Parametric and Non-Parametric Estimation

Experiment Description

基本要求 (3’)

在两个数据集合上应用“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。

中级要求 (2’)
在两个数据集合上使用高斯核函数估计方法，应用“似然率测试规则”分类，在 [0.1, 0.5, 1, 1.5, 2] 范围内交叉验证找到最优 h 值，分析实验结果。

Data Generation

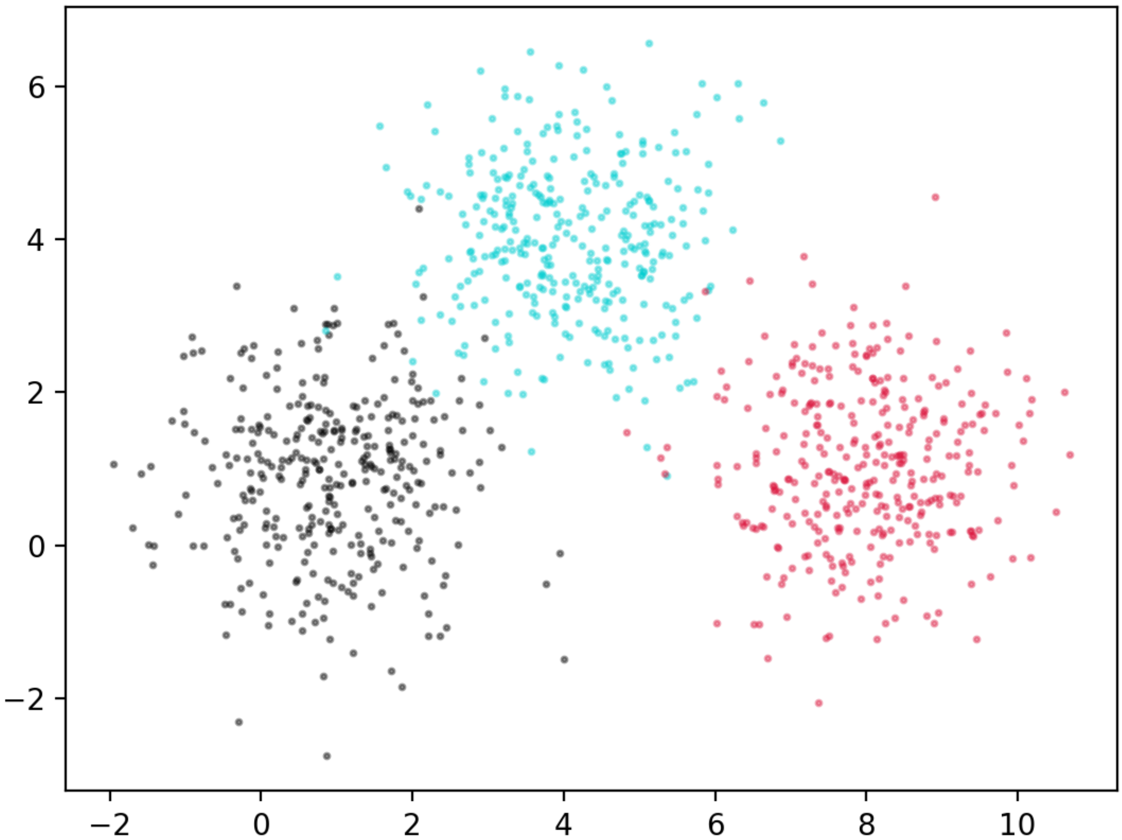
Data: generate two dataset X_1, X_2, including N=1000 two-dimentional random vectors respectively. The random vectors are from three distributions, which has mean value $m1 = [1, 1]^T$, $m2 = [4, 4]^T$, $m3 = [8, 1]^T$ and covariance matrix S1 = S2 = S3 = 2(I is 2*2 indentity matrix).

