实验一：贝叶斯判别

实验目的：

分别利用最小错误决策准则、最小风险决策准则、Neyman-Person决策准则，对 Iris 数据进行两两分类。

实验要求：

1. 在假设模型服从正态分布的前提下，对数据分类，并分别写出条件概率密度以及阈值；
2. 写明实验数据的选取方式，说明选取的原因，分析不同选取方式的影响，并给出先验概率；

实验原理：

1. 最小错误决策准则

   最小错误决策准则从最小错误率的要求出发，利用概率论中的贝叶斯公式，得到是错误率最小的分类决策。根据贝叶斯决策公式，错误率最小的决策就是是后验概率最大的决策。因此，对弈两两分类问题，得到的决策规则为：

   如果 $P(w_1|x) > P(w_2|x)$ => $x \in w_1$;反之，则$x \in w_2$

   根据后验概率计算公式，上式转化为：

   $$l(x) = \frac{p(x|w_1)}{p(x|w_2)} > \lambda = \frac{P(w_2)}{P(w_1)}, x \in w_1, \text{反之，则} x \in w_2$$

2. 最小风险决策准则

相较于最小错误决策准则，最小风险决策准则考虑各种错误造成损失，进而得到新的最有决策。

对于二分类问题，在最小错误概率决策准则的基础上，最小风险决策准则的分类公式转换为：

$$l(x)=\frac{p(x|w_1)}{p(x|w_2)}>\frac{P(w_2)}{P(w_1)}*\frac{L_{12}-L_{22}}{L_{21}-L_{11}},x\in w_1,\text{反之，则}x\in w_2$$

3. Neyman-Person 决策准则：

在贝叶斯分类器的设计过程中，如果先验概率未知且固定，但不易求得，可采用 Neyman-Person 决策准则来设计分类器。Neyman-Person 准则，在控制一类错误概率为一定范围内，来是另一类错误概率或者是总的错误概率最小。这里控制在一定范围内的一类错误概率往往是代价更高的错误。在将这个有边界条件的优化问题，利用 Lagrange 乘子法转换为无约束优化问题后，可以求得用于决策面的阈值μ。

公式表示为：

$$\text{如果}l(x)=\frac{p(x|w_1)}{p(x|w_2)}>\mu,x\in w_1,\text{反之，则}x\in w_2$$

其中μ根据以下公式计算：

$$\int_{R_1}p(x|w_2)dx=\varepsilon_0$$

$$\mu=\frac{p(t|w_1)}{p(t|w_2)}$$

实验过程：

本实验利用 Jupyter 软件，python 语言在线建模

0. 数据预处理：

0.1. 首先利用 sklearn 库里面自带的 iris 数据库，并且对数据显示与课本数据进行对照，发现两者数据无相差，可以直接利用此处数据。

```
from sklearn import datasets
import numpy as np
```

```
iris=datasets.load_iris()

#data对应了样本的4个特征，150行4列
print(iris.data.shape)

#target对应样本的标签，159行1列
print(iris.target.shape)

#显示每个类别前2个样本特征
print('iris dataset: \n',iris.data[(0,1,50,51,100,101),:])

#显示样本特征对应的标签
print('corresponding labels: \n', iris.target[(0,1,50,51,100,101),])
```

```
(150, 4)
(150,)
iris dataset:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]]
corresponding labels:
 [0 0 1 1 2 2]
```

