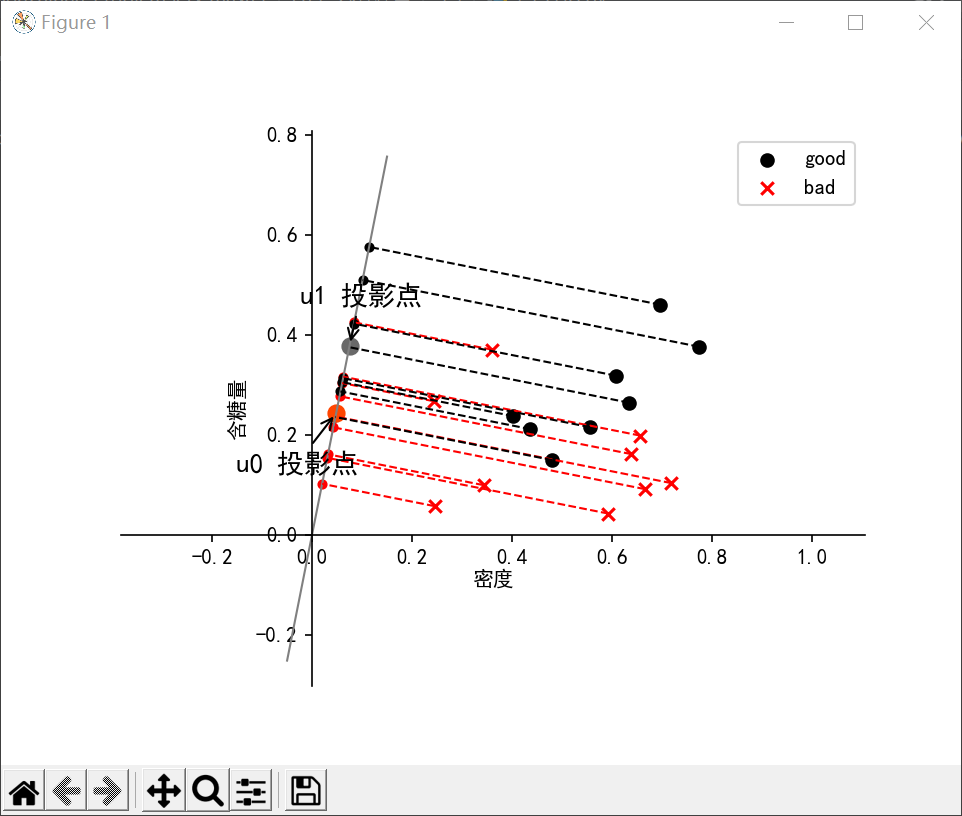
## 

# （1）3.5

## 1.代码：

import numpy as np  
import pandas as pd  
from matplotlib import pyplot as plt  
  
  
class LDA(object):  
  
 def fit(self, X\_, y\_, plot\_=False):  
 pos = y\_ == 1  
 neg = y\_ == 0  
 X0 = X\_[neg]  
 X1 = X\_[pos]  
  
 u0 = X0.mean(0, keepdims=True) # (1, n)  
 u1 = X1.mean(0, keepdims=True)  
  
 sw = np.dot((X0 - u0).T, X0 - u0) + np.dot((X1 - u1).T, X1 - u1)  
 w = np.dot(np.linalg.inv(sw), (u0 - u1).T).reshape(1, -1) # (1, n)  
  
 if plot\_:  
 fig, ax = plt.subplots()  
 ax.spines['right'].set\_color('none')  
 ax.spines['top'].set\_color('none')  
 ax.spines['left'].set\_position(('data', 0))  
 ax.spines['bottom'].set\_position(('data', 0))  
  
 plt.scatter(X1[:, 0], X1[:, 1], c='k', marker='o', label='good')  
 plt.scatter(X0[:, 0], X0[:, 1], c='r', marker='x', label='bad')  
  
 plt.xlabel('密度', labelpad=1)  
 plt.ylabel('含糖量')  
 plt.legend(loc='upper right')  
  
 x\_tmp = np.linspace(-0.05, 0.15)  
 y\_tmp = x\_tmp \* w[0, 1] / w[0, 0]  
 plt.plot(x\_tmp, y\_tmp, '#808080', linewidth=1)  
  
 wu = w / np.linalg.norm(w)  
  
 # 正负样板店  
 X0\_project = np.dot(X0, np.dot(wu.T, wu))  
 plt.scatter(X0\_project[:, 0], X0\_project[:, 1], c='r', s=15)  
 for i in range(X0.shape[0]):  
 plt.plot([X0[i, 0], X0\_project[i, 0]], [X0[i, 1], X0\_project[i, 1]], '--r', linewidth=1)  
  
 X1\_project = np.dot(X1, np.dot(wu.T, wu))  
 plt.scatter(X1\_project[:, 0], X1\_project[:, 1], c='k', s=15)  
 for i in range(X1.shape[0]):  
 plt.plot([X1[i, 0], X1\_project[i, 0]], [X1[i, 1], X1\_project[i, 1]], '--k', linewidth=1)  
  
