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：王鹏

学　　号：14030130101

完成时间：2017.5.20

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：

$$P(\omega_i|X)=\frac{P(X|\omega_i)P(\omega_i)}{\sum_{j=1}^{c}P(X|\omega_i)P(\omega_i)}$$ 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

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

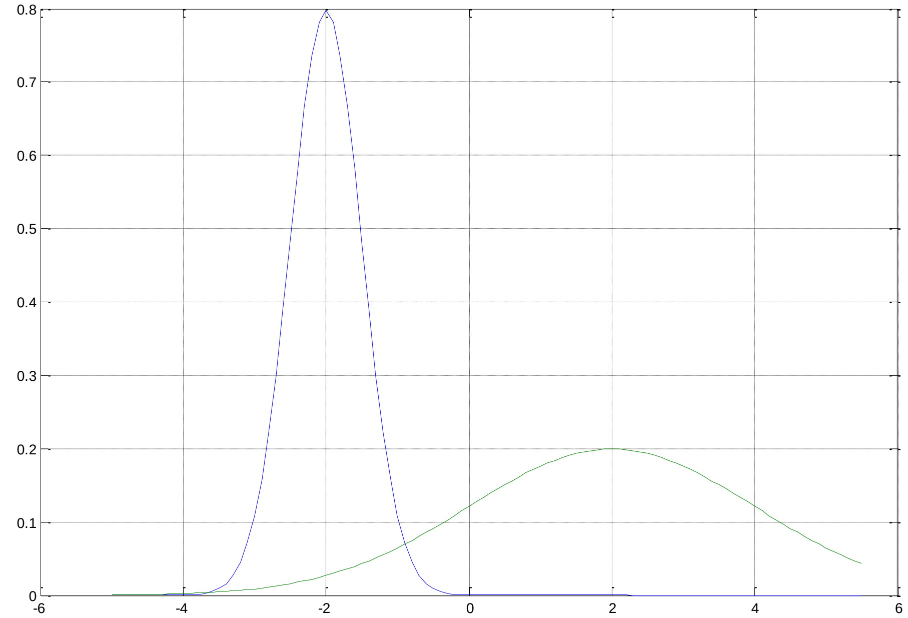
正常状态：P（ω_1）=0.9；

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

$p(x|\omega_1)$ 和 $p(x|\omega_2)$ 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

```
#类条件
def p1(x):
    return 1/(sqrt(2*pi)*sigma1)*exp(-1.0/2*(x-mu1)**2/sigma1**2);

def p2(x):
    return 1/(sqrt(2*pi)*sigma2)*exp(-1.0/2*(x-mu2)**2/sigma2**2);
```

【2】.求解后验概率

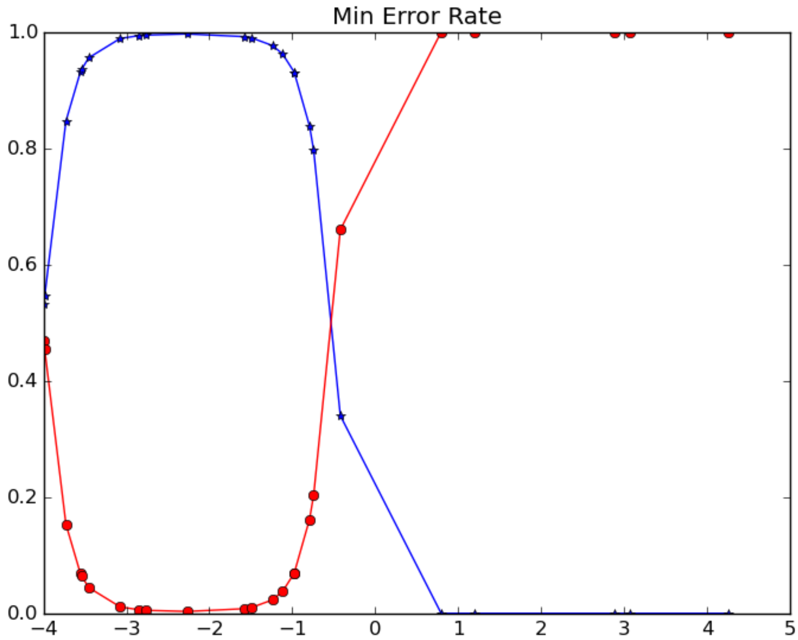
```
#后验概率
def pw1x(x):
#       print "pw1x:",pw1*p1(x)/(pw1*p1(x)+pw2*p2(x))
    return pw1*p1(x)/(pw1*p1(x)+pw2*p2(x))

def pw2x(x):
#       print "pw2x:",pw2*p2(x)/(pw1*p1(x)+pw2*p2(x))
    return pw2*p2(x)/(pw1*p1(x)+pw2*p2(x))
```

