上海理工大学光电信息与计算机工程学院

**《机器学习》实验报告**

****

**专　　业 智能科学与技术**

**姓 名　　 高浩琦**

**学　 号 13**

**年　　级 2020级**

**指导教师 傅迎华**

**成 绩：**

**教师签字：**

目录

[**实验一 线性回归机器学习** 5](#_Toc74657770)

[**实验二 Logistic回归** 9](#_Toc74657771)

[**实验三 SVM** 14](#_Toc74657772)

[**实验四 神经网络学习** 21](#_Toc74657773)

**实验一 线性回归机器学习**

一、实验目的和要求

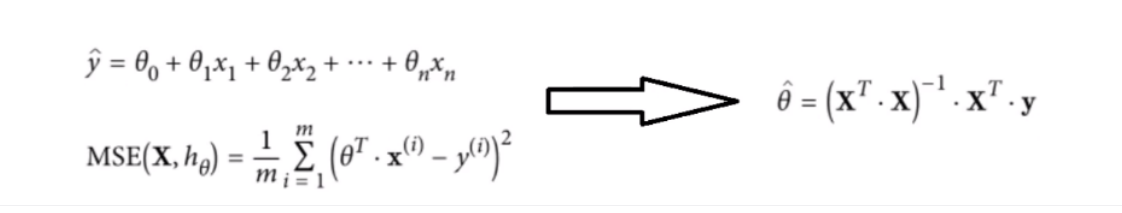
1.编程实现Linear Regression模型，计算线性回归损失的函数，求解实际问题

2.编程实现梯度下降算法（BGD、SGD、minibatch-GD），比较学习率α对结果影响

3.使用Linear Regression的标准方程法求解最优解

二、实验算法

**回归方程：**



**梯度下降算法：**

1. 确定参数的初始值，计算损失函数的偏导数

2．将参数代入偏导数计算出梯度。若梯度为0，结束；否则转到3

3. 用步长乘以梯度，并对参数进行更新

4. 重复2-3

三、实验过程记录

1.计算损失函数

线性回归的目的是使成本函数最小化



function J = computeCost(X, y, theta)

m = length(y);

J = 0;

J = sum((X \* theta - y).^2) / (2\*m);

end

2.梯度下降算法

for iter = 1:num\_iters

theta(1) = theta(1) - alpha / m \* sum(X \* theta\_s - y);

theta(2) = theta(2) - alpha / m \* sum((X \* theta\_s - y) .\* X(:,2));

theta\_s=theta;

J\_history(iter) = computeCost(X, y, theta);

End

3.Linear Regression的标准方程法

function [theta] = normalEqn(X, y)

theta = zeros(size(X, 2), 1);

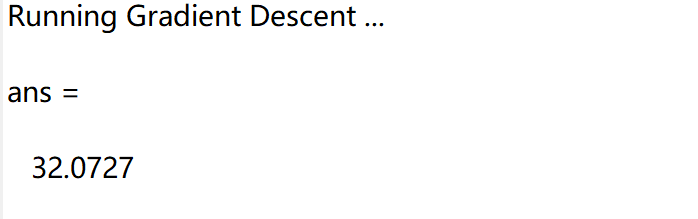
theta = pinv( X' \* X ) \* X' \* y;

