哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型：选修

实验项目名称： 逻辑回归

实验题目： 逻辑回归

班级：1503104
学号：1153710506
姓名：薛仲豪

设计成绩	报告成绩	指导老师

一、 实验目的

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

二、 实验要求及实验环境

实现两种损失函数的参数估计（1，无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等（建议对比梯度下降和牛顿法）。

验证：
1.可以手工生成两个分别类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素贝叶斯假设，会得到什么样的结果。
2. 逻辑回归有广泛的用处，例如广告预测。可以到 UCI 网站上，找一实际数据加以测试。
实验环境：
OS 名称: Microsoft Windows 10 家庭中文版
OS 版本: 10.0.15063 暂缺 Build 15063
处理器: 安装了 1 个处理器。
[01]: Intel64 Family 6 Model 60 Stepping 3 GenuineIntel ~2601 Mhz
物理内存总量: 16,269 MB
虚拟内存: 最大值: 19,198 MB

三、 设计思想（本程序中的用到的主要算法及数据结构）

1．算法原理

Logistic 回归，可以表达为优化问题：

$$W_{MCLE} = \arg\max_{W} \prod_{l} P(Y^l|W, X^l)$$

其中，

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=1|X,W) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

由于连乘在计算、推到中不方便，因此根据习惯对齐取对数，使得乘法变成加法，并将其定义为 l(W)，经计算可得：

$$\begin{aligned} l(W) &\equiv \ln \prod_{l} P(Y^l|X^l,W) \\ &= \sum_{l} Y^l(w_0+\sum_{i}^{n} w_i X_i^l) - \ln(1+exp(w_0+\sum_{i}^{n} w_i X_i^l)) \end{aligned}$$

针对这个函数，就可以使用各种基于梯度的方法进行求解。
梯度上升法：

梯度上升最好理解，也最容易实现，只需求出方程的一阶梯度即可，结果如下：

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

具体的计算过程与上次实验的梯度下降完全一致，这里不再重复。
共轭梯度法：

共轭梯度法所用到的信息与梯度上升法完全一致，因此不需要更多推导。

具体计算过程也与上次实验完全一致，不再重复
牛顿法：

牛顿法的一个难点在于其用到了二阶导，高维中就是海森矩阵，推导如下：

$$\begin{aligned}
H_{ij} &= \frac{\partial^2 l(\theta)}{\partial\theta_i \partial\theta_j} \\
&= \frac{1}{m} \frac{\partial}{\theta_j} \sum_{t=1}^{m} (y^{(t)} - h_\theta(x^{(t)})) x_i^{(t)} \\
&= \frac{1}{m} \sum_{t=1}^{m} \frac{\partial}{\theta_j} (y^{(t)} - h_\theta(x^{(t)})) x_i^{(t)} \\
&= \frac{1}{m} \sum_{t=1}^{m} -x_i^{(t)} \frac{\partial}{\partial\theta_j} h_\theta(x^{(t)}) \\
&= \frac{1}{m} \sum_{t=1}^{m} -x_i^{(t)} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)})) \frac{\partial}{\theta_j} (\theta^T x^{(t)}) \\
&= \frac{1}{m} \sum_{t=1}^{m} h_\theta(x^{(t)})(h_\theta(x^{(t)}) - 1) x_i^{(t)} x_j^{(t)}
\end{aligned}$$

迭代的公式为：

$$\theta \leftarrow \theta - H^{-1}\nabla$$

其中∇表示梯度，就是前两个方法中需要用到的梯度。

另外，对于惩罚项，在逻辑回归中有一个新的理解，可以理解为我们对参数有一个先验知识，满足一个均值为 0 的正态分布，这样其实就是等价于我们认为参数的值越小越好。

2．算法的实现

本次算法的实现是直接修改上次实验的代码进行的。

由于上次实验使用的是梯度下降法，所以本次实验沿用了这一方法，也就是直接将 likelihood function 等价为 lost function。这样导致的结果是算法事实上并没有最大化似然函数，而是最小化了似然函数。然而，直观上来说，最小化似然函数就是尽量将 0 预测为 1，将 1 预测为 0，取反后不影响实际的预测效果，从理论上来说，logistic 函数具有对称性，正反没有什么区别。

