西安电子科技大学

计算机学院

模式识别上机报告

班　　级：1403013

姓　　名：周林茂

学　　号：14030130098

完成时间：2017.5.22

实验一、Bayes 分类器设计

1.1 实验类型：

基础型：Bayes 分类器设计

1.2 实验目的：

本实验旨在让同学对模式识别有一个初步的理解，能够根据自己的设计对贝叶斯决策理论算法有一个深刻地认识，理解二类分类器的设计原理。

1.3 实验原理：

最小风险贝叶斯决策可按下列步骤进行：

(1)在已知 $P(\omega_i)$，$P(X|\omega_i)$，i=1,…，c 及给出待识别的 X 的情况下，根据贝叶斯公式计算出后验概率：

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

(2)利用计算出的后验概率及决策表，按下面的公式计算出采取 a_i, i=1,…，a 的条件风险

$$R(a_i|X)=\sum_{j=1}^{c}\lambda(a_i,\omega_j)P(\omega_j|X)\text{, i=1, 2, …, a}$$

(3)对(2)中得到的 a 个条件风险值 $R(a_i|X)$, i=1,…，a 进行比较，找出使其条件风险最小的决策 a_k，则 a_k 就是最小风险贝叶斯决策。

1.4 实验内容：

假定某个局部区域细胞识别中正常（ω_1）和非正常（ω_2）两类先验概率分别为

正常状态：P（ω_1）=0.9；

异常状态：P（ω_2）=0.1。

现有一系列待观察的细胞，其观察值为 x：

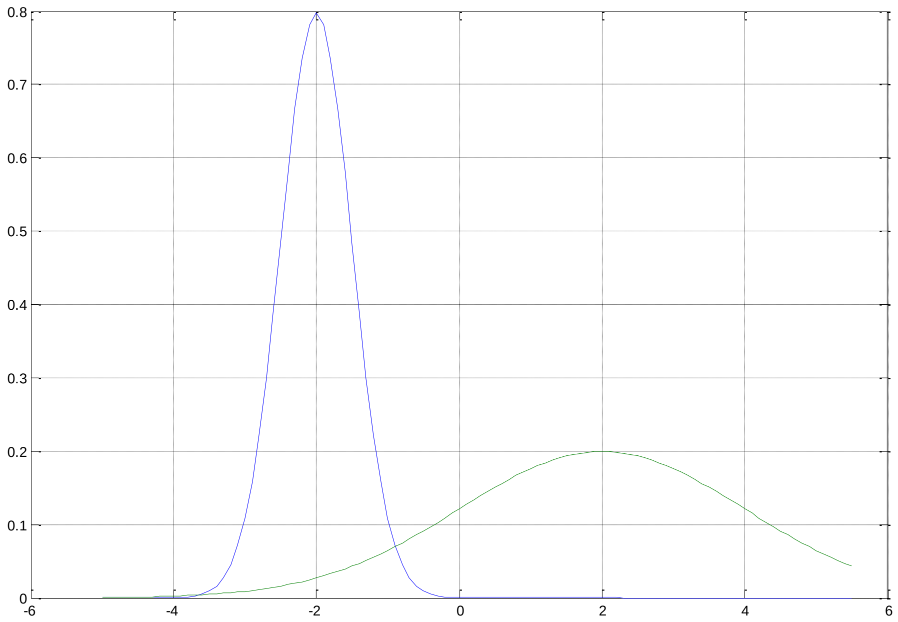
-3.9847　-3.5549　-1.2401　-0.9780　-0.7932　-2.8531

-2.7605　-3.7287　-3.5414　-2.2692　-3.4549　-3.0752

-3.9934　2.8792　-0.9780　0.7932　1.1882　3.0682

-1.5799　-1.4885　-0.7431　-0.4221　-1.1186　4.2532

$p(x\mid\omega_1)$ 和 $p(x\mid\omega_2)$ 类条件概率分布正态分布分别为（-2，0.25）和（2,4），试对观察的结果进行分类。



1.5 实验要求：

1） 完成分类器的设计，要求程序相应语句有说明文字。

2） 如果是最小风险贝叶斯决策，决策表如下：

最小风险贝叶斯决策表：

状态 决策	ω_1	ω_2
α 1	0	6
α 2	1	0

请重新设计程序，画出相应的后验概率的分布曲线和分类结果，并比较两个结果

解决问题

设计思路：根据 bayes 分类器的思想及其相应公式实现分类器，一种是最小错误率 bayes 分类器，使用后验概率进行判别，另一种是最小风险 bayes 分类器，在最小错误率分类器的基础上加入每一种判别的风险，最后根据风险最小原则判别。

主要函数：

【1】.求解类条件概率

```
def getClassConditionP(self):
returnmath.exp(-math.pow(x-self.mu,2)/(2.0*self.segma))/math.sqrt(2.0*math.pi*self.segma)
```

【2】.求解后验概率

```
def getAfterP(self,b,x):
    return self.getClassConditionP(x)*self.p/(self.getClassConditionP(x)*self.p+
            b.getClassConditionP(x)*b.p)
```

