School of Automation Science and Electrical Engineering
160324, 16711094, 李翰韬

Experiment of pattern recognition I

I. Perceptron Learning toward Linear Classification

i). Principle and Theory

If we obtained the samples of two classes are linearly separable in the feature space. The goal of perceptron learning is to find a separating plane which can separate the two classes of points of training set. We can define a plane which has the following expression:

$$w \cdot x + b = 0$$

Where w is the parameter and x is the data.

Take the two-dimensional plane as an example, w has two parameters x_0, x_1. So, the hyperplane $w*x+b=0$ in two dimensions is a straight line. After finding the hyperplane, the upper part of the hyperplane is a positive class and the lower part is a negative class, which can separate the linear separable pattern datasets.

ii). Contents and Procedure

Stage 1:

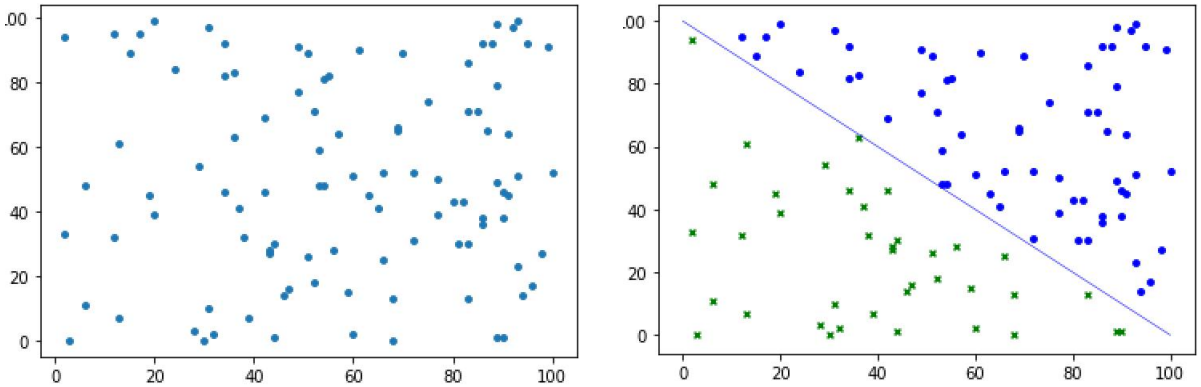
To create the samples of the two pattern datasets which are linearly separable, we can manually create a line in a series of random lattice. In this way, the points on both sides of the straight line can be automatically classified into two classes of linear separable pattern datasets.

The program takes 100 random points in the first quadrant coordinate, horizontal and vertical coordinates are x_0, x_1. The dividing line is $L: x_0-x_1+100=0$. Each pattern dataset contains approximately 50 samples, which meets the experimental requirements. The code is as follows:

```
def data(num):

    X = np.random.randint(0,101,(2,num))
    y = np.zeros(num)
    y[X[0,:]>=(100-X[1,:])] = 1
    y[X[0,:]<(100-X[1,:])] = -1
    return X,y
```

The point graph and the separated pattern datasets are shown in the following figure:



According to the definition, we can clearly find that this line is a hyperplane that meets the requirements. Now we use perceptron learning to recalculate a new line(hyperplane).

Initialization parameter: *w* is initialized to [0.5, 0.5], *b* is initialized to 0, and learning rate is set to 0.02.

```
w[0] = 0.5
w[1] = 0.5
b = 0
rho = 0.02 #parameters
```

For the misclassification points (x_i,y_i), $-y_i(w*x_i+b)>0$, write program to determine which data has misclassification and return a Boolean array:

```
def wrongclass(X,y,w,b):

    return y*(w.dot(X)+b)<0
```

Therefore, the loss function can be obtained by adding up all misclassification points:

$$-\sum y_i(w \cdot x_i + b)$$

Write program to calculate the cost:

```
def costcompute(X,y,w,b,c):

    return -np.sum(y[c]*(w.dot(X[:,c])+b))
```

It is obviously that as long as this function is continuously optimized, *w* and *b* can be obtained. Gradient descent method is used as optimization method.