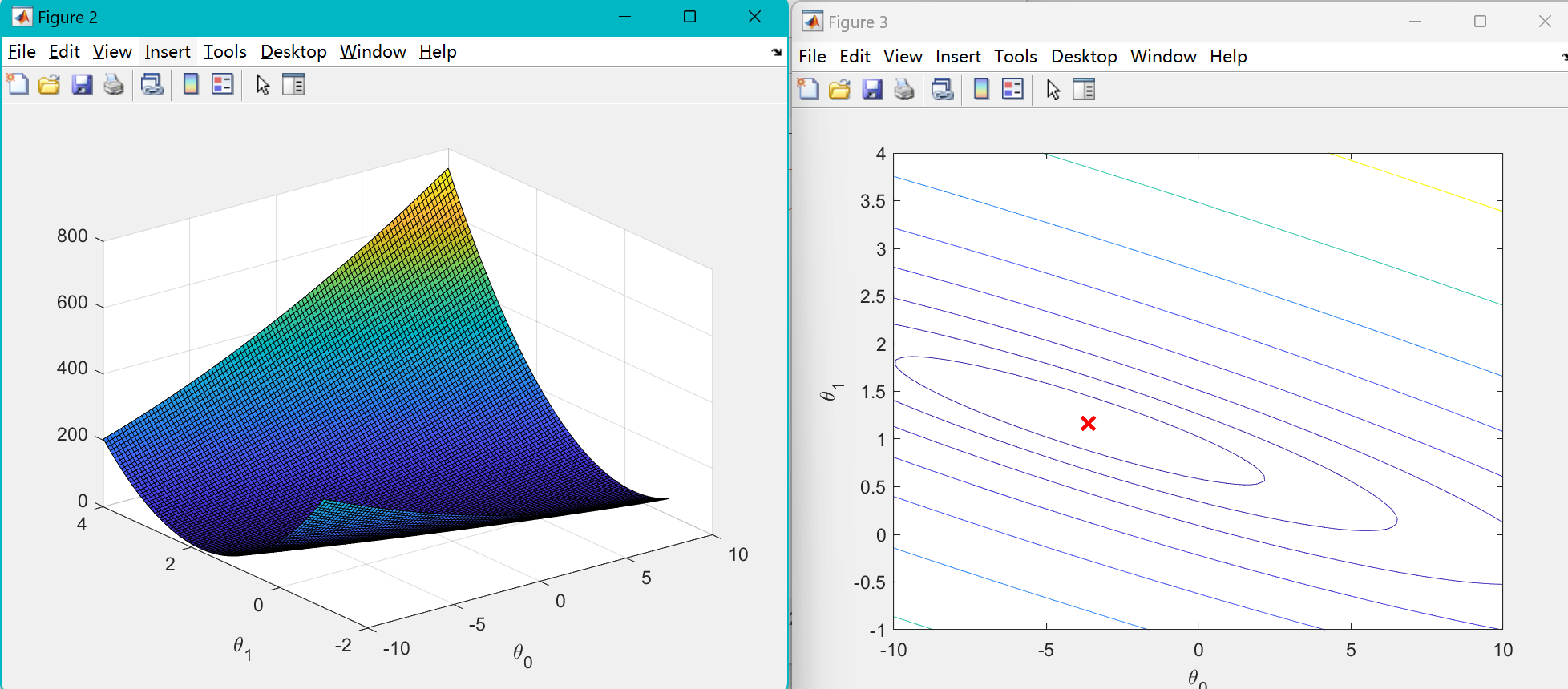
end

四、结果分析

1.计算损失函数

计算J(theta)的函数。使用初始化为零的theta运行一次calculateCost.m

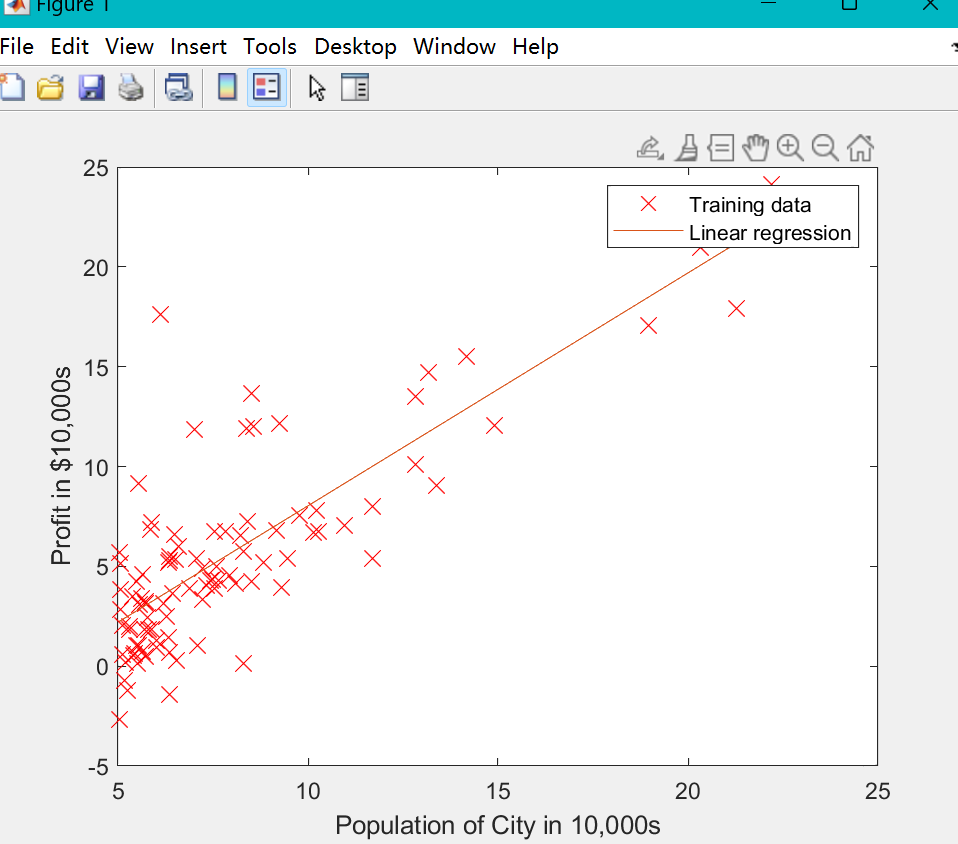




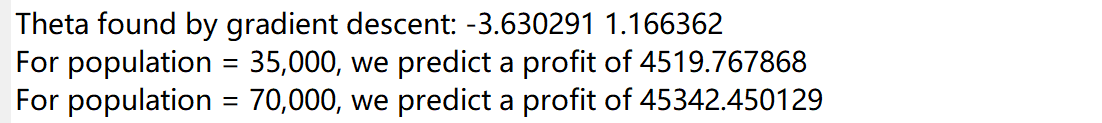
图中展示J(theta)如何随θ0和θ1的变化而变化。成本函数J(theta)是碗形的，并且具有全局最小值。该最小值是θ0和θ1的最佳点，并且梯度下降的每一步都靠近该点。

2.梯度下降算法

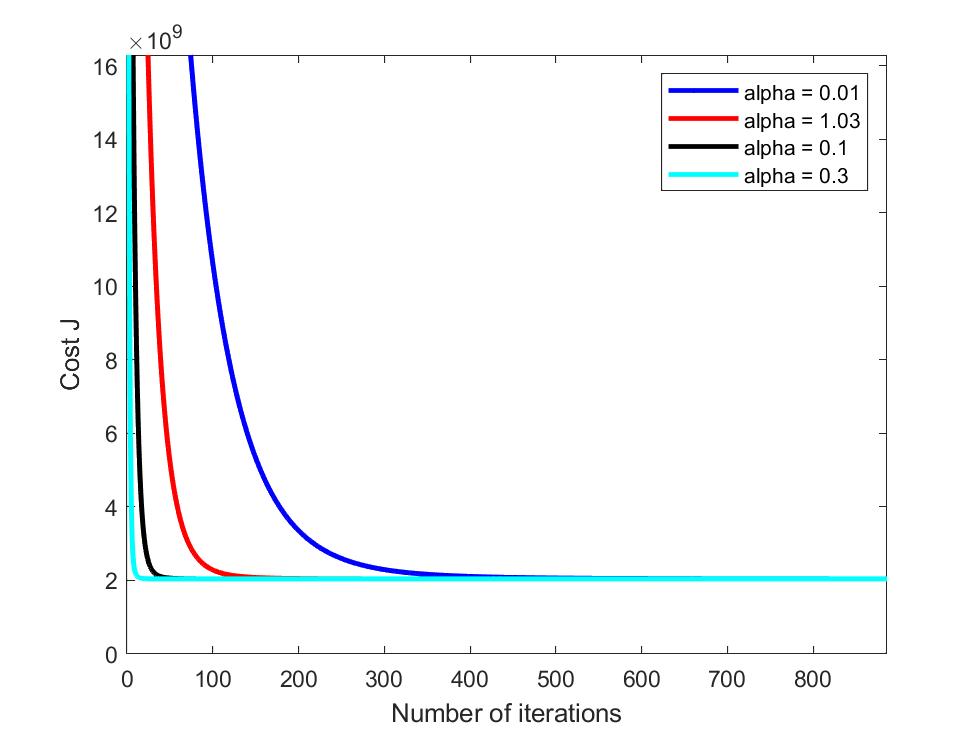
计算J(theta)的函数。使用初始化为零的运行一次calculateCost.m