0.2. 接着，在基于模型服从高斯分布的假设条件下，利用数据分别估计三类数据的高斯分布公式中待确定参数：均值向量和协方差矩阵

```
#计算3类数据的均值和协方差矩阵
#均值
iris1 = iris.data[:50,:]
iris2 = iris.data[50:100,:]
iris3 = iris.data[101:150,:]

ave1 = np.mean(iris1,0)
ave2 = np.mean(iris2,0)
ave3 = np.mean(iris3,0)

#方差
varin1 = np.cov(iris1.T)
varin2 = np.cov(iris2.T)
varin3 = np.cov(iris3.T)

print('数据集1的均值：\n',ave1)
print('数据集2的均值：\n',ave2)
print('数据集3的均值：\n',ave3)

print('数据集1的方差：\n',varin1)
print('数据集2的方差：\n',varin2)
print('数据集3的方差：\n',varin3)
```

```
数据集1的均值：
 [5.006 3.428 1.462 0.246]
数据集2的均值：
 [5.936 2.77  4.26  1.326]
数据集3的均值：
 [6.59387755 2.96734694 5.54285714 2.01632653]
数据集1的方差：
 [[0.12424898 0.09921633 0.0163551  0.01033061]
 [0.09921633 0.1436898  0.01169796 0.00929796]
 [0.0163551  0.01169796 0.03015918 0.00606939]
 [0.01033061 0.00929796 0.00606939 0.01110612]]
数据集2的方差：
 [[0.26643265 0.08518367 0.18289796 0.05577959]
 [0.08518367 0.09846939 0.08265306 0.04120408]
 [0.18289796 0.08265306 0.22081633 0.07310204]
 [0.05577959 0.04120408 0.07310204 0.03910612]]
数据集3的方差：
 [[0.4110034  0.09771259 0.31235119 0.05301871]
 [0.09771259 0.10391156 0.0697619  0.04533588]
 [0.31235119 0.0697619  0.30666667 0.04532738]
 [0.05301871 0.04533588 0.04532738 0.07222789]]
```

所以3类数据条件概率模型为以上均值和协方差作为参数的多维高斯分布。

0.3. 然后定义用于计算条件概率的高斯函数：

其中主要参数为：

Data:用于计算条件概率的数据 x;

Dimension: 数据 x 的维度;

Mean:条件概率密度函数的高斯分布的均值向量;

Variance:条件概率密度函数的高斯分布的协方差矩阵;

Return 计算的得到的条件概率

```
#定义高斯函数用于计算条件概率
def Gaussfunction(data,dimension,mean,variance):
    constant = 1/(np.power(2 * np.pi,dimension/2 ) * np.sqrt(abs(np.linalg.det(variance))))
    variance_inv = np.linalg.inv(variance)
    error = data - mean
    value = constant * np.exp(-(error.T @ variance_inv @ error)/2.0)
    return value
```

0.4. 定义最小错误决策函数:

重要参数为:

Dim: dimension

M1、M2:待分类的两类数据条件概率密度函数的均值向量;

V1、V2:两类数据条件概率密度函数的协方差矩阵;

Label:2 维列向量, 分别为两类数据的标签;

Threshold:待分类的两类数据的先验概率之比;

返回的 P_x_w1、P_x_w2 以及 decision 分别为: data 在第一类数据是条件概率密度函数中的得到的概率密度;

data 在第二类数据条件概率密度函数中得到的概率密度;

根据最小错误决策准则做出的决策。

```
#定义两两比较的最小错误函数
#其中data为待分类的数据
#dim(ension)为数据集的维度
#m(ean) v(ariance) label = [2 1]
def MiniErrorDecFunc(data, dim, m1, m2, v1, v2, label, threshold):
    P_x_w1 = Gaussfunction(data, dimension, m1, v1)
    P_x_w2 = Gaussfunction(data, dimension, m2, v2)
    likelihood = P_x_w1 / P_x_w2
    decision = label[0]
    if likelihood < threshold:
        decision = label[1]
    return P_x_w1, P_x_w2, decision
```

0.5. 定义最小风险决策函数：

ValueMatrixFunc 函数输入为损失矩阵，输出为最小风险决策函数阈值相对最小错误决策函数阈值的变化因子；MiniRiskDecFunc 函数的重要参数与 MiniErrorDecFunc 函数的参数化相同，唯一不同的 VMatrix 为损失矩阵。

```
#公式：计算权值对阈值影响
def ValueMatrixFunc(V_matrix):
    return (V_matrix[0][1] - V_matrix[1][1]) / (V_matrix[1][0] - V_matrix[0][0])

def MiniRiskDecFunc(data, dimension, m1, m2, v1, v2, label, threshold, VMatrix):
    P_x_w1 = Gaussfunction(data, dimension, m1, v1)
    P_x_w2 = Gaussfunction(data, dimension, m2, v2)
    likelihood = P_x_w1 / P_x_w2
    decision = label[0]

    threshold = ValueMatrixFunc(VMatrix) * threshold

    if likelihood < threshold:
        decision = label[1]
    return P_x_w1, P_x_w2, decision
```

0.6. 定义 Neyman-Person 决策函数：

Calculate_threshold 函数根据第二类分类样本允许的最小错误mini_error 以及待分类两类样本条件概率密度服从的正态分布的均值和方差（m1,v1,m2,v2）计算出 Neyman-Person 决策的阈值 threshold.