The partial derivatives of *w* and *b* are calculated respectively. With the gradient and learning rate, we can obtain the new *w* and *b:*

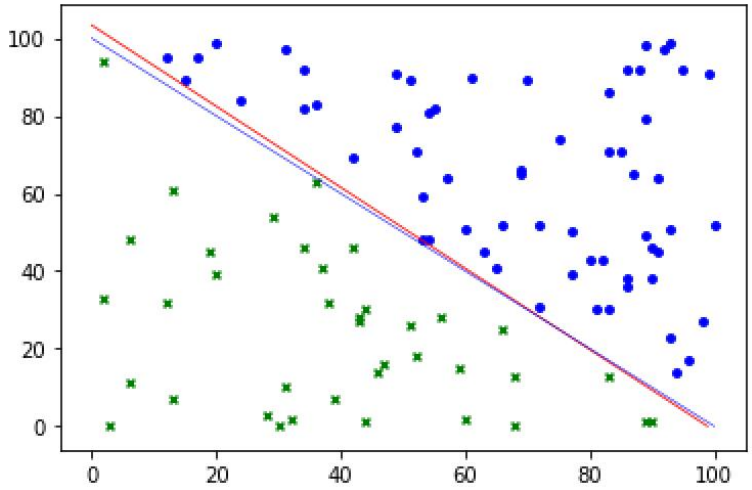
```
def grad(X,y,c):

    dw = -np.sum(y[c]*X[:,c],axis=1)
    db = -np.sum(y[c])
    return dw,db

def updata_parameters(w,b,dw,db,learning_rate):

    w = w-learning_rate*dw
    b = b-learning_rate*db*100
    return w,b
```

Run the program and we can get the line which is calculated by the program. The result of the program is as follows:



w=[26.12, 25]; b=-2584.0; the number of iterations for the convergence is 191.

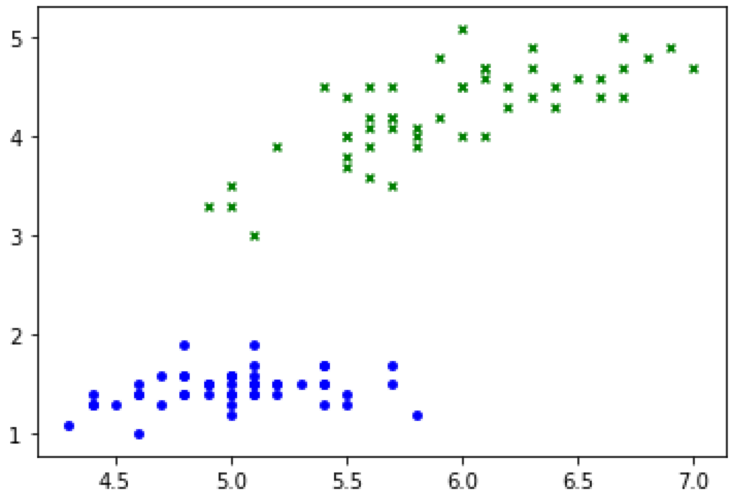
It can be seen that the final hyperplane line is very similar to the line splitting the pattern datasets, which meets the experimental requirements.

Stage 2:

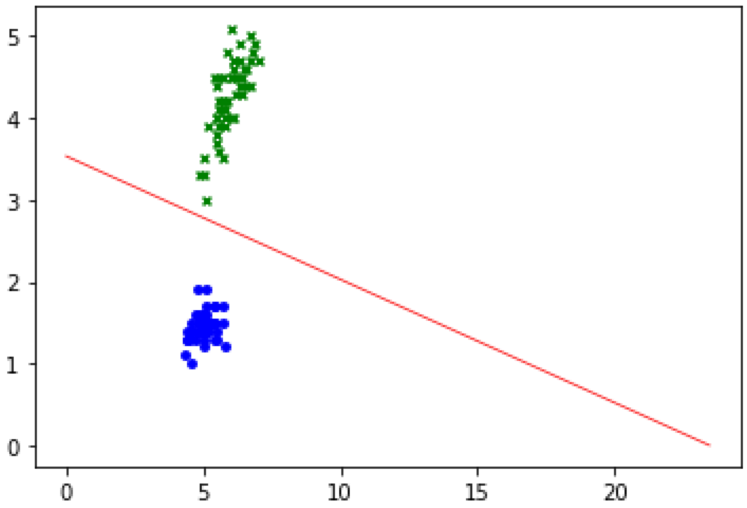
Repeat the test, use the data of 100 iris leaves in learning database *https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data* instead of the previous data.

```
def irisdata(num):

    df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
    y = df.iloc[0:num, 4].values
    y = np.where(y == 'Iris-setosa', 1, -1)
    X = df.iloc[0:num, [0, 2]].values
    X = X.T
    return X,y
```