最终的结果预测为如下：



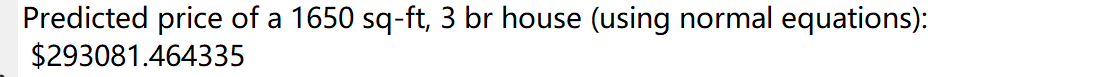
关于alptha学习率的变化：



如图可得，随着学习速率的变化，收敛曲线也会发生变化。学习率小的情况下，你应该会发现梯度下降需要很长时间才能收敛到最优值。相反，在学习率较大的情况下，梯度下降可能不会收敛，甚至可能会发散。

3.Linear Regression的标准方程法

使用这个公式不需要任何特征缩放，一次计算就可以得到精确解:不存在梯度下降中的“循环直到收敛”。



五、自评

在求解[机器学习算法](https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0%E7%AE%97%E6%B3%95/18635836?fromModule=lemma_inlink)的模型参数，即无约束优化问题时，梯度下降（Gradient Descent）是最常采用的方法之一，它还是一个[最优化](https://baike.baidu.com/item/%E6%9C%80%E4%BC%98%E5%8C%96?fromModule=lemma_inlink)算法，常用于机器学习和[人工智能](https://baike.baidu.com/item/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD/9180?fromModule=lemma_inlink)当中用来递归性地逼近最小偏差模型。

**实验二 Logistic回归**

一、实验目的和要求

1.编程实现Logistic Regression并应用于数据集；

2.绘制二元分类函数、sigmoid函数和代价函数；

3.正则化logistic回归代价。

二、实验算法

1．Logistic Regression

这是一种广义线性回归（generalized linear model），因此与多重线性回归分析有很多相同之处。它们的模型形式基本上相同，都具有 wx+b，其中w和b是待求参数，其区别在于他们的因变量不同，多重线性回归直接将wx+b作为因变量，即y =wx+b，而logistic回归则通过函数L将wx+b对应一个隐状态p，p =L(wx+b),然后根据p 与1-p的大小决定因变量的值。

2. 正则化logistic回归代价

正则化思想是参数值较小意味着更简单的假设模型，一般来说，theta的值越小，我们得到的曲线越平滑，也越简单，更不容易出现过拟合。

三、实验过程记录

1．Logistic Regression

调用fminunc，fminunc将收敛到正确的优化参数

function [J, grad] = costFunction(theta, X, y)

m = length(y);

J = 0;

grad = zeros(size(theta));

J= -1 \* sum( y .\* log( sigmoid(X\*theta) ) + (1 - y ) .\* log( (1 - sigmoid(X\*theta)) ) ) / m ;

grad = ( X' \* (sigmoid(X\*theta) - y ) )/ m ;

end

2. 绘制二元分类函数、sigmoid函数和代价函数

function g = sigmoid(z)

g = zeros(size(z));

g = 1 ./ ( 1 + exp(-z) ) ;

end

3.正则化logistic回归代价

function p = predict(theta, X)

m = size(X, 1);

p = zeros(m, 1);

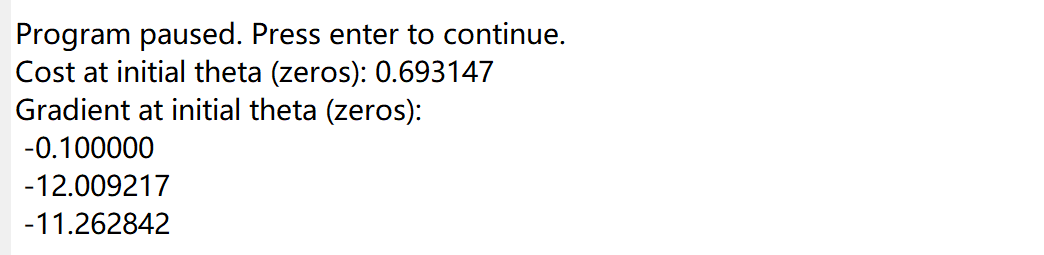
k = find(sigmoid( X \* theta) >= 0.5 );

p(k)= 1;