【3】.求解条件风险

```
def decision(data,dtable):
    print "decision"
    m,n=shape(data);
    kase=zeros([m,n])
    for i in range(m):
        for j in range(n):
            x=data[i,j];
#                print x,p1(x),p2(x),pw1x(x),pw2x(x),rw1(x,dtable),rw2(x,dtable)
            if rw1(data[i,j],dtable)>rw2(data[i,j],dtable):
                kase[i,j]=1;
    return kase
```

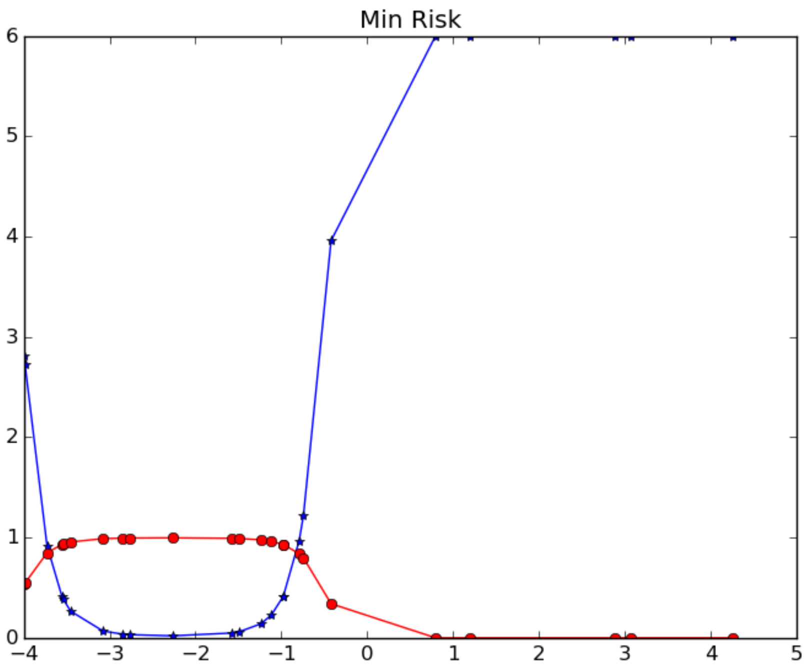
实验结果分析

【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

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 W_0 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

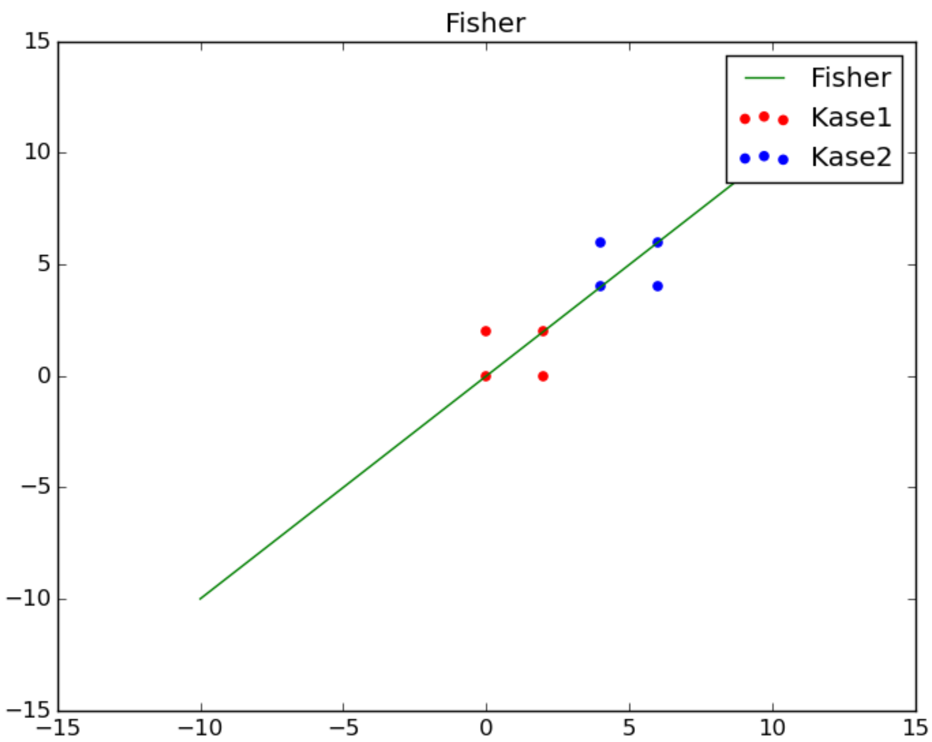
设计思路：根据 fisher 准则可以知道最佳的投影方向为$W^* = S_W^{-1}(m_1 - m_2)$,所以只需要求解总的类内三都矩阵和每类的均值即可