It can be found that the pattern datasets called in this experiment are far more scattered than the previous random number datasets. Still run the program with the same initial parameters as the last experiment and the learning rate parameter of 0.02.

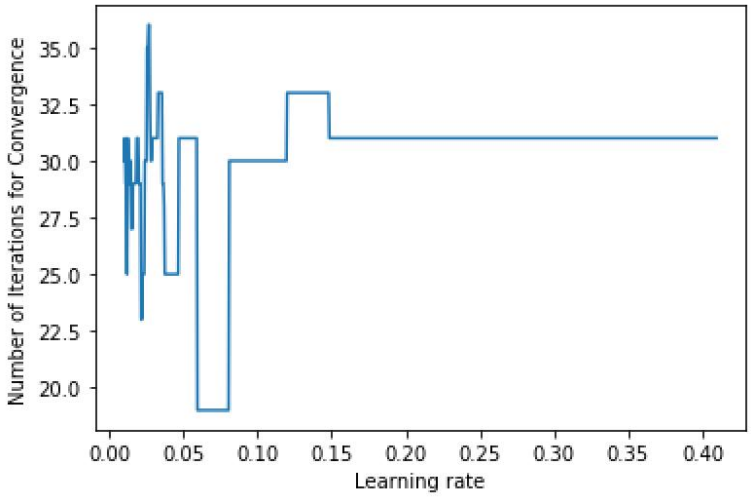


w=[-5.03, -33.376]; b=118.0; the number of iterations for the convergence is 29.

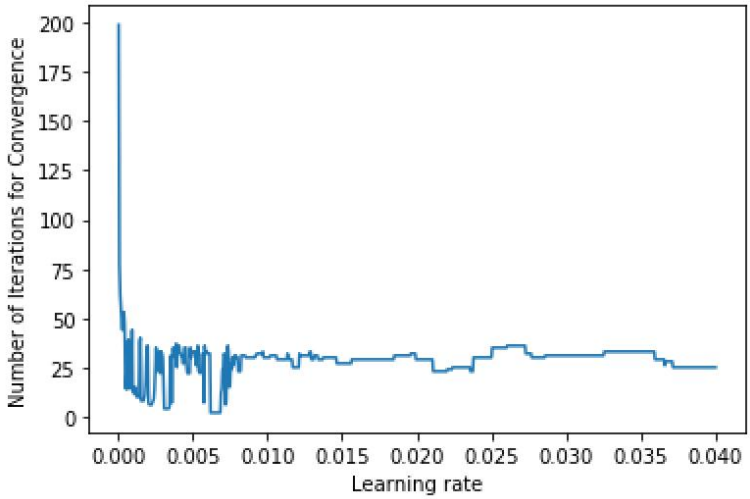
When the separation between the two pattern datasets increases, the number of iterations needed for the convergence in the program operation decreases significantly.

Stage 3:

Repeated the experiments with different learning rates, we can obtain the following results:



For small learning rate, the following results can be obtained:



Stage 4:

It can be seen that for the pattern datasets of iris, when learning rate approaches 0^+, the number of iterations for the convergence will approach infinity.

It is very clearly to understand that phenomenon. When the learning rate is very small, the step size of each gradient descent will also very small, thus it needs huge number of times of gradient descent to get the minimum value of gradient.

II. Synthetical Design of Bayesian Classifier

i). Principle and Theory

The minimum risk Bayes decision is to choose the decision that causes the minimum loss (i.e. risk) when all kinds of wrong decisions cause different losses.