end

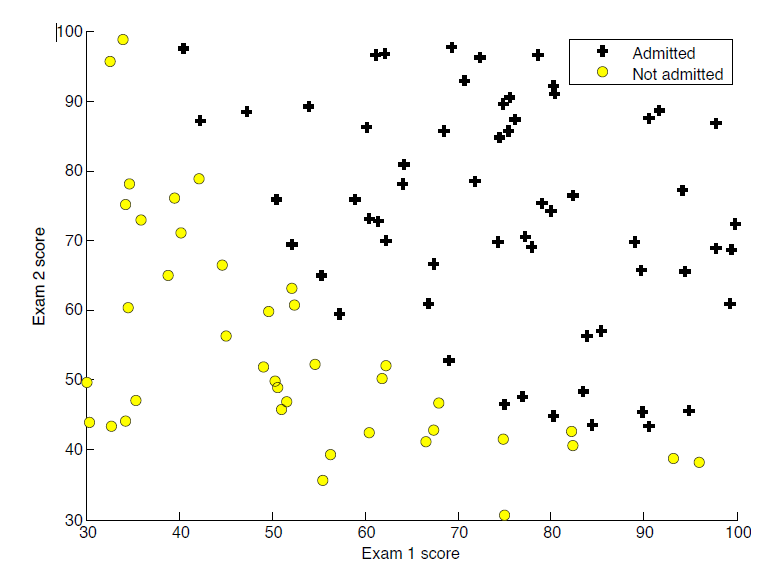
四、结果分析

1．Logistic Regression

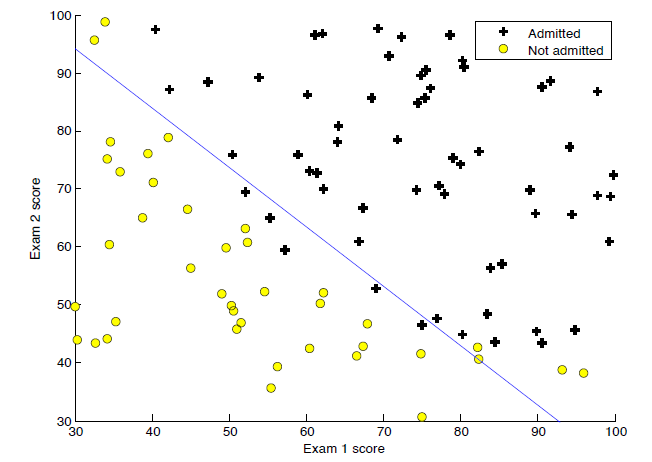


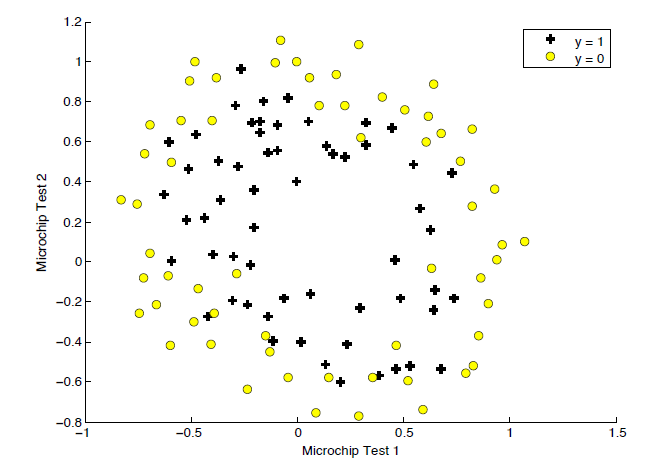
2. 绘制二元分类函数、sigmoid函数和代价函数

**Sigmoid**

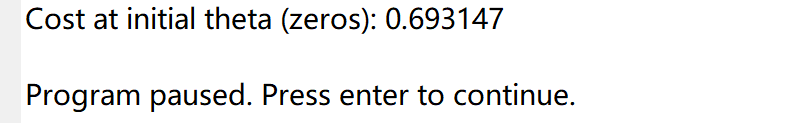


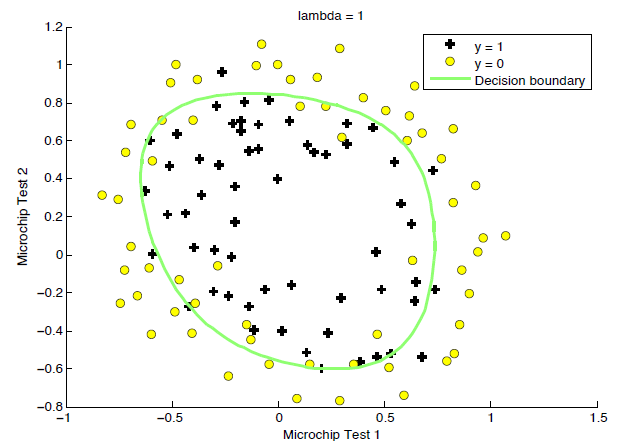
**二元分类**



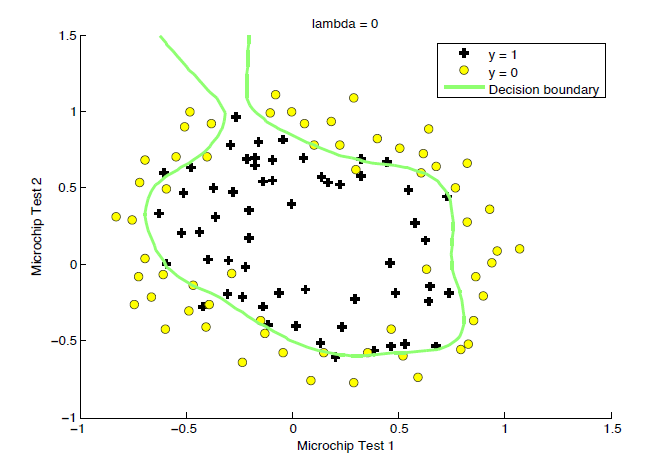


3.正则化logistic回归代价

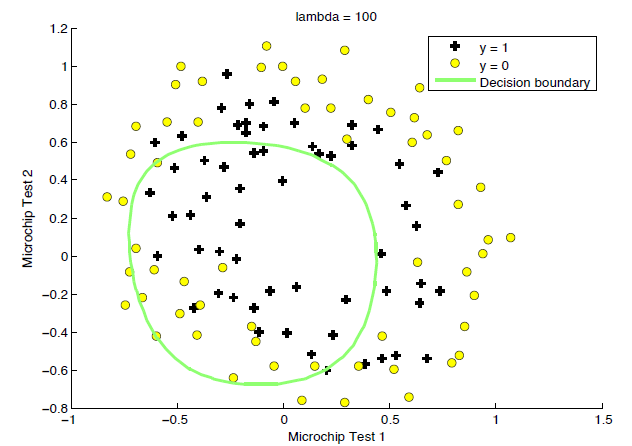




具有决策边界的训练数据（λ=1）



无正则化（过拟合）（λ=0）



正则化过多（欠拟合）（λ=100）

五、自评

logistic回归又称logistic[回归分析](https://baike.baidu.com/item/%E5%9B%9E%E5%BD%92%E5%88%86%E6%9E%90?fromModule=lemma_inlink)，是一种广义的线性回归分析模型，常用于数据挖掘，疾病自动诊断，经济预测等领域。例如，探讨引发疾病的危险因素，并根据危险因素预测疾病发生的概率等。以胃癌病情分析为例，选择两组人群，一组是胃癌组，一组是非胃癌组，两组人群必定具有不同的体征与生活方式等。因此[因变量](https://baike.baidu.com/item/%E5%9B%A0%E5%8F%98%E9%87%8F?fromModule=lemma_inlink)就为是否胃癌，值为“是”或“否”，自变量就可以包括很多了，如年龄、性别、饮食习惯、[幽门螺杆菌](https://baike.baidu.com/item/%E5%B9%BD%E9%97%A8%E8%9E%BA%E6%9D%86%E8%8F%8C?fromModule=lemma_inlink)感染等。自变量既可以是连续的，也可以是分类的。然后通过logistic回归分析，可以得到自变量的权重，从而可以大致了解到底哪些因素是胃癌的危险因素。同时根据该权值可以根据危险因素预测一个人患癌症的可能性。

**实验三 SVM**

一、实验目的和要求

1.编程实现SVM训练函数和预测函数；

2.绘制线性和非线性边界；

3.编写线性核函数

二、实验算法

**SVM训练函数和预测函数**

支持向量机（Support Vector Machine, SVM）是一类按监督学习（supervised learning）方式对数据进行二元分类的广义线性分类器（generalized linear classifier），其决策边界是对学习样本求解的最大边距超平面（maximum-margin hyperplane）。

三、实验过程记录

1.编程实现SVM训练函数和预测函数；

**svmTrain**

function [model] = svmTrain(X, Y, C, kernelFunction, ...

tol, max\_passes)

if ~exist('tol', 'var') || isempty(tol)

tol = 1e-3;

end

if ~exist('max\_passes', 'var') || isempty(max\_passes)

max\_passes = 5;

end

% Data parameters

m = size(X, 1);

n = size(X, 2);

% Map 0 to -1

Y(Y==0) = -1;

% Variables

alphas = zeros(m, 1);

b = 0;

E = zeros(m, 1);

passes = 0;

eta = 0;

L = 0;

H = 0;

if strcmp(func2str(kernelFunction), 'linearKernel')

% Vectorized computation for the Linear Kernel

% This is equivalent to computing the kernel on every pair of examples

K = X\*X';

elseif strfind(func2str(kernelFunction), 'gaussianKernel')

% Vectorized RBF Kernel

% This is equivalent to computing the kernel on every pair of examples