其实主函数不需要进行过多的改动，主要是需要修改 l(W)，梯度，以及海森矩阵的计算，具体的计算如下：

l(W)：

```
J = -(sum(y.*log(h) + (ones(m,1)-y).*log(ones(m,1)-h)))/m;
```

这里并为了方便计算（h 可以通过矩阵运算一次性全算出来），与之前的理论推导的计算过程有一些区别，但结果一样。

梯度：

```
grad = -(X'*(h-y))./m + lambda*[0; W(2:end)];
```

海森矩阵：

```
H = zeros(n,n);
for i=(1:n)
    for j=(1:n)
        SUM = 0;
        for t=(1:m)
            SUM = SUM + h(t)*(h(t)-1)*X(t,i)*X(t,j);
        end
        H(i,j) = SUM./m;
    end
    if i~=1
        H(i,i) = H(i,i) + lambda;
    end
end
```

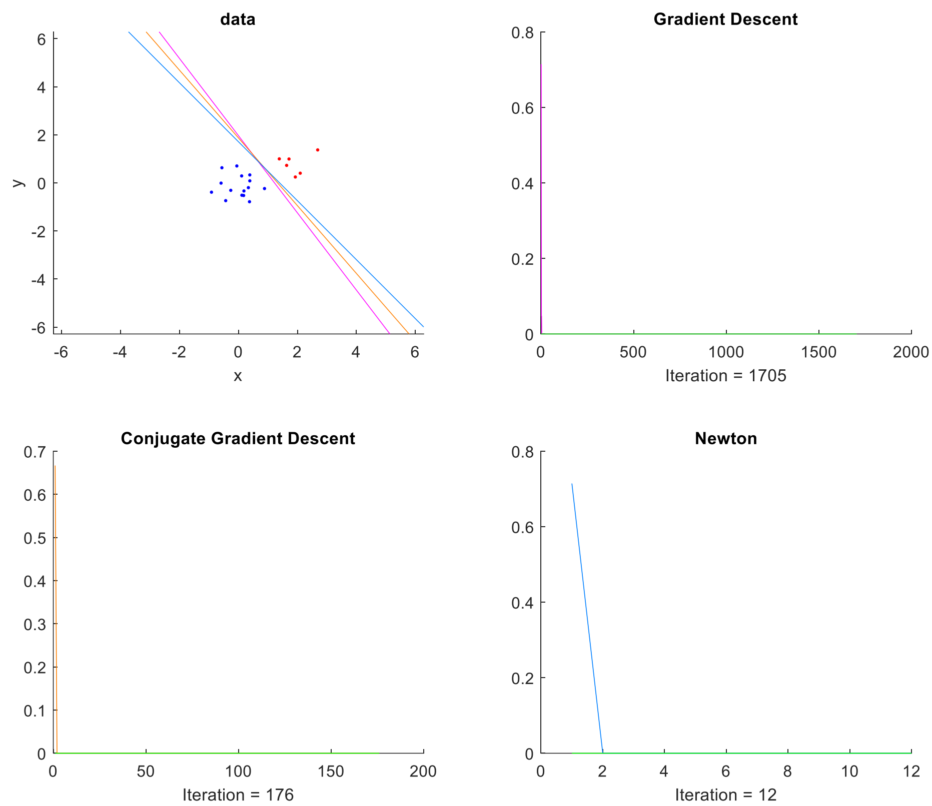
四、 测试结果

生成数据进行测试：

直接使用 MATLAB 的 randn 函数生成正态分布的数据，然后通过增加一个向量，平移一半的数据，使得数据能够被分为两类。

图示说明：

每一个实验结果都用四张图来表示，第一张图用红点和蓝点标记两类数据，用紫线、橘黄线、浅蓝线分别表示梯度下降、共轭梯度、牛顿法迭代的最终结果。第二张图表示梯度下降法的迭代过程，紫色线表示训练集准确率，绿色表示测试集准确率；第三张图表示共轭梯度法的迭代过程，橘黄线表示训练集准确率，绿色表示测试集准确率；第四张图表示牛顿法的迭代过程，浅蓝线表示训练集准确率，绿色表示测试集准确率。