The posterior probability can be derived as follows:

$$P(\omega_i|X)=\frac{P(X|\omega_i)P(\omega_i)}{\sum_{j=1}^{c}P(X|\omega_j)P(\omega_j)} \qquad \text{i=1,…, c}$$

The conditional risk of decision action a can be computed by:

$$R(a_i|X)=\sum_{j=1,j\neq i}^{c}\lambda(a_i,\omega_j)P(\omega_j|X), \qquad \text{i=1,…, c}$$

Thus the minimum risk Bayesian decision can be found as:

$$a_k^* = \text{Arg min}_i\ R(a_i|X), \qquad \text{i=1,…, c}$$

ii). Contents and Procedure

Stage 1:

In order to get the probability density function more simply, we use the Matlab to code the program.

Firstly, input the cell data given in the text into Matlab. Because that the sampling data accords with the Gaussian normal distribution, we can get the mean value and standard deviation of the normal distribution through the data, in this way, we can get the prior probability density functions.

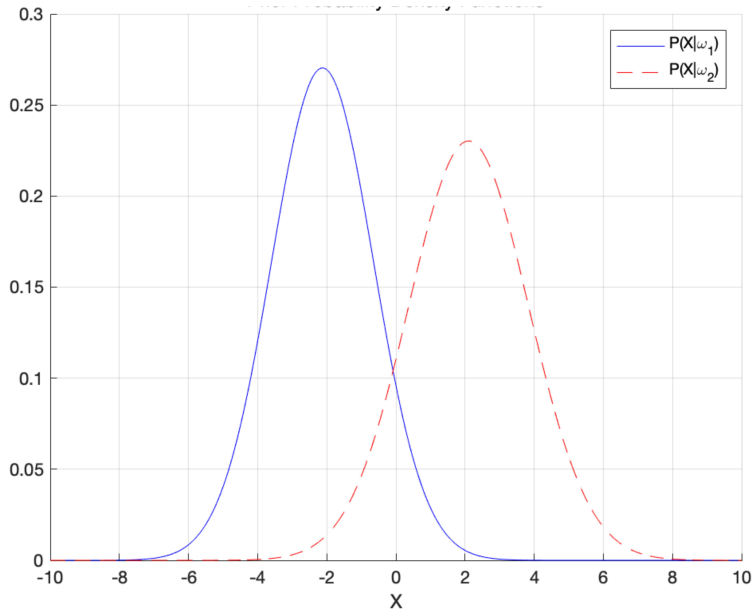
Although the values given by the lost parameters are contrary to the general judgment of cancer cells, we still calculate according to the original data.

```
x1 = [-3.9847,-3.5549,-1.2401,-0.9780,-0.7932,-2.8531,-2.7605,-3.7287,-3.5414,...
    -2.2692,-3.4549,-3.0752,-3.9934,-0.9780,-1.5799,-1.4885,-0.7431,-0.4221,...
    -1.1186,-2.3462,-1.0826,-3.4196,-1.3193,-0.8367,-0.6579,-2.9683];
x2 = [2.8792,0.7932,1.1882,3.0682,4.2532,0.3271,0.9846,2.7648,2.6588];
[miu1,sigma1]=normfit(x1);
[miu2,sigma2]=normfit(x2);
sqsigma1 = sigma1*sigma1;
sqsigma2 = sigma2*sigma2;

P_w1=0.9;
P_w2=0.1;

y(1,1)=0;y(1,2)=1;
y(2,1)=6;y(2,2)=0;
```

We can draw the curves of conditional probability density functions:



For the cells to be tested, according to the Bayesian formula, the posterior probability and conditional risk of their pattern data under two classifications are calculated by the program.

