机器学习作业三

- 姓名：周延霖
- 学号：2013921
- 专业：信息安全

实验要求

题目：回归模型

实验要求：

1. 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 X1 和 X2 ，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 μ1=[1,4] , μ2=[4,1] , μ3=[8,4] 和协方差矩阵 D1=D2=D3=2I ，其中 I 是 2∗2 的单位矩阵。在生成数据集合 X1 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 X2 时，先验概率如下： p(w1)=0.6 , p(w2)=0.1 , p(w3)=0.3

- 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。
- 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。

1. 中级要求

根据初级要求中使用的一个核函数，在数据集 X2 上应用交叉验证法，在 h∈[0.1,0.5,1,1.5,2] 中寻找最优的h值。

1. 高级要求

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。

截止日期：11月4日

- 以.ipynb形式的文件提交，输出运行结果，并确保自己的代码能够正确运行
- 发送到邮箱：2120220594@mail.nankai.edu.cn

In [1]:
```
import numpy as np
import sys
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
```

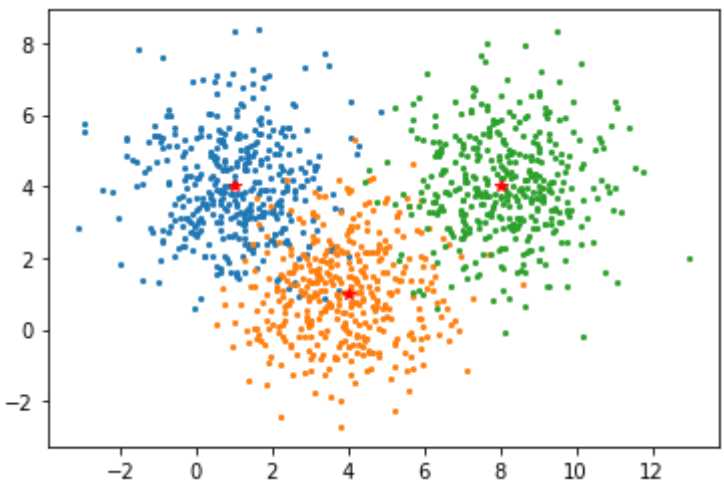
基本要求

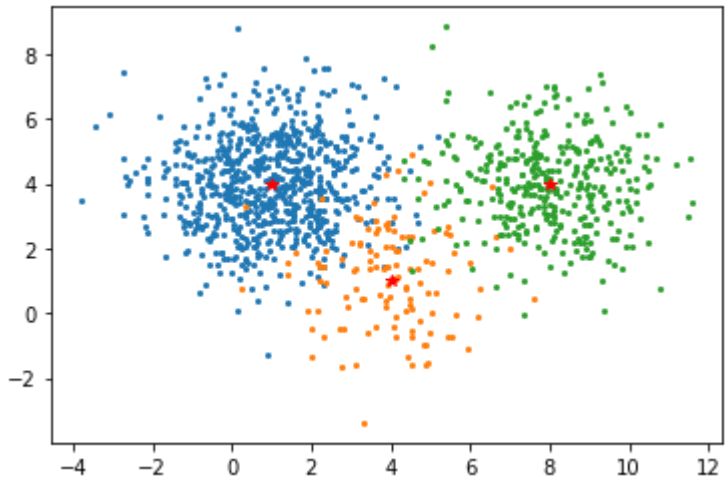
In [2]:
```
# 生成正态分布数据
```

```
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
```

In [3]:
```
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 [4]:
```
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 [5]:
```
X1.shape, X2.shape  # 前两列是坐标，最后一列是标签
```

Out[5]:
```
((1200, 3), (1200, 3))
```

In [6]:
```
print(X1[400])
print(X1[40])
print(X1[1100])
```

```
[2.7753132158160145 1.410975549046935 2.0]
[2.19470630456755 3.2634862970727587 1.0]
[7.1051341918081565 3.769889943473418 3.0]
```

似然率测试规则

概率密度公式

In [7]:
```
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
    return p
```

In [8]:
```
def Normal_distribution(data, m):
    m = np.array(m)
    cov = np.array([[1, 0], [0, 1]])
    return 1 / math.sqrt((2 * np.pi) ** 2 * 1) * math.exp(-0.5 * np.dot((dat
```