X2 = sum(X.^2, 2);

K = bsxfun(@plus, X2, bsxfun(@plus, X2', - 2 \* (X \* X')));

K = kernelFunction(1, 0) .^ K;

else

% Pre-compute the Kernel Matrix

% The following can be slow due to the lack of vectorization

K = zeros(m);

for i = 1:m

for j = i:m

K(i,j) = kernelFunction(X(i,:)', X(j,:)');

K(j,i) = K(i,j); %the matrix is symmetric

end

end

end

% Train

fprintf('\nTraining ...');

dots = 12;

while passes < max\_passes,

num\_changed\_alphas = 0;

for i = 1:m,

% Calculate Ei = f(x(i)) - y(i) using (2).

% E(i) = b + sum (X(i, :) \* (repmat(alphas.\*Y,1,n).\*X)') - Y(i);

E(i) = b + sum (alphas.\*Y.\*K(:,i)) - Y(i);

if ((Y(i)\*E(i) < -tol && alphas(i) < C) || (Y(i)\*E(i) > tol && alphas(i) > 0)),

j = ceil(m \* rand());

while j == i, % Make sure i \neq j

j = ceil(m \* rand());

end

% Calculate Ej = f(x(j)) - y(j) using (2).

E(j) = b + sum (alphas.\*Y.\*K(:,j)) - Y(j);

% Save old alphas

alpha\_i\_old = alphas(i);

alpha\_j\_old = alphas(j);

% Compute L and H by (10) or (11).

if (Y(i) == Y(j)),

L = max(0, alphas(j) + alphas(i) - C);

H = min(C, alphas(j) + alphas(i));

else

L = max(0, alphas(j) - alphas(i));

H = min(C, C + alphas(j) - alphas(i));

end

if (L == H),

% continue to next i.

continue;

end

% Compute eta by (14).

eta = 2 \* K(i,j) - K(i,i) - K(j,j);

if (eta >= 0),

% continue to next i.

continue;

end

% Compute and clip new value for alpha j using (12) and (15).

alphas(j) = alphas(j) - (Y(j) \* (E(i) - E(j))) / eta;

% Clip

alphas(j) = min (H, alphas(j));

alphas(j) = max (L, alphas(j));

% Check if change in alpha is significant

if (abs(alphas(j) - alpha\_j\_old) < tol),

% continue to next i.

% replace anyway

alphas(j) = alpha\_j\_old;

continue;

end

% Determine value for alpha i using (16).

alphas(i) = alphas(i) + Y(i)\*Y(j)\*(alpha\_j\_old - alphas(j));

% Compute b1 and b2 using (17) and (18) respectively.

b1 = b - E(i) ...

- Y(i) \* (alphas(i) - alpha\_i\_old) \* K(i,j)' ...

- Y(j) \* (alphas(j) - alpha\_j\_old) \* K(i,j)';

b2 = b - E(j) ...

- Y(i) \* (alphas(i) - alpha\_i\_old) \* K(i,j)' ...