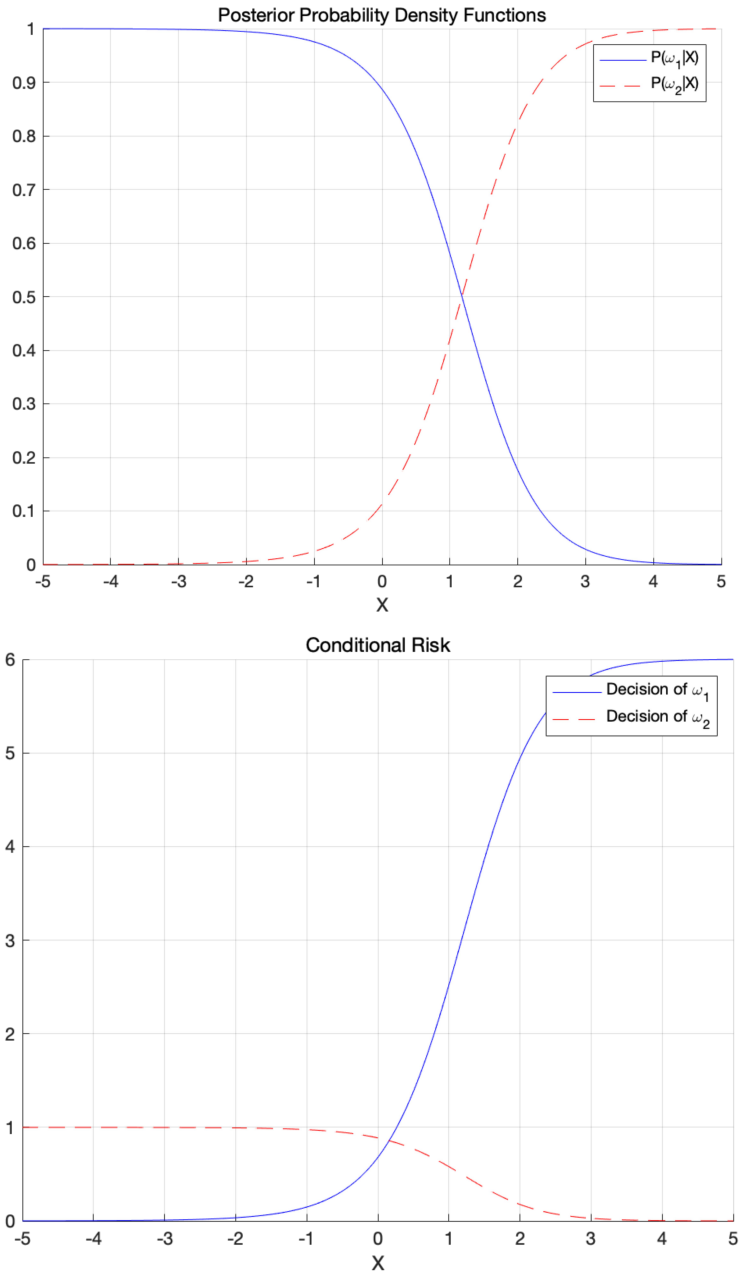
```
for i = 1:m
pw1_x(i)=(P_w1*normpdf(x(i),miu1,sqsigma1))/...
    (P_w1*normpdf(x(i),miu1,sqsigma1)+P_w2*normpdf(x(i),miu2,sqsigma2));

pw2_x(i)=(P_w2*normpdf(x(i),miu2,sqsigma2))/...
    (P_w1*normpdf(x(i),miu1,sqsigma1)+P_w2*normpdf(x(i),miu2,sqsigma2));
end

for i=1:m
    r1_x(i)=y(1,1)*pw1_x(i)+y(2,1)*pw2_x(i);
    r2_x(i)=y(1,2)*pw1_x(i)+y(2,2)*pw2_x(i);
end
```

By comparing the posterior probability, we can get the minimum error Bayesian classifier; by comparing risk, we can get the minimum risk Bayesian classifier.

We can create sample points, and then calculate them, so as to get the image of posterior probability density functions and the conditional risk:



We can see that the existence of loss parameter affects the result of classification obviously.