在这个函数里，mini_error 为根据第二类样本所允许的

分类错误 sigma=0.06 计算得到的下侧分位数-1.65。

```
#calculate_threshold 根据最小允许的错误率和条件概率分布来计算阈值
def calculate_threshold(mini_error,m1,v1,m2,v2):
    data = m2 + mini_error * np.sqrt(v2)
    P1 = new_gauss_function(data,m1,v1)
    P2 = new_gauss_function(data,m2,v2)
    threshold = P1 / P2
    return threshold
```

```
#Neyman_Person决策函数定义
def Neyman_Person(data,m1,m2,v1,v2,labels,mini_error):
    threshold = calculate_threshold(mini_error,m1,v1,m2,v2)
    P_x_w1 = new_gauss_function(data,m1,v1)
    P_x_w2 = new_gauss_function(data,m2,v2)
    likelihood = P_x_w1 / P_x_w2
    decision = labels[0]

    if likelihood < threshold:
        decision = labels[1]
    return P_x_w1,P_x_w2,decision
```

另外，因为四位样本数据的正态分布的下侧分位数我不知道怎么计算，针对 Neyman-Person 决策实验，我只取了样本数据的第三列特征用于分类，并重新计算了均值、方差，定义了新的 gauss 函数。

```
def new_gauss_function(data,mean,variance):
    constant = 1/(np.sqrt(2 * np.pi * variance))
    error = data - mean
    value = constant * np.exp(-(error * error)/(2.0*variance))
    return value
```

```
iris1 = iris1[:,2]
iris2 = iris2[:,2]
iris3 = iris3[:,2]

junzhi1 = np.mean(iris1,0)
junzhi2 = np.mean(iris2,0)
junzhi3 = np.mean(iris3,0)

#方差
fangcha1 = np.cov(iris1.T)
fangcha2 = np.cov(iris2.T)
fangcha3 = np.cov(iris3.T)

print('数据集1的均值：',junzhi1)
print('数据集2的均值：',junzhi2)
print('数据集3的均值：',junzhi3)

print('数据集1的方差：',fangcha1)
print('数据集2的方差：',fangcha2)
print('数据集3的方差：',fangcha3)
```

```
数据集1的均值： 1.4620000000000002
数据集2的均值： 4.26
数据集3的均值： 5.5520000000000005
数据集1的方差： 0.030159183673469387
数据集2的方差： 0.22081632653061226
数据集3的方差： 0.30458775510204084
```

0.7. 定义计算分类错误率的函数：

Decision 是 100 维列向量，记录待分类两类 100 个样本数据的分类结果。前 50 个第一类，后 50 个为第二类。

```
#定义计算错误率的函数
def rate_function(decision):
    error1 = abs(decision[:50] - label1)
    error2 = abs(decision[50:100] - label2)

    error1[error1>0] = 1
    error2[error2>0] = 1

    error_rate1 = sum(error1) / 50
    error_rate2 = sum(error2) / 50

    print('error_rate1 = %f error_rate2 = %f ' % (error_rate1,error_rate2))
```