梯度下降：

迭代次数：1705

Lost Function：0.067691

训练集错误率：0.000000

测试集错误率：0.000000

共轭梯度法：

迭代次数：176

Lost Function：0.003067

训练集错误率：0.000000

测试集错误率：0.000000

牛顿法：

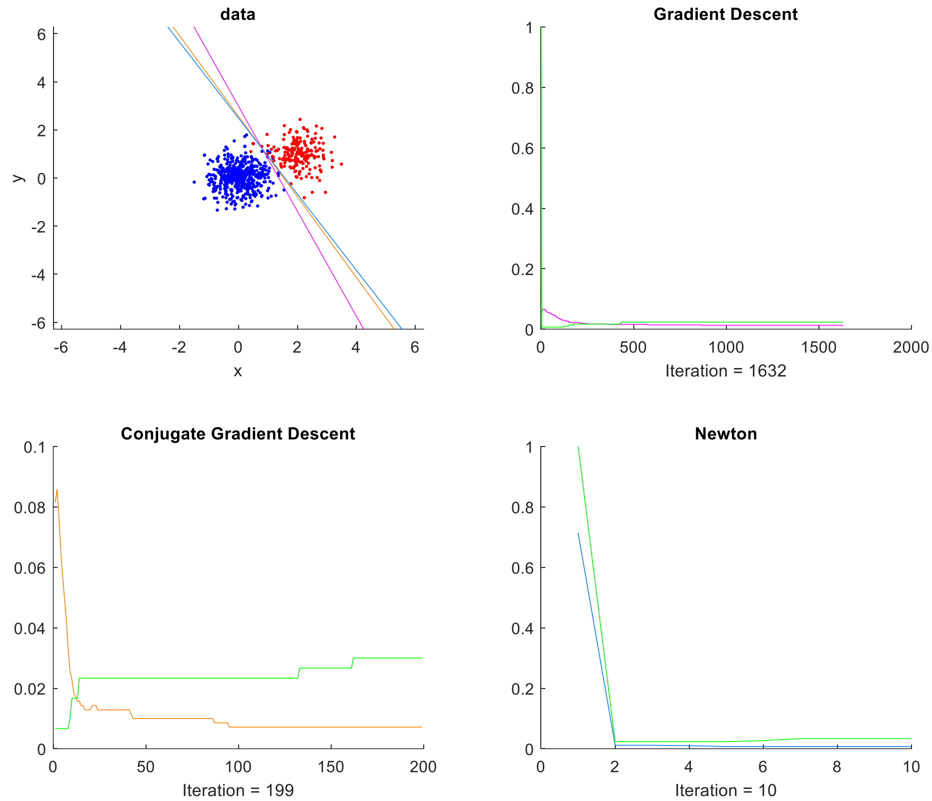
迭代次数：12

Lost Function：0.000014

训练集错误率：0.000000

测试集错误率：0.000000

从上图看出，在数据较少，且明显线性可分时，使用三种方法中的任意一种都可以得到 100% 的准确率，只是 Lost Function 的值有所不同，牛顿法要明显好于梯度下降法。在效率上，实验结果与理论一致，牛顿法明显快于共轭梯度法，而共轭梯度法又明显快于普通的梯度下降法。