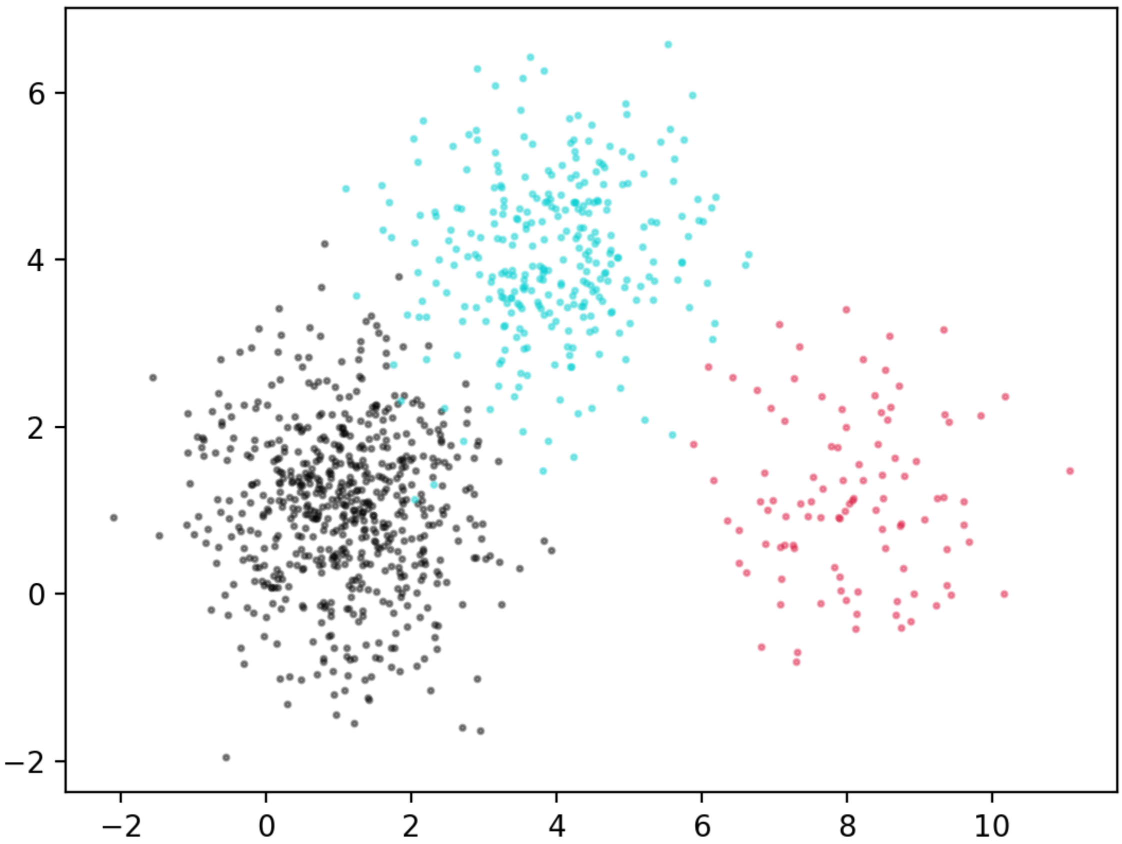
Use random.multivariate_normal from numpy library to generate data.

```
y as np
import matplotlib.pyplot as plt
import math
def f(m, cnt):
    x = []
    y = []
    for i in range(cnt):
        x.append(m[i][0])
        y.append(m[i][1])
    return x, y

def random_x(cnt1, cnt2, cnt3, name):
    cov = [[1, 0], [0, 1]]
    a1 = np.random.multivariate_normal((1, 1), cov, cnt1)
    a2 = np.random.multivariate_normal((4, 4), cov, cnt2)
    a3 = np.random.multivariate_normal((8, 1), cov, cnt3)
    colors0 = '#000000'
    colors1 = '#00CED1' # color
    colors2 = '#DC143C'
    area = np.pi   # area
    x, y = f(a1, cnt1)
    plt.scatter(x, y, s=area, c=colors0, alpha=0.4)
    x, y = f(a2, cnt2)
    plt.scatter(x, y, s=area, c=colors1, alpha=0.4)
    x, y = f(a3, cnt3)
    plt.scatter(x, y, s=area, c=colors2, alpha=0.4)
    ls = []
    result = []
    for i in range(cnt1):
        ls.append(a1[i])
```

```
        result.append(1)
    for i in range(cnt2):
        ls.append(a2[i])
        result.append(2)
    for i in range(cnt3):
        ls.append(a3[i])
        result.append(3)
    plt.figure(num = name)
    return ls, result

x1, c1 = random_x(333, 333, 334)
x2, c2 = random_x(600, 300, 100)
```