实验过程：

1. 最小错误决策：

```
#最小错误决策分类 阈值为1，区分1 2类
P_x_w1 = np.zeros((100,1))
P_x_w2 = P_x_w1
decision = P_x_w1
label1 = 1
label2 = 2
threshold = 1
for i in range(100):
    [P_x_w1[i], P_x_w2[i], decision[i]] = MiniErrorDecFunc(iris.data[i,:],4,ave1,ave2,varin1,varin2,[1,2],threshold)
rate_function(decision)
```

```
error_rate1 = 0.000000 error_rate2 = 0.000000
```

```
# 最小错误决策分类 阈值为1，区分1 3类
P_x_w1 = np.zeros((100,1))
P_x_w3 = P_x_w1
decision = P_x_w1
label1 = 1
label2 = 3
threshold = 1
for i in range(100):
    if i > 49 :
        [P_x_w1[i], P_x_w3[i], decision[i]] = MiniErrorDecFunc(iris.data[i+50,:],4,ave1,ave3,varin1,varin3,[1,3],threshold)

    [P_x_w1[i], P_x_w3[i], decision[i]] = MiniErrorDecFunc(iris.data[i,:],4,ave1,ave3,varin1,varin3,[1,3],threshold)

rate_function(decision)
```

```
error_rate1 = 0.000000 error_rate2 = 0.000000
```

```
#最小错误决策分类 阈值为1，区分1 2类
P_x_w2 = np.zeros((100,1))
P_x_w3 = P_x_w2
decision = P_x_w2
label1 = 2
label2 = 3
threshold = 1
for i in range(100):
    [P_x_w2[i], P_x_w3[i], decision[i]] = MiniErrorDecFunc(iris.data[i+50,:],4,ave2,ave3,varin2,varin3,[2,3],threshold)

rate_function(decision)
```

```
error_rate1 = 0.040000 error_rate2 = 0.020000
```

```
#最小错误决策分类 阈值为10， 区分2 3 类
P_x_w2 = np.zeros((100,1))
P_x_w3 = P_x_w2
decision = P_x_w2
label1 = 2
label2 = 3

for threshold in range(10):
    for i in range(100):
        [P_x_w2[i], P_x_w3[i], decision[i]] = MiniErrorDecFunc(iris.data[i+50,:],4,ave2,ave3,varin2,varin3,[2,3],threshold)
    rate_function(decision)
```

```
error_rate1 = 0.000000 error_rate2 = 1.000000
error_rate1 = 0.040000 error_rate2 = 0.020000
error_rate1 = 0.040000 error_rate2 = 0.000000
error_rate1 = 0.040000 error_rate2 = 0.000000
error_rate1 = 0.060000 error_rate2 = 0.000000
error_rate1 = 0.060000 error_rate2 = 0.000000
error_rate1 = 0.060000 error_rate2 = 0.000000
error_rate1 = 0.060000 error_rate2 = 0.000000
error_rate1 = 0.060000 error_rate2 = 0.000000
error_rate1 = 0.080000 error_rate2 = 0.000000
```

结果分析：前三幅图片分别对（1,2），（1,3），（2,3）类各 50 样本数据在阈值为 1 的条件下进行了分类，发现（1,2），（1，3）两对的所有数据都正确分类，而（2,3）两类数据存在分类错误，错误率分别为 0.04 和 0.02。另外，针对（2,3）两类数据，通过改

变阈值条件从 0 到 10 变化，发现，第二类样本数据的准确度逐渐提升，第一类样本数据准确地逐渐下降。从先验概率考虑的话，就是我们第二类数据的先验概率越大的话，阈值也就越大，进而分类的时候，我们更倾向于把样本分类到第二类，以确保分类的错误率最小。