梯度下降：

迭代次数：1632

Lost Function：0.095874

训练集错误率：0.012857

测试集错误率：0.023333

共轭梯度法：

迭代次数：199

Lost Function：0.026876

训练集错误率：0.007143

测试集错误率：0.030000

牛顿法：

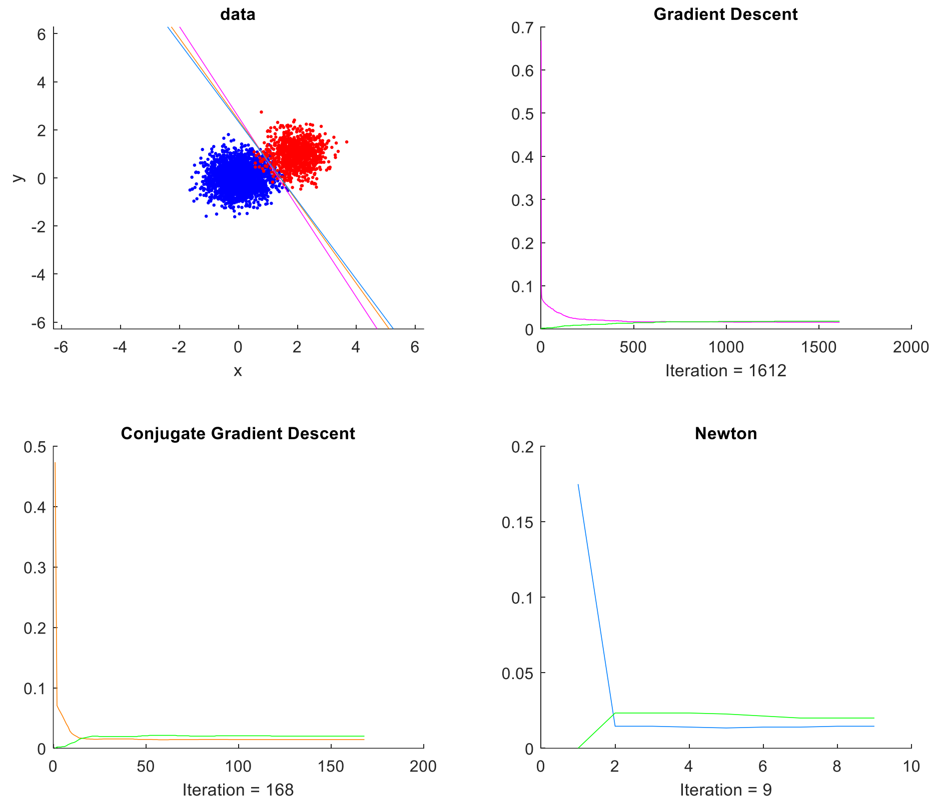
迭代次数：10

Lost Function：0.024273

训练集错误率：0.007143

测试集错误率：0.033333

从上图看出，在数据较多，且数据不容易线性分割（或线性不可分）时，三种算法都能够达到不错的准度度（超过 95%）。但也可以看出，三种算法都出现了过拟合的趋势（训练集的准确度不断提升，测试集的准确度不断下降），其中共轭梯度下降法最为严重。



梯度下降：

迭代次数：1612

Lost Function：0.102859

训练集错误率：0.015429

测试集错误率：0.018000

共轭梯度法：

迭代次数：168

Lost Function：0.039855

训练集错误率：0.014286

测试集错误率：0.020000

牛顿法：

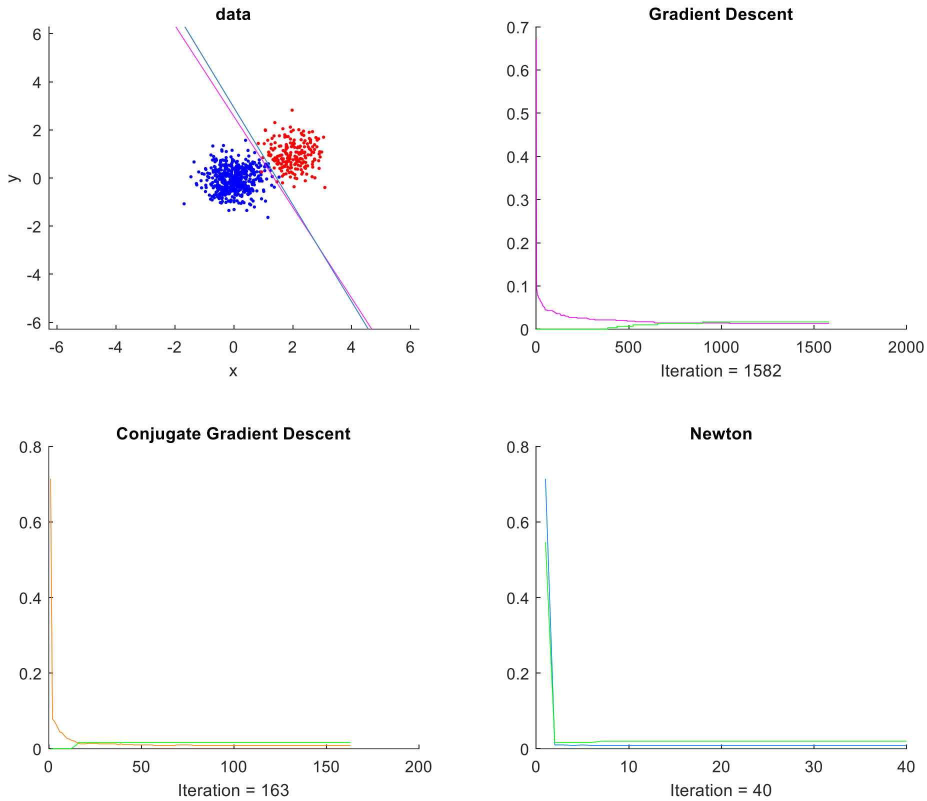
迭代次数：9

Lost Function：0.038134

训练集错误率：0.014571

测试集错误率：0.020000

如果进一步提高数据的数量，会发现不仅结果的准确率有所提高，并且过拟合的现象也有所好转。这与上一次曲线拟合实验的结论“增加数据不仅能够克服过拟合，还能使拟合效果更好”相一致。