主要函数：

【1】.求最佳投影方向

```
def fisher(data1,data2):
    m1,n1=shape(data1);
    m2,n2=shape(data2);
    if(n1!=n2):
        print "Data1 must is the same array as data2!"
        return

    M1=average(data1,axis=0);
    M2=average(data2,axis=0);
    Sw1=cov(transpose(data1));
    Sw2=cov(transpose(data2));
    print shape(Sw1)
    lammda=1e-16
    Sw=Sw1+Sw2+lammda*eye(n1);
    return array(linalg.inv(Sw).dot(M1-M2))
```

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

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

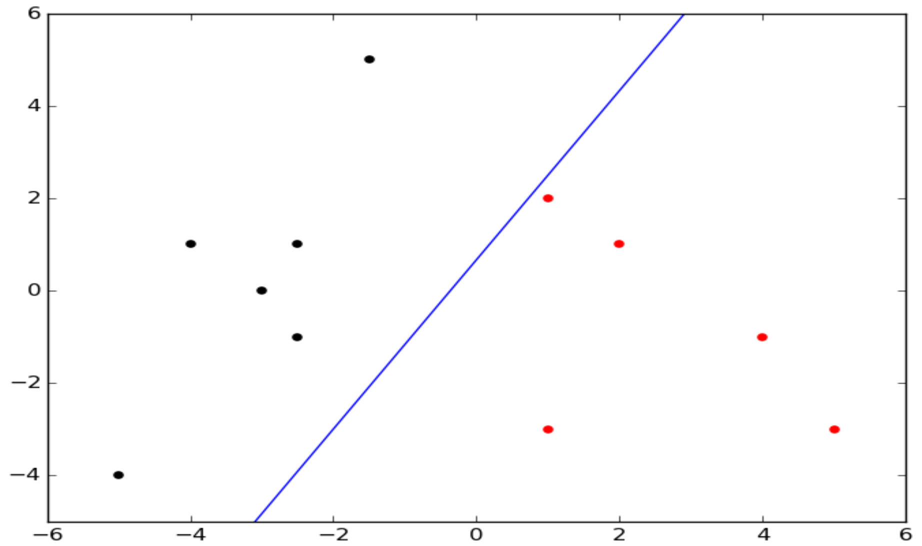
$$\Delta w_j = \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}$$

主要函数：

为了加快数据处理速度，采用了随机梯度下降的方法

```
def training(train_datas):
    weight=[1,1]
    bias=0;
    learning_rate=0.05
    wb=[];
    train_num=10000
    for i in range(train_num):
        m,n=shape(train_datas);
        index=random.randint(0,m)
        train=train_datas[index,:];
        x1,x2,y=train;
        predict=sign(weight[0]*x1+weight[1]*x2+bias)
        if y*predict>=0:
            weight[0]=weight[0]-y*learning_rate*x1;
            weight[1]=weight[1]-y*learning_rate*x2;
            bias=bias+learning_rate*y;
            wb+=[[weight[0],weight[1],bias]];
    return weight,bias,array(wb);
```