2. 最小风险决策：

```
#最小风险决策分类 阈值为1 L12为10
#对1 2 类数据进行分类
P_x_w1 = np.zeros((100,1))
P_x_w1 = P_x_w1
decision = P_x_w1
label1 = 1
label2 = 2

#损失矩阵
Vmatrix=[[0,10],[1,0]]
threshold = 1
for i in range(100):
    [P_x_w1[i], P_x_w2[i], decision[i]] = MiniRiskDecFunc(iris.data[i,:],4,ave1,ave2,varin1,varin2,[1,2],threshold,Vmatrix)
rate_function(decision)
```

```
error_rate1 = 0.000000 error_rate2 = 0.000000
```

```
#最小风险决策分类 阈值为1 L12为10
#对1 3 类数据进行分类
P_x_w1 = np.zeros((100,1))
P_x_w3 = P_x_w1
decision = P_x_w1
label1 = 1
label2 = 3

#损失矩阵
Vmatrix=[[0,10],[1,0]]
threshold = 1
for i in range(100):
    if i > 49:
        [P_x_w1[i], P_x_w3[i], decision[i]] = MiniRiskDecFunc(iris.data[i+50,:],4,ave1,ave3,varin1,varin3,[1,3],threshold,Vmatrix)

    [P_x_w1[i], P_x_w3[i], decision[i]] = MiniRiskDecFunc(iris.data[i,:],4,ave1,ave3,varin1,varin3,[1,3],threshold,Vmatrix)

rate_function(decision)
```

```
error_rate1 = 0.000000 error_rate2 = 0.000000
```

```
#最小风险决策分类 阈值为1 L12为10
#对2 3 类数据进行分类
P_x_w2 = np.zeros((100,1))
P_x_w3 = P_x_w2
decision = P_x_w2
label1 = 2
label2 = 3

#损失矩阵L12=10
Vmatrix=[[0,10],[1,0]]
threshold = 1

for i in range(100):
    [P_x_w2[i], P_x_w3[i], decision[i]] = MiniRiskDecFunc(iris.data[i+50,:],4,ave2,ave3,varin2,varin3,[2,3],threshold,Vmatrix)

rate_function(decision)
```

```
#最小风险决策分类 阈值为1 L12为100
Vmatrix=[[0,100],[1,0]]
threshold = 1

for i in range(100):
    [P_x_w2[i], P_x_w3[i], decision[i]] = MiniRiskDecFunc(iris.data[i+50,:],4,ave2,ave3,varin2,varin3,[2,3],threshold,Vmatrix)

rate_function(decision)
```

```
#最小风险决策分类 阈值为10 L12为100
Vmatrix=[[0,100],[1,0]]
threshold = 10

for i in range(100):
    [P_x_w2[i], P_x_w3[i], decision[i]] = MiniRiskDecFunc(iris.data[i+50,:],4,ave2,ave3,varin2,varin3,[2,3],threshold,Vmatrix)

rate_function(decision)
```

```
error_rate1 = 0.100000 error_rate2 = 0.000000
error_rate1 = 0.220000 error_rate2 = 0.000000
error_rate1 = 0.400000 error_rate2 = 0.000000
```

结果分析：在阈值为 1，损失矩阵为$\begin{bmatrix}0 & 10\\1 & 0\end{bmatrix}$的条件下，(1,2)，(1,3)分类结果与最小错误决策一样，所用样本都被正确地分类，(2,3）两类分类因为第二类样本数据分类错误的代价相对较大，所以从最后的分类结果来看，此时 2 类数据的错误率相对于最小错误决策有所增加，但是保全了第二类 3 类样本数据的分类准确率；在阈值为 1，损失矩阵为$\begin{bmatrix}0 & 100\\1 & 0\end{bmatrix}$的条件下，(2,3）类中 2 类样本数据随着第二类数据分类错误代价的提升出现更多的错误；在阈值为 10，损失矩阵为$\begin{bmatrix}0 & 100\\1 & 0\end{bmatrix}$的条件下，(2,3）2 类样本数据的错误率进一步增加。所以，增大阈值或者增大第二类样本数据分类错误的代价，都能够通过牺牲第一类分类数据的正确率来保全第二类样本分类正确率的提升。

3. Neyman-Person 决策：

```
#Neyman-Person决策 第二类错误率0.06 对1 2 类样本分类
mini_error = -1.65
threshold = 0
label1 = 1
label2 = 2
for i in range(100):
    if i < 50:
        [P_x_w1[i],P_x_w2[i],decision[i],threshold] = Neyman_Person(iris1[i],junzhi1,junzhi2,fangcha1,fangcha2,[1,2],mini_error)
    else:
        [P_x_w1[i],P_x_w2[i],decision[i],threshold] = Neyman_Person(iris2[i-50],junzhi1,junzhi2,fangcha1,fangcha2,[1,2],mini_error)

print('threshold = ',threshold)
rate_function(decision)
```

```
threshold =  3.693297815403447e-29
error_rate1 = 0.000000 error_rate2 = 0.060000
```

```
#Neyman-Person决策 第二类错误率0.06 对1 3 类样本分类
mini_error = -1.65
label1 = 1
label2 = 3
for i in range(100):
    if i < 50:
        [P_x_w1[i],P_x_w2[i],decision[i],threshold] = Neyman_Person(iris1[i],junzhi1,junzhi3,fangcha1,fangcha3,[1,3],mini_error)
    else:
        [P_x_w1[i],P_x_w2[i],decision[i],threshold] = Neyman_Person(iris3[i-50],junzhi1,junzhi3,fangcha1,fangcha3,[1,3],mini_error)

print('threshold = ',threshold)
rate_function(decision)
```

```
threshold =  2.052812212979856e-72
error_rate1 = 0.000000 error_rate2 = 0.020000
```

```
#Neyman-Person决策 第二类错误率0.06 对2 3类样本分类
mini_error = -1.65
label1 = 2
label2 = 3
for i in range(100):
    if i < 50:
        [P_x_w1[i],P_x_w2[i],decision[i],threshold] = Neyman_Person(iris2[i],junzhi2,junzhi3,fangcha2,fangcha3,[2,3],mini_error)
    else:
        [P_x_w1[i],P_x_w2[i],decision[i],threshold] = Neyman_Person(iris3[i-50],junzhi2,junzhi3,fangcha2,fangcha3,[1,3],mini_error)

print('threshold = ',threshold)
rate_function(decision)
```

```
threshold =  3.2960575426499115
error_rate1 = 0.220000 error_rate2 = 0.020000
```