 # 中心点的投影  
 u0\_project = np.dot(u0, np.dot(wu.T, wu))  
 plt.scatter(u0\_project[:, 0], u0\_project[:, 1], c='#FF4500', s=60)  
 u1\_project = np.dot(u1, np.dot(wu.T, wu))  
 plt.scatter(u1\_project[:, 0], u1\_project[:, 1], c='#696969', s=60)  
  
 ax.annotate(r'u0 投影点',  
 xy=(u0\_project[:, 0], u0\_project[:, 1]),  
 xytext=(u0\_project[:, 0] - 0.2, u0\_project[:, 1] - 0.1),  
 size=13,  
 va="center", ha="left",  
 arrowprops=dict(arrowstyle="->",  
 color="k",  
 )  
 )  
  
 ax.annotate(r'u1 投影点',  
 xy=(u1\_project[:, 0], u1\_project[:, 1]),  
 xytext=(u1\_project[:, 0] - 0.1, u1\_project[:, 1] + 0.1),  
 size=13,  
 va="center", ha="left",  
 arrowprops=dict(arrowstyle="->",  
 color="k",  
 )  
 )  
 plt.axis("equal") # 两坐标轴的单位刻度长度保存一致  
   
 plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文标签  
 plt.rcParams['axes.unicode\_minus'] = False # 设置正常显示符号  
  
 plt.show()  
  
 self.w = w  
 self.u0 = u0  
 self.u1 = u1  
 return self  
  
 def predict(self, X):  
 project = np.dot(X, self.w.T)  
  
 wu0 = np.dot(self.w, self.u0.T)  
 wu1 = np.dot(self.w, self.u1.T)  
  
 return (np.abs(project - wu1) < np.abs(project - wu0)).astype(int)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data\_path = r'D:\machinelearning\watermelon3\_0\_Ch.csv'  
  
 data = pd.read\_csv(data\_path).values  
  
 X = data[:, 7:9].astype(float)  
 y = data[:, 9]  
  
 y[y == '是'] = 1  
 y[y == '否'] = 0  
 y = y.astype(int)  
 lda = LDA()  
 lda.fit(X, y, plot\_=True)  
 print(lda.predict(X)) # 和逻辑回归的结果一致  
 print(y)

## 2.运行结果：

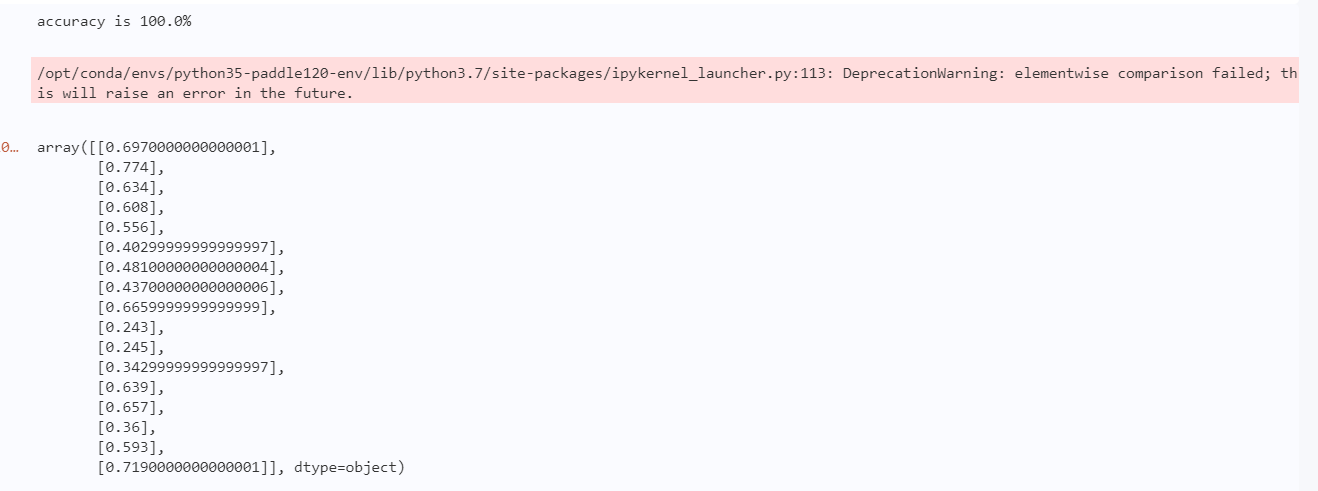


# （2）4.10

## 代码：

import numpy as np  
import copy  
np.set\_printoptions(suppress=True) # 禁用科学计数  
import pandas as pd  
import matplotlib.pyplot as plt  
path = 'work/西瓜数据集 3.0.txt'  
data = pd.read\_csv(  
 path,  
 )  
data.head()  
  