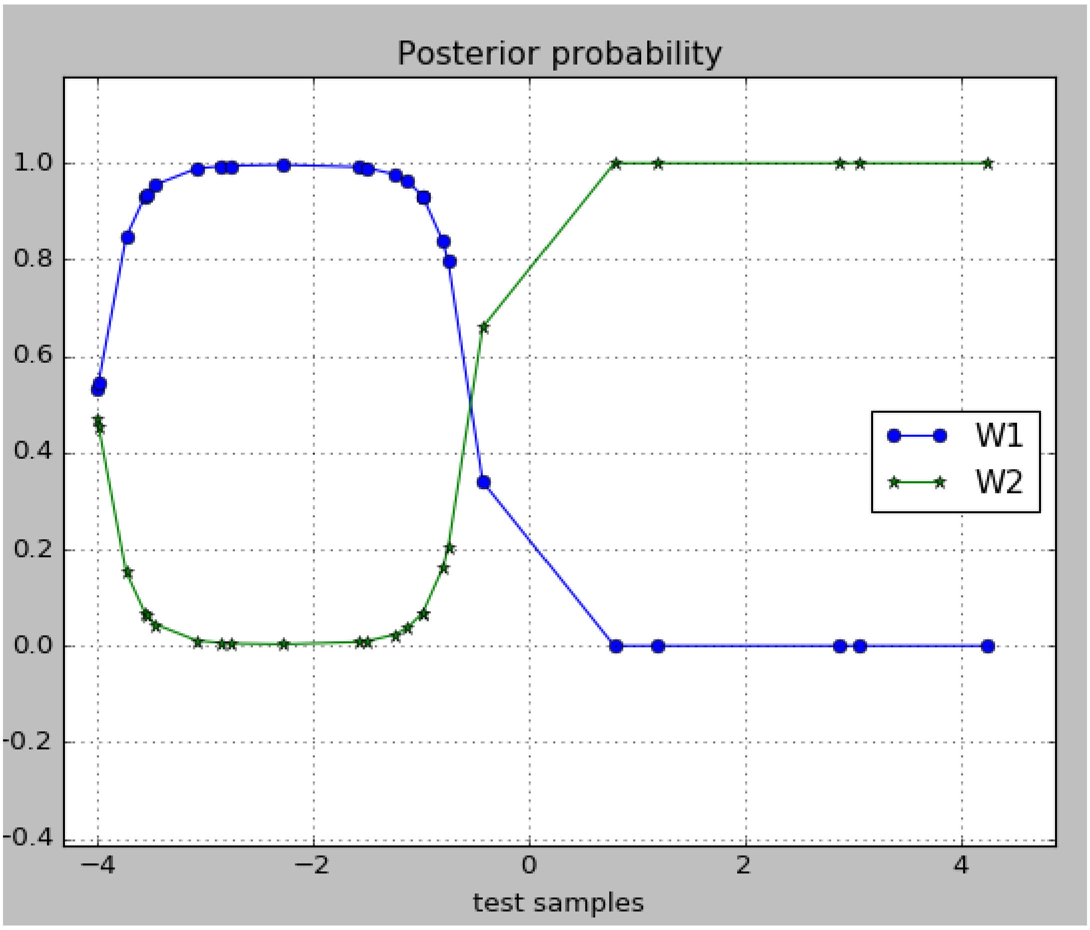
【3】.求解条件风险

```
def getDecisionTable(self,p1,p2,table,x):
    res=np.zeros((1,2))
    for i in range(2):
            res[0,i]=table[i,0]*p1+table[i,1]*p2
    return res
```

实验结果分析

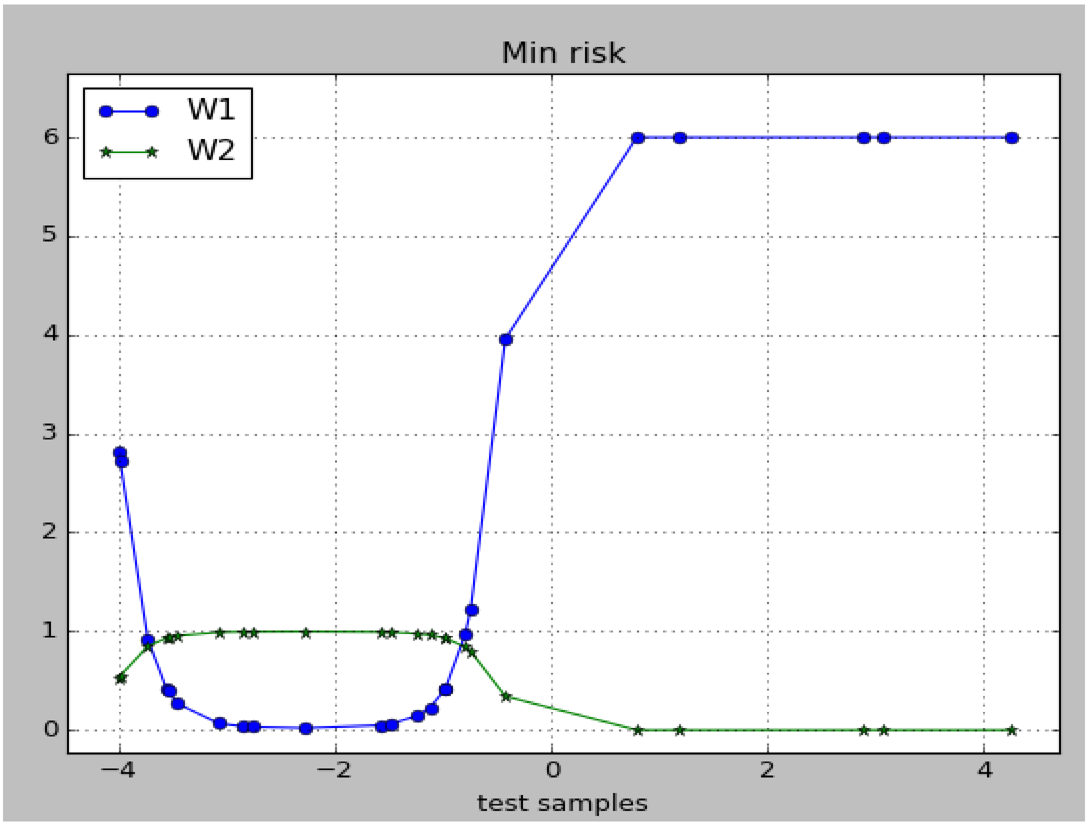
【1】.若采用后验概率（最小错误率）做判别，从-4 到-0.5 之间的样本均属于第一类，从-0.5 到 4.5 的样本均属于第二类。