The scatter plot of generated data:





Use Maximum Posterior Probability for Classification

1. Calculate according to probability density function $\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(\theta-u)^2}{2\sigma^2}}$.

```
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((data - m).T,
(data - m)))
```

2. Calculate posterior probability respectively and classify the data point into the cluster with maximum posterior probability.

```
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
```

3. Calculate accuracy and visualize with scatter plot.

```
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 = '#000000'
    colors1 = '#00CED1' # color
    colors2 = '#DC143C'
    area = np.pi
    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(x1, (1, 1), (4, 4), (8, 1))
result2 = Classification(x2, (1, 1), (4, 4), (8, 1))

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

show_result(x1, result1)
show_result(x2, result2)
```

```
    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)
    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)
```

Use Kernel Density Estimation (KDE) for Classification

1. Calculate according to probability density function $\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\sqrt{2\pi h^2}}exp\{-\frac{||x-x_n||^2}{2h^2}\}$

```
    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] - data[i]
[1]) ** 2)) / (2 * h * h))
        result = result / (n * math.sqrt(2 * np.pi * h * h))
        return result
```

2. Calculate likelihood respectively and classify the data point into the cluster with maximum likelyhood.

```
    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)
```

3. Classification and visualization.

```
r1_1 = Classification_2(x1, 334, 333, 333, 0.1, c1)
r1_5 = Classification_2(x1, 334, 333, 333, 0.5, c1)
r1_10 = Classification_2(x1, 334, 333, 333, 1, c1)
r1_15 = Classification_2(x1, 334, 333, 333, 1.5, c1)
r1_20 = Classification_2(x1, 334, 333, 333, 2, c1)
```

```
r2_1 = Classification_2(x2, 600, 300, 100, 0.1, c2)
r2_5 = Classification_2(x2, 600, 300, 100, 0.5, c2)
r2_10 = Classification_2(x2, 600, 300, 100, 1, c2)
r2_15 = Classification_2(x2, 600, 300, 100, 1.5, c2)
r2_20 = Classification_2(x2, 600, 300, 100, 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)
```

Experiment Result

The main difference of parametric and non-parametric estimation is that whether assumpt data is from a specific distribution.