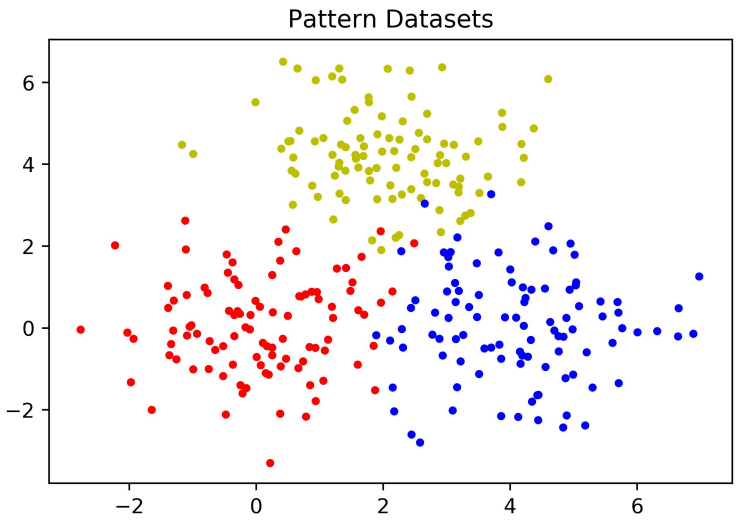
Stage 2:

In order to fulfill the experimental requirements, we directly use the *numpy* function in python to create a two-dimensional, three categories of normal distribution pattern dataset.

Each category contains 100 data points, the mean value and standard deviation of the normal distribution are as follows:

```
sampleNo = 100
mu1 = np.array([[0,0]])
mu2 = np.array([[2,4]])
mu3 = np.array([[4,0]])
Sigma1 = np.array([[1.2,0], [0,1.2]])
Sigma2 = np.array([[1.3,0], [0,1.3]])
Sigma3 = np.array([[1.4,0], [0,1.4]])
```

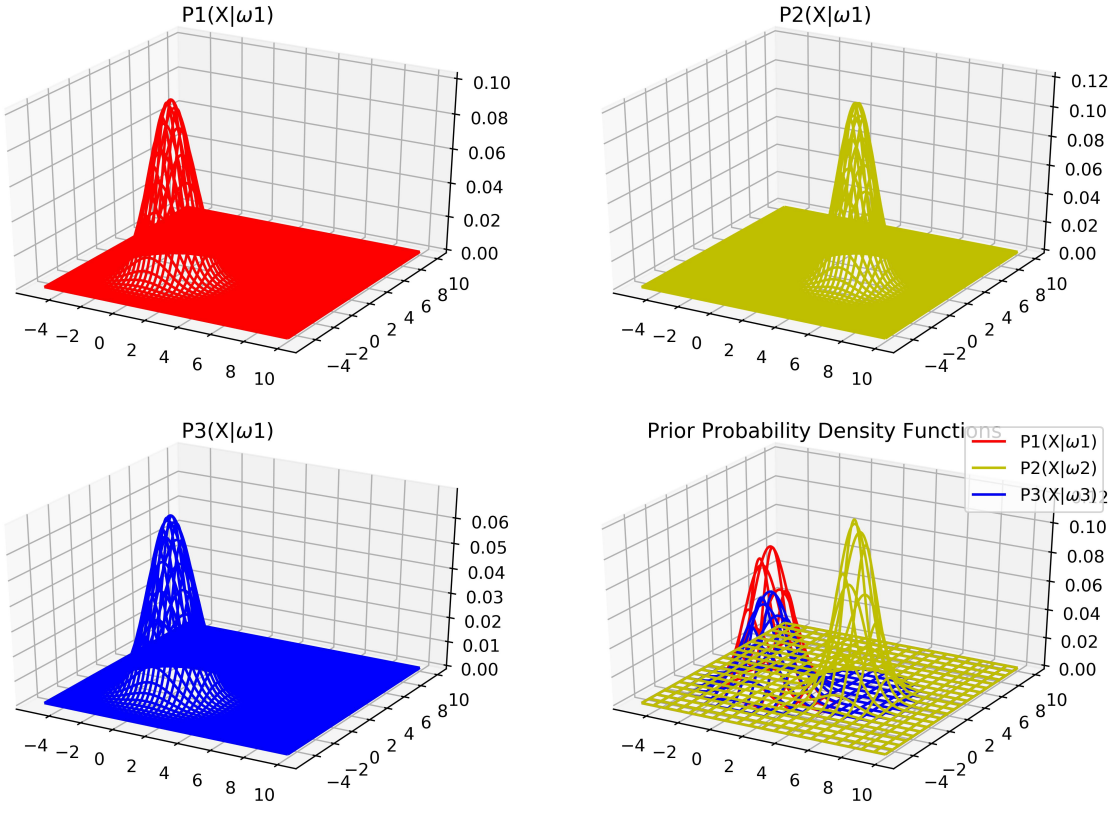
The two-dimensional Gaussian scatter diagram is shown in the figure below:



For the convenience of the observation of experimental results, our prior probability and the loss parameters of different decisions are set as:

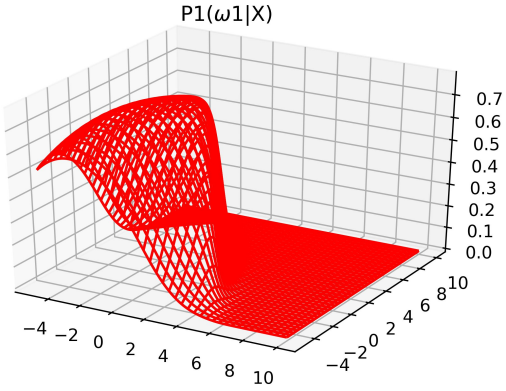
$$\begin{cases} P(\omega_1) = 0.5 \\ P(\omega_2) = 0.3 \\ P(\omega_3) = 0.2 \end{cases} , \quad \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 4 \\ 2 & 4 & 0 \end{bmatrix}$$

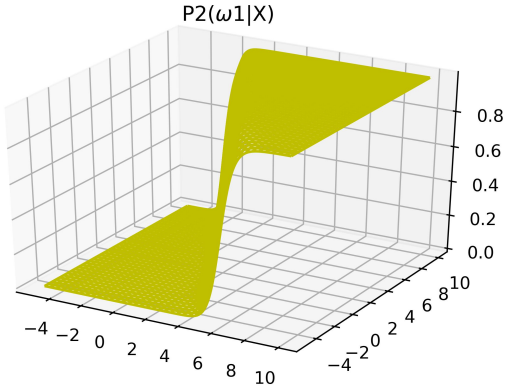
We can draw the curves of conditional probability density functions:

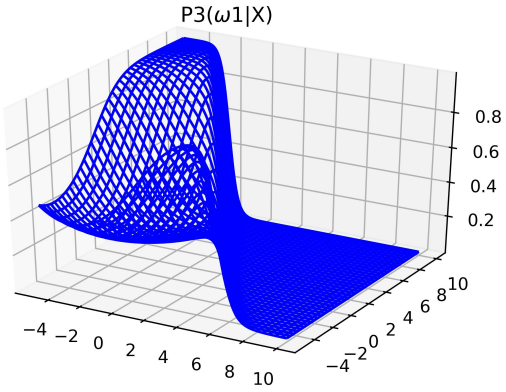


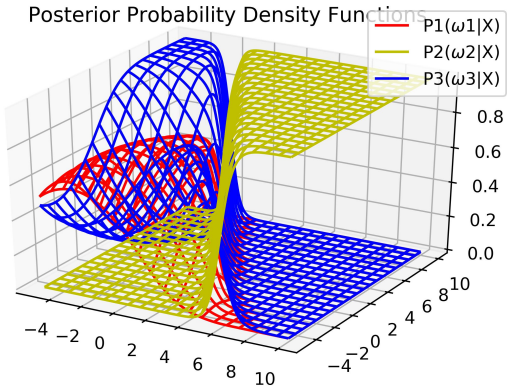
We can also create a series of sample points, and then calculate them, so as to get the image of posterior probability density functions and the conditional risk:

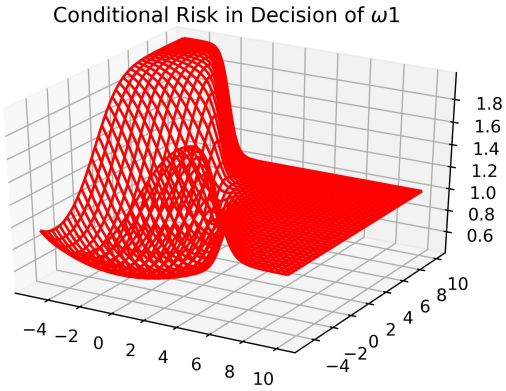
The function codes of the two steps of the experiment are very similar, thus they are not repeated again.