结果分析：在 sigma=0.06 的条件下，查询标准正态分布表得到标准正态分布的下侧分位数为 mini-error = -1.65。基于此，通过对（1,2）（1,3）（2,3）类数据两两分类，验证了第二类样本数据的错误率保证在 0.06 以内。但是，（2,3）类数据特征比较相似，在保证 3 类样本数据错误率 0,06 以内的条件下，2 类样本数据错误率较高。

4. 实验要求回复：

实验数据以及先验概率的选取：

错误率		20个样本数据		30个样本数据		40个样本数据		50个样本数据	
最小错误	(1,2)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	(1,3)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	(2,3)	0.04	0.00	0.04	0.02	0.04	0.00	0.04	0.02
最小风险	(1,2)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	(1,3)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	(2,3)	0.06	0.00	0.06	0.00	0.01	0.00	0.01	0.00
NP	(1,2)	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.06
	(1,3)	0.00	0.02	0.00	0.02	0.00	0.02	0.00	0.02
	(2,3)	0.28	0.02	0.28	0.02	0.22	0.02	0.22	0.02

以上表格，分别利用样本数据的前 20、30、40 和 50 个数据，得到应用于最小错误决策和最小风险决策不同的条件概率密

度函数，以及应用于Neyman-Person决策不同的阈值，反映了样本数据对实验结果的影响。总的来说，20个样本数据及更多对最后的样本分类结果影响不大。在以上样本数目中，40个样本数据比其他数据要相对好一些，但是影响不大。

因为先验概率的定义，是为了计算阈值。不同的先验概率得到不同的阈值。所以，本实验中，相比于定义不同的先验概率在计算阈值，我直接定义了不同的阈值，来测试不同阈值条件下，分类准确度的变化。

实验结论：

综上所有实验数据来看：

1. 40个样本数据比其他样本数据得到的样本数据分类结果要相对好一些；
2. 阈值越大，L12越大，第二类样本数据的分类准确度越高；
3. 1类样本数据特征与2、3类样本数据的特征差别较大，无论是用最小错误决策还是最小风险决策，甚至Neyman-Person决策只用了1个特征，都能够较容易对其进行分类。但是2、3类样本数据的特征比较相似，最小错误决策和最小风险决策存在一定的分类错误，Neyman-Person决策利用1维特征，对其分类时错误率甚至高达0.28.

实验思考：

本实验中应用 Neyman-Person 决策准则对 iris 数据 4 维特征进行分类时，Neyman-Person 中求解阈值所提及的解方程组方法和利用试探法计算似然比密度函数积分的方法，我都尝试了，但是因为知识水平有限，没有找到正确的思路来求解。

解方程组的方法，需要计算 4 重积分然后解方程，直接求解不容易；试探法计算似然比密度函数积分的方法，无法得到似然比密度函数的表达式形式，我也没有找到可行的计算方法。另外，我尝试通过样本数据计算 l(x),P(x|w2),然后利用一些插值方法来近似求解积分，但是因为我们的样本数据中大部分比较容易分开,也就是说,大多数 l(x)在 0 附近，可以用于插值的点不多，所以这种拟合函数求解的方法也应用不起来。所以，我最终选择了利用 1 维样本特征结合正态分布表来进行了 Neyman-Perosn 决策实验。

如果有同学或者老师找到了可行计算的话，我完善再修正我的实验。