将概率密度得到的最大值分到此类

In [9]:
```
def Classification(data, mean1, mean2, mean3):
    result = []
    for d in data:
        t1 = Normal_distribution(d, mean1)
        t2 = Normal_distribution(d, mean2)
        t3 = Normal_distribution(d, mean3)
        i = 1
        max = t1
        if t2 > max:
            max = t2
            i = 2
        if t3 > max:
            max = t3
            i = 3
```

```
        result.append(i)
    return result
```

计算准确率，并画出散点图

In [10]:
```
x_w = X1[:,[0, 1]]
y_w = X2[:,[0, 1]]

print(x_w.shape)
```

```
(1200, 2)
```

> 计算真正的均值

x1三个样本集的均值

In [11]:
```
x1_all1_x = 0
x1_all1_y = 0
x1_num1 = 0

x1_all2_x = 0
x1_all2_y = 0
x1_num2 = 0

x1_all3_x = 0
x1_all3_y = 0
x1_num3 = 0

# for i in range(0, 5):
#     print(i)

for i in range(0, 1200):
    if X1[i][2] == 1:
        x1_all1_x += X1[i][0]
        x1_all1_y += X1[i][1]
        x1_num1 += 1
    elif X1[i][2] == 2:
        x1_all2_x += X1[i][0]
        x1_all2_y += X1[i][1]
        x1_num2 += 1
    elif X1[i][2] == 3:
        x1_all3_x += X1[i][0]
        x1_all3_y += X1[i][1]
        x1_num3 += 1

x1_1x = x1_all1_x / x1_num1
x1_1y = x1_all1_y / x1_num1

x1_2x = x1_all2_x / x1_num2
x1_2y = x1_all2_y / x1_num2

x1_3x = x1_all3_x / x1_num3
x1_3y = x1_all3_y / x1_num3

# print(x1_1x)
# print('\n')
# print(x1_1y)
# print('\n')

# print(x1_2x)
# print('\n')
```

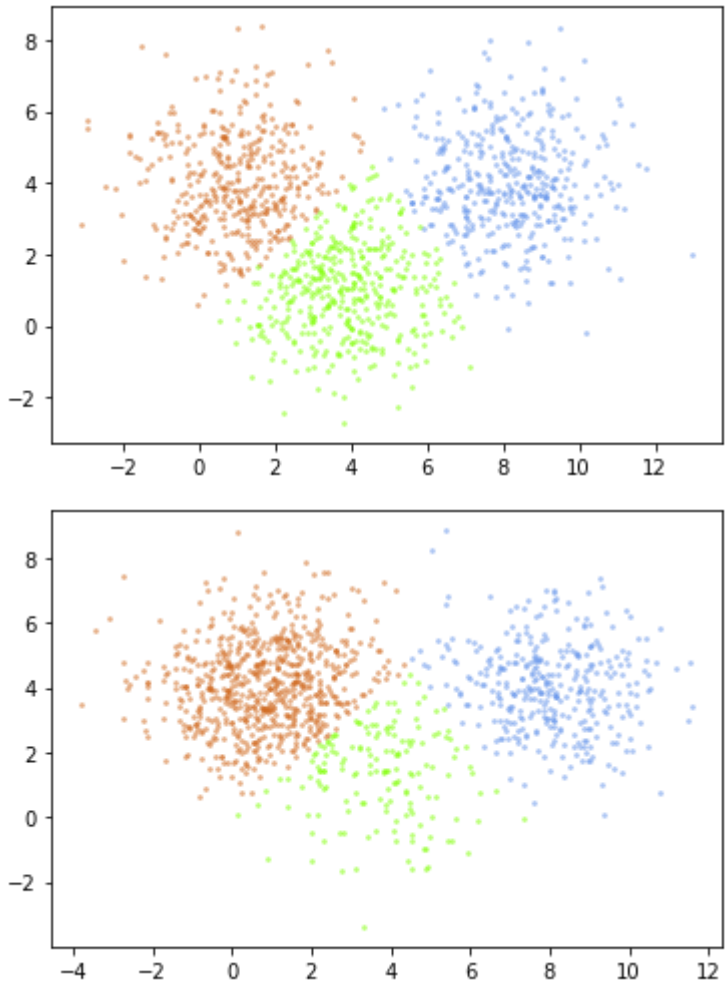
```
# print(x1_2y)
# print('\n')

# print(x1_3x)
# print('\n')
# print(x1_3y)
```