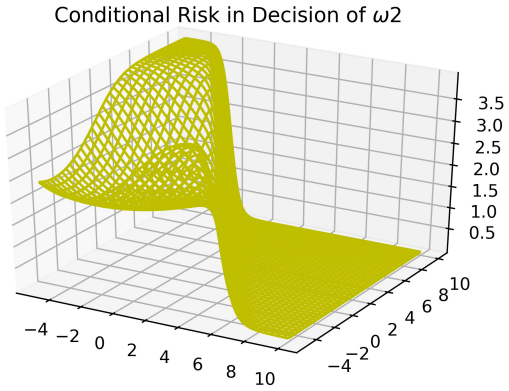


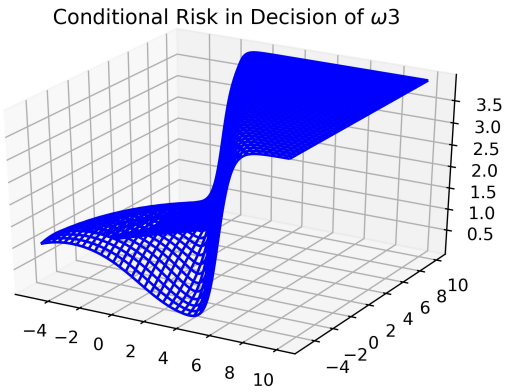


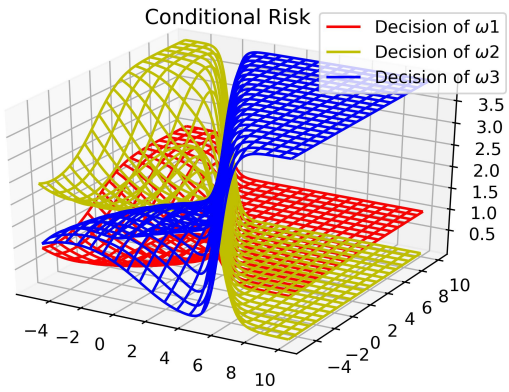




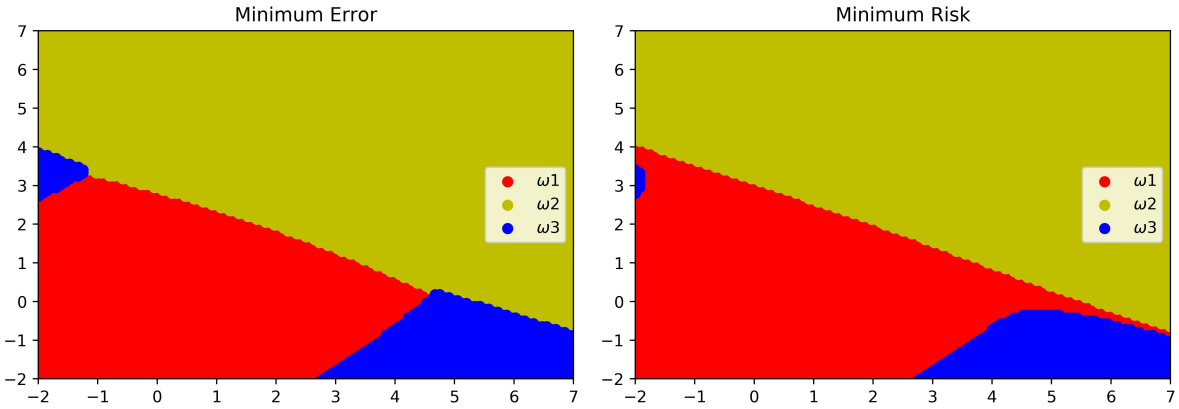








Through these images, we can easily get the results of two types of classifiers:



It can be seen that the program can meet the experimental requirements.

Stage 3:

By the programming, we can understand that although the data dimensions of the two experiments are different, the number of categories is different, and even the programming language which we used is different, there is no obviously difference between the algorithm of the program and the way of coding the program.

The loss parameters will affect the classification results. But in the two experimental results above, the degree of influence is not very obvious.