实验三：参数估计 & 非参数估计

姓名：雷贺奥

学号：2013551

专业：计算机科学与技术

实验要求

基本要求

生成两个各包含 $N=1200$ 个二维随机向量的数据集合 X_1 和 X_2，数据集合中随机向量来自于三个分布模型，分别满足均值向量$\mu_1=[1,4],\mu_2=[4,1],\mu_3=[8,4]$和协方差矩阵 $D_1=D_2=D_3=2I$，其中I是$2*2$的单位矩阵。在生成数据集合 X_1时，假设来自三个分布模型的先验概率相同；而在生成数据集合 X_2 时，先验概率如下：
$p(w_1)=0.6,p(w_2)=0.1,p(w_3)=0.3$

1. 在两个数据集合上分别应用“似然率测试规则” 、“最大后验概率规则” 进行分类实验，计算分类错误率，分析实验结果。
2. 在两个数据集合上分别应用 $h=1$ 时的方窗核函数或高斯核函数估计方法，应用“似然率测试规则”进行分类实验，计算分类错误率，分析实验结果。

中级要求

1. 根据初级要求中使用的一个核函数，在数据集 X_2 上应用交叉验证法，在 $h\in[0.1,0.5,1,1.5,2]$ 中寻找最优的h值。

高级要求

1. 任选一个数据集，在该数据集上应用k-近邻概率密度估计，任选3个k值输出概率密度分布图。

生成数据集

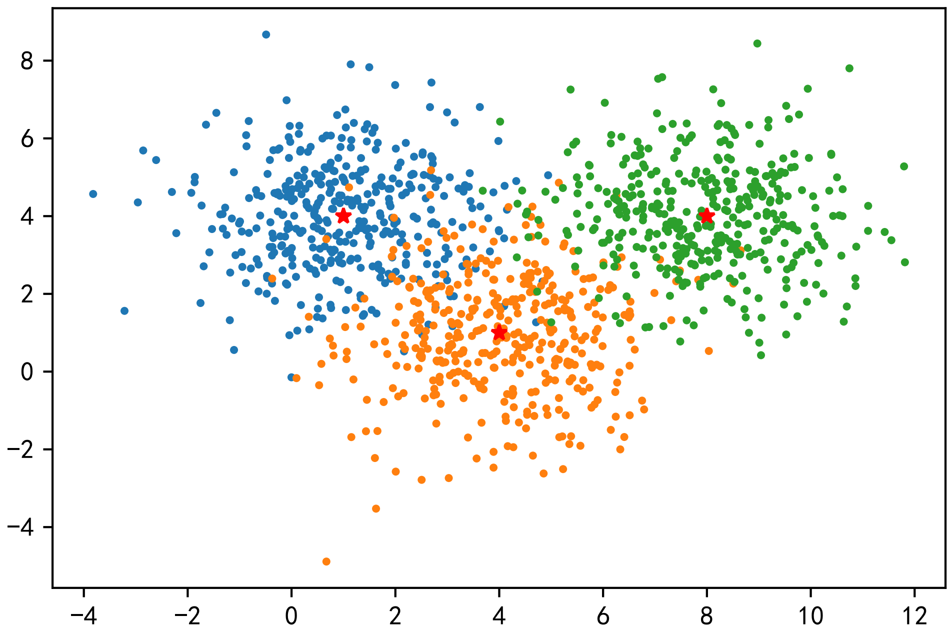
In [164…
```
import numpy as np
import sys
import matplotlib.pyplot as plt
import copy
from mpl_toolkits.mplot3d import Axes3D
```