class Value():  
 def \_\_init\_\_(self, name, values):  
 *"""  
 name是属性名, values是属性具体取值  
 """* self.name = name  
 self.values = values  
  
class DecisionTree():  
 *"""  
 决策树算法  
 """* class Node():  
 *"""  
 节点类  
 """* def \_\_init\_\_(self, genre=None):  
 self.next = {}  
 self.is\_genre = False  
 def add\_genre(self, genre):  
 *"""  
 genre是当该节点是根节点时的类别  
 """* self.genre = int(genre)  
 def add\_next\_node(self, next\_node, value, name):  
 *"""  
 next\_node是下一个节点, value是到该节点的取值(条件), next\_name是取决划分的属性名  
 """* self.next\_name = name  
 self.next[value] = next\_node  
 def \_\_init\_\_(self, pattern):  
 *"""  
 pattern是决策树划分准则  
 """* self.pattern = pattern  
 self.pruning = {}  
 self.pruning['none'] = self.none\_pruning  
 self.pruning['pre'] = self.pre\_pruning  
 self.pruning['after'] = self.after\_pruning  
 def fit(self, X, y, A = None, pruning='none'):  
 *"""  
 A为属性集, 可以手动传入也可以自动创建  
 """* self.X = X  
 self.y = y  
 self.head = self.Node() # 定义头节点  
 self.now\_pruning = pruning  
 if A == None:  
 A = []  
 i = 0  
 for i in np.arange(X.shape[1]): # 创建A  
 A.append(Value(i, np.arange(np.unique(X[:, i]).shape[0])))  
 i += 1  
 temp\_X = X.copy()  
 temp\_y = y.copy()  
 self.X\_to\_temp\_X = {} # 映射集合  
 for i in range(temp\_X.shape[1]):  
 name = np.unique(temp\_X[:, i])  
 for j in range(name.shape[0]):  
 self.X\_to\_temp\_X[name[j]] = j  
 temp\_X[temp\_X[:, i] == name[j], i] = j  
 name = np.unique(temp\_y)  
 self.temp\_y\_to\_y = {}  
 for i in range(name.shape[0]):  
 temp\_y[temp\_y == name[i]] = i  
 self.temp\_y\_to\_y[i] = name[i]  
 self.X\_to\_temp\_X = pd.DataFrame(self.X\_to\_temp\_X, index=[0])  
 self.temp\_y\_to\_y = pd.DataFrame(self.temp\_y\_to\_y, index=[0])  
 temp\_y = temp\_y.astype(int)  
 temp\_X = temp\_X.astype(int)  
 self.pruning[pruning](temp\_X, temp\_y, A, self.head)  
 def after\_pruning(self, X, y, A, now\_node):  
 *"""  
 后剪枝  
 """* self.none\_pruning(X, y, A, now\_node)  
 def recursive(now\_node, front\_node): # 考察front\_node是否要替换为叶节点  
 if not now\_node.is\_genre:  
 for name in now\_node.next:  
 recursive(now\_node.next[name], now\_node)  
 front\_node.is\_genre = True  
 pruning\_acc = self.accuracy(self.X, self.y)  
 front\_node.is\_genre = False  
 acc = self.accuracy(self.X, self.y)  
 if(pruning\_acc >= acc):  
 front\_node.is\_genre = True  
 return  
 for name in self.head.next:  
 recursive(self.head.next[name], self.head)  
 def pre\_pruning(self, X, y, A, now\_node):  
 *"""  
 预剪枝  
 """* self.none\_pruning(X, y, A, now\_node)  
 def recursive(now\_node): #考察该节点是否要替换为叶节点  
 if now\_node.is\_genre:  
 return  
 now\_node.is\_genre = True  
 pruning\_acc = self.accuracy(self.X, self.y)  
 now\_node.is\_genre = False  
 self.next\_set\_is\_genre(now\_node, True)  
 acc = self.accuracy(self.X, self.y)  
 self.next\_set\_is\_genre(now\_node, False)  
 if(pruning\_acc >= acc):  
 now\_node.is\_genre = True  
 return  
 else:  
 for name in now\_node.next:  
 recursive(now\_node.next[name])  
 recursive(self.head)  
 def none\_pruning(self, X, y, A, now\_node):  
 *"""  
 A是属性集, front\_node是前一个节点, 该函数寻找的是前一个节点的下一个最优节点  
 """* judge = np.unique(y)  
 if judge.shape[0] == 1: # 判断是否属于同一类别  
 now\_node.add\_genre(judge)  
 now\_node.is\_genre = True  
 return  
 now\_node.add\_genre(np.argmax(np.bincount(y.reshape(len(y))))) # D中样本数最多的类  
 if (len(A) == 0) or (np.unique(np.unique(X) == X[0])):  
 now\_node.is\_genre = True  
 return  
 a = self.find\_best(X, y, A, self.pattern) # 返回最优的属性子集  
 temp\_A = copy.deepcopy(A)  
 for i in range(len(temp\_A)):  
 if temp\_A[i].name == a.name:  
 break  
 temp\_A.pop(i)  
 for a\_v in a.values:  
 temp = (X[:, a.name] == a\_v)  
 X\_v = X[temp]  
 y\_v = y[temp]  
 if X\_v.shape[0] == 0:  
 now\_node.is\_genre = True  
 return  
 else:  
 next\_node = self.Node()  
 now\_node.add\_next\_node(next\_node, a\_v, a.name)  
 self.none\_pruning(X\_v, y\_v, temp\_A, next\_node)  
 def next\_set\_is\_genre(self, node, setting):  
 for name in node.next:  
 if setting == True:  
 if node.next[name].is\_genre == True:  
 node.next[name].tag = True  
 else:  
 node.next[name].tag = False  
 node.next[name].is\_genre = True  
 else:  
 if node.next[name].tag == True:  
 continue  
 else:  
 node.next[name].is\_genre = False  
 def find\_best(self, X, y, A, pattern): # 寻找最好的属性集  
 bigger = np.NINF  
 for a in A:  
 temp = pattern(X, y, a)  
 if temp > bigger:  
 bigger = temp  
 greater\_a = a  
 return greater\_a  
 def predict(self, X):  
 y = np.zeros(X.shape[0]).reshape(1, X.shape[0]).T.astype('object')  
 i = 0  
 for x in X: # 对每个向量x进行决策树的预测  
 x = np.array(self.X\_to\_temp\_X[x]).reshape(x.shape)  
 temp = self.head  
 while not temp.is\_genre:  
 temp = temp.next[x[temp.next\_name]]  
 y[i] = self.temp\_y\_to\_y[temp.genre][0]  
 i += 1  
 return y  
 def accuracy(self, X, y):  
 y\_pre = self.predict(X)  
 return (np.sum(y\_pre == y) / y.shape[0])  
  