梯度下降：

　　迭代次数：1582

　　Lost Function：0.098104

　　训练集错误率：0.012857

　　测试集错误率：0.016667

共轭梯度法：

　　迭代次数：163

　　Lost Function：0.035854

　　训练集错误率：0.008571

　　测试集错误率：0.016667

牛顿法：

　　迭代次数：40

　　Lost Function：0.034443

　　训练集错误率：0.008571

　　测试集错误率：0.020000

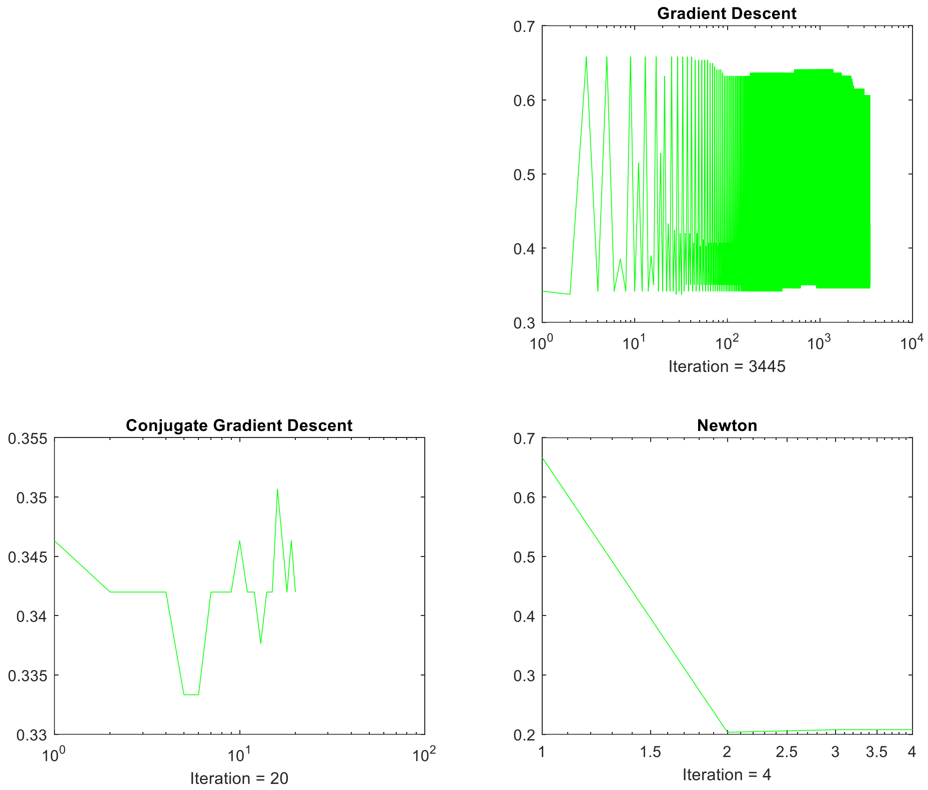
接下来，进行增加惩罚项的实验，上图中，λ的值为 0.001，而数据的个数选择了前面实验中比较容易出现过拟合的 1000 个数据，可以看出，增加了惩罚项后，虽然测试集的错误率仍然要高于训练姐，但对比之前已经有所好转，因此惩罚项在一定程度上克服了过拟合现象。