实验结果



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:utf8 -*-
from numpy import *
from matplotlib.pyplot import *
import sys
datapath="data.txt"
decisionpath="decisiontable.txt"
datamat=genfromtxt(datapath,delimiter=' ');
decisiontable=genfromtxt(decisionpath,delimiter=' ');
m,n=shape(datamat);

datamat=datamat.reshape(1,m*n);
datamat=sort(datamat);
#print datamat
pw1=0.9;
pw2=0.1;

mu1=-2;sigma1=0.5;
mu2=2;sigma2=2;
#plot(datamat[0,:],datamat[0,:])
#show()
def test():
    x=linspace(-6,6,1000)
    pxw2=1/(sqrt(2*pi)*sigma2)*exp(-1.0/2*(x-mu2)**2/sigma2**2);
    pxw1=1/(sqrt(2*pi)*sigma1)*exp(-1.0/2*(x-mu1)**2/sigma1**2);
    plot(x,pxw1,label="$pxw1$",color="red");
    plot(x,pxw2,label="$pxw2$");
    xlabel("x");
    ylabel("p(x)")
    ylim(0,2.0)
    legend();
    show();
#test()
#类条件
def p1(x):
```

```
    return 1/(sqrt(2*pi)*sigma1)*exp(-1.0/2*(x-mu1)**2/sigma1**2);

def p2(x):
    return 1/(sqrt(2*pi)*sigma2)*exp(-1.0/2*(x-mu2)**2/sigma2**2);

#后验概率
def pw1x(x):
#    print "pw1x:",pw1*p1(x)/(pw1*p1(x)+pw2*p2(x))
    return pw1*p1(x)/(pw1*p1(x)+pw2*p2(x))

def pw2x(x):
#    print "pw2x:",pw2*p2(x)/(pw1*p1(x)+pw2*p2(x))
    return pw2*p2(x)/(pw1*p1(x)+pw2*p2(x))


def rw1(x,dtable):
#    print "rw1:",dtable(0,0)*pw1x(x)+dtable(0,1)*pw2x(x);
    return dtable[0,0]*pw1x(x)+dtable[0,1]*pw2x(x);

def rw2(x,dtable):
#    print "rw2:",dtable(1,0)*pw1x(x)+dtable(1,1)*pw2x(x);
    return dtable[1,0]*pw1x(x)+dtable[1,1]*pw2x(x);

def decision(data,dtable):
    print "decision"
    m,n=shape(data);
    kase=zeros([m,n])
    for i in range(m):
        for j in range(n):
            x=data[i,j];
#            print x,p1(x),p2(x),pw1x(x),pw2x(x),rw1(x,dtable),rw2(x,dtable)
            if rw1(data[i,j],dtable)>rw2(data[i,j],dtable):
                kase[i,j]=1;
    return kase


figure(1);
#后验概率图:
y1=zeros([1,m*n]);
y2=y1.copy()
for i in range(1):
    for j in range(m*n):
        y1[i,j]=pw1x(datamat[i,j]);
        y2[i,j]=pw2x(datamat[i,j]);
```

```
#print sort(y1)
#print sort(y2)
#print datamat[0,:];
#print y1[0,:];
#x=y1[0,:];
#y=y1[0,:];
#print x.shape;
#print y.shape
#scatter(x,y);
plot(datamat[0,:],y1[0,:],'*-');
plot(datamat[0,:],y2[0,:],'o-',color="red");
title('Min Error Rate')
figure(2)

#最小风险：
y1=zeros([1,m*n]);
y2=y1.copy();
for i in range(1):
    for j in range(m*n):
        y1[i,j]=rw1(datamat[i,j],decisiontable);
        y2[i,j]=rw2(datamat[i,j],decisiontable);
plot(datamat[0,:],y1[0,:],'*-');
plot(datamat[0,:],y2[0,:],'o-',color='red');
title('Min Risk')

#ylim(-1,1)
#print datamat
#print y1
#print y2
#分类图
#print datamat.shape
figure(3)
kase=decision(datamat,decisiontable);
print kase.shape
print datamat.shape
scatter(datamat,kase,color='green',s=20)
title("Case")
show();
```

【2】.基于 fisher 准则的线性分类器设计

```
#coding:utf8
from numpy import *
from matplotlib.pyplot import *
import sys
```

```
datapath="fisherdata.txt"
data=genfromtxt(datapath,delimiter=' ');

#列表示维，行表示样本数量
def fisher(data1,data2):
    m1,n1=shape(data1);
    m2,n2=shape(data2);
    if(n1!=n2):
        print "Data1 must is the same array as data2!"
        return

    M1=average(data1,axis=0);
    M2=average(data2,axis=0);
    Sw1=cov(transpose(data1));
    Sw2=cov(transpose(data2));
    print shape(Sw1)
    lammda=1e-16
    Sw=Sw1+Sw2+lammda*eye(n1);
    return array(linalg.inv(Sw).dot(M1-M2))


data1=array([[0,0],[0,2],[2,0],[2,2]]);
data2=array([[4,4],[4,6],[6,4],[6,6]]);
W=fisher(data1,data2);

print data1[:,0]
print data1[:,1]
scatter(data1[:,0],data1[:,1],color='red',label='Kase1');
scatter(data2[:,0],data2[:,1],color='blue',label='Kase2');
if W[0]!=0:
    x=linspace(-10,10,1000)
    y=x*1.0*W[1]/W[0]
    plot(x,y,color='green',label='Fisher')
else:
    x=zeros(100)
    y=linspace(-10,10,1000)
    plot(x,y,color='green')
title("Fisher")
legend();
show();
```

【3】.基于感知器准则的线性分类器设计

```
#coding:utf8
from numpy import *
```

```
from matplotlib.pyplot import *
from matplotlib.animation import *
import sys

datapath="perceptrondata.txt"
data=genfromtxt(datapath,delimiter=' ');
#print min(data[1,:]),max(data[1,:])

#符号函数
def sign(v):
    if v>0:
        return 1;
    else:
        return -1;

def training(train_datas):

    weight=[1,1]
    bias=0;
    learning_rate=0.05
    wb=[];
#     train_num=int(raw_input("train num: "))
#     print train_datas
    train_num=10000
    for i in range(train_num):
        m,n=shape(train_datas);
        index=random.randint(0,m)
#         index=i%shape(train_datas)[0]
        train=train_datas[index,:];
#         train=random.choice(train_datas);
#         print train
        x1,x2,y=train;
#         print index,weight[0],weight[1],bias,": ",weight[0]*x1+weight[1]*x2+bias
#         print " "
        predict=sign(weight[0]*x1+weight[1]*x2+bias)

        if y*predict>=0:
            weight[0]=weight[0]-y*learning_rate*x1;
            weight[1]=weight[1]-y*learning_rate*x2;
            bias=bias+learning_rate*y;
            wb+=[[weight[0],weight[1],bias]];
#         print x1,x2,":",y,y*predict
#         print " "
    return weight,bias,array(wb);
```

```
fig=figure();
window=fig.add_subplot(111)
window.axis([min(data[:,0])-1,max(data[:,0])+1,min(data[:,1])-1,max(data[:,1])+1])
def test(data):
    weight,bias=training(data);
    while True:
        test_data=[];
        data=raw_input("Enter data test (x1,x2):");
        if data=='1':break;
        test_data+=[int(n) for n in data.split(',')]
        predict=sign(weight[0]*test_data[0]+weight[1]*test_data[1]+bias);
#           print predict
#

def picture(weight,bias):
    m,n=shape(data);
    for i in range(m):
        if data[i,2]>0:
            window.scatter(data[i,0],data[i,1],color='red');
        else:
            window.scatter(data[i,0],data[i,1],color='black');
#     x=linspace(min(data[:,0]),max(data[:,1]),1000);
#     window.plot(x,weight[0]/weight[1]*x-bias/weight[1])
#     show()
weight,bias,wb=training(data)
#print shape(wb)
print wb
picture(weight,bias)
x=linspace(min(data[:,0]),max(data[:,0]),1000);
#print x
#x=list(x)
#print 'x',x
m_wb,n_wb=shape(wb);
y=[]
for i in range(m_wb):
    if wb[i,1]==0:
#           y.append()
        continue
    y.append(-x*wb[i,0]/wb[i,1]-wb[i,2]/wb[i,1])

print "start"
#print y
y=array(y)
```

```
if shape(y)[0]==0:
	kase=5
	mid=1/2.0*(max(data[0:5,0])+min(data[kase:shape(data)[0],0]))
#	print min(data[1,:]),max(data[1,:])
	plot([mid,mid],[min(data[:,1]),max(data[:,1])])
	show()
else:
	p=min(data[:,0])
	q=max(data[:,0])
#	line,=window.plot(x,y[0,:])
	def update(data):
		line.set_xdata(linspace(p,q,1000));
		line.set_ydata(data);
		return line
#	ani = FuncAnimation(fig, update, y, interval=200)

	plot(x,y[shape(y)[0]-1,:])
	show()
```