def ent(y):  
 res = 0  
 num = y.shape[0]  
 for k in np.unique(y):  
 p\_k = np.sum(y == k) / num  
 res += p\_k \* np.log2(p\_k)  
 return -res  
def gain(X, y, a): # 信息增益  
 res = 0  
 num = y.shape[0]  
 for value in a.values:  
 label = (X[:, a.name] == value)  
 res += (np.sum(label) / num) \* ent(y[label])  
 return ent(y) - res  
  
X = np.array(data.iloc[:, :6])  
y = np.array(data.iloc[:, 7]).reshape(1, data.shape[0]).T  
  
decisiontree = DecisionTree(gain)  
  
decisiontree.fit(X, y, pruning='none')  
print('accuracy is {}%'.format(decisiontree.accuracy(X, y) \* 100))  
decisiontree.predict(X)  
  
decisiontree.fit(X, y, pruning='none')  
print('accuracy is {}%'.format(decisiontree.accuracy(X, y) \* 100))  
decisiontree.predict(X)

## 运行结果：

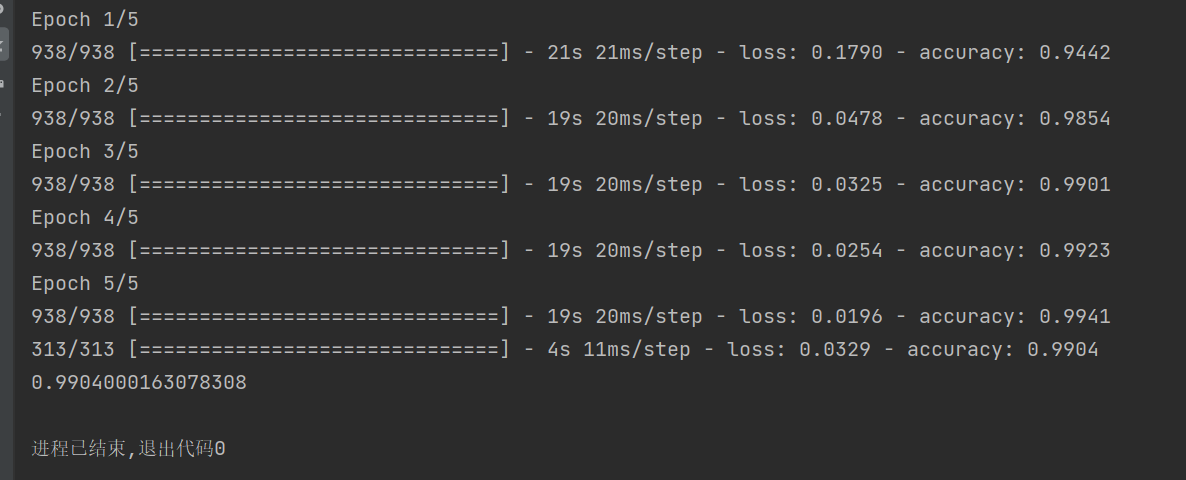


# （3）5.10

## 代码：

from keras import layers  
from keras import models  
from keras.datasets import mnist  
from keras.utils import to\_categorical # one-hot编码  
# 获得数据  
from keras.datasets import mnist  
(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()  
  