后验概率曲线图

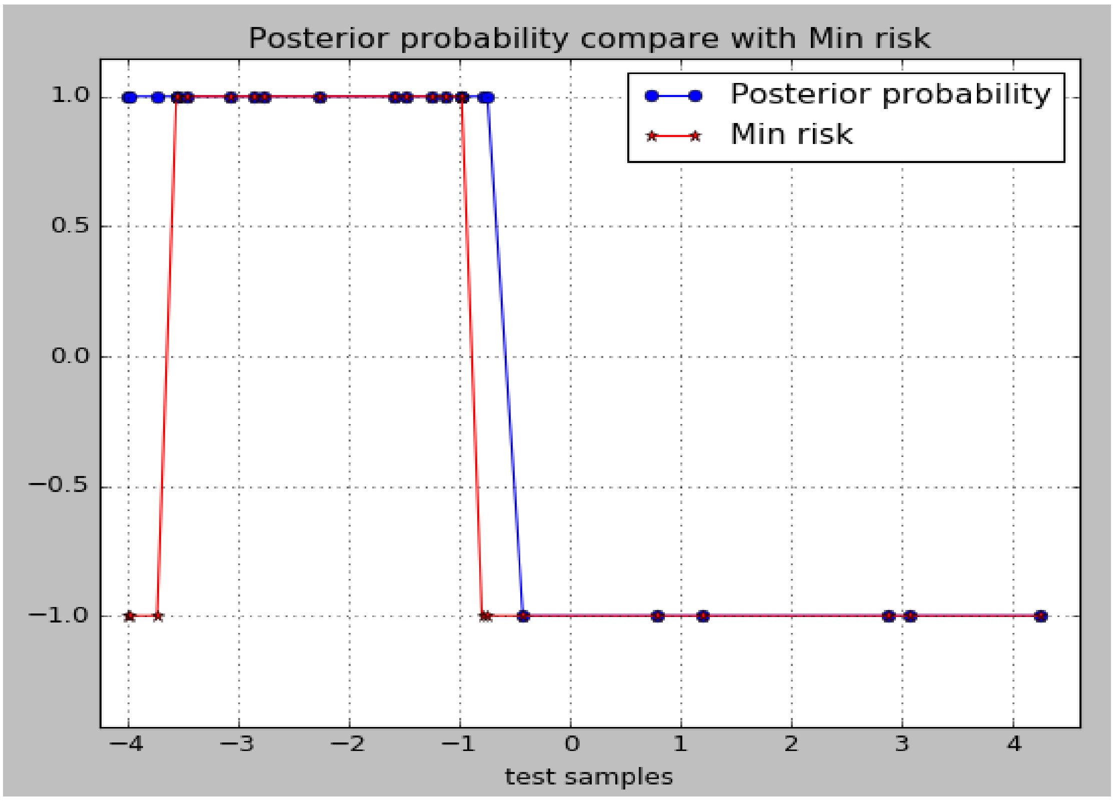


【2】.若采用最小风险做判别，从-4.0 到-3.7 之间有三个样本属于第二类，从-3.7 到-0.8 的样本属于是第一类，-0.8 到 4.5 之间的样本属于第二类

最小风险曲线图



【3】.最后，我将两种判别方式画在一个图中进行比较，可以直观的看出两种判别方式的差异。



实验二、基于 Fisher 准则线性分类器设计

2.1 实验类型：

线性分类器设计（Fisher 准则）

2.2 实验目的：

本实验旨在让同学进一步了解分类器的设计概念，能够根据自己的设计对线性分类器有更深刻地认识，理解 Fisher 准则方法确定最佳线性分界面方法的原理。

2.3 实验原理：

线性判别函数的一般形式可表示成

$g\ (X) = W^T X + w_0$　　其中

$$X = \begin{pmatrix} x_1 \\ \cdots \\ x_d \end{pmatrix} \qquad W = \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{pmatrix}$$

根据 Fisher 选择投影方向 W 的原则，即使原样本向量在该方向上的投影能兼顾类间分布尽可能分开，类内样本投影尽可能密集的要求，用以评价投影方向 W 的函数为：

$$J_F(W) = \frac{(\tilde{m}_1 - \tilde{m}_2)^2}{\tilde{S}_1^2 + \tilde{S}_2^2}$$

$$W^* = S_W^{-1}(m_1 - m_2)$$

上面的公式是使用 Fisher 准则求最佳法线向量的解，该式比较重要。另外，该式这种形式的运算，我们称为线性变换，其中 $m_1 - m_2$ 式一个向量，S_W^{-1} 是 S_W 的逆矩阵，如 $m_1 - m_2$ 是 d 维，S_W 和 S_W^{-1} 都是 d×d 维，得到的 W^* 也是一个 d 维的向量。

向量 W^* 就是使 Fisher 准则函数 $J_F(W)$ 达极大值的解，也就是按 Fisher 准则将 d 维 X 空间投影到一维 Y 空间的最佳投影方向，该向量 W^* 的各分量值是对原 d 维特征向量求加权和的权值。

以上讨论了线性判别函数加权向量 W 的确定方法，并讨论了使 Fisher 准则函数极大的 d 维向量 W^* 的计算方法，但是判别函数中的另一项 W_0 尚未确定，一般可采用以下几种方法确定 W_0 如

$$W_0 = -\frac{\tilde{m}_1 + \tilde{m}_2}{2}$$

或者 $$W_0 = -\frac{N_1\tilde{m}_1 + N_2\tilde{m}_2}{N_1 + N_2} = \tilde{m}$$

或当 $p(\omega)_1$ 与 $p(\omega)_2$ 已知时可用

$$W_0 = \left[\frac{\tilde{m}_1 + \tilde{m}_2}{2} - \frac{\ln\left[p(\omega_1)/p(\omega_2)\right]}{N_1 + N_2 - 2}\right]$$

……

当 W0 确定之后，则可按以下规则分类，

$$W^T X > -w_0 \rightarrow X \in \omega_1$$
$$W^T X > -w_0 \rightarrow X \in \omega_2$$

使用 Fisher 准则方法确定最佳线性分界面的方法是一个著名的方法，尽管提出该方法的时间比较早，仍见有人使用。

2.4 实验内容：

利用 Fisher 准则对自行建立的样本或应用下面数据求出投影变换向量。

假设已经获得两类二维的模式样本： $\omega_1:\left\{\begin{pmatrix}0\\0\end{pmatrix},\begin{pmatrix}2\\0\end{pmatrix},\begin{pmatrix}2\\2\end{pmatrix},\begin{pmatrix}0\\2\end{pmatrix}\right\}$，$\omega_2:\left\{\begin{pmatrix}4\\4\end{pmatrix},\begin{pmatrix}6\\4\end{pmatrix},\begin{pmatrix}6\\6\end{pmatrix},\begin{pmatrix}4\\6\end{pmatrix}\right\}$，两类均服从正态分布，且先验概率相等。试用 Fisher 准则求出投影变换向量（权向量）。

2.5 实验要求：

请把数据作为样本，根据 Fisher 选择投影方向 W 的原则，使原样本向量在该方向上的投影能兼顾类间分布尽可能分开，类内样本投影尽可能密集的要求，完成 Fisher 线性分类器的设计，程序的语句要求有注释。

解决问题

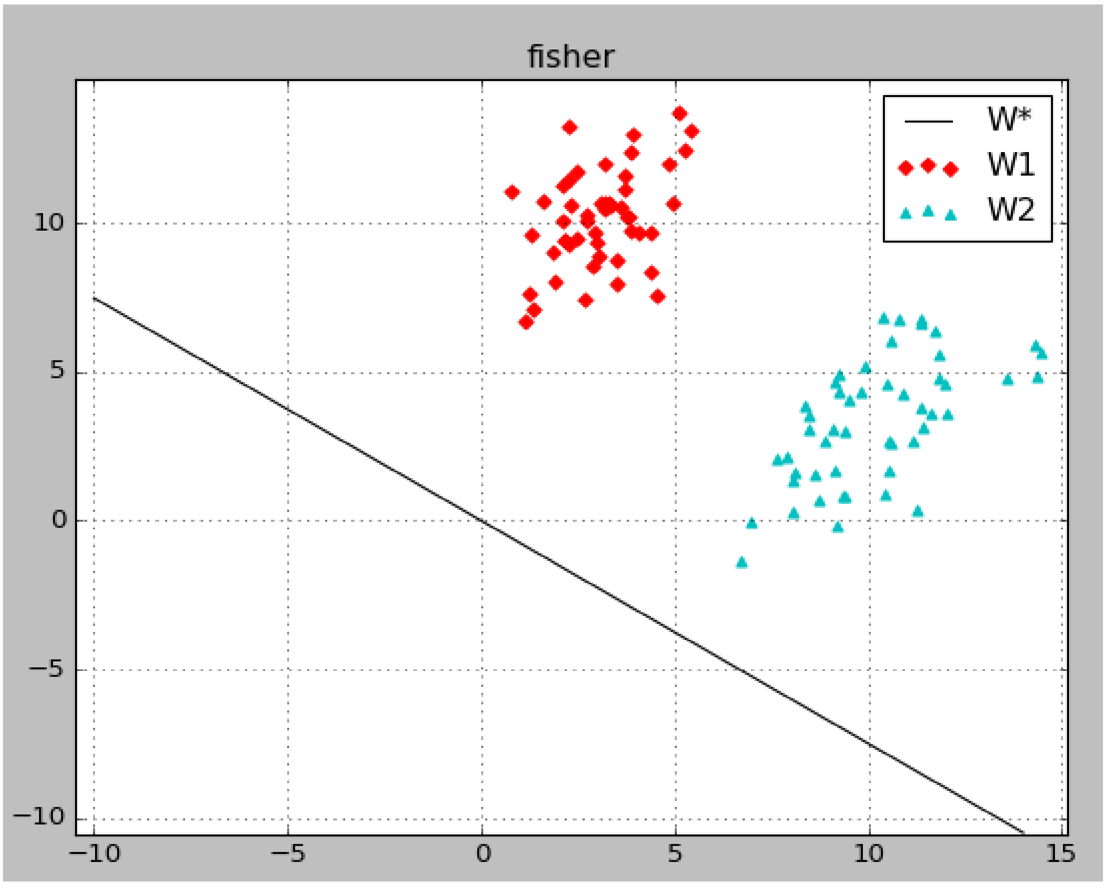
设计思路：根据 fisher 准则可以知道最佳的投影方向为$W^* = S_W^{-1}(m_1 - m_2)$,所以只需要求解总的类内三都矩阵和每类的均值即可

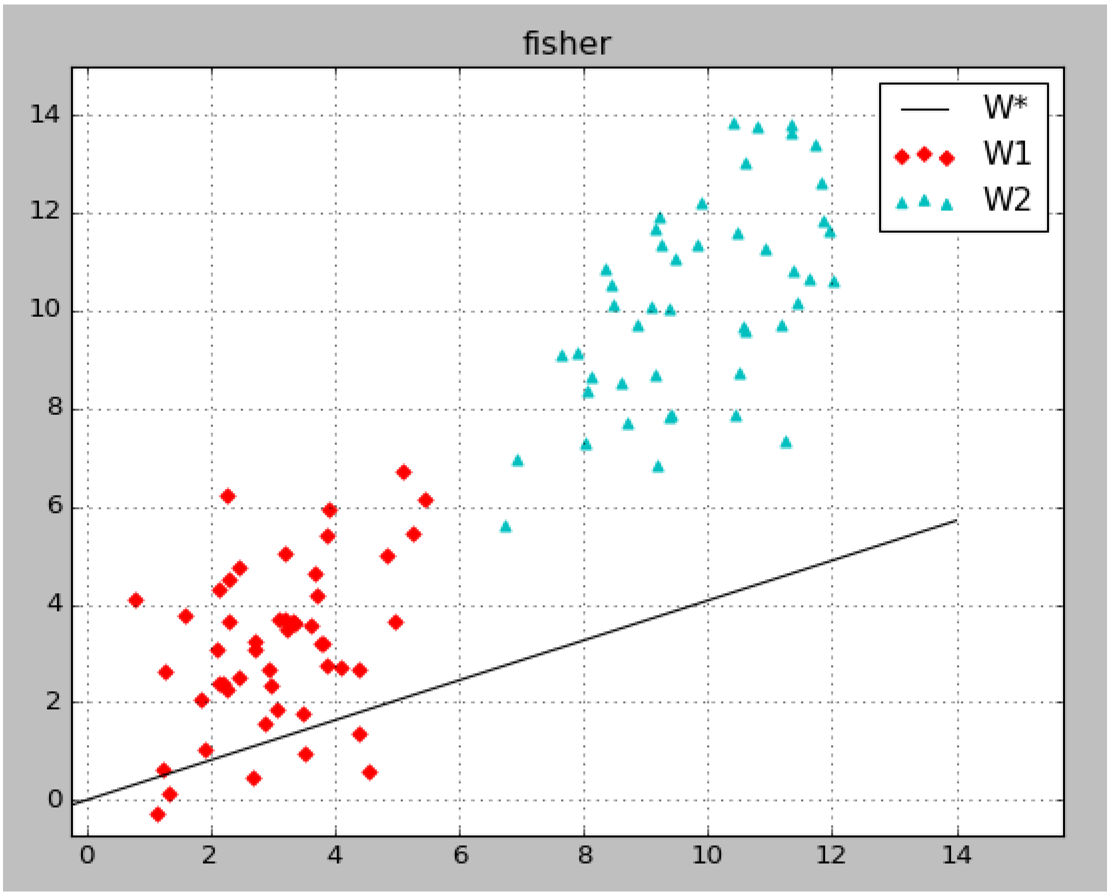
主要函数：

【1】.求最佳投影方向

```
def getW(x1,x2):
    m1 = x1.mean(0);m2 = x2.mean(0);s1 = np.zeros((2, 2));s2 = np.zeros((2, 2))
    m1 = np.mat(m1);m2 = np.mat(m2)
    #求第一类的类内散度矩阵
    for x in x1:
        x = np.mat(x)
        s1 += np.dot((x - m1).T, (x - m1))
    #求第二类的类内散度矩阵
    for x in x2:
        x = np.mat(x)
        s2 += np.dot((x - m2).T, (x - m2))
    sw = (s1 + s2)
    sw = np.mat(sw)
    w = np.mat(m1 - m2).dot(sw.I)#根据公式求最佳的投影方向
    return w
```

实验结果：





实验三、基于感知函数准则线性分类器设计

3.1 实验类型：

线性分类器设计（感知函数准则）

3.2 实验目的：

本实验旨在让同学理解感知准则函数的原理，通过软件编程模拟线性分类器，理解感知函数准则的确定过程，掌握梯度下降算法求增广权向量，进一步深刻认识线性分类器。

3.3 实验原理：

感知准则函数是五十年代由 Rosenblatt 提出的一种自学习判别函数生成方法，由于 Rosenblatt 企图将其用于脑模型感知器，因此被称为感知准则函数。其特点是随意确定的判别函数初始值，在对样本分类训练过程中逐步修正直至最终确定。

感知准则函数利用梯度下降算法求增广权向量的做法，可简单叙述为：任意给定一向量初始值 $\overline{a}(1)$，第 k+1 次迭代时的权向量 $\overline{a}(k+1)$ 等于第 k 次的权向量 $\overline{a}(k)$ 加上被错分类

的所有样本之和与 ρ_k 的乘积。可以证明，对于线性可分的样本集，经过有限次修正，一定可以找到一个解向量 $\overline{a}$ ，即算法能在有限步内收敛。其收敛速度的快慢取决于初始权向量 $\overline{a}(1)$ 和系数 ρ_k 。

3.4 实验内容

随机生成生成满足正态分布的数据，使用感知器算法求解权向量

3.5 实验任务：

完成感知准则函数确定判决权向量的程序设计。

解决问题

设计思路：感知器算法能完成线性可分的分类，属于有监督的学习算法，只要样本是线性可分的，感知器感知器算法通过迭代一定可以找到分界面的权向量，我是用下面的公式更新权值

$$w_j := w_j + \Delta w_j$$

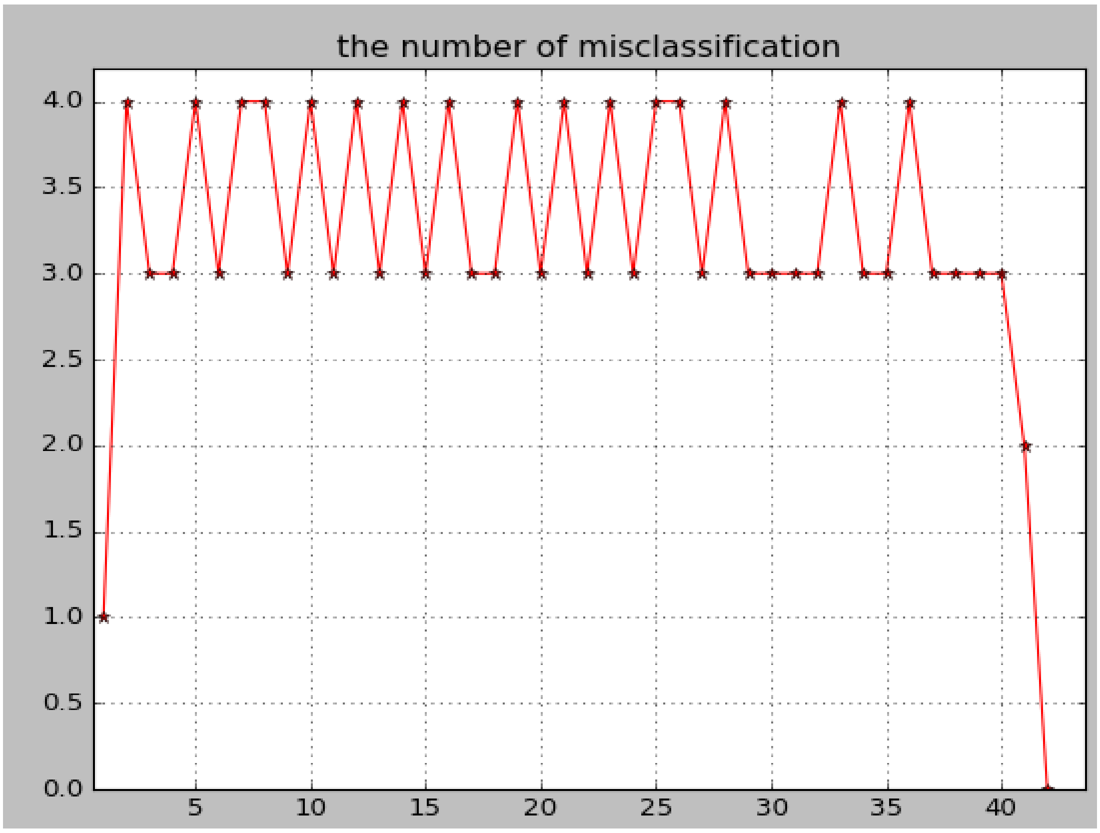
$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$

主要函数：

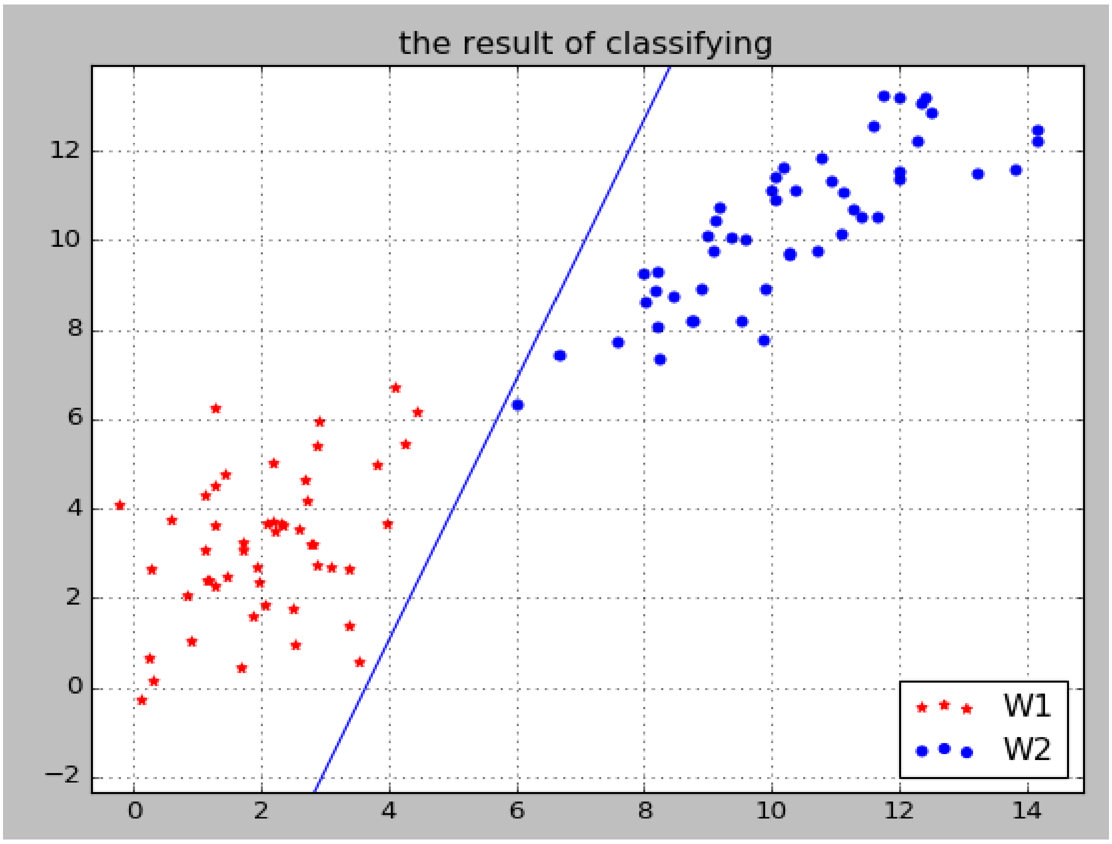
```
def fit(self,X,y):
#跟训练样本集及其标签训练权值
    self.w_=np.zeros(1+X.shape[1])
    self.errors_=[]#记录每次迭代完成后分类错误的个数
    for _ in range(self.n_iter):
        es=0
        for xi,target in zip(X,y):
            upd=self.eta*(target-self.predict(xi))
            self.w_[1:]+=upd*xi#更新权值
            self.w_[0]+=upd
            es += int(update != 0)
        self.errors_.append(es)
        if errors == 0:#如果已经没有错分，结束迭代
            break
    return self

def net_input(self,X):
    return X[0]*self.w_[1]+X[1]*self.w_[2]+self.w_[0]#计算样本与权值的乘积
def predict(self,X):
    return np.where(self.net_input(X)>=0.0,1,-1)#判断所属类别
```

实验结果

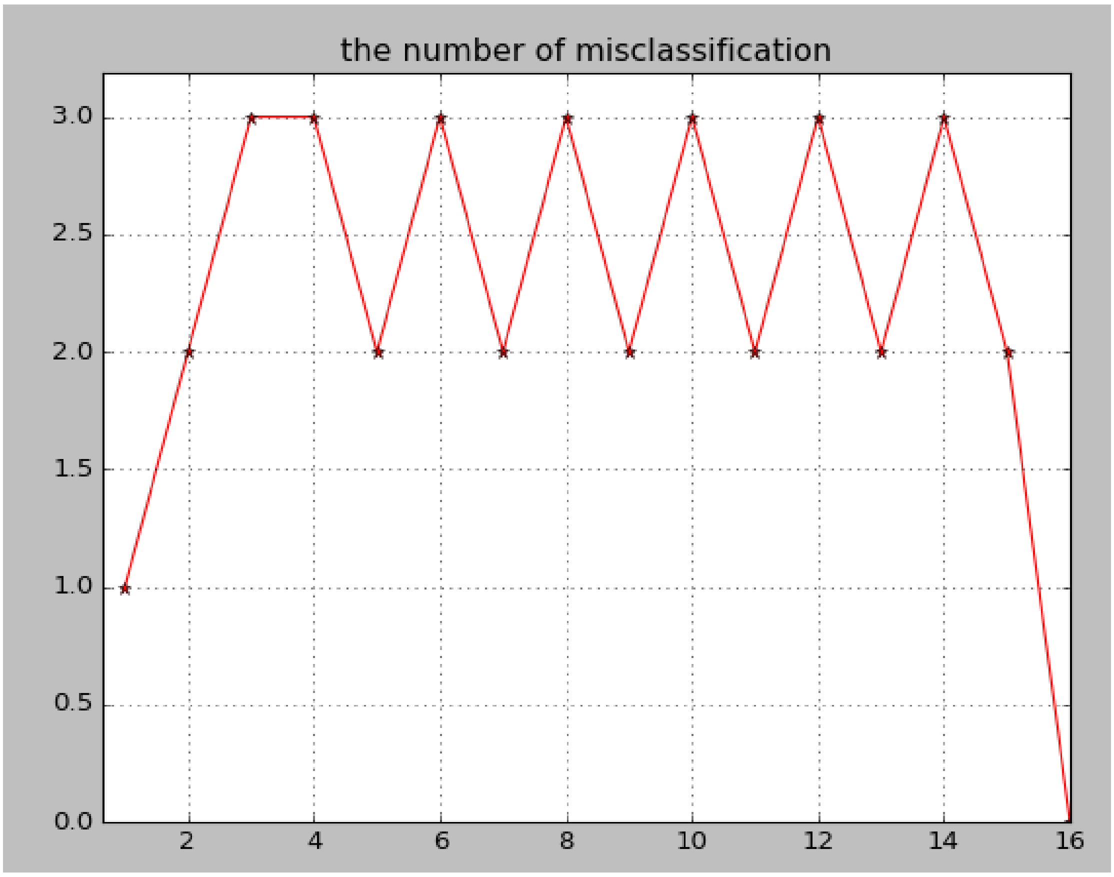
每次迭代过程中分类错误的数量图



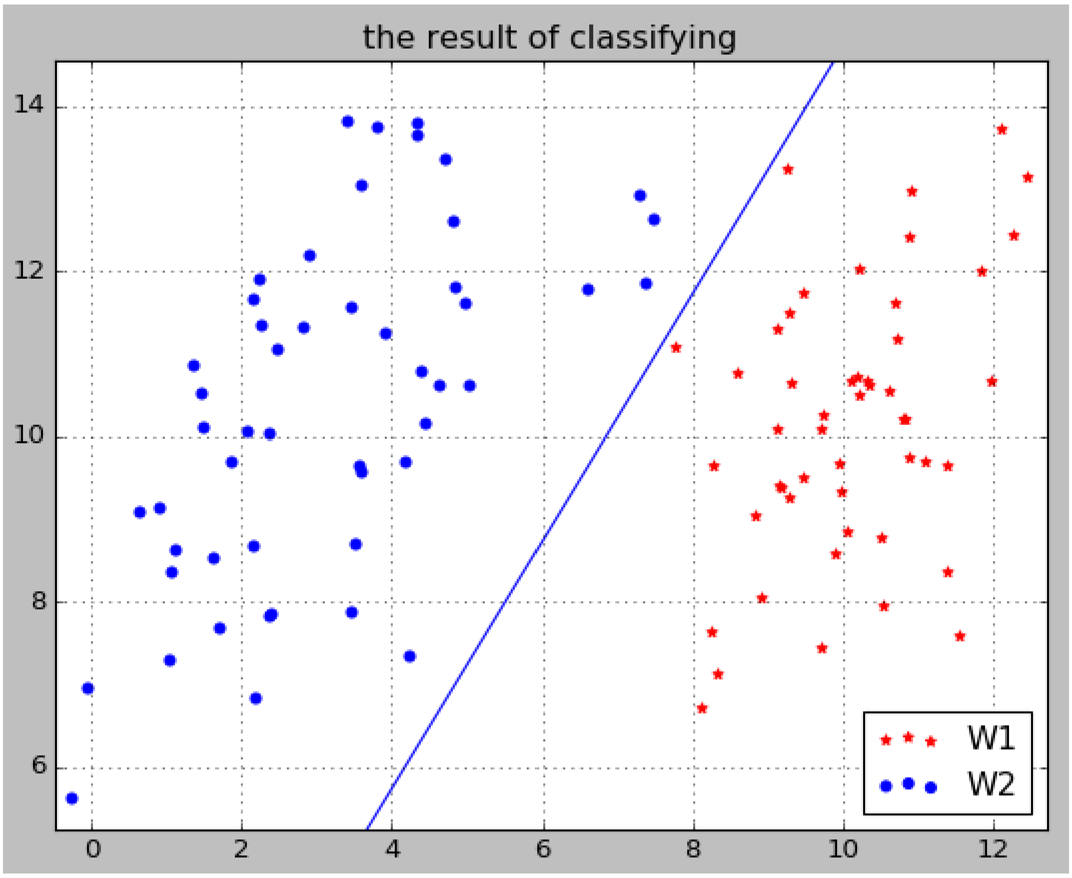
分类结果



每次迭代过程中分类错误的数量图



分类结果



结果分析

根据实验结果可以看出，对于线性可分的问题，感知器算法通过有限步的迭代，总是可以找分类界面，但是他找到的分类界面并不是最优的。而且他的收敛熟速度与学习率和采用学习规则有很大关系。

实验总结

通过模式识别上机实验，学会了简单分类器的设计，同时也深刻理解理论课程中讲解的分类器，在调试程序的同时也深深感受到不同参数,规则对分类效果的影响。

源代码

【1】.bayes 分类器数据

```
# -*- coding: utf-8 -*-
__author__ = 'zlm'
import math
import numpy as np
import matplotlib.pyplot as plt
class Bayes(object):
    def __init__(self,p,mu,segma):#通先验概率和均值方差进行初始化
        self.p=p
        self.mu=mu
        self.segma=segma
    # p(x|w)计算类条件概率
    def getClassConditionP(self,x):
    returnmath.exp(-math.pow(x-self.mu,2)/(2.0*self.segma))/math.sqrt(2.0*math.pi*self.segma)
    #p(w|x)计算后验概率
    def getAfterP(self,b,x):
        return
self.getClassConditionP(x)*self.p/(self.getClassConditionP(x)*self.p+b.getClassConditionP(x)*b.
p)
    def getDecisionTable(self,p1,p2,table,x):#计算条件风险
        res=np.zeros((1,2))
        for i in range(2):
                res[0,i]=table[i,0]*p1+table[i,1]*p2
        return res
if __name__=='__main__':
    x=np.loadtxt('bayes.txt')
    x=x.reshape((x.shape[0]*x.shape[1]))
    table=np.array([[0,6],[1,0]])
    b1=Bayes(0.9,-2,0.25)
    b2=Bayes(0.1,2,4)
    res=[]
    p1=np.zeros(24)
```

```
    p2=np.zeros(24)
    res=np.zeros((24,2))
    for i in range(24):
        p1[i]=b1.getAfterP(b2,x[i])
        p2[i]=b2.getAfterP(b1,x[i])
        res[i,:]=b1.getDecisionTable(p1[i],p2[i],table,x[i])
    id=np.argsort(x)
    #以下为作图程序
    plt.figure(1)
    plt.plot(x[id],p1[id],marker='o',label='W1')
    plt.plot(x[id],p2[id],marker='*',label='W2')
    plt.ylim(-1,2)
    plt.grid(True)
    plt.legend()
    plt.xlim(-5,5)
    plt.xlabel('test samples')
    plt.title('Posterior probability')
    plt.figure(2)
    plt.plot(x[id],res[id,0],marker='o',label='W1')
    plt.plot(x[id],res[id,1],marker='*',label='W2')
    plt.legend()
    plt.ylim(-1,7)
    plt.xlabel('test samples')
    plt.xlim(-5,5)
    plt.grid(True)
    plt.title('Min risk')
    temp1=np.where(p1[id]>p2[id],1,-1)
    temp2=np.where(res[id,0] < res[id,1],1,-1)
    plt.figure(3)
    plt.plot(x[id],temp1,marker='o',color='b',label='Posterior probability')
    plt.plot(x[id],temp2,marker='*',color='r',label='Min risk')
    plt.legend()
    plt.grid(True)
    plt.xlabel('test samples')
    plt.ylim(-2,2)
    plt.xlim(-4.5,6)
    plt.title('Posterior probability compare with Min risk')
    plt.show()
```

【2】.基于 fisher 准则的线性分类器设计

```
# -*- coding: utf-8 -*-
__author__ = 'zlm'
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import cholesky
```

```python
def getData():
    x1=np.array([[0,0],[2,0],[2,2],[0,2]])
    x2 = np.array([[4, 4], [6, 4], [6, 6], [4, 6]])
    return x1,x2
def getRandomData():#随机生成满足正态分布的二维数据
    np.random.seed(0)
    sampleNo = 50
    mu = np.array([[2, 3]])
    Sigma = np.array([[1, 0.5], [0.5, 3]])
    R = cholesky(Sigma)
    x1 = np.dot(np.random.randn(sampleNo, 2), R) + mu
    mu = np.array([[10, 10]])
    Sigma = np.array([[1, 1.5], [1.5, 5]])
    R = cholesky(Sigma)
    x2 = np.dot(np.random.randn(sampleNo, 2), R) + mu
    return x1,x2
def getW(x1,x2):#根据 fisher 准则求解最佳投影方向
    m1 = x1.mean(0)
    m2 = x2.mean(0)
    s1 = np.zeros((2, 2))
    s2 = np.zeros((2, 2))
    m1 = np.mat(m1)
    m2 = np.mat(m2)
    #求第一类的类内散度矩阵
    for x in x1:
        x = np.mat(x)
        s1 += np.dot((x - m1).T, (x - m1))
    ##求第二类的类内散度矩阵
    for x in x2:
        x = np.mat(x)
        s2 += np.dot((x - m2).T, (x - m2))
    sw = (s1 + s2)
    sw = np.mat(sw)
    w = np.mat(m1 - m2).dot(sw.I)#根据公式求最佳的投影方向
    return w
if __name__=='__main__':
    x1,x2=getRandomData()
    w=getW(x1,x2)
    k=w[0,1]/w[0,0]
    #以下为作图程序
    plt.figure(1)
    plt.scatter(x1[:,0],x1[:,1],marker='D',color='r',label='W1')
    plt.scatter(x2[:,0],x2[:,1],marker='^',color='c',label='W2')
```

```
    mx=max(x1[:,0].max(),x2[:,0].max())
    mn=min(x1[:,0].min(),x2[:,0].min())
    x=np.arange(-10,mx,1)
    plt.plot(x,k*x,color='black',label='W*')
    plt.title('fisher')
    plt.legend(loc='upper right')
    plt.grid(True)
    plt.show()
```

【3】.基于感知器准则的线性分类器设计

```
# -*- coding: utf-8 -*-
__author__ = 'zlm'
import numpy as np
import matplotlib.pyplot as plt
from fisher import getRandomData
class Perceptron(object):
    def __init__(self,eta=0.01,n_iter=200):#使用学习率和最大迭代次数初始化
        self.eta=eta
        self.n_iter=n_iter
    def fit(self,X,y):
    #跟训练样本集及其标签训练权值
        self.w_=np.zeros(1+X.shape[1])
        self.errors_=[]#记录每次迭代完成后分类错误的个数
        for _ in range(self.n_iter):
            es=0
            for xi,target in zip(X,y):
                upd=self.eta*(target-self.predict(xi))
                self.w_[1:]+=upd*xi#更新权值
                self.w_[0]+=upd
                es += int(update != 0)
            self.errors_.append(es)
            if errors == 0:#如果已经没有错分，结束迭代
                break
        return self

    def net_input(self,X):
        return X[0]*self.w_[1]+X[1]*self.w_[2]+self.w_[0]#计算样本与权值的乘积
    def predict(self,X):
        return np.where(self.net_input(X)>=0.0,1,-1)#判断所属类别
if __name__=='__main__':
    x1,x2=getRandomData()#调用第二次实验时随机生数据的函数
    b=np.ones((x1.shape[0],1))
    x=np.vstack((x1,x2))
    y=np.vstack((b,-b))
```

```
p=Perceptron(0.001,200)
p.fit(x,y)
#以下为作图程序
plt.figure(1)
plt.plot(range(1,len(p.errors_)+1),p.errors_,marker='*',color='r')
plt.title('the number of misclassification')
plt.ylim(0,max(p.errors_)+1)
plt.grid(True)
plt.figure(2)
plt.scatter(x[0:x1.shape[0],0],x[0:x1.shape[0],1],marker='*',color='r',label='W1')
plt.scatter(x[x1.shape[0]:x.shape[0],0],x[x1.shape[0]:x.shape[0],1],marker='o',color='b',label='W2')
plt.title('the result of classifying')
plt.legend(loc='lower right')
X=np.arange(x.min(),x.max(),1)
plt.plot(X,-(X*p.w_[1]+p.w_[0])/p.w_[2])
plt.grid(True)
plt.show()
```