x2三个样本集的均值

In [12]:
```
x2_all1_x = 0
x2_all1_y = 0
x2_num1 = 0

x2_all2_x = 0
x2_all2_y = 0
x2_num2 = 0

x2_all3_x = 0
x2_all3_y = 0
x2_num3 = 0

# for i in range(0, 5):
#     print(i)

for i in range(0, 1200):
    if X2[i][2] == 1:
        x2_all1_x += X2[i][0]
        x2_all1_y += X2[i][1]
        x2_num1 += 1
    elif X2[i][2] == 2:
        x2_all2_x += X2[i][0]
        x2_all2_y += X2[i][1]
        x2_num2 += 1
    elif X2[i][2] == 3:
        x2_all3_x += X2[i][0]
        x2_all3_y += X2[i][1]
        x2_num3 += 1

x2_1x = x2_all1_x / x2_num1
x2_1y = x2_all1_y / x2_num1

x2_2x = x2_all2_x / x2_num2
x2_2y = x2_all2_y / x2_num2

x2_3x = x2_all3_x / x2_num3
x2_3y = x2_all3_y / x2_num3

# print(x2_1x)
# print('\n')
# print(x2_1y)
# print('\n')

# print(x2_2x)
# print('\n')
# print(x2_2y)
# print('\n')

# print(x2_3x)
# print('\n')
# print(x2_3y)
```

In [13]:
```
def simrate(ls1, ls2):
    num = 0
```

```
    l = len(ls1)
    for i in range(l):
        if ls1[i] != ls2[i]:
            num += 1
    return format(num / l, '.2%')

def show_result(data, result, name):
    colors0 = '#D2691E'
    colors1 = '#7FFF00' #点的颜色
    colors2 = '#6495ED'
    area = np.pi   # 点面积
    x1 = []
    x2 = []
    x3 = []
    y1 = []
    y2 = []
    y3 = []
    for i in range(len(data)):
        if result[i] == 1:
            x1.append(data[i][0])
            y1.append(data[i][1])
        elif result[i] == 2:
            x2.append(data[i][0])
            y2.append(data[i][1])
        elif result[i] == 3:
            x3.append(data[i][0])
            y3.append(data[i][1])
    plt.scatter(x1, y1, s=area, c=colors0, alpha=0.4)
    plt.scatter(x2, y2, s=area, c=colors1, alpha=0.4)
    plt.scatter(x3, y3, s=area, c=colors2, alpha=0.4)
    plt.figure(num = name)

result1 = Classification(x_w, (x1_1x, x1_1y), (x1_2x, x1_2y), (x1_3x, x1_3y))
result2 = Classification(y_w, (x2_1x, x2_1y), (x2_2x, x2_2y), (x2_3x, x2_3y))

c1 = X1[:,[2]]
c2 = X2[:,[2]]

Error_rate1 = simrate(result1, c1)
Error_rate2 = simrate(result2, c2)

show_result(X1, result1, "X_1最大似然估计分类图")
show_result(X2, result2, "X_2最大似然估计分类图")


plt.show()

print("\nX_1数据集使用似然率测试规则进行分类的错误率是", Error_rate1)
print("X_2数据集使用似然率测试规则进行分类的错误率是", Error_rate2)
```