  
# 构造模型  
model = models.Sequential()  
model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))  
model.add(layers.MaxPooling2D((2, 2)))  
model.add(layers.Conv2D(64, (3, 3), activation='relu'))  
model.add(layers.MaxPooling2D((2, 2)))  
model.add(layers.Conv2D(64, (3, 3), activation='relu'))  
model.add(layers.Flatten())  
model.add(layers.Dense(64, activation='relu'))  
model.add(layers.Dense(10, activation='softmax'))  
  
# 处理数据  
train\_images = train\_images.reshape((60000, 28, 28, 1))  
train\_images = train\_images.astype('float32') / 255  
test\_images = test\_images.reshape((10000, 28, 28, 1))  
test\_images = test\_images.astype('float32') / 255  
train\_labels = to\_categorical(train\_labels)  
test\_labels = to\_categorical(test\_labels)  
  
# 编译模型  
model.compile(optimizer='rmsprop',  
loss='categorical\_crossentropy',  
metrics=['accuracy'])  
  
# 训练模型  
model.fit(train\_images, train\_labels, epochs=5, batch\_size=64)  
  
# 测试集精度  
test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)  
print(test\_acc)

## 运行结果：

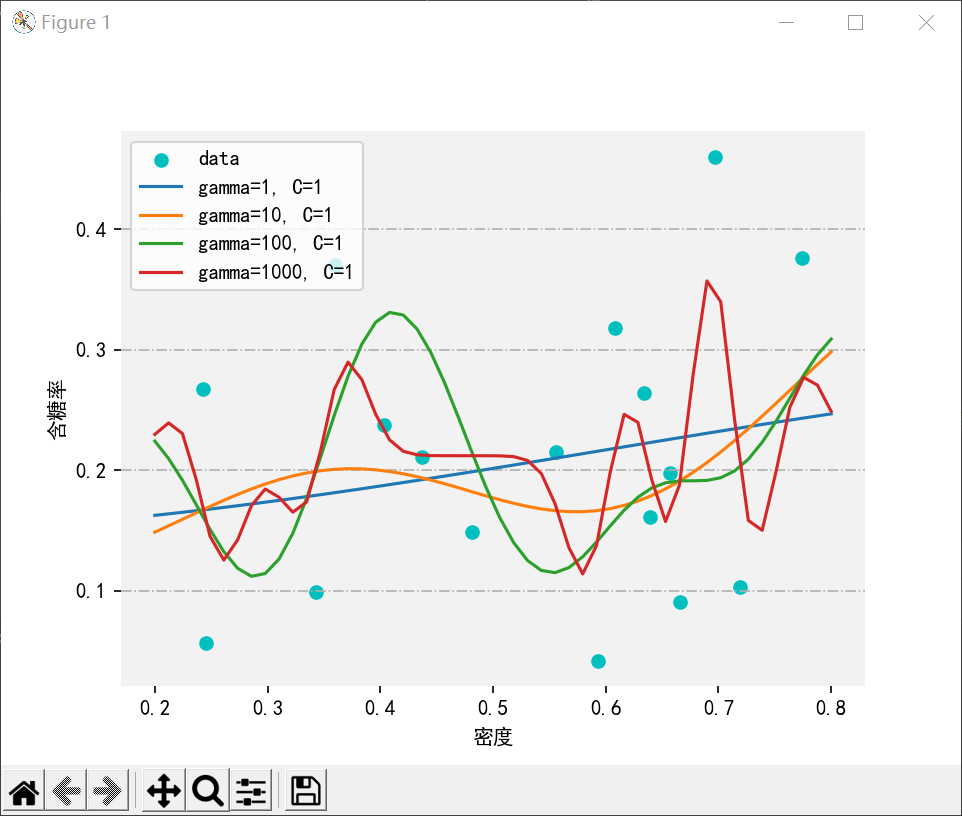


# （4）6.8

## 代码：

import pandas as pd  
from sklearn import svm  
import matplotlib.pyplot as plt  
import numpy as np  
  
  
def set\_ax\_gray(ax):  
 ax.patch.set\_facecolor("gray")  
 ax.patch.set\_alpha(0.1)  
 ax.spines['right'].set\_color('none') # 设置隐藏坐标轴  
 ax.spines['top'].set\_color('none')  
 ax.spines['bottom'].set\_color('none')  
 ax.spines['left'].set\_color('none')  
 ax.grid(axis='y', linestyle='-.')  
  
  
path = r'D:\machinelearning\watermelon3\_0a\_Ch.txt'  
data = pd.read\_table(path, delimiter=' ', dtype=float)  
  