实际数据：

采用 UCI 上的数据集 pima-indians-diabetes 进行实验

这个数据的标签只有 0 和 1，非常适合逻辑回归

首先，由于此数据数量较多，维度较高，如果不改变参数，就会出现无法收敛，活在收敛过程中剧烈抖动的问题，如图：



经过多次调整参数，最终所使用的参数为：

梯度下降：

学习速率：0.000003

ε = 0.00000000001

正则项：0

共轭梯度下降：

学习速率：0.0001

ε = 0.00000000001

正则项：0

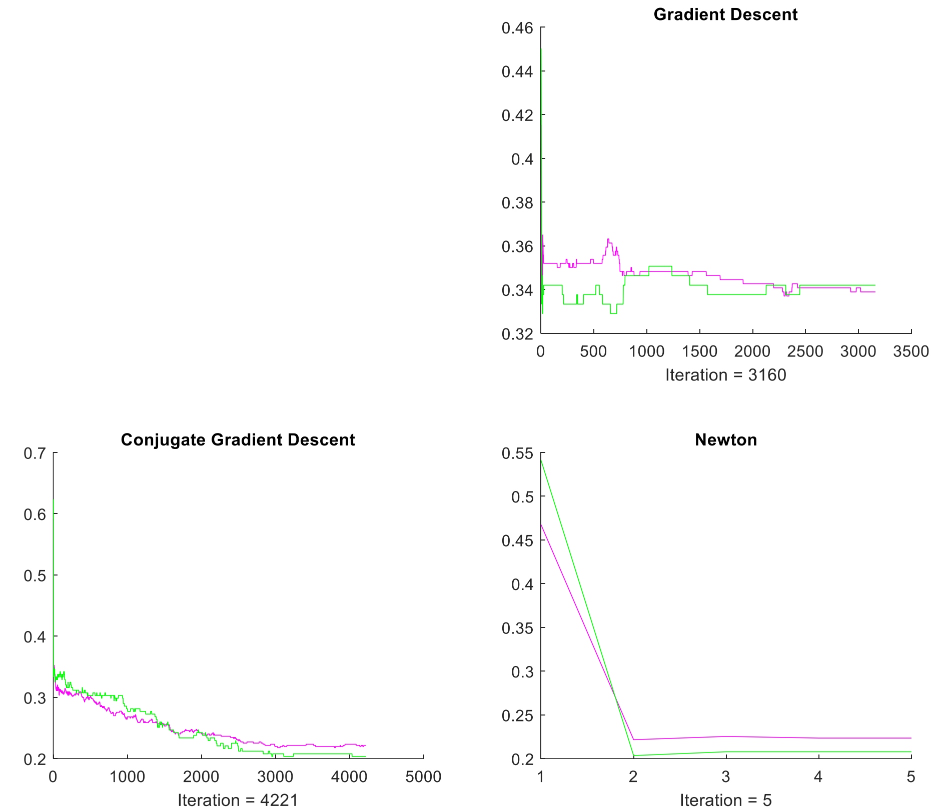
牛顿法：

正则项：0

需要特别说明的是，每一个图象上都有两条曲线，一条紫色的线，表示训练集上的正确率，一条绿色曲线，表示测试集上的正确率，由图像可以看出，训练集上的准确率与测试集上的准确率基本一致，甚至有事测试集上的准确率更高，因此认为这里没有出现过拟合现象，因此没有加入验证集，并将正则项设置为 0.

实验结果如下：

由于数据有 8 个属性，无法直观地在图像中将点画出，查看逻辑回归的效果，因此只能说从错误率来看，逻辑回归最高能够达到将近 80%的正确率。



梯度下降：

迭代次数：3160

Lost Function：0.630084

训练集错误率：0.338920

测试集错误率：0.341991

共轭梯度：

迭代次数：4221

Lost Function：0.489519

训练集错误率：0.221601

测试集错误率：0.203463

牛顿法：

迭代次数：5

Lost Function：0.485152

训练集错误率：0.223464

测试集错误率：0.207792

由于数据有 8 个属性，无法直观地在图像中将点画出，查看逻辑回归的效果，因此只能说从错误率来看，逻辑回归最高能够达到将近 80%的正确率。

另外一个在上个实验中没有遇到的现象是，共轭梯度以较高的迭代次数获得了较高的准确度，但普通的梯度下降，即使进一步调整参数，也很难迭代更多次，且正确率也无法进一步提高。也就是说，对于这个数据，共轭梯度并没有明显的提升效率，而是显著地提升了正确率。

五、 系统不足与经验体会

本次实验能够说明，对于线性可分的数据，这些算法都是有能力进行分割的，但当数据之间有一些重合，不再线性可分时，各自的缺点就开始凸显。比如梯度下降准确度不足，速度慢，共轭梯度有时会过拟合，而牛顿法个别时候会出现不收敛的情况等。

从 UCI 的数据集测试结果来看，这些算法都具有客观的实用性，但还不能直接进行应用，80%的正确率虽然已经不错了，但要真正的能够应用还是远远不够的。这些算法可以作为基础，加入到一些更加复杂的算法中起作用，例如 boosting 方法。