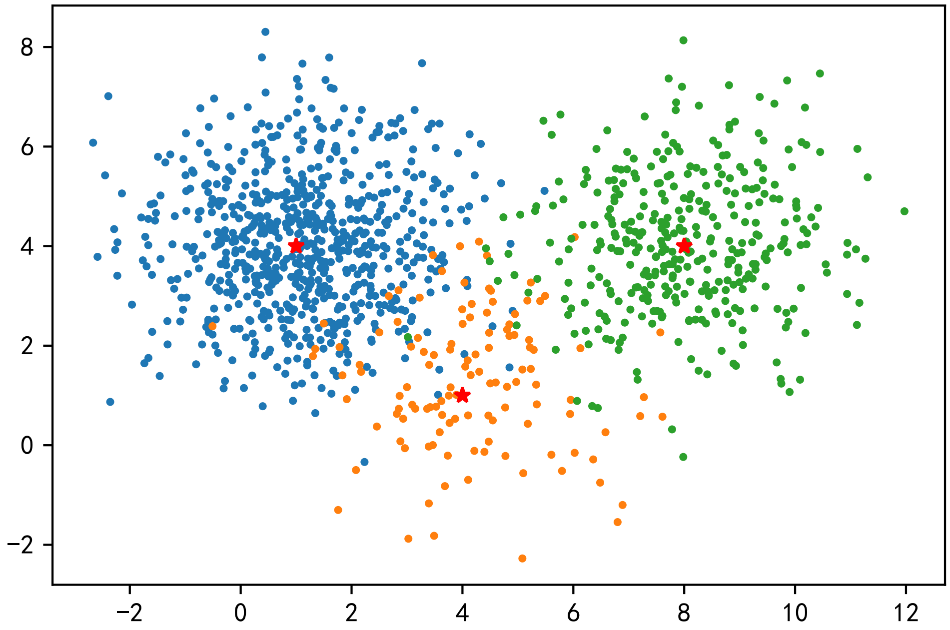
In [165…
```
# 生成正态分布数据
def Generate_Sample_Gaussian(mean, cov, P, label):
    '''
        mean 为均值向量
        cov 为方差矩阵a
        P 为单个类的先验概率
        return 单个类的数据集
    '''
```

```
    temp_num = round(1200 * P)
    x, y = np.random.multivariate_normal(mean, cov, temp_num).T
    z = np.ones(temp_num) * label
    X = np.array([x, y, z])
    return X.T

def Generate_DataSet_plot(mean, cov, P):
    # 画出不同先验对应的散点图
    xx = []
    label = 1
    for i in range(3):
        xx.append(Generate_Sample_Gaussian(mean[i], cov, P[i], label))
        label += 1
        i = i + 1
    # 画图
    plt.figure()
    for i in range(3):
        plt.plot(xx[i][:, 0], xx[i][:, 1], '.', markersize=4.)
        plt.plot(mean[i][0], mean[i][1], 'r*')
    plt.show()
    return xx
```

In [166…

```
mean = np.array([[1, 4], [4, 1], [8, 4]])  # 均值数组
cov = [[2, 0], [0, 2]]  # 方差矩阵
num = 1200  # 样本个数
P1 = [1 / 3, 1 / 3, 1 / 3]  # 样本X1的先验概率
P2 = [0.6, 0.1, 0.3]  # 样本X2的先验概率
X1 = np.array(Generate_DataSet_plot(mean, cov, P1), dtype=object)
X2 = np.array(Generate_DataSet_plot(mean, cov, P2), dtype=object)
X1 = np.vstack(X1)
X2 = np.vstack(X2)
```