X = data.iloc[:, [0]].values  
y = data.iloc[:, 1].values  
  
gamma = 10  
C = 1  
  
ax = plt.subplot()  
set\_ax\_gray(ax)  
ax.scatter(X, y, color='c', label='data')  
  
for gamma in [1, 10, 100, 1000]:  
 svr = svm.SVR(kernel='rbf', gamma=gamma, C=C)  
 svr.fit(X, y)  
  
 ax.plot(np.linspace(0.2, 0.8), svr.predict(np.linspace(0.2, 0.8).reshape(-1, 1)),  
 label='gamma={}, C={}'.format(gamma, C))  
ax.legend(loc='upper left')  
ax.set\_xlabel('密度')  
ax.set\_ylabel('含糖率')  
  
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体  
plt.rcParams['axes.unicode\_minus'] = False # 解决保存图像是负号'-'显示为方块的问题  
plt.show()

## 运行结果：

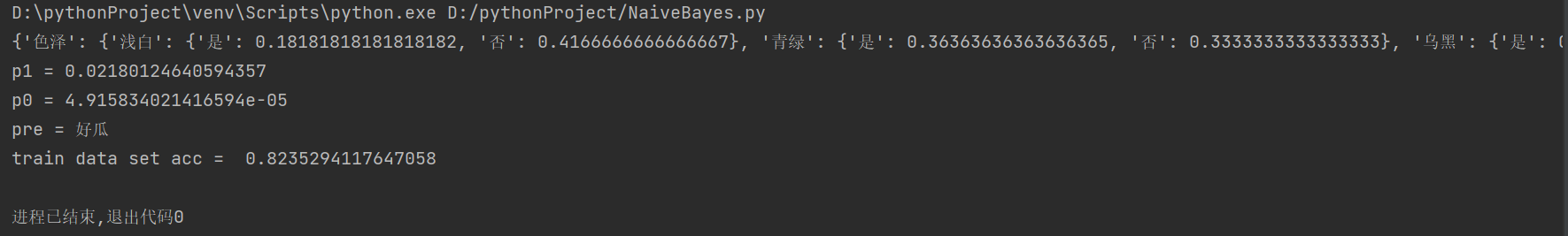


# （5）7.3

## 代码：

import numpy as np  
import matplotlib.pyplot as plt  
from pylab import \*  
import operator  
  
# 特征字典，后面用到了好多次，干脆当全局变量了  
featureDic = {  
 '色泽': ['浅白', '青绿', '乌黑'],  
 '根蒂': ['硬挺', '蜷缩', '稍蜷'],  
 '敲声': ['沉闷', '浊响', '清脆'],  
 '纹理': ['清晰', '模糊', '稍糊'],  
 '脐部': ['凹陷', '平坦', '稍凹'],  
 '触感': ['硬滑', '软粘']}  
  
  
def getDataSet():  
 *"""  
 get watermelon data set 3.0.* ***:return****: 编码好的数据集以及特征的字典。  
 """* dataSet = [  
 ['青绿', '蜷缩', '浊响', '清晰', '凹陷', '硬滑', 0.697, 0.460, 1],  
 ['乌黑', '蜷缩', '沉闷', '清晰', '凹陷', '硬滑', 0.774, 0.376, 1],  
 ['乌黑', '蜷缩', '浊响', '清晰', '凹陷', '硬滑', 0.634, 0.264, 1],  
 ['青绿', '蜷缩', '沉闷', '清晰', '凹陷', '硬滑', 0.608, 0.318, 1],  
 ['浅白', '蜷缩', '浊响', '清晰', '凹陷', '硬滑', 0.556, 0.215, 1],  
 ['青绿', '稍蜷', '浊响', '清晰', '稍凹', '软粘', 0.403, 0.237, 1],  
 ['乌黑', '稍蜷', '浊响', '稍糊', '稍凹', '软粘', 0.481, 0.149, 1],  
 ['乌黑', '稍蜷', '浊响', '清晰', '稍凹', '硬滑', 0.437, 0.211, 1],  
 ['乌黑', '稍蜷', '沉闷', '稍糊', '稍凹', '硬滑', 0.666, 0.091, 0],  
 ['青绿', '硬挺', '清脆', '清晰', '平坦', '软粘', 0.243, 0.267, 0],  
 ['浅白', '硬挺', '清脆', '模糊', '平坦', '硬滑', 0.245, 0.057, 0],  
 ['浅白', '蜷缩', '浊响', '模糊', '平坦', '软粘', 0.343, 0.099, 0],  
 ['青绿', '稍蜷', '浊响', '稍糊', '凹陷', '硬滑', 0.639, 0.161, 0],  
 ['浅白', '稍蜷', '沉闷', '稍糊', '凹陷', '硬滑', 0.657, 0.198, 0],  
 ['乌黑', '稍蜷', '浊响', '清晰', '稍凹', '软粘', 0.360, 0.370, 0],  
 ['浅白', '蜷缩', '浊响', '模糊', '平坦', '硬滑', 0.593, 0.042, 0],  
 ['青绿', '蜷缩', '沉闷', '稍糊', '稍凹', '硬滑', 0.719, 0.103, 0]  
 ]  
  
 features = ['色泽', '根蒂', '敲声', '纹理', '脐部', '触感', '密度', '含糖量']  
  