本次实验多数都是在人造数据上进行，且都是通过正态分布造的，情况比实际的应用要简单很多，因此所得出的结论是否能够推广尚未可知。

本次实验比较令我惊讶的是牛顿法的收敛速度竟如此之快，在实验中它的迭代很少超过 10 次，看来二阶导的加入引起了质的变化。

六、附录：源代码（带注释）

不知道为什么出现了乱码，不过压缩文件中的代码应该不会乱。

```
clear;clf;clc;

%==============µ÷Õû²ÎÊý==============
n = 1000;%Ñù±¾ÊýÁ¿
M = 2;%½×Êý
lmd = 0.001;%ÕýÔòÏîµÄÏµÊý£¬Ã»ÓÐÕýÔòÏîÊ±Îª0

%==============»ñÈ¡Êý¾Ý==============
[data,Y] = DataGenerator(n, M);
N = floor(n*0.7);
train_set = data(1:N,:);%»ñµÃÑµÁ·¼¯
test_set = data((N+1):n, :);%»ñµÃ²âÊÔ¼¯
y = Y(1:N);
test_y = Y(N+1:n);
%==============ÏÔÊ¾Êý¾Ý==============
subplot(2, 2, 1);
hold on;
% title('origin');
data0 = train_set(y==0, :);
data1 = train_set(y==1, :);
plot(data0(:,1), data0(:,2), '.', 'color', [0 0 1]);
plot(data1(:,1), data1(:,2), '.', 'color', [1 0 0]);
% %xlabel(sprintf('Erms = %f', (sum((sin(data*2*pi)-y).^2))^0.5));
hold off;

X = [ones(N,1) train_set];
X2 = [ones(n-N,1) test_set];
```

```
%==============ÌÝ¶ÈÏÂ½µ==============
alpha = 0.03;
epsilon = 0.000000001;
W = randn(M+1, 1)./1000;%Ëæ»ú³õÖµ
J = [];%¼ÇÂ¼Ã¿´Îµü´úµÄ¾ù·½²î
J1 = [];
J2 = [];
[J(end+1), grad] = costFunction(W, X, y, lmd);
J1(end+1) = sum((X*W>0)==y)/N;
J2(end+1) = sum((X2*W>0)==test_y)/(n-N);%costFunction(W, X2, test_y, lmd);
W = W - alpha.*grad;
[J(end+1), grad] = costFunction(W, X, y, lmd);
J1(end+1) = sum((X*W>0)==y)/N;
J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
W = W - alpha.*grad;
while (J(end)-J(end-1))^2>epsilon
    [J(end+1), grad] = costFunction(W, X, y, lmd);
    J1(end+1) = sum((X*W>0)==y)/N;
    J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
    W = W - alpha.*grad;
    if length(J) > 10000
        break;
    end
end
fprintf('ÌÝ¶ÈÏÂ½µ£º\n\tµü´ú´ÎÊý£º%d\n\tLost
Function£º%f\n\tÑµÁ·¼¯´íÎóÂÊ£º%f\n\t²âÊÔ¼¯´íÎóÂÊ£º%f\n', [length(J), J(end),
J1(end), J2(end)]);
%»æÍ¼
subplot(2, 2, 2);
hold on;
%semilogx(1:size(J,2),J, 'color', [1 0 1]);
semilogx(1:size(J1,2),J1, 'color', [1 0 1]);
semilogx(1:size(J2,2),J2, 'color', [0 1 0]);
title('Gradient Descent');
xlabel(sprintf('Iteration = %d', size(J,2)));
hold off;
if M==2
    subplot(2, 2, 1);
    hold on;
    p1 = ezplot(sprintf('%f+(%f.*x)+(%f.*y)=0', W'));
    set(p1, 'Color', [1 0 1]);
    hold off;
end
```

```
%==============¹²éîÌÝ¶È==============
epsilon2 = epsilon;
lambda = 0.3;
W = randn(M+1, 1)/1000;
J = [];
J1 = [];
J2 = [];
[J(end+1), gradk] = costFunction(W, X, y, lmd);
J1(end+1) = sum((X*W>0)==y)/N;
J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
pk = -gradk;
W = W + lambda*pk;
[J(end+1), gradk1] = costFunction(W, X, y, lmd);
J1(end+1) = sum((X*W>0)==y)/N;
J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
beta = sum(gradk1.^2)/sum(gradk.^2);
pk1 = beta*pk-gradk1;
W = W + lambda*pk1;
gradk = gradk1;
pk = pk1;
while (J(end)-J(end-1))^2>epsilon2
    [J(end+1), gradk1] = costFunction(W, X, y, lmd);
    J1(end+1) = sum((X*W>0)==y)/N;