In [167…

```
X1.shape, X2.shape  # 前两列是坐标，最后一列是标签
```

Out[167]:

```
((1200, 3), (1200, 3))
```

HINT FOR THIS LAB

In [168…

```
# 极大似然估计
# 输入n*2维数据
def LikelyHood(X):
    mu = np.mean(X, axis=0)
    # python把向量转化成矩阵需要用reshape
    cov = np.array([np.dot((X[i] - mu).reshape(2,1), (X[i] - mu).reshape(1, 2)) for
    return mu, cov

# Hint for 初级要求：二元高斯分布概率密度函数计算
# 在公式中，x和mean应该是列向量，但是为了方便，这里接收的都是行向量（维度：1*2）
def Gaussian_function(x, mu, cov):
    det_cov = np.linalg.det(cov)  # 计算方差矩阵的行列式
    inv_cov = np.linalg.inv(cov)  # 计算方差矩阵的逆
    # 计算概率p(x|w)
    p = 1 / (2 * np.pi * np.sqrt(det_cov)) * np.exp(-0.5 * np.dot(np.dot((x - mu),
    return p

# Hint for 初级要求：高斯核概率密度函数计算
# 在公式中，x和mean应该是列向量，但是为了方便，这里接收的都是行向量（维度：1*2）
def Gaussian_Kernel(x, X, h=2):
    # 计算概率p(x|w)
    p = (1 / (np.sqrt(2 * np.pi) * h)) * np.array([np.exp(-0.5 * np.dot(x - X[i],
    return p
```

基本要求

算出均值和协方差

In [169…

```
def comunicate(X):
    # 分别筛选出3类数据
```

```python
    X1_label_1 = X[np.where((X[:, 2] == 1))]  #label=1
    X1_label_2 = X[np.where((X[:, 2] == 2))]  #label=2
    X1_label_3 = X[np.where((X[:, 2] == 3))]  #label=3
    print("划分结果：",X1_label_1.shape,X1_label_2.shape,X1_label_3.shape)
    #删除标签列
    X1_label_1=np.delete(X1_label_1, 2, axis=1)
    X1_label_2=np.delete(X1_label_2, 2, axis=1)
    X1_label_3=np.delete(X1_label_3, 2, axis=1)

    #计算三种分类mu和cov，均值和协方差，用于后续预测
    mus = dict()
    covs = dict()
    for i in range(1,4):
        if i==1:
            X1_temp=X1_label_1;
            mu,cov=LikelyHood(X1_temp)
            mus[i]=mu
            covs[i]=cov.astype('float64')
        elif i==2:
            X1_temp=X1_label_2;
            mu,cov=LikelyHood(X1_temp)
            mus[i]=mu
            covs[i]=cov.astype('float64')
        elif i==3:
            X1_temp=X1_label_3;
            mu,cov=LikelyHood(X1_temp)
            mus[i]=mu
            covs[i]=cov.astype('float64')
    return mus,covs
```

In [170…

```python
mus,cov=comunicate(X1)
print("X1的mu均值如下：\n")
mus
```

```
划分结果： (400, 3) (400, 3) (400, 3)
X1的mu均值如下：
```

Out[170]:

```
{1: array([1.0091680042056148, 4.001259178710441], dtype=object),
 2: array([4.044748310547373, 0.9418498351806275], dtype=object),
 3: array([7.919340957720434, 4.000328944694721], dtype=object)}
```

In [171…

```python
print("X1的cov协方差如下:\n")
covs
```

```
X1的cov协方差如下:
```

Out[171]:

```
{1: array([[1.95979265, 0.00730369],
        [0.00730369, 2.02783407]]),
 2: array([[1.80729886, 0.08529894],
        [0.08529894, 2.00639163]]),
 3: array([[ 1.92092912, -0.01060068],
        [-0.01060068,  2.02390551]])}
```

In [172…

```python
mus,cov=comunicate(X2)
print("X2的mu均值如下：\n")
mus
```

```
划分结果： (720, 3) (120, 3) (360, 3)
X2的mu均值如下：
```

Out[172]:

```
{1: array([1.11081973, 4.0348242 ]),
 2: array([4.14518399, 1.27677016]),
 3: array([8.0059258 , 4.05281484])}
```

In [173…
```
print("X2的cov协方差如下:\n")
covs
```

X2的cov协方差如下:

Out[173]:
```
{1: array([[1.95979265, 0.00730369],
        [0.00730369, 2.02783407]]),
 2: array([[1.80729886, 0.08529894],
        [0.08529894, 2.00639163]]),
 3: array([[ 1.92092912, -0.01060068],
        [-0.01060068,  2.02390551]])}
```