 # 每种特征的属性个数  
 numList = [] # [3, 3, 3, 3, 3, 2]  
 for i in range(len(features) - 2):  
 numList.append(len(featureDic[features[i]]))  
  
 dataSet = np.array(dataSet)  
 return dataSet, features  
  
def cntProLap(dataSet, index, value, classLabel, N):  
 *"""  
 用拉普拉斯修正估计概率值* ***:param*** *dataSet:* ***:param*** *index:* ***:param*** *value:* ***:param*** *classLabel:* ***:param*** *N:* ***:return****:  
 """* extrData = dataSet[dataSet[:, -1] == classLabel]  
 cnt = 0  
 for data in extrData:  
 if data[index] == value:  
 cnt += 1  
 return (cnt + 1) / (float(len(extrData)) + N)  
  
  
def naiveBayesClassifier(dataSet, features):  
 *"""  
 拉普拉斯修正的朴素贝叶斯分类器。所谓拉普拉斯修正的意义在于，避免训练集中某些属性没有出现，导致概率为零，而使得整个连  
 乘为零。  
 修正的方法是：  
 1.对于类c(好瓜/坏瓜)的先验概率(样本中的 P(c=好瓜)/P(c=坏瓜)),分子加1，分母加类别数，本题中两类所以为2。  
 2.对于条件概率(P(xi|c), xi表示第i个属性取值为x的值)，分子加1，分母加第i个属性可能的取值数。* ***:param*** *dataSet: 训练集* ***:param*** *features: 特征列表。['色泽', '根蒂', '敲声', '纹理', '脐部', '触感', '密度', '含糖量']* ***:return****: 一个字典，保存了3部分内容。  
 1.对于类别型的变量，保存其是好瓜和不是好瓜的概率。  
 2.对于数值型的属性，保存其实好瓜和不是好瓜的均值和方差。  
 3.保存了类别的先验概率。即P(c=好瓜)和P(c=坏瓜)。  
 """* dict = {}  
 for feature in features:  
 index = features.index(feature)  
 dict[feature] = {}  
 if feature != '密度' and feature != '含糖量':  
 featIList = featureDic[feature]  
 for value in featIList:  
 PisCond = cntProLap(dataSet, index, value, '1', len(featIList))  
 pNoCond = cntProLap(dataSet, index, value, '0', len(featIList))  
 dict[feature][value] = {}  
 dict[feature][value]["是"] = PisCond  
 dict[feature][value]["否"] = pNoCond  
 else:  
 for label in ['1', '0']:  
 dataExtra = dataSet[dataSet[:, -1] == label]  
 extr = dataExtra[:, index].astype("float64")  
 aver = extr.mean()  
 var = extr.var()  
  
 labelStr = ""  
 if label == '1':  
 labelStr = '是'  
 else:  
 labelStr = '否'  
  
 dict[feature][labelStr] = {}  
 dict[feature][labelStr]["平均值"] = aver  
 dict[feature][labelStr]["方差"] = var  
  
 length = len(dataSet)  
 classLabels = dataSet[:, -1].tolist()  
 dict["好瓜"] = {}  
 dict["好瓜"]['是'] = (classLabels.count('1') + 1) / (float(length) + 2)  
 dict["好瓜"]['否'] = (classLabels.count('0') + 1) / (float(length) + 2)  
  
 return dict  
  
def NormDist(mean, var, xi):  
 *"""  
 计算连续属性的概率密度。* ***:param*** *mean: 第c类在第i个属性上的均值* ***:param*** *var: 第c类在第i个属性上的方差* ***:param*** *xi: 第c类在第i个属性上的取值* ***:return****: 概率密度  
 """* return exp(-((float(xi) - mean) \*\* 2) / (2 \* var)) / (sqrt(2 \* pi \* var))  
  
def predict(data, features, bayesDis):  
 *"""  
 通过贝叶斯预测数据的类型。* ***:param*** *data: 待预测的数据。* ***:param*** *features: 特征列表。* ***:param*** *bayesDis: 字典。  
 对于类别型的变量，保存其是好瓜和不是好瓜的概率。  
 对于数值型的属性，保存其实好瓜和不是好瓜的均值和方差。* ***:return****: 预测类型值。  
 """* pGood = bayesDis['好瓜']['是']  
 pBad = bayesDis['好瓜']['否']  
 for feature in features:  
 index = features.index(feature)  
 if feature != '密度' and feature != '含糖量':  
 pGood \*= bayesDis[feature][data[index]]['是']  
 pBad \*= bayesDis[feature][data[index]]['否']  
 else:  
 # NormDist(mean, var, xi)  
 pGood \*= NormDist(bayesDis[feature]['是']['平均值'],  
 bayesDis[feature]['是']['方差'],  
 data[index])  
 pBad \*= NormDist(bayesDis[feature]['否']['平均值'],  
 bayesDis[feature]['否']['方差'],  
 data[index])  
  
 retClass = ""  
 if pGood > pBad:  
 retClass = "好瓜"  
 else:  
 retClass = "坏瓜"  
  
 return pGood, pBad, retClass  
  
  
def calcAccRate(dataSet, features, bayesDis):  
 *"""  
 计算训练集在朴素贝叶斯分类器上的准确率。* ***:param*** *dataSet:* ***:param*** *features:* ***:param*** *bayesDis:* ***:return****:  
 """* cnt = 0.0  
 for data in dataSet:  
 \_, \_, pre = predict(data, features, bayesDis)  
 if (pre == '好瓜' and data[-1] == '1') \  
 or (pre == '坏瓜' and data[-1] == '0'):  
 cnt += 1  
  
 return cnt / float(len(dataSet))  
  
dataSet, features = getDataSet()  
dic = naiveBayesClassifier(dataSet, features)  
print(dic)  
p1, p0, pre = predict(dataSet[0], features, dic)  
print(f"p1 = {p1}")  
print(f"p0 = {p0}")  
print(f"pre = {pre}")  
print("train data set acc = ", calcAccRate(dataSet, features, dic))

## 运行结果：



# （6）8.5

## 代码：

import numpy as np  
import pandas as pd  
from matplotlib import pyplot as plt  
from sklearn.utils import resample  
  
  
def stumpClassify(X, dim, thresh\_val, thresh\_inequal):  
 ret\_array = np.ones((X.shape[0], 1))  
  
 if thresh\_inequal == 'lt':  
 ret\_array[X[:, dim] <= thresh\_val] = -1  
 else:  
 ret\_array[X[:, dim] > thresh\_val] = -1  
  
 return ret\_array  
  
  
def buildStump(X, y):  
 m, n = X.shape  
 best\_stump = {}  
  
 min\_error = 1  
  
 for dim in range(n):  
  
 x\_min = np.min(X[:, dim])  
 x\_max = np.max(X[:, dim])  
  
 # 这里第一次尝试使用排序后的点作为分割点，效果很差，因为那样会错过一些更好的分割点；  
 # 所以后来切割点改成将最大值和最小值之间分割成20等份。  
  
 # sorted\_x = np.sort(X[:, dim])  
 # split\_points = [(sorted\_x[i] + sorted\_x[i + 1]) / 2 for i in range(m - 1)]  
  
 split\_points = [(x\_max - x\_min) / 20 \* i + x\_min for i in range(20)]  
  
 for inequal in ['lt', 'gt']:  
 for thresh\_val in split\_points:  
 ret\_array = stumpClassify(X, dim, thresh\_val, inequal)  
  
 error = np.mean(ret\_array != y)  
  
 if error < min\_error:  
 best\_stump['dim'] = dim  
 best\_stump['thresh'] = thresh\_val  
 best\_stump['inequal'] = inequal  
 best\_stump['error'] = error  
 min\_error = error  
  
 return best\_stump  
  
  
def stumpBagging(X, y, nums=20):  
 stumps = []  
 seed = 16  
 for \_ in range(nums):  
 X\_, y\_ = resample(X, y, random\_state=seed) # sklearn 中自带的实现自助采样的方法  
 seed += 1  
 stumps.append(buildStump(X\_, y\_))  
 return stumps  
  
  
def stumpPredict(X, stumps):  
 ret\_arrays = np.ones((X.shape[0], len(stumps)))  
  
 for i, stump in enumerate(stumps):  
 ret\_arrays[:, [i]] = stumpClassify(X, stump['dim'], stump['thresh'], stump['inequal'])  
  
 return np.sign(np.sum(ret\_arrays, axis=1))  
  
  
def pltStumpBaggingDecisionBound(X\_, y\_, stumps):  
 pos = y\_ == 1  
 neg = y\_ == -1  
 x\_tmp = np.linspace(0, 1, 600)  
 y\_tmp = np.linspace(-0.1, 0.7, 600)  
  
 X\_tmp, Y\_tmp = np.meshgrid(x\_tmp, y\_tmp)  
 Z\_ = stumpPredict(np.c\_[X\_tmp.ravel(), Y\_tmp.ravel()], stumps).reshape(X\_tmp.shape)  
  
 plt.contour(X\_tmp, Y\_tmp, Z\_, [0], colors='orange', linewidths=1)  
  
 plt.scatter(X\_[pos, 0], X\_[pos, 1], label='1', color='c')  
 plt.scatter(X\_[neg, 0], X\_[neg, 1], label='0', color='lightcoral')  
 plt.legend()  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data\_path = r'D:\machinelearning\watermelon3\_0a\_Ch.txt'  
  
 data = pd.read\_table(data\_path, delimiter=' ')  
  
 X = data.iloc[:, :2].values  
 y = data.iloc[:, 2].values  
  
 y[y == 0] = -1  
  
 stumps = stumpBagging(X, y, 21)  
  
 print(np.mean(stumpPredict(X, stumps) == y))  
 pltStumpBaggingDecisionBound(X, y, stumps)

## 运行结果：