    J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
    beta = sum(gradk1.^2)/sum(gradk.^2);
    pk1 = beta*pk-gradk1;
    W = W + lambda*pk1;
    gradk = gradk1;
    pk = pk1;
end
fprintf('¹²éîÌÝ¶È·¨£º\n\tµü´ú´ÎÊý£º%d\n\tLost
Function£º%f\n\tÑµÁ·¼¯´íÎóÂÊ£º%f\n\t²âÊÔ¼¯´íÎóÂÊ£º%f\n', [length(J), J(end),
J1(end), J2(end)]);
%»æÍ¼
subplot(2, 2, 3);
hold on;
semilogx(1:size(J1,2),J1, 'color', [1 0.5 0]);
semilogx(1:size(J2,2),J2, 'color', [0 1 0]);
title('Conjugate Gradient Descent');
xlabel(sprintf('Iteration = %d', size(J,2)));
hold off;
if M==2
    subplot(2, 2, 1);
```

```
    hold on;
    p2 = ezplot(sprintf('%f+(%f.*x)+(%f.*y)=0', W'));
    set(p2, 'Color', [1 0.5 0]);
    hold off;
end
%==============Å£¶Ù·¨==============
epsilon3 = epsilon;
W = randn(M+1, 1)/1000;%Ëæ»ú³õÖµ
J = [];%¼ÇÂ¼Ã¿´Îµü´úµÄ¾ù·½²î
J1 = [];
J2 = [];
[J(end+1), grad, H] = costFunction(W, X, y, lmd);
J1(end+1) = sum((X*W>0)==y)/N;
J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
W = W + pinv(H)*grad;
[J(end+1), grad, H] = costFunction(W, X, y, lmd);
J1(end+1) = sum((X*W>0)==y)/N;
J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
W = W + pinv(H)*grad;
while (J(end)-J(end-1))^2>epsilon3
    [J(end+1), grad, H] = costFunction(W, X, y, lmd);
    J1(end+1) = sum((X*W>0)==y)/N;
    J2(end+1) = sum((X2*W>0)==test_y)/(n-N);
    W = W + pinv(H)*grad;
end
fprintf('Å£¶Ù·¨£º\n\tµü´ú´ÎÊý£º%d\n\tLost
Function£º%f\n\tÑµÁ·¼¯´íÎóÂÊ£º%f\n\t²âÊÔ¼¯´íÎóÂÊ£º%f\n', [length(J), J(end),
J1(end), J2(end)]);
%»æÍ¼
subplot(2, 2, 4);
hold on;
%semilogx(1:size(J,2),J, 'color', [0 0.5 1]);
semilogx(1:size(J1,2),J1, 'color', [0 0.5 1]);
semilogx(1:size(J2,2),J2, 'color', [0 1 0]);
title('Newton');
xlabel(sprintf('Iteration = %d', size(J,2)));
hold off;
if M==2
    subplot(2, 2, 1);
    hold on;
    p3 = ezplot(sprintf('%f+(%f.*x)+(%f.*y)=0', W'));
    set(p3, 'Color', [0 0.5 1]);
    title('data');
    % xlabel(sprintf('Erms = %f', (sum((X*W-y).^2))^0.5));
```

```
    hold off;
end

function [ J, grad, H ] = costFunction( W, X, y, lmd)

lambda = lmd;
m = length(y);
n = size(X,2);
h = ones(m,1)./(1+exp(X*W));
J = -(sum(y.*log(h) + (ones(m,1)-y).*log(ones(m,1)-h)))/m;
%grad = (X'*(h-y)+2*lambda*W)./m;
grad = -(X'*(h-y))./m + lambda*[0; W(2:end)];

H = zeros(n,n);
for i=(1:n)
    for j=(1:n)
        SUM = 0;
        for t=(1:m)
            SUM = SUM + h(t)*(h(t)-1)*X(t,i)*X(t,j);
        end
        H(i,j) = SUM./m;
    end
    if i~=1
        H(i,i) = H(i,i) + lambda;
    end
end

end


function [ X, y ] = DataGenerator( N, M )

if M==2
    p = 2;
    n = N/2;
    X = [repmat([0, 0], n, 1); repmat([2, 1], N-n, 1)];
    X = X + randn(N,2)./p;
    y = [zeros(n,1); ones(n,1)];
else
    %Õâ¸öÊý¾Ý¾ÍÊÇ±¨¸æÖÐÌáµ½µÄÕæÊµÊý¾Ý
    data = importdata('data.mat');
    X = data(:,1:8);
    y = data(:,9);
end
```

```
end
```