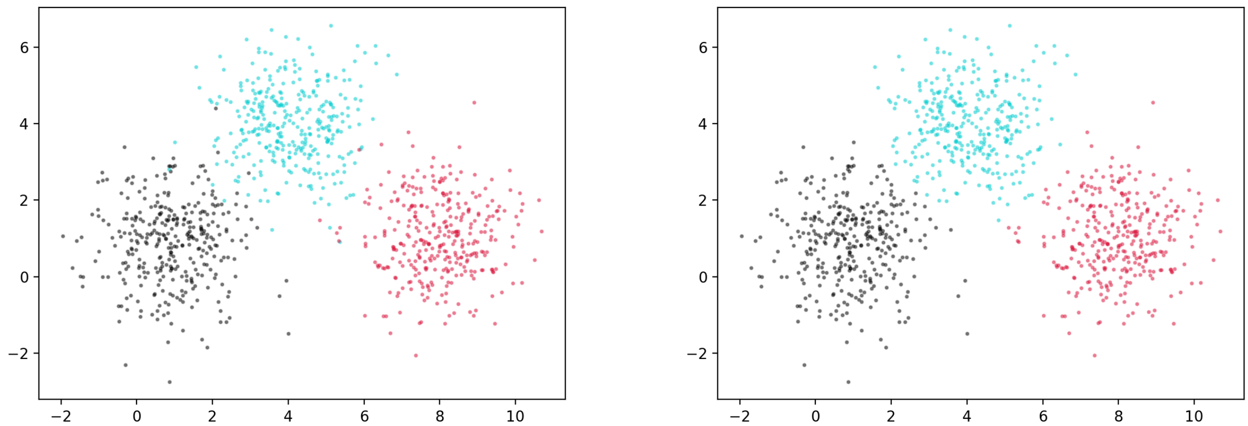
Use Maximum Posterior Probability for Classification

The accuracy is shown as below:

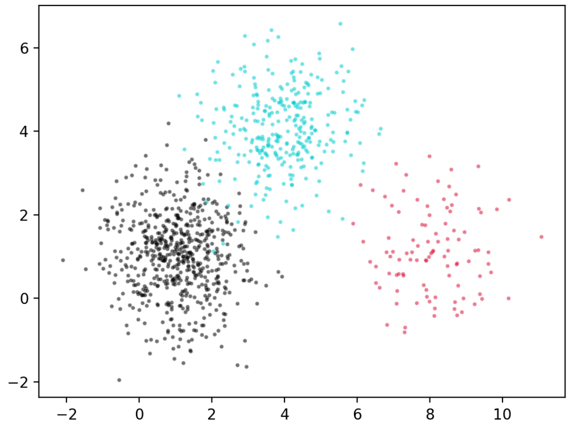
```
X_1数据集使用最大后验概率规则进行分类的错误率是 1.20%
X_2数据集使用最大后验概率规则进行分类的错误率是 2.00%
```

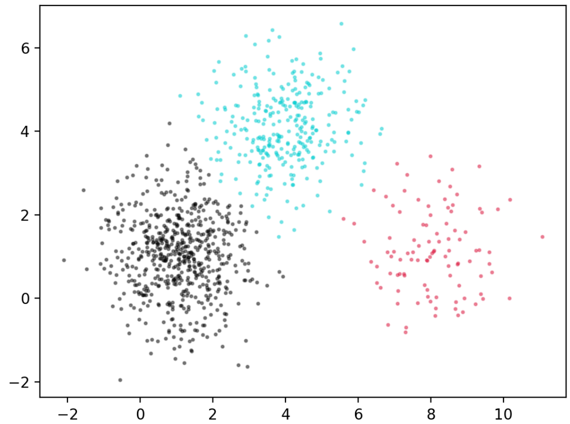
Use scatter plot to compare with original data:

X_1(Original data, left) compared with data classified with Maximum Posterior Probability in X_1(Right)



X_2(Original data, left) compared with data classified with Maximum Posterior Probability in X_2(Right)





In conclusion, using Maximum Posterior Probability for classification has high accuracy, and the classification errors mostly happen to peripheral samples. The clssification technique has a ideal effect and can be implemented.

Use Kernel Density Estimation (KDE) for Classification

```
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   0.20%
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   0.90%
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   1.10%
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   1.20%
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   1.20%
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   0.20%
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   1.90%
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   1.90%
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   2.00%
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   2.00%
```

It can be seen that in the KDE classification experiment, the large size of window(h) can generate an overly smooth probability desity estimation. It makes the structure of data inaccurate and vague, and overlooks data details. If h is too small, the probability desity estimation has a lot of peaks, which makes it hard to find a clear trend. The experiment shows that for the two datasets, the window size h=0.1 has the best effect, where the error rates are both 0.2%.