- Y(j) \* (alphas(j) - alpha\_j\_old) \* K(j,j)';

% Compute b by (19).

if (0 < alphas(i) && alphas(i) < C),

b = b1;

elseif (0 < alphas(j) && alphas(j) < C),

b = b2;

else

b = (b1+b2)/2;

end

num\_changed\_alphas = num\_changed\_alphas + 1;

end

end

if (num\_changed\_alphas == 0),

passes = passes + 1;

else

passes = 0;

end

fprintf('.');

dots = dots + 1;

if dots > 78

dots = 0;

fprintf('\n');

end

if exist('OCTAVE\_VERSION')

fflush(stdout);

end

end

fprintf(' Done! \n\n');

% Save the model

idx = alphas > 0;

model.X= X(idx,:);

model.y= Y(idx);

model.kernelFunction = kernelFunction;

model.b= b;

model.alphas= alphas(idx);

model.w = ((alphas.\*Y)'\*X)';

end

**svmPredict**

function pred = svmPredict(model, X)

if (size(X, 2) == 1)

% Examples should be in rows

X = X';

end

% Dataset

m = size(X, 1);

p = zeros(m, 1);

pred = zeros(m, 1);

if strcmp(func2str(model.kernelFunction), 'linearKernel')

p = X \* model.w + model.b;

elseif strfind(func2str(model.kernelFunction), 'gaussianKernel')

X1 = sum(X.^2, 2);

X2 = sum(model.X.^2, 2)';

K = bsxfun(@plus, X1, bsxfun(@plus, X2, - 2 \* X \* model.X'));

K = model.kernelFunction(1, 0) .^ K;

K = bsxfun(@times, model.y', K);

K = bsxfun(@times, model.alphas', K);

p = sum(K, 2);

else

% Other Non-linear kernel

for i = 1:m

prediction = 0;

for j = 1:size(model.X, 1)

prediction = prediction + ...

model.alphas(j) \* model.y(j) \* ...

model.kernelFunction(X(i,:)', model.X(j,:)');

end

p(i) = prediction + model.b;

end

end

% Convert predictions into 0 / 1

pred(p >= 0) = 1;

pred(p < 0) = 0;

end

2.绘制线性和非线性边界；

**非线性**

function visualizeBoundary(X, y, model, varargin)

plotData(X, y)

x1plot = linspace(min(X(:,1)), max(X(:,1)), 100)';

x2plot = linspace(min(X(:,2)), max(X(:,2)), 100)';

[X1, X2] = meshgrid(x1plot, x2plot);

vals = zeros(size(X1));

for i = 1:size(X1, 2)

this\_X = [X1(:, i), X2(:, i)];

vals(:, i) = svmPredict(model, this\_X);

end

hold on

contour(X1, X2, vals, [0 0], 'Color', 'b');

hold off;

end

**线性边界**

function visualizeBoundaryLinear(X, y, model)

w = model.w;

b = model.b;

xp = linspace(min(X(:,1)), max(X(:,1)), 100);

yp = - (w(1)\*xp + b)/w(2);

plotData(X, y);

hold on;

plot(xp, yp, '-b');

hold off

end

3.编写线性核函数

function sim = linearKernel(x1, x2)

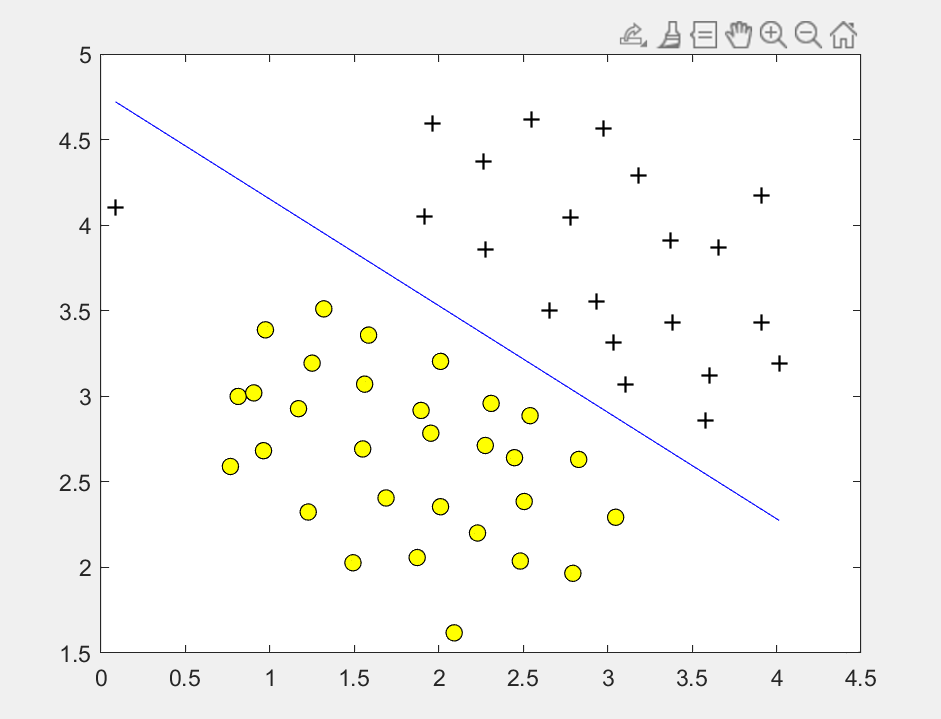
x1 = x1(:); x2 = x2(:);

sim = x1' \* x2; % dot product

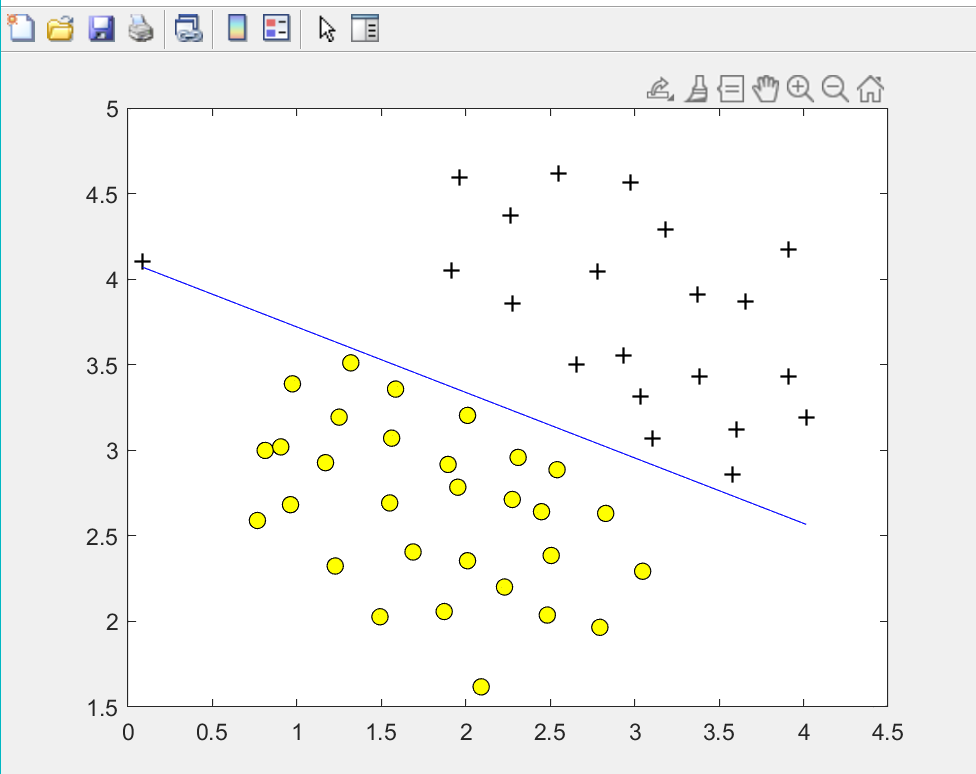
end

四、结果分析

C = 1时，高斯核函数拟合S，将决策边界放在两个数据集之间的间隙中，并将数据点错误地分类在最左边。



C = 100时， SVM现在对每个例子都进行了正确的分类，但是它的决策边界不能自然地适合数据。



五、自评

SVM是一种二类分类模型。它的基本思想是在特征空间中寻找间隔最大的分离超平面使数据得到高效的二分类，具体来讲，有三种情况(不加核函数的话就是个线性模型，加了之后才会升级为一个非线性模型)∶

当训练样本线性可分时，通过硬间隔最大化，学习一个线性分类器，即线性可分支持向量机;

当训练数据近似线性可分时，引入松弛变量，通过软间隔最大化，学习一个线性分类器，即线性支持向量机;

当训练数据线性不可分时，通过使用核技巧及软间隔最大化，学习非线性支持向量机。

**实验四 神经网络学习**

**一、实验目的和要求**

1.计算sigmoid函数的梯度；

2.随机初始化网络权重；

3.编写网络的代价函数。

**二、实验算法**

sigmoid函数也叫logistic函数，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。Sigmoid作为激活函数有以下优缺点：

优点：平滑、易于求导。

缺点：激活函数计算量大，反向传播求误差梯度时，求导涉及除法；反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练。

**三、实验过程记录**

1.计算sigmoid函数的梯度；

function g = sigmoidGradient(z)

g = zeros(size(z));

g = sigmoid(z) .\* (1 - sigmoid(z));

end

2.随机初始化网络权重；

function W = randInitializeWeights(L\_in, L\_out)

W = zeros(L\_out, 1 + L\_in);

epsilon\_init = 0.12;

W = rand(L\_out, 1 + L\_in) \* 2 \* epsilon\_init - epsilon\_init;

End

3.编写网络的代价函数。

%% 计算前向传输

X = [ones(m, 1) X];

a2 = sigmoid(X \* Theta1'); % ?????????

a2 = [ones(m, 1) a2]; % ?????b

a3 = sigmoid(a2 \* Theta2');

temp1 = [zeros(size(Theta1,1),1) Theta1(:,2:end)]; % ??theta(1)?????????

temp2 = [zeros(size(Theta2,1),1) Theta2(:,2:end)];

temp1 = sum(temp1 .^2); % ??????????????

temp2 = sum(temp2 .^2);

cost = Y .\* log(a3) + (1 - Y ) .\* log( (1 - a3)); % cost?m\*K(5000\*10)????? sum(cost(:))????

J= -1 / m \* sum(cost(:)) + lambda/(2\*m) \* ( sum(temp1(:))+ sum(temp2(:)) );

%% 计算gradient

delta\_1 = zeros(size(Theta1));

delta\_2 = zeros(size(Theta2));

for t = 1:m

% step 1

a\_1 = X(t,:)';

% a\_1 = [1 ; a\_1];

z\_2 = Theta1 \* a\_1;

a\_2 = sigmoid(z\_2);

a\_2 = [1 ; a\_2];

z\_3 = Theta2 \* a\_2;

a\_3 = sigmoid(z\_3);

% step 2

err\_3 = zeros(num\_labels,1);

for k = 1:num\_labels

err\_3(k) = a\_3(k) - (y(t) == k);

end

% step 3

err\_2 = Theta2' \* err\_3;

err\_2 = err\_2(2:end) .\* sigmoidGradient(z\_2);

% step 4

delta\_2 = delta\_2 + err\_3 \* a\_2';

delta\_1 = delta\_1 + err\_2 \* a\_1';

end

Theta1\_temp = [zeros(size(Theta1,1),1) Theta1(:,2:end)];

Theta2\_temp = [zeros(size(Theta2,1),1) Theta2(:,2:end)];

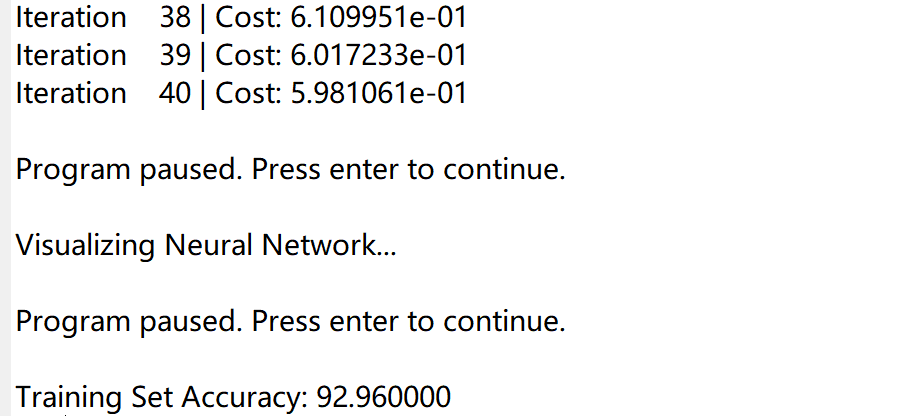
Theta1\_grad = 1 / m \* delta\_1 + lambda/m \* Theta1\_temp;

Theta2\_grad = 1 / m \* delta\_2 + lambda/m \* Theta2\_temp ;

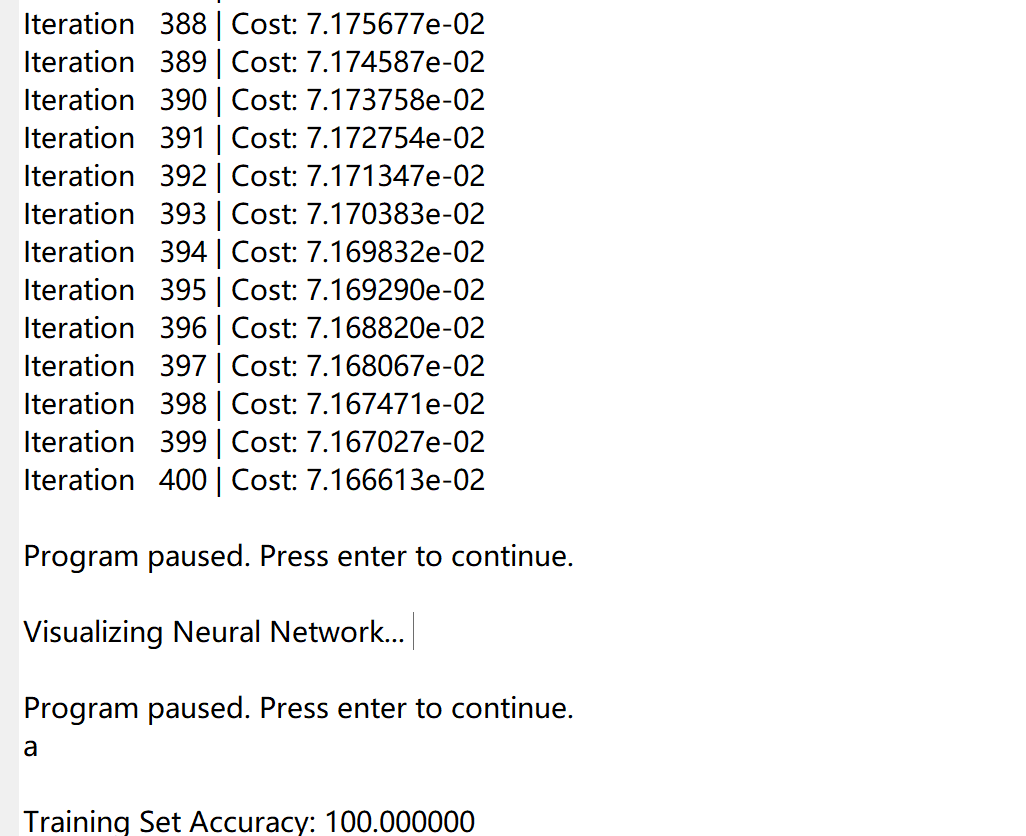
grad = [Theta1\_grad(:) ; Theta2\_grad(:)];

**四、结果分析**

**Lamda = 1,maxlter = 40,** **通过对神经网络进行多次迭代训练，我们就可以获得较高的训练精度。**



然而在训练过程中,当lambda = 0.1, maxlter = 400 时,得到准确率为100%,但是神经网络“过度拟合”了训练集。



**五、自评**

人工神经网络( ANN )，通常简称为神经网络( NN ) 或神经网络，[[1]](https://en.wikipedia.org/wiki/Artificial_neural_network#cite_note-1)是受构成动物[大脑的](https://en.wikipedia.org/wiki/Brain)[生物神经网络](https://en.wikipedia.org/wiki/Biological_neural_network)启发的计算系统。

ANN 基于称为[人工神经元](https://en.wikipedia.org/wiki/Artificial_neuron)的连接单元或节点的集合，这些单元或节点松散地模拟生物大脑中的[神经元。](https://en.wikipedia.org/wiki/Neuron)每个连接，就像生物大脑中的[突触一样，可以将信号传输到其他神经元。](https://en.wikipedia.org/wiki/Synapse)人工神经元接收信号然后对其进行处理，并可以向与之相连的神经元发出信号。连接处的“信号”是[实数](https://en.wikipedia.org/wiki/Real_number)，每个神经元的输出由其输入总和的某个非线性函数计算得出。这些连接称为边。神经元和边缘通常有一个[权重](https://en.wikipedia.org/wiki/Weighting)随着学习的进行而调整。权重增加或减少连接处的信号强度。神经元可能有一个阈值，只有当聚合信号超过该阈值时才会发送信号。

通常，神经元聚集成层。不同的层可能对其输入执行不同的转换。信号从第一层（输入层）传播到最后一层（输出层），可能在多次遍历这些层之后。