似然率测试规则 、 最大后验概率规则实现

似然率测试规则

In [174…
```
def LikelyHood_test(X,labels):
    right=0
    for sample in X:
        p = np.zeros(len(labels))
        for i in range(len(labels)):
            p[i] = Gaussian_function(sample[0:2],mus[labels[i]],covs[labels[i]])
        exp_label = np.argmax(p)+1;#np.argmax(p)下标
        if exp_label==sample[2]:
            right+=1
    return 1-right/X.shape[0]
```

最大后验概率测试规则

In [175…
```
def MaxPosterior_test(X,labels,Post):
    right=0
    for sample in X:
        p = np.zeros(len(labels))
        for i in range(len(labels)):
            p[i] = Gaussian_function(sample[0:2],mus[labels[i]],covs[labels[i]])*Pos
        exp_label = np.argmax(p)+1;#np.argmax(p)下标
        if exp_label==sample[2]:
            right+=1
    return 1-right/X.shape[0]
```

似然和最大后验结果对比

In [176…
```
print("数据集\t似然概率规则\t最大后验概率规则")
mus,covs=comunicate(X1)
print(f"X1\t{LikelyHood_test(X1, [1, 2, 3])}\t{MaxPosterior_test(X1, [1, 2, 3], P1)}
mus,covs=comunicate(X2)
print(f"X2\t{LikelyHood_test(X2, [1, 2, 3])}\t{MaxPosterior_test(X2, [1, 2, 3], P2)}
```

```
数据集	似然概率规则	最大后验概率规则
划分结果: (400, 3) (400, 3) (400, 3)
X1	0.07083333333333333	0.07083333333333333
划分结果: (720, 3) (120, 3) (360, 3)
X2	0.06333333333333335	0.04333333333333335
```

结果分析

若每类的先验概率相同，则似然率测试规则和最大后验概率规则的预测错误率相同。

当先验概率不同时，最大后验概率测试规则的预测错误率更小。这是因为后验概率，在高斯概率密度后乘了一项后验概率，会使得误差变小。

高斯核函数结合似然率测试规则

In [177…

```
def LikelyHood_test_Gausskernel(X, labels, h=1.0):
    #错误数量
    error_nums = 0
    # 分别筛选出3类数据
    X1_label_1 = X[np.where((X[:, 2] == 1))]  #label=1
    X1_label_2 = X[np.where((X[:, 2] == 2))]  #label=2
    X1_label_3 = X[np.where((X[:, 2] == 3))]  #label=3
    for sample in X:
        p = np.zeros(len(labels))
        for i in range(0,3):
            #根据不同的类确定 Gaussian_Kernel(x, X, h=2)中X
            if i==0:
                p[i] = Gaussian_Kernel(sample[0:2],X1_label_1[:,0:2], h)
            elif i==1:
                p[i] = Gaussian_Kernel(sample[0:2],X1_label_2[:,0:2], h)
            elif i==2:
                p[i] = Gaussian_Kernel(sample[0:2],X1_label_3[:,0:2],h)
        exp_label = labels[np.argmax(p)]
        if exp_label != sample[2]:
            error_nums += 1
    return error_nums / X.shape[0]
print("数据集\t似然概率规则")
print(f"X1\t{LikelyHood_test_Gausskernel(X1, [1, 2, 3])}")
print(f"X2\t{LikelyHood_test_Gausskernel(X2, [1, 2, 3])}")
```

```
数据集	似然概率规则
X1	0.07
X2	0.06166666666666667
```