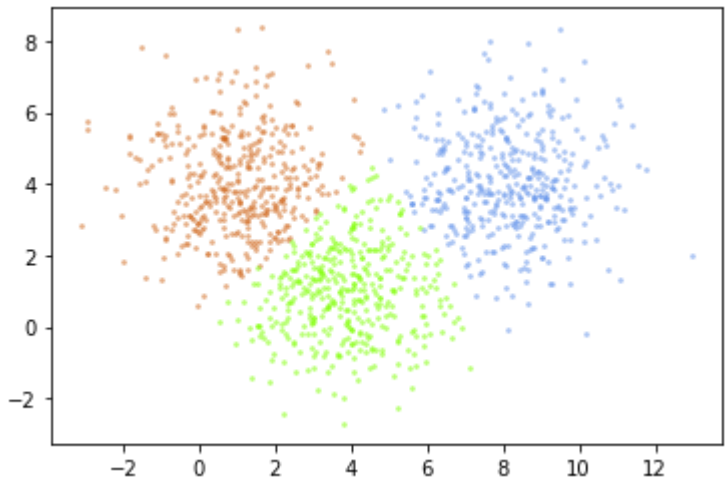
```
<Figure size 432x288 with 0 Axes>
X_1数据集使用似然率测试规则进行分类的错误率是 5.75%
X_2数据集使用似然率测试规则进行分类的错误率是 6.17%
```

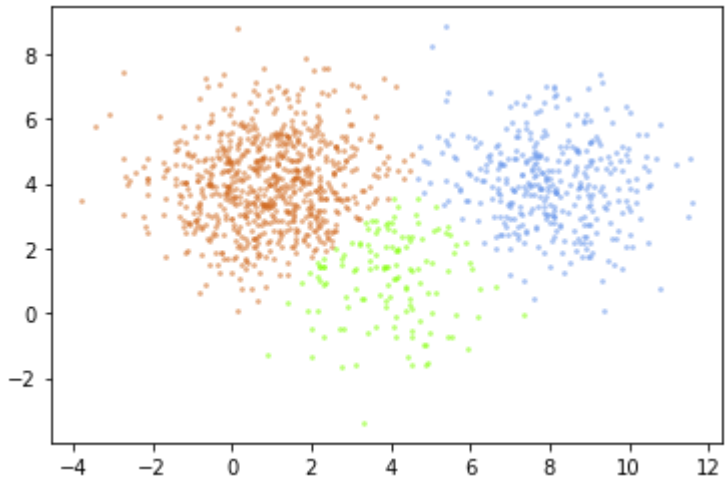
最大后验概率规则

In [14]:
```
def Classification_after(data, mean1, mean2, mean3, P):
    result = []
    for d in data:
        t1 = Normal_distribution(d, mean1) * P[0]
        t2 = Normal_distribution(d, mean2) * P[1]
        t3 = Normal_distribution(d, mean3) * P[2]
        i = 1
        max = t1
        if t2 > max:
            max = t2
            i = 2
        if t3 > max:
            max = t3
            i = 3
        result.append(i)
    return result
```

In [15]:
```
result1_after = Classification_after(x_w, (x1_1x, x1_1y), (x1_2x, x1_2y), (x
result2_after = Classification_after(y_w, (x2_1x, x2_1y), (x2_2x, x2_2y), (x


Error_rate1_after = simrate(result1_after, c1)
Error_rate2_after = simrate(result2_after, c2)

show_result(X1, result1_after, "X_1最大后验概率规则分类图")
```

```
show_result(X2, result2_after, "X_2最大后验概率规则分类图")


plt.show()

print("\nX_1数据集使用最大后验概率规则进行分类的错误率是", Error_rate1_after)
print("X_2数据集使用最大后验概率规则进行分类的错误率是", Error_rate2_after)
```






```
<Figure size 432x288 with 0 Axes>
X_1数据集使用最大后验概率规则进行分类的错误率是 5.75%
X_2数据集使用最大后验概率规则进行分类的错误率是 4.75%
```

结果分析

可以看出，利用“最大后验概率规则”进行分类比利用“似然率测试规则”的错误率较低，且分类错误基本处于边缘或离中心较远的点，故该分类方法在此数据集下效果较好，可以采用。

h = 1 + 高斯核函数估计方法 + 似然率测试规则

In [16]:
```
def guss(x, data, h):
    n = len(data)
    result = 0
    for i in range(n):
        result += math.exp(- (math.sqrt((x[0] - data[i][0]) ** 2 + (x[1] - d
    result = result / (n * math.sqrt(2 * np.pi * h * h))
    return result
```

In [17]:
```
def Classification_2(data, m1, m2, m3, h, r):
    result = []
    data1 = data[0 : m1]
```

```
    data2 = data[m1 : m1+m2]
    data3 = data[m1+m2 : m1+m2+m3]
    for d in data:
        t1 = guss(d, data1, h)
        t2 = guss(d, data2, h)
        t3 = guss(d, data3, h)
        i = 1
        max = t1
        if t2 > max:
            max = t2
            i = 2
        if t3 > max:
            max = t3
            i = 3
        result.append(i)
    return simrate(result, r)
```

In [18]:
```
r1_10 = Classification_2(X1, 400, 400, 400, 1, c1)
r2_10 = Classification_2(X2, 720, 120, 360, 1, c2)

print("X_1数据集利用似然率测试规则在h = 1.0情况下分类错误率是 ", r1_10)
print("X_2数据集利用似然率测试规则在h = 1.0情况下分类错误率是 ", r2_10)
```

```
X_1数据集利用似然率测试规则在h = 1.0情况下分类错误率是  5.75%
X_2数据集利用似然率测试规则在h = 1.0情况下分类错误率是  5.75%
```

中级要求

利用不同的h进行交叉验证

In [19]:
```
r1_1 = Classification_2(X1, 400, 400, 400, 0.1, c1)
r1_5 = Classification_2(X1, 400, 400, 400, 0.5, c1)
r1_10 = Classification_2(X1, 400, 400, 400, 1, c1)
r1_15 = Classification_2(X1, 400, 400, 400, 1.5, c1)
r1_20 = Classification_2(X1, 400, 400, 400, 2, c1)

r2_1 = Classification_2(X2, 720, 120, 360, 0.1, c2)
r2_5 = Classification_2(X2, 720, 120, 360, 0.5, c2)
r2_10 = Classification_2(X2, 720, 120, 360, 1, c2)
r2_15 = Classification_2(X2, 720, 120, 360, 1.5, c2)
r2_20 = Classification_2(X2, 720, 120, 360, 2, c2)

print("X_1数据集利用似然率测试规则在h = 0.1情况下分类错误率是 ", r1_1)
print("X_1数据集利用似然率测试规则在h = 0.5情况下分类错误率是 ", r1_5)
print("X_1数据集利用似然率测试规则在h = 1.0情况下分类错误率是 ", r1_10)
print("X_1数据集利用似然率测试规则在h = 1.5情况下分类错误率是 ", r1_15)
print("X_1数据集利用似然率测试规则在h = 2.0情况下分类错误率是 ", r1_20)

print("X_2数据集利用似然率测试规则在h = 0.1情况下分类错误率是 ", r2_1)
print("X_2数据集利用似然率测试规则在h = 0.5情况下分类错误率是 ", r2_5)
print("X_2数据集利用似然率测试规则在h = 1.0情况下分类错误率是 ", r2_10)
print("X_2数据集利用似然率测试规则在h = 1.5情况下分类错误率是 ", r2_15)
print("X_2数据集利用似然率测试规则在h = 2.0情况下分类错误率是 ", r2_20)
```

```
X_1数据集利用似然率测试规则在h = 0.1情况下分类错误率是  0.00%
X_1数据集利用似然率测试规则在h = 0.5情况下分类错误率是  5.58%
X_1数据集利用似然率测试规则在h = 1.0情况下分类错误率是  5.75%
X_1数据集利用似然率测试规则在h = 1.5情况下分类错误率是  5.83%
X_1数据集利用似然率测试规则在h = 2.0情况下分类错误率是  5.83%
X_2数据集利用似然率测试规则在h = 0.1情况下分类错误率是  0.25%
X_2数据集利用似然率测试规则在h = 0.5情况下分类错误率是  4.83%
X_2数据集利用似然率测试规则在h = 1.0情况下分类错误率是  5.75%
X_2数据集利用似然率测试规则在h = 1.5情况下分类错误率是  5.83%
X_2数据集利用似然率测试规则在h = 2.0情况下分类错误率是  6.00%
```

结果分析

可以看出，在应用“似然率测试规则”分类的实验中，窗口h的选择在较大取值将产生过度平滑的密度估计，模糊了数据的空间结构；较小取值将产生又长又尖的密度估计，解释比较困难。通过实验可以看出，在这两个数据集下窗口值为0.1时效果较好，此时的错误率在两数据集上明显均比其他h的情况低.

高级要求

In [20]:
```python
def judge_label(mid, X):
    n1 = 0
    n2 = 0
    n3 = 0
    l = len(mid)
    PI = math.pi
    s = PI * (mid[l - 1][1]**2)
    for i in mid:
        # print(i[0])
        if X[int(i[0])][2] == 1:
            n1 += 1
        elif X[int(i[0])][2] == 2:
            n2 += 1
        else:
            n3 += 1

    label = []
    label.append(n1 / (s * 1200))
    label.append(n2 / (s * 1200))
    label.append(n3 / (s * 1200))

    return label
```

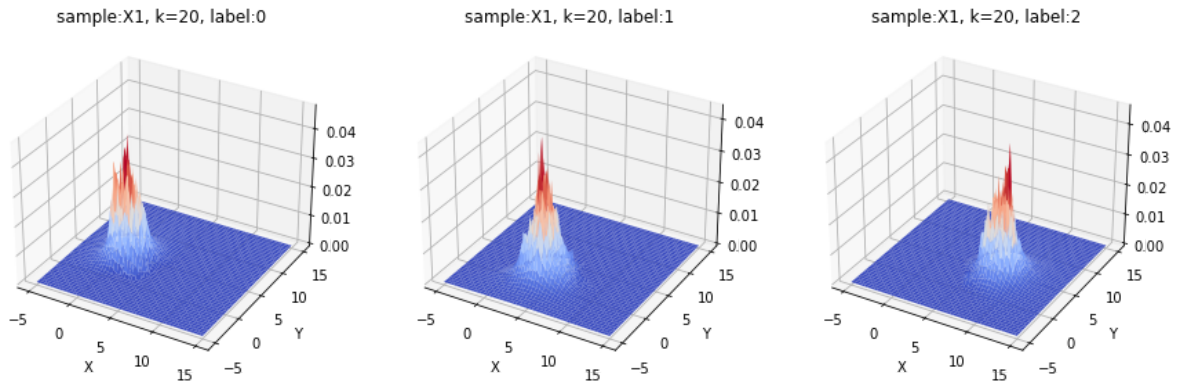
In [21]:
```python
def Kneibor_Eval(X, k):
    num = len(X)
    # Xtrain = np.array(X)
    # 生成200*200=40000个采样点，每个采样点对应三类的不同概率
    p = np.zeros((200, 200, 3))
    # 在[-5,15]的范围内，以0.1为步长估计概率密度
    for i in np.arange(0, 200):
        for j in np.arange(0, 200):
            '''
            # 生成标准差距离
            # 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
            '''
            l = []
            x_now = -5 + 0.1 * i
```

```
            y_now = -5 + 0.1 * j
            for t in np.arange(0, 1200):
                s1 = math.sqrt((x_now - X[t][0]) ** 2 + (y_now - X[t][1]) **
                l1 = [t,s1]
                l.append(l1)
            # 按最后一列排序
            l = np.array(l)
            mid = l[np.lexsort(l.T)]
            mid = mid[:k, :]

            p[i][j] = judge_label(mid, X)


    return p
```

In [22]:
```
p = Kneibor_Eval(X1, 20)  # 获得概率密度估计

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

In [23]:
```
Z0 = p[:, :, 0]
Z1 = p[:, :, 1]
Z2 = p[:, :, 2]
```

In [24]:
```
fig = plt.figure(figsize=(15,5))
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()
```

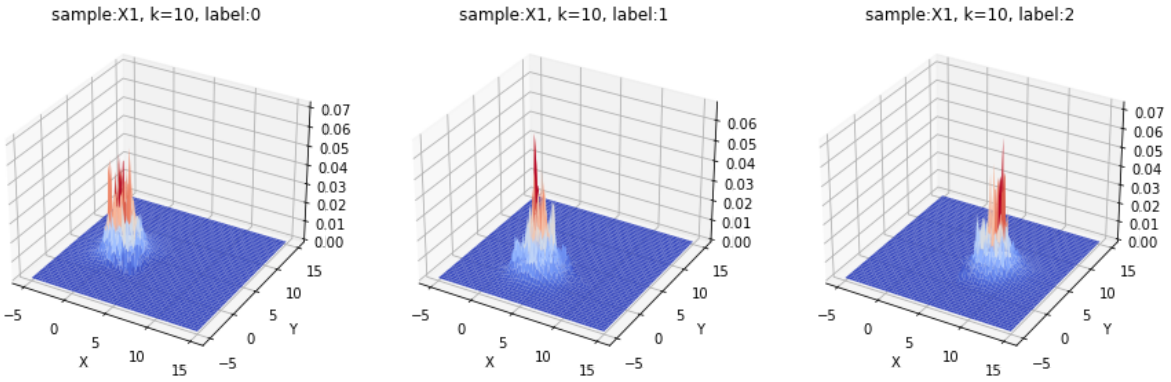


In [25]:
```
p = Kneibor_Eval(X1, 10)  # 获得概率密度估计

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

In [26]:
```
Z0 = p[:, :, 0]
```

```
Z1 = p[:, :, 1]
Z2 = p[:, :, 2]
```

In [27]:
```
fig = plt.figure(figsize=(15,5))
ax = plt.subplot(1, 3, 1,projection='3d')
ax.plot_surface(X, Y, Z0,cmap=plt.cm.coolwarm)
ax.set_title("sample:X1, k=10, 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=10, 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=10, label:2")
ax.set_xlabel('X')
ax.set_ylabel('Y')
plt.show()
```

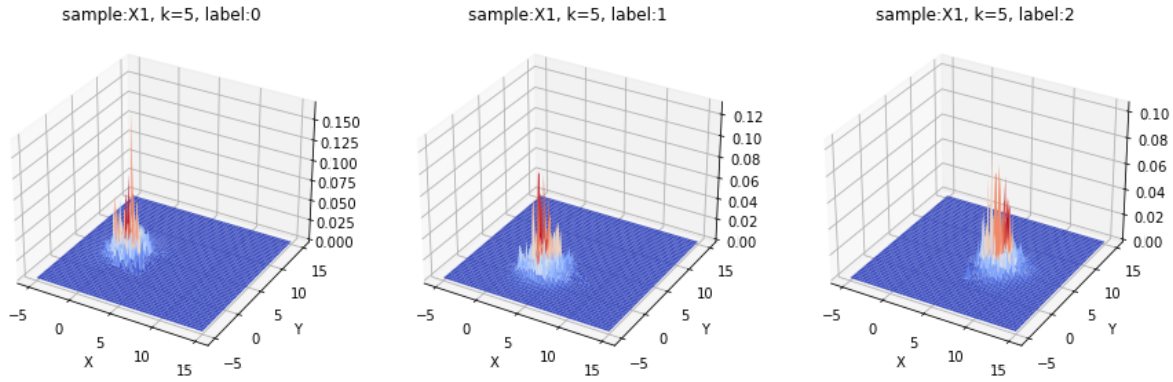


In [28]:
```
p = Kneibor_Eval(X1, 5)  # 获得概率密度估计

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

In [29]:
```
Z0 = p[:, :, 0]
Z1 = p[:, :, 1]
Z2 = p[:, :, 2]
```

In [30]:
```
fig = plt.figure(figsize=(15,5))
ax = plt.subplot(1, 3, 1,projection='3d')
ax.plot_surface(X, Y, Z0,cmap=plt.cm.coolwarm)
ax.set_title("sample:X1, k=5, 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=5, 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=5, label:2")
ax.set_xlabel('X')
```

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



总结与展望

总结

- 本次是机器学习的第三次实验，在做实验的过程中感受到了似然概率估计函数以及最大后验估计密度相应的优缺点
- 在本次实验中也了解了参数估计和非参数估计的具体区别
- 在本次实验中也知道了什么是方窗函数，以及该如何应用这个函数的来对结果进行估计，也知道了这个函数的高明之处
- 最后也对k近邻又一次应用，对此方法也更加的熟悉

展望

通过第三次实验，发现自己对机器学习有了更近一步的认识，希望自己能在本学期的课程中学到更多，也希望自己未来能有更好的发展✌