中级要求

交叉检验

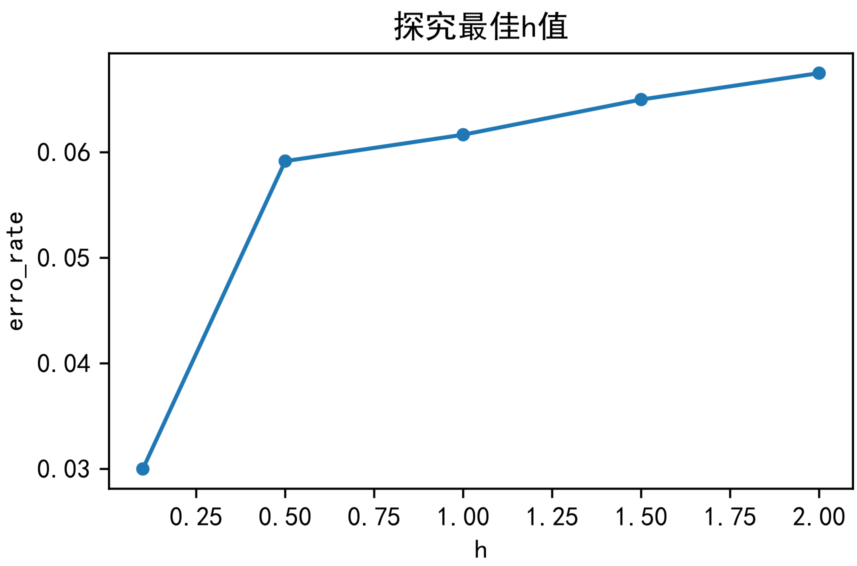
In [178…

```
def LikelyHood_test_Gausskernel_CrossValid(X, labels, h):
    #错误数量
    error_nums = 0
    for sample in X:
        #用于交叉检验
        index=0;
        temp_X=np.delete(X,index,axis=0)
        # 分别筛选出3类数据
        X1_label_1 =temp_X[np.where((temp_X[:, 2] == 1))]  #label=1
        X1_label_2 =temp_X[np.where((temp_X[:, 2] == 2))]  #label=2
        X1_label_3 =temp_X[np.where((temp_X[:, 2] == 3))]  #label=3
        p = np.zeros(len(labels))
        for i in range(0,3):
            #根据不同的类确定 Gaussian_Kernel(x, X, h=2)中X
            if i==0:
                p[i] = Gaussian_Kernel(sample[0:2],X1_label_1[:,0:2], h)
            elif i==1:
                p[i] = Gaussian_Kernel(sample[0:2],X1_label_2[:,0:2], h)
            elif i==2:
                p[i] = Gaussian_Kernel(sample[0:2],X1_label_3[:,0:2],h)
```

```
        exp_label = labels[np.argmax(p)]
        if exp_label != sample[2]:
            error_nums += 1
        index+=1
    return error_nums / X.shape[0]
```

In [179…

```
h_test=[0.1, 0.5, 1.0, 1.5, 2.0]
score=[]
for h_temp in h_test:
    score.append(LikelyHood_test_Gausskernel_CrossValid(X2,[1,2,3],h_temp))
#绘图,让图像清晰
from matplotlib import pyplot as plt
%matplotlib inline
#让图像清晰
%config InlineBackend.figure_format = 'svg'
#设置画布大小像素点
plt.figure(figsize=(5,3),dpi=100)
plt.rc('font', family='SimHei', size=10)
plt.plot(h_test,score,marker='o',markersize=4)
plt.xlabel('h')
plt.ylabel('erro_rate')
plt.title('探究最佳h值')
plt.show()
```



从图中找到$erro_rate$最低点，对应的$h=0.1$，所以最佳的h值为0.1。（不知道对不对）

高级要求

K近邻算法

In [180…

```
#定义距离
def distance(x,y):
    dis=0
    for i in range(len(x)):
        dis+=(x[i]-y[i])**2
    return np.sqrt(dis)

def Kneibor_Eval(X,k,labels):
    N_k = list()
    X1_label_1 = X[np.where((X[:, 2] == 1))]  #label=1
```

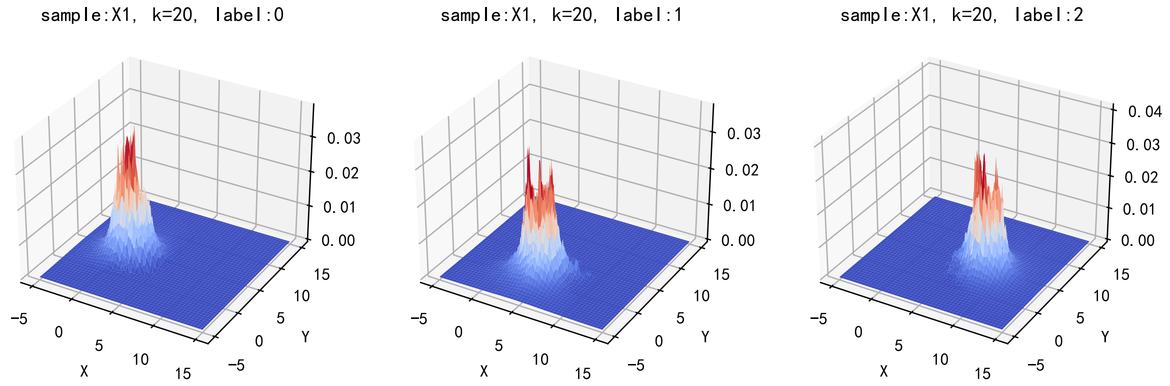
```python
    X1_label_2 = X[np.where((X[:, 2] == 2))]  #label=2
    X1_label_3 = X[np.where((X[:, 2] == 3))]  #label=3
    #X.shape[0]为公式中的n
    N_k.append(X1_label_1.shape[0])
    N_k.append(X1_label_2.shape[0])
    N_k.append(X1_label_3.shape[0])
    # 生成200*200=40000个采样点，每个采样点对应三类的不同概率
    p = np.zeros((200, 200, 3))
    # 在[-5,15]的范围内，以0.1为步长估计概率密度
    for i in range(200):
        for j in range(200):
            '''
            # 生成标准差距离
            # 根据第k个数据点的位置计算V
            # 找到前k个数据点的类别,分别加到对应类的权重上
            # 计算每个采样点的概率密度函数
            '''
            x = [-5 + 0.1 * i, -5 + 0.1 * j]
            dists = list()
            for sample in X:
                dists.append([distance(x, sample[0:2]), sample[2]])
            dists.sort(key=lambda x:x[0])
            # V为公式中的体积
            V = np.pi * (dists[k - 1][0] ** 2)
            for index in range(len(labels)):
                K_k = 0
                for _i in range(k):
                    if dists[_i][1] == labels[index]:
                        #K_k为公式中的k
                        K_k += 1
                p[i][j][index] = (K_k /X.shape[0]) / V
    return p
```

实验结果

In [181…

```python
p = Kneibor_Eval(X1,20,[1,2,3])  # 获得概率密度估计

# 高级要求1
X,Y = np.mgrid[-5:15:200j, -5:15:200j]

Z0 = p[:, :, 0]
Z1 = p[:, :, 1]
Z2 = p[:, :, 2]
#绘图
plt.rcParams['axes.unicode_minus']=False
fig = plt.figure(figsize=(12,6))
ax = plt.subplot(1, 3, 1,projection='3d')
ax.plot_surface(X, Y, Z0,cmap=plt.cm.coolwarm)
ax.set_title("sample:X1, k=20, label:0")
ax.set_xlabel('X')
ax.set_ylabel('Y')

ax = plt.subplot(1, 3, 2,projection='3d')
ax.plot_surface(X, Y, Z1,cmap=plt.cm.coolwarm)
ax.set_title("sample:X1, k=20, label:1")
ax.set_xlabel('X')
ax.set_ylabel('Y')

ax = plt.subplot(1, 3, 3,projection='3d')
ax.plot_surface(X, Y, Z2,cmap=plt.cm.coolwarm)
ax.set_title("sample:X1, k=20, label:2")
ax.set_xlabel('X')
```

```
ax.set_ylabel('Y')
plt.show()
```



In [ ]: