# 第3章 逻辑回归

CHAPTER 3

逻辑回归(Logistic regression)与第2章中讨论的线性回归相似,是一种利用线性函数构建模型,但是线性回归假设模型的输出结果服从高斯分布,所以线性回归模型输出连续的预测值,解决机器学习中的回归问题;而逻辑回归假设模型输出服从伯努利分布,基于线性模型引入非线性函数,例如Sigmoid函数,让模型输出非线性离散的预测值,解决机器学习中的分类问题。

## 3.1 逻辑回归模型

逻辑回归模型虽然名字中包含“回归”二字,但它却是一种应用广泛的分类算法。对于分类问题,是希望建立的模型输出结果为某一个确定的类别,例如正确或者错误,也就是1或者0的二分类问题。实际应用中判断一封电子邮件是否垃圾邮件,判断一次金融交易是否欺诈,此前肿瘤分类问题中区分肿瘤是恶性还是良性的,手写数字是0、1、2,还是3等十个类别的多分类问题等。

其中最简单的二分类可以定义为

$$y=g(x) \quad y\in\{0,1\} \tag{3-1}$$

其中0表示负类(Negative class),1表示正类(Positive class)。

基于线性函数的逻辑回归分类模型定义为

$$H_{\boldsymbol{\theta}}(\boldsymbol{x})=g(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}) \tag{3-2}$$

其中,$\boldsymbol{x}$为输入特征向量;$g$为逻辑函数,让模型输出范围在0和1之间,一般常用的逻辑函数有S型函数Sigmoid,其表达式为

$$g(z)=\frac{1}{1+\mathrm{e}^{-z}} \tag{3-3}$$

该函数图形如图3.1所示。

因此,逻辑回归模型的假设函数可以记作:

$$H_{\boldsymbol{\theta}}(\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}}} \tag{3-4}$$

式(3-4)可以理解为:对于给定的输入变量$\boldsymbol{x}$,根据所选择的参数计算输出变量为1的概率(Estimated probability),即:

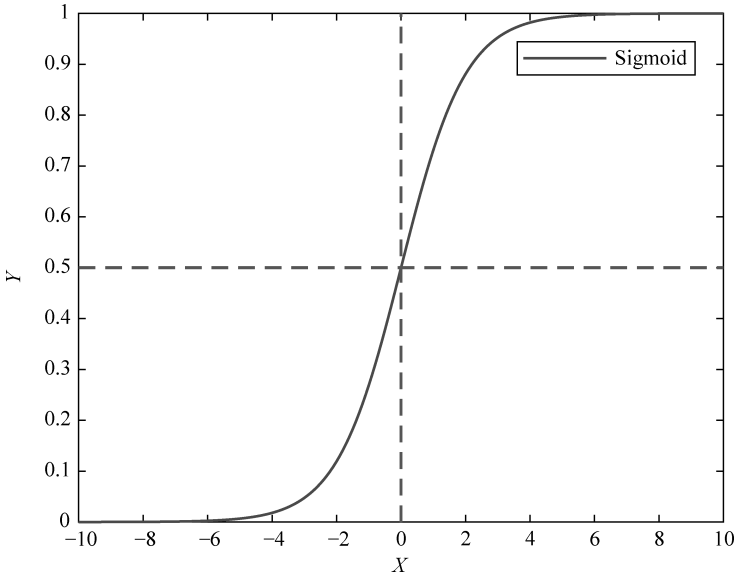



**图 3.1　Sigmoid 函数图形**

$$H_{\boldsymbol{\theta}}(\boldsymbol{x}) = P(y=1 \mid \boldsymbol{x};\boldsymbol{\theta}) \tag{3-5}$$

例如，对于给定的特征向量 $\boldsymbol{x}$，选择参数$\boldsymbol{\theta}$，计算出 $H_{\boldsymbol{\theta}}(\boldsymbol{x})=0.7$，则表示这组样本 $\boldsymbol{x}$ 为正类 1 的概率为 70%，而为负类的概率为 1−0.7=0.3=30%。

做出这样的分类决策的依据称为决策边界(Decision boundary)，对于 S 型函数的决策边界可以表示伪代码：

```
if  H_θ(x)≥0.5  then:  输出正类 "1"
if  H_θ(x)<0.5  then:  输出负类 "0"
```

决策边界函数可以是简单的直线，也可以是更复杂的二次函数，例如圆形分界函数，根据不同的样本分布特征需要确定不同的判定边界函数。

## 3.2　逻辑回归的代价函数

逻辑回归中的代价函数的含义与线性回归中一致，是为逼近参数$\boldsymbol{\theta}$而定义的优化函数，在线性回归中，用模型的误差平方和作为代价函数，理论上，在逻辑回归中也可以采用模型的误差平方和为代价函数，然而将得到一个非凸函数(Non-convex function)。这意味这个代价函数有很多局部最小值，这样非常不利于采用梯度下降算法寻找全局最小值。所以，可以重新定义逻辑回归的代价函数为：

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\mathrm{Cost}(H_{\boldsymbol{\theta}}(x_i), y_i) \tag{3-6}$$

其中，

$$\mathrm{Cost}(H_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = \begin{cases} -\log(H_{\boldsymbol{\theta}}(\boldsymbol{x})), & y=1 \\ -\log(1-H_{\boldsymbol{\theta}}(\boldsymbol{x})), & y=0 \end{cases} \tag{3-7}$$

则逻辑回归的代价函数可以表示为：

$$J(\boldsymbol{\theta})=-\frac{1}{n}\left[\sum_{i=1}^{n}y_i\log H_{\boldsymbol{\theta}}(x_i)+(1-y_i)\log(1-H_{\boldsymbol{\theta}}(x_i))\right] \tag{3-8}$$

得到这样的代价函数后，就可以用梯度下降法来求使得代价函数取最小值的参数了。逻辑回归中的梯度下降法可以表示为：

重复：
{

$$\theta_j:=\theta_j-\alpha\frac{\partial}{\partial\theta_j}J(\theta)=\theta_j-\alpha\sum_{i=1}^{n}(H_\theta(x_i)-y_i)x_{i,j}$$

}

对于逻辑回归模型的代价函数的凸性分析可以参考相应的数学知识，这里只给出逻辑回归模型的代价函数是凸函数的结论，并且没有局部最优值。当然和线性回归模型一样，在梯度下降更新时，要同时更新所有输入特征，尽管逻辑回归的梯度下降算法和线性回归下降算法很相似，但是这里逻辑回归模型和线性回归模型的输出是不一样的，所以实际上逻辑回归中的梯度下降和线性回归中的梯度下降是不一样的。此外，在运行逻辑回归梯度下降算法之前，如果多特征输入变量的范围存在很大差别，那么进行归一化的特征缩放依然是必要的。

对逻辑回归模型的代价函数除了用梯度下降法寻找最小值外，也可以用其他更快、更优越的算法来求解，这些算法有共轭梯度(Conjugate gradient)、局部优化法(Broyden Fletcher Goldfarb Shanno，BFGS)、有限内存局部优化法(Limited-memory BFGS)等。

## 3.3 优化函数

为求代价函数的最小值，除了可以采用一阶导数的梯度下降法迭代求解，也可以采用二阶导数的海森矩阵(Hessen matrix)对代价函数求解最小值。其中比较常用的就是牛顿法(Newton method)。牛顿法是一种在实数域和复数域上近似求解方程的方法，相对于梯度下降法最大的优点是收敛速度很快，但是因为基于二阶导数，所以如同 3.2 节提到的共轭梯度等算法一样，其不足是计算复杂度非常高，但是随着硬件计算平台性能依据摩尔定律不断提升，这些收敛速度更快的复杂算法也逐渐在工程实践中得到广泛应用。

对于逻辑回归的代价函数 $J(\boldsymbol{\theta})$，无论采用梯度下降法，还是牛顿法，其目的都是想找到使得 $\min J(\boldsymbol{\theta})$的$\boldsymbol{\theta}$ 值，可以表示为：

$$\frac{\partial}{\partial\theta_j}J(\boldsymbol{\theta})\overset{\text{set}}{=}0\quad(\boldsymbol{\theta}\in\mathbf{R}^{m+1}) \tag{3-9}$$

对于式(3-9)，通过梯度下降法或者牛顿法来寻找最接近的$\boldsymbol{\theta}$ 组合。上式中如果假设 $\boldsymbol{\theta}\in\mathbf{R}$，可以记作：

$$f(\boldsymbol{\theta})=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}J(\boldsymbol{\theta})=0 \tag{3-10}$$

对于上述函数 $f(\boldsymbol{\theta})$，牛顿法的原理是将函数在某点 $\theta_0$ 做线性化近似，即：

$$f(\boldsymbol{\theta})\approx f(\theta_0)+f'(\theta_0)(\boldsymbol{\theta}-\theta_0) \tag{3-11}$$

令式(3-11)左边为零，可以得到：

$$\theta = \theta_0 - \frac{f(\theta_0)}{f'(\theta_0)} \tag{3-12}$$

这样经过若干次迭代后，可以得到一系列$\{\theta_0, \theta_1, \theta_2, \theta_3, \cdots\}$，这样牛顿法会很快收敛。

用代价函数来表示牛顿法为：

$$\theta_{t+1} = \theta_t - \frac{J'(\theta_t)}{J''(\theta_t)} \tag{3-13}$$

其中 $t$ 为迭代次数。如果考虑$\boldsymbol{\theta} \in \mathbf{R}^{m+1}$，则牛顿法表示为如下公式：

$$\theta_{t+1} = \theta_t - \frac{V[J(\theta_t)]}{H[J(\theta_t)]} \tag{3-14}$$

其中 $V[J(\boldsymbol{\theta})]$对应单变量一阶导数是多变量的一阶偏导数，定义为梯度向量：

$$V[J(\boldsymbol{\theta})] = \begin{bmatrix} \dfrac{\partial J(\boldsymbol{\theta})}{\partial \theta_0} \\ \dfrac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \dfrac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \dfrac{\partial J(\boldsymbol{\theta})}{\partial \theta_m} \end{bmatrix}_{(m+1)\times 1} \tag{3-15}$$

其中 $H[J(\boldsymbol{\theta})]$对应单变量二阶导数是多变量的二阶偏导数，定义为海森矩阵：

$$H[J(\boldsymbol{\theta})] = \begin{bmatrix} \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_0 \partial \theta_0} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_0 \partial \theta_1} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_0 \partial \theta_2} & \cdots & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_0 \partial \theta_m} \\ \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_0} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_1} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2} & \cdots & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_m} \\ \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_0} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_1} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_2} & \cdots & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_m} \\ \vdots & \vdots & \vdots & & \vdots \\ \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_m \partial \theta_0} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_m \partial \theta_1} & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_m \partial \theta_2} & \cdots & \dfrac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_m \partial \theta_m} \end{bmatrix}_{(m+1)\times(m+1)} \tag{3-16}$$

所以牛顿公式常常也写为：

$$\theta_{t+1} = \theta_t - H^{-1}(J(\theta_t))\nabla J(\theta_t) \tag{3-17}$$

其中$-H^{-1}(J(\theta_t))\nabla J(\theta_t)$称为牛顿方向，当海森矩阵正定时，可以保证牛顿搜索方向下降。

尽管牛顿法为二阶收敛，并且可以快速收敛，但是牛顿法因引入海森矩阵而增加了复杂性，当矩阵维度过大时，求解海森矩阵的逆矩阵会带来巨大的计算量。如果海森矩阵不可逆，则牛顿算法无解。再如果海森矩阵不是正定矩阵，也就是函数不是严格的凸函数，也可能会导致算法无法收敛。如果初值选择偏离极值点太远，也会导致算法无法收敛，也就是说，基本牛顿法并不是全局优化算法。

牛顿法与梯度下降法优缺点对比如表 3.1 所示。

表 3.1 牛顿法与梯度下降法对比

| 梯度下降法 | 牛 顿 法 |
|---|---|
| 更简单 | 较复杂 |
| 需要设置参数，如学习率等 | 不需要设置参数 |
| 更多迭代次数 | 较少迭代次数 |
| 每次迭代的计算成本较低，复杂度为 $O(m)$，其中 $m$ 为样本特征数 | 每次迭代计算成本很高，复杂度为 $O(m^3)$，其中 $m$ 为样本特征数 |
| 当 $m$ 较大时推荐使用，推荐 $m>10\,000$ 时使用梯度下降法较为合理 | 当 $m$ 较小时推荐使用，推荐 $m<1000$ 时，计算海森矩阵比较容易，使用牛顿法较为合理 |

大部分的机器学习算法的本质是建立优化模型，通过优化方法对目标函数进行优化，从而训练出最好的模型，除了常见的梯度下降法和牛顿法，还有改进梯度下降法的随机梯度下降法（Stochastic Gradient Descent，SGD）、批梯度下降法（Batch Gradient Descent，BGD）、改进牛顿法的拟牛顿法（Quasi-Newton Methods）。还有根据人类在解决问题所采取的经验规则而提出的启发式优化方法，例如基于物理中固体物质的冷却过程，提出组合优化的模拟退火（Simulated Annealing，SA）算法，基于自然界遗传规律的仿生算法，提出解决复杂非线性优化问题的遗传算法（Genetic Algorithm，GA），新的近似 GA 的差分进化算法（Differential Evolution Algorithm，DE），仿生种群觅食的寻优算法如粒子群（Particle Swarm Optimization，PSO）算法和人工蜂群（Artificial Bee Colony，ABC）算法等，具体可参考后面相关章节。

视频讲解

## 3.4 逻辑回归解决分类问题

### 3.4.1 实例一：牛顿法实现逻辑回归模型

#### 1. 问题描述

1）数据

假设一个高中生的数据集，其中 40 个学生被大学接收，40 个被大学拒绝。每个训练样本可以表示为$(x_i, y_i)$，其中包含一个学生在两次标准化考试中的分数，一个是否被大学接收的标记。需要解决的问题是：建立一个二分类模型，基于学生两次考试分数来预测被大学录取的概率。在训练数据中：

（1）矩阵 $\boldsymbol{x}$，第一列是所有学生第一次考试分数，第二列是所有学生第二次考试分数；

（2）向量 $\boldsymbol{y}$，用 1 标记被大学录取的学生，用 0 标记被大学拒绝的学生。

数据集文件 ex341x.dat 和 ex341y.dat 见本书配套的数据和源码清单。

2）画图

导入训练数据，在矩阵 $\boldsymbol{x}$ 中加入常数项 $x_0=1$。在开始使用牛顿法之前，首先用不同的符号表示两类数据画出数据图形。在 MATLAB/Octave 中，可以用以下命令实现正和负样本的分离：

```
% 找到并返回标签 1 和 0 的行和列
pos = find(y == 1);
```

```
neg = find(y == 0);
% 特征在变量 x 的第 2 列和第 3 列
plot(x(pos, 2), x(pos,3), 'r + ');
hold on;
plot(x(neg, 2), x(neg, 3), 'bo');
```

运行程序可以得到如图 3.2 所示图形。

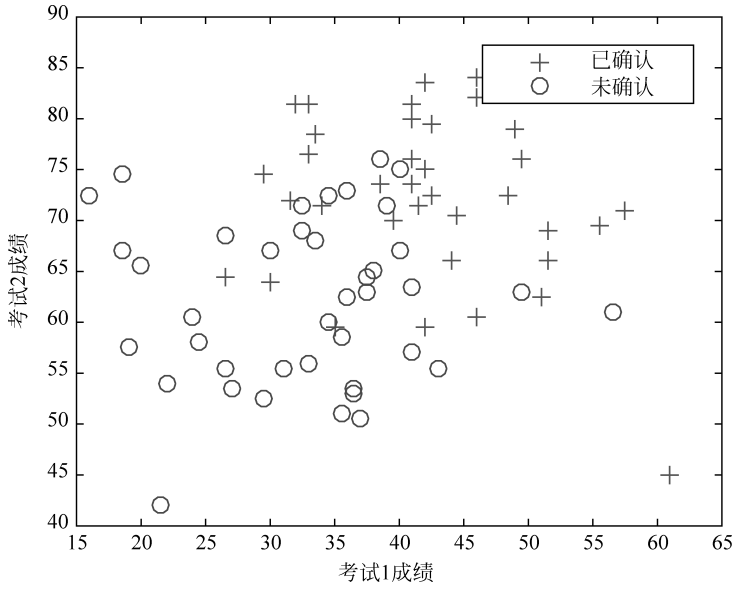



**图 3.2　训练数据图**

3）牛顿法

回顾逻辑回归，假设函数为

$$H_{\boldsymbol{\theta}}(\boldsymbol{x})=g(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x})=P(y=1\mid \boldsymbol{x};\boldsymbol{\theta})$$

其中 $g(\cdot)$ 是激活函数，常见的选择 Sigmoid 函数。如果 MATLAB/Octave 中，没有 Sigmoid 库函数，而 S 函数的数学表达式为：

$$g(z)=\frac{1}{1+\mathrm{e}^{-z}}$$

所以，最简单的方式是通过以下内联函数来实现：

```
% 定义 Sigmoid 函数
g = inline('1.0 ./ (1.0 + exp( - z))');
```

回顾代价函数 $J(\boldsymbol{\theta})$ 的定义：

$$J(\boldsymbol{\theta})=-\frac{1}{n}\left[\sum_{i=1}^{n}y_i\log H_{\boldsymbol{\theta}}(x_i)+(1-y_i)\log(1-H_{\boldsymbol{\theta}}(x_i))\right]$$

采用牛顿法来最小化代价函数。回归牛顿法的更新规则如下：

$$\theta_{t+1}=\theta_t-H^{-1}(J(\theta_t))\nabla J(\theta_t)$$

在逻辑回归中，梯度向量和海森矩阵如下：

$$\nabla J(\boldsymbol{\theta})=\frac{1}{n}\sum_{i=1}^{n}(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)-y_i)\boldsymbol{x}_i$$

$$H(\boldsymbol{\theta})=\frac{1}{n}\sum_{i=1}^{n}[(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)(1-(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\boldsymbol{x}_i(\boldsymbol{x}_i)^{\mathrm{T}}]$$

注意：以上公式是向量版本的公式。也就明确意味着，当 $H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ 和 $y_i$ 是一个实数表示的标量时，其中 $\boldsymbol{x}_i\in\mathbf{R}^{m+1}$，$\boldsymbol{x}_i(\boldsymbol{x}_i)^{\mathrm{T}}\in\mathbf{R}^{(m+1)\times(m+1)}$。

4）实现分类

现在开始采用代码来实现牛顿法，首先初始化 $\boldsymbol{\theta}=\vec{0}$。为确定需要多少次迭代，可以计算每个迭代的 $J(\boldsymbol{\theta})$，同时画出结果图，一般可以在 5～15 次迭代之后收敛。如果需要更多次迭代，请检查实现代码是否有错误。在算法收敛之后，用 $\boldsymbol{\theta}$ 找到分类问题的分界线。分界线被定义为如下直线：

$$P(y=1\mid\boldsymbol{x};\boldsymbol{\theta})=g(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x})=0.5$$

画出分界线就等于画出 $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}=0$ 的直线。完成后，可以得到如图 3.3 所示的图形。

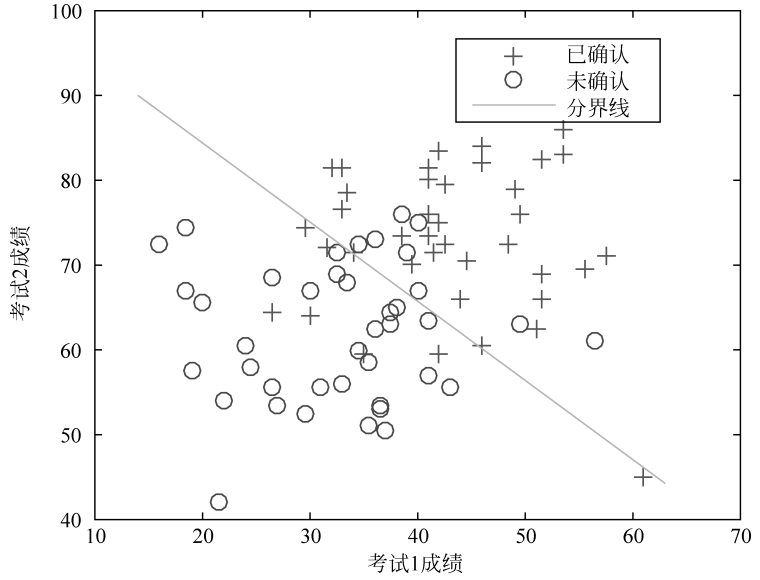



**图 3.3　分界线图示**

5）问题

根据以上理论，请回答以下两个问题：

（1）得到的 $\boldsymbol{\theta}$ 值是多少？需要多少次迭代可以收敛？

（2）当一个学生在第一次考试中分数为 20，第二次考试中分数为 80，该学生被大学录取的概率是多少？

## 2. 实例分析参考解决方案

请参考以下解决方案，检查你的实现和答案是否正确。如果在相同的参数/函数描述的情况下，得到不一样的结果，请调试代码直到得到相同的结果。

源码 ex341.m 文件见本书配套的源码清单。

牛顿法的最终的 $\boldsymbol{\theta}$ 值应该为

$$\theta_0=-16.38\quad\theta_1=0.1483\quad\theta_2=0.1589$$

代价函数的图形显示类似图 3.4。

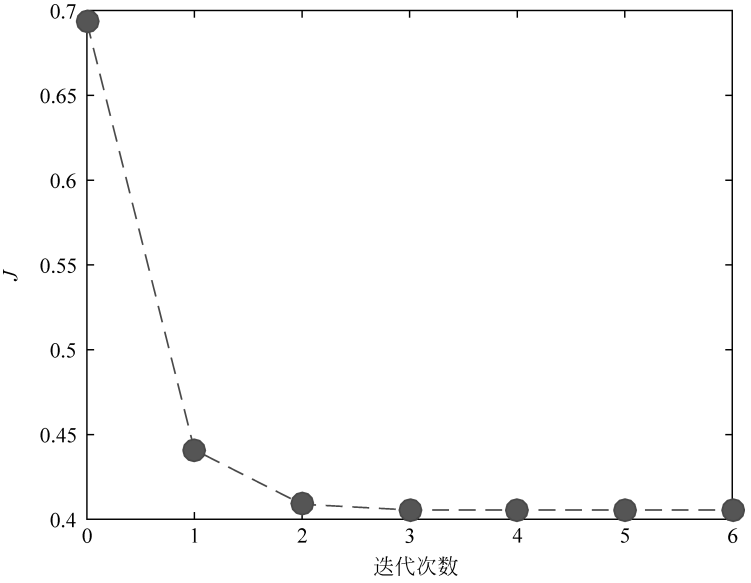



**图 3.4　代价函数曲线**

由图 3.4 可知，牛顿法在大约 5 次迭代后收敛。实际上，查看并打印输出 $J$ 的值，可以发现 $J$ 的值在第 4 次和第 5 次迭代之间就已经小于 $10^{-7}$。通常梯度下降需要几百次甚至几千次迭代才可以收敛。与之相比，牛顿法速度要更快一些。

当一个学生在第一次考试中分数为 20，第二次考试中分数为 80，预测该学生被大学拒绝的概率是 0.668。

## 3.4.2　实例二：逻辑回归解决二分类问题

### 1. 问题描述

1）数据

假设某班级有 20 名学生，采集学生复习课程的时长（单位为小时），学生复习效率，对应学生考试成绩为通过标记为 1，不通过标记为 0，数据如表 3.2 所示。

**表 3.2　某班级学生复习与考试结果数据**

| 样　本 | 时　长 | 效　率 | 结　果 | 样　本 | 时　长 | 效　率 | 结　果 |
|---|---|---|---|---|---|---|---|
| Student 0# | 1 | 0.1 | 0 | Student 10# | 7 | 0.9 | 1 |
| Student 1# | 2 | 0.9 | 0 | Student 11# | 8 | 0.1 | 0 |
| Student 2# | 2 | 0.4 | 0 | Student 12# | 8 | 0.6 | 1 |
| Student 3# | 4 | 0.9 | 1 | Student 13# | 8 | 0.8 | 1 |
| Student 4# | 5 | 0.4 | 0 | Student 14# | 3 | 0.9 | 0 |
| Student 5# | 6 | 0.4 | 0 | Student 15# | 8 | 0.5 | 1 |
| Student 6# | 6 | 0.8 | 1 | Student 16# | 7 | 0.2 | 0 |
| Student 7# | 6 | 0.7 | 1 | Student 17# | 4 | 0.5 | 0 |
| Student 8# | 7 | 0.2 | 0 | Student 18# | 4 | 0.7 | 1 |
| Student 9# | 7 | 0.8 | 1 | Student 19# | 2 | 0.9 | 1 |

2）任务

对学生复习与考试结果数据集，建立逻辑回归模型，实现分类，基于 Python 语言机器学习库 Sklearn 的逻辑回归函数 LogisticRegression 实现，返回模型精度和参数，并显示混淆矩阵。

### 2. 实例分析参考解决方案

Python 实现逻辑回归分类源码清单如下：

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix
from prettytable import PrettyTable
import numpy as np
X = np.matrix('1 0.1;2 0.9;2 0.4;4 0.9;5 0.4;\
               6 0.4;6 0.8;6 0.7;7 0.2;7 0.8;\
               7 0.9;8 0.1;8 0.6;8 0.8;3 0.9;\
               8 0.5;7 0.2;4 0.5;4 0.7;2 0.9')
y_true = np.matrix('0; 0; 0; 1; 0;\
                    0; 1; 1; 0; 1;\
                    1; 0; 1; 1; 0;\
                    1; 0; 0; 1; 1')
#导入数据,并设置训练和测试数据比例为 8:2
X_train, X_test, y_train, y_test = train_test_split(X, np.ravel(y_true),test_size = 0.2)
reg = LogisticRegression(C = 1e5, solver = 'lbfgs')
#训练
reg.fit(X_train,y_train)
#测试
y_pred = reg.predict(X_test)
print('test accuracy:\n',accuracy_score(y_test,y_pred))     #打印模型精度
print('weights:\n', reg.coef_, '\nbias:\n', reg.intercept_) #打印模型参数
pre = reg.predict(X)
cm = confusion_matrix(y_true, pre)
print("confusion_matrix:")                                  #打印混淆矩阵
cm_table = PrettyTable(["","predict: 0 class", "predict: 1 class"])
cm_table.add_row(["true: 0 class",cm[0,0], cm[0,1]])
cm_table.add_row(["true: 1 class",cm[1,0], cm[1,1]])
print(cm_table)
```

运行以上程序输出结果为：

```
test accuracy:
 0.75
weights:
 [[ 1.27608256 14.39485034]]
bias:
 [-15.15313307]
confusion_matrix:
+---------------+------------------+------------------+
|               | predict: 0 class | predict: 1 class |
+---------------+------------------+------------------+
| true: 0 class |        8         |        2         |
| true: 1 class |        0         |        10        |
+---------------+------------------+------------------+
```

从程序运行结果可知，逻辑回归模型的测试精度为 75%，模型可以表示为：

$$H(\boldsymbol{x}) = 1.276x_0 + 14.395x_1 - 15.153$$

从模型的混淆矩阵可知，共有 8 个标签为 0 的样本分类正确，有 10 个标签为 1 的样本分类正确，有 2 个标签为 0 的样本被错误分类为 1。

## 3.5　正则化

正则化(Regularization)是通过向机器学习的模型引入额外信息，从而防止过拟合(Overfitting)，提高模型的泛化能力。首先，对机器学习算法进行训练会出现 3 种结果。第一为欠拟合(Underfitting)，或者叫作高偏差(High bias)，也就是所建立的模型不能很好地拟合训练数据，一般出现在模型刚开始训练的阶段，需要通过不断地调整算法参数来提高模型对数据的表达能力。第二种为过拟合，或者叫作高方差(High variance)，也就是所建立的模型对训练数据的特征表现得太彻底，几乎拟合到了所有训练集中的数据，导致将数据中的噪声也当作特征来学习，一般出现在训练的最后阶段，或者训练数据集较小，而模型较为复杂的情况。当过拟合出现时，模型对新的样本的预测能力会很糟糕，也就是在测试集上精度严重下降。第三种就是想要保留下来的刚刚好(Just right)模型和模型参数，对训练数据的特征有很好的表达力，对测试数据有很好的适应能力，这样的模型就有很好的泛化能力。

当机器学习模型出现过拟合时，可以从数据集和模型两个角度来寻求解决方案。训练数据集样本量太小，可以增大数据集规模。样本特征过多，就要舍弃一部分冗余的特征，放弃对模型精度贡献很小的特征。从模型的角度，为解决过拟合问题，可以采用正则化的方法，对代价函数引入一个正则化项。

回顾线性回归模型的代价函数表达式为：

$$J(\boldsymbol{\theta})=\frac{1}{n}\sum_{j=0}^{n}[H_{\boldsymbol{\theta}}(x_j)-y_j]^2$$

对于引入正则项后的线性回归模型，其代价函数可以表示为：

$$J(\boldsymbol{\theta})=\frac{1}{2n}\left[\sum_{j=0}^{n}[H_{\boldsymbol{\theta}}(x_j)-y_j]^2+\lambda\sum_{j=1}^{m}(\theta_j)^2\right]$$

其中 λ 为正则化参数，可以控制在训练模型和保持参数值较小间达到较好的平衡，保证对训练数据集的拟合模型形式相对简单，从而较好地避免过拟合。对线性回归的正则化后的代价函数，也同样可以采用梯度下降法和最小二乘正规化法优化。

回顾逻辑回归模型的代价函数表达式为：

$$J(\boldsymbol{\theta})=-\frac{1}{n}\left[\sum_{i=1}^{n}y_i\log H_{\boldsymbol{\theta}}(x_i)+(1-y_i)\log(1-H_{\boldsymbol{\theta}}(x_i))\right]$$

对于引入正则项后的逻辑回归模型，其代价函数可以表示为：

$$J(\boldsymbol{\theta})=-\frac{1}{n}\left[\sum_{i=1}^{n}y_i\log H_{\boldsymbol{\theta}}(x_i)+(1-y_i)\log(1-H_{\boldsymbol{\theta}}(x_i))\right]+\frac{\lambda}{2n}\sum_{j=1}^{m}(\theta_j)^2 \quad (3\text{-}18)$$

当然，对逻辑回归的正则化后的代价函数，同样可以采用梯度下降法和牛顿法优化。

## 3.6　正则化后的线性回归和逻辑回归模型实例分析

本节通过实例来分析引入正则项后的线性回归和逻辑回归解决回归和分类问题。其中问题描述为：

1）数据

数据包含两个数据集：一个 ex361x. dat 和 ex361y. dat 用于线性回归，另一个是 ex362x. dat 和 ex362y. dat 用于逻辑回归。

2）画图

数据对应于程序中需要处理的变量 $x$ 和 $y$，其中输入 $x$ 是一个单特征，所以可以画出标签 $y$ 关于 $x$ 的二维图，请参考本书配套的源码自己动手，在 MATLAB/Octave 中编写代码画出数据图，画出的数据图如图 3.5 所示。

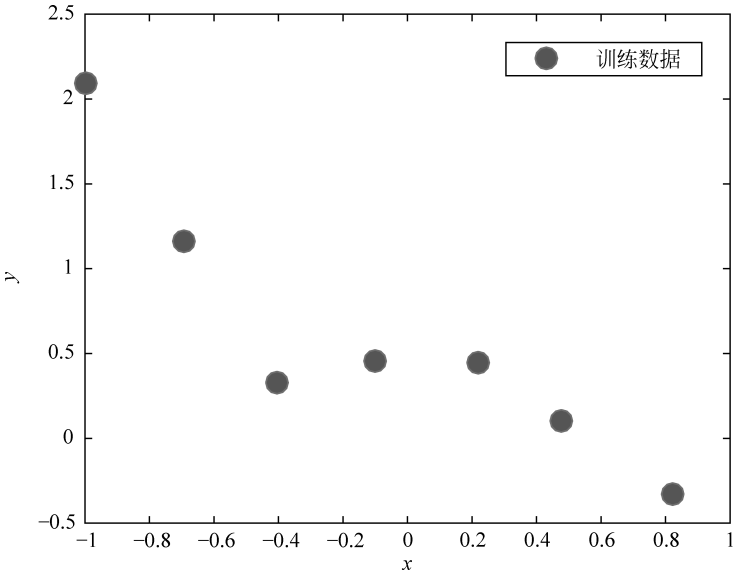



**图 3.5　线性回归函数图**

从图 3.5 中可知，如果用一个直线来逼近数据似乎过于简单。因此，可以尝试一个高阶的多项式来拟合数据，从而更好地表现各个数据点的变化。

可以尝试一个五阶的多项式，表示为：

$$H_{\boldsymbol{\theta}}(x)=\theta_0+\theta_1 x+\theta_2 x^2+\theta_3 x^3+\theta_4 x^4+\theta_5 x^5$$

这意味着一个 6 个特征的假设多项式模型，因为$(x^0,x^1,x^2,x^3,x^4,x^5)$是回归的所有特征。注意尽管用多项式逼近数据，但是依然在讨论线性回归问题，原因在于假设函数对于每一个特征都是线性相关的。

用五阶的多项式逼近一个仅有 7 个点的数据集，很可能出现过拟合。为防止过拟合出现，对模型要进行正则化。

回顾正则化问题，正则化目的就是，如下公式所表示的代价函数关于$\boldsymbol{\theta}$获得最小值：

$$\text{Min}: J(\boldsymbol{\theta})=\frac{1}{2n}\left[\sum_{j=0}^{n}[H_{\boldsymbol{\theta}}(x_j)-y_j]^2+\lambda\sum_{j=1}^{m}(\theta_j)^2\right]$$

其中 λ 是正则化参数，是控制拟合的参数。当拟合参数的数量(Magnitudes)增加时，代价函数的正则化惩罚力度将随之增加。这个惩罚同时依赖于参数的平方和 λ 的大小。注意，此处正则化求和的参数并不包含 $\theta_0^2$。

### 3.6.1　实例一：最小二乘正规方程法优化正则化线性回归模型


视频讲解

对于线性回归模型的代价函数最小化，可以采用梯度下降法，也可以采用最小二乘正规方程法，由于训练集规模较小，所以在这个实例的分析中，采用正规方程来求解正则化后的代价函数。

回顾线性回归模型的最小二乘正规方程表达式为：

$$\boldsymbol{\theta}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}$$

对于正则化的代价函数，其正规方程表示如下：

$$\boldsymbol{\theta}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{A})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}$$

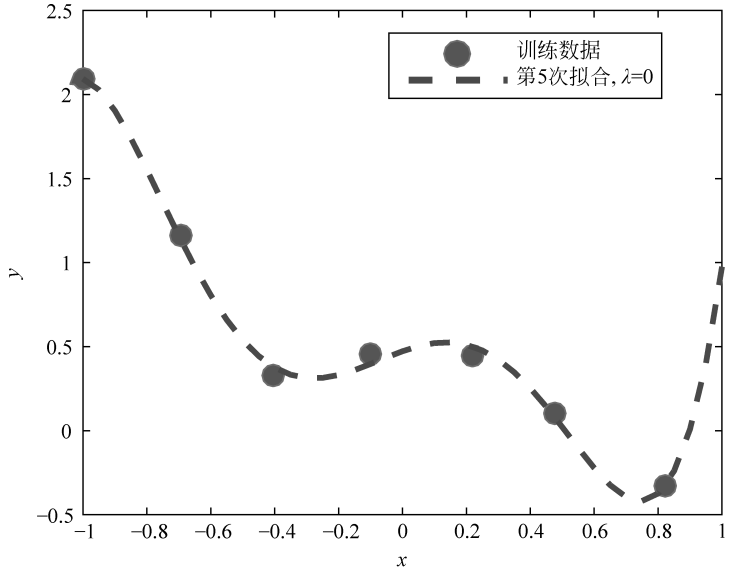
$$\boldsymbol{A}=\begin{bmatrix}0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1\end{bmatrix}_{(m+1)\times(m+1)}$$

其中正则化单位矩阵 $\boldsymbol{A}$ 是一个$(m+1)\times(m+1)$的对角阵，对角线上从左上到右下，最左上一个元素为0，其他元素均为1。注意，这里 $m$ 为特征数，不包含常数项，输入矩阵 $\boldsymbol{X}$ 和标签向量 $\boldsymbol{Y}$ 定义保持不变。其中对于输入矩阵的实现代码为：

```
x = [ones(n, 1), x, x.^2, x.^3, x.^4, x.^5];
```

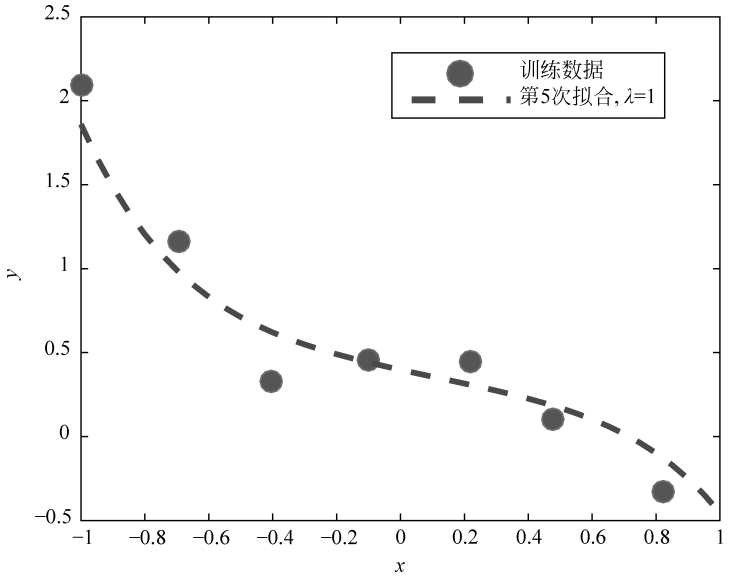
可以获得一个 $n\times(m+1)$的输入矩阵 $\boldsymbol{X}$，其中包含 $n$ 个训练样本；$m$ 个特征；一个常数项。因为本实例数据集中，仅提供了特征的一次项，其他的高次特征需要通过以上代码计算获得。

接下来对不同的正则化参数 $\lambda$ 取值，例如 $\lambda=0$，$\lambda=1$，$\lambda=10$ 这3种情况，根据最小二乘正规方程求解 $\boldsymbol{\theta}$。当找到合适的 $\boldsymbol{\theta}$ 时，可以对照解决方案中的值来检测答案。然后参考源码清单中的代码，自己动手画出对应每一个 $\lambda$ 值的多项式拟合结果，应该获得类似如图3.6所示的图形。观察图3.6，总结正则化参数 $\lambda$ 是怎么影响你的模型的。

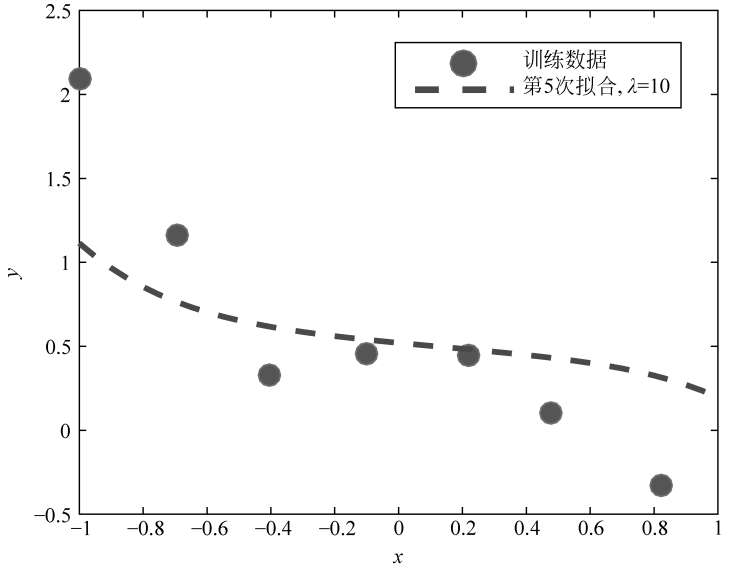



(a) $\lambda$=0

**图3.6　不同的正则化参数模型对比**




(b) $\lambda$=1




(c) $\lambda$=10

**图 3.6** （续）

## 3.6.2 实例二：牛顿法优化正则化逻辑回归模型

在实例分析的第二部分，将采用牛顿法来优化引入正则化的逻辑回归模型。首先，导入逻辑回归训练数据集，其中包含两个特征。为避免与上面的线性回归混淆，假设两个输入数据 $\boldsymbol{x}$ 包含两个特征，分别表示为 $u$ 和 $v$，标签表示为 $y$。由于为二分类问题，所以 $y$ 的值为 1 或者 0。将一个特征 $u$ 作为横轴变量，另一个特征 $v$ 作为纵轴变量，对样本标签为 1 的标记为正样本，为 0 的标记为负样本，自己动手在 MATLAB/Octave 中编写代码：

```
x = load('ex362x.dat');
```

```
y = load('ex362y.dat');
figure;pos = find(y == 1);
neg = find(y == 0);
plot(x(pos,1), x(pos,2), '+');
hold on;
plot(x(neg,1), x(neg, 2), 'o');
```

运行以上代码，可以得到如图 3.7 所示的数据图。

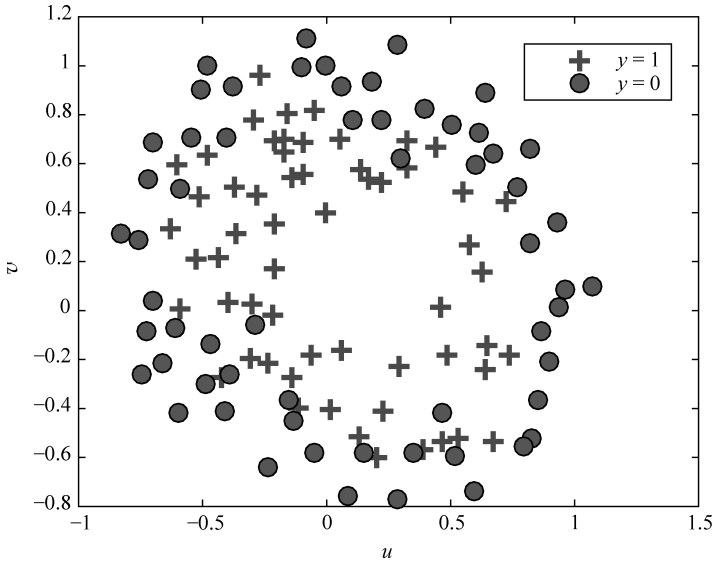



**图 3.7　逻辑回归数据图**

由图 3.7 可知，数据呈现内外两层同心分布。现在开始对这个数据集建立正则化的逻辑回归模型实现分类。首先，设计输入矩阵。指定特征 $u$ 和 $v$ 的任意单项式，也就是包含在多项式中的某一项，构成六阶输入特征因子，如式(3-19)所示。其中 $\boldsymbol{x}$ 是一个有 28 个特征的向量，如式(3-20)所示。

$$\boldsymbol{x}=\begin{bmatrix}1\\u\\v\\u^2\\uv\\v^2\\u^3\\u^2v\\uv^2\\v^3\\u^4\\u^3v\\u^2v^2\\uv^3\\v^4\\\vdots\\v^6\end{bmatrix} \tag{3-19}$$

$$(x_0=1,x_1=u,x_2=v,x_3=u^2,x_4=uv,x_5=v^2,\cdots,x_{28}=v^6) \tag{3-20}$$

注意：输入逻辑回归模型的特征量不是 $u$ 和 $v$，而是 $x_0,x_1,x_2,\cdots$。

为避免编写枚举 $\boldsymbol{x}$ 的各项代码的困难，可以直接调用源码清单中的一个函数，用于映射原始输入数据到特征向量的实现，代码文件名为 map_feature.m。

参考代码所实现的函数可以转换单个训练样本，也可以转换整个训练数据集，只要将该文件复制到工作目录，并采用以下命令调用函数：

```
x = map_feature(u, v);
```

函数返回转换后的输入特征向量。代码实现中假设原始特征数据被存储在列向量 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 中，所以，如果仅有一个训练样本输入，每个列向量将是一个标量，也就是变量。

注意，在使用这个函数转换数据时，要确保输入是相同长度的两个列向量。

在建立模型之前，要明确优化的目标是让引入正则项的逻辑回归模型的代价函数最小化，回顾代价函数表达式为：

$$\text{Min}:J(\boldsymbol{\theta})=-\frac{1}{n}\left[\sum_{i=1}^{n}y_i\log H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)+(1-y_i)\log(1-H_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\right]+\frac{\lambda}{2n}\sum_{j=1}^{m}(\theta_j)^2$$

该代价函数与未正则化的逻辑回归模型的代价函数相似，除了在末尾添加了一个正则化项，所以依然可以采用梯度下降法或者牛顿法来找到以上函数的最小化参数组合。

回顾牛顿法的更新规则：

$$\theta_{t+1}=\theta_t-H^{-1}(J(\theta_t))\nabla J(\theta_t)$$

由于引入了正则化项，所以梯度向量和海森矩阵的形式变为：

$$\nabla J(\theta)=\begin{bmatrix}\frac{1}{n}\sum_{i=1}^{n}(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)-y_i)(\boldsymbol{x}_0)_i\\ \frac{1}{n}\sum_{i=1}^{n}(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)-y_i)(\boldsymbol{x}_1)_i+\frac{\lambda}{n}\theta_1\\ \frac{1}{n}\sum_{i=1}^{n}(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)-y_i)(\boldsymbol{x}_2)_i+\frac{\lambda}{n}\theta_2\\ \vdots\\ \frac{1}{n}\sum_{i=1}^{n}(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)-y_i)(\boldsymbol{x}_m)_i+\frac{\lambda}{n}\theta_m\end{bmatrix}$$

$$H(\boldsymbol{\theta})=\frac{1}{n}\sum_{i=1}^{n}[(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i)(1-(H_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\boldsymbol{x}_i(\boldsymbol{x}_i)^{\mathrm{T}}]+\frac{\lambda}{n}\begin{bmatrix}0&&&\\&1&&\\&&\ddots&\\&&&1\end{bmatrix}_{(m+1)\times(m+1)}$$

以上计算中，如果正则化参数 $\lambda=0$，则可以获得与未正则化之前的梯度向量和海森矩阵相同的公式。公式中的 $\boldsymbol{x}_i$ 是一个 $(m+1)\times1$ 的向量，在本实例中是一个 $28\times1$ 的向量，所以求得的梯度向量 $\nabla J(\boldsymbol{\theta})$ 也是一个 $28\times1$ 的向量。

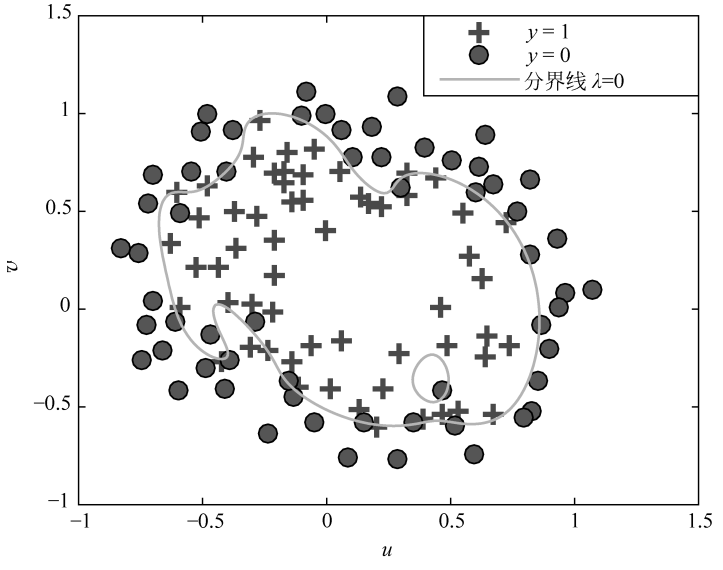
其中 $\boldsymbol{x}_i(\boldsymbol{x}_i)^{\mathrm{T}}$ 和海森矩阵的维度相同，均为 $(m+1)\times(m+1)$，本实例中，是 $28\times28$ 的矩阵。其中 $y_i$ 和 $H_\theta(\boldsymbol{x}_i)$ 都是标量，也就是单变量。

在海森公式中的正则化单位矩阵是一个 $28\times28$ 的对角阵，其中除了最左上元素为 0，对角线上元素从左上到右下，均为 1。接下来对不同的正则化参数 $\lambda$ 取值，例如 $\lambda=0$，$\lambda=1$，

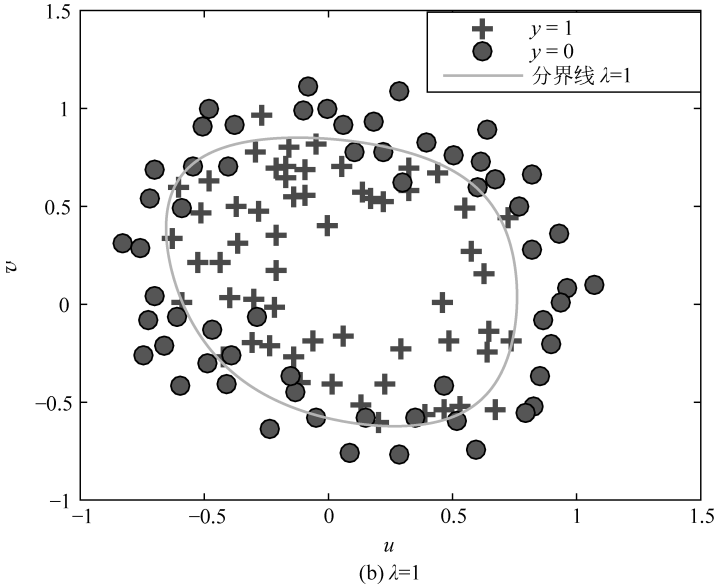
$\lambda=10$ 这 3 种情况，运行牛顿法来求解$\boldsymbol{\theta}$。为确定牛顿法是否收敛，可以在每一步迭代中输出代价函数 $J(\boldsymbol{\theta})$的值，观察图形，是否在迭代最终的若干点上，代价函数应该不会再减小，如果其值减小了，就需要检查是否代价函数的定义正确，同时检查梯度向量和海森矩阵的定义是否正确，确保正则化的部分没有错误。如果牛顿法收敛，可以用获得的$\boldsymbol{\theta}$去求解分类问题的分界线，分界线的定义为：

$$P(y=1 \mid \boldsymbol{x};\boldsymbol{\theta})=0.5 \Rightarrow \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}=0$$

画出分界线相对于画出线性回归的最佳拟合曲线更为困难，需要以画轮廓线的形式，先画出隐含的直线$\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}=0$。具体实现是通过在网格化后的原始输入数据图，也就是 $u$ 为横轴和 $v$ 为纵轴的二维图上，评估$\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$后，画出$\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$为 0 的曲线，在 MATLAB/Octave 上画出的图形如图 3.8 所示。为获得最佳呈现效果，可以采用图中所示的坐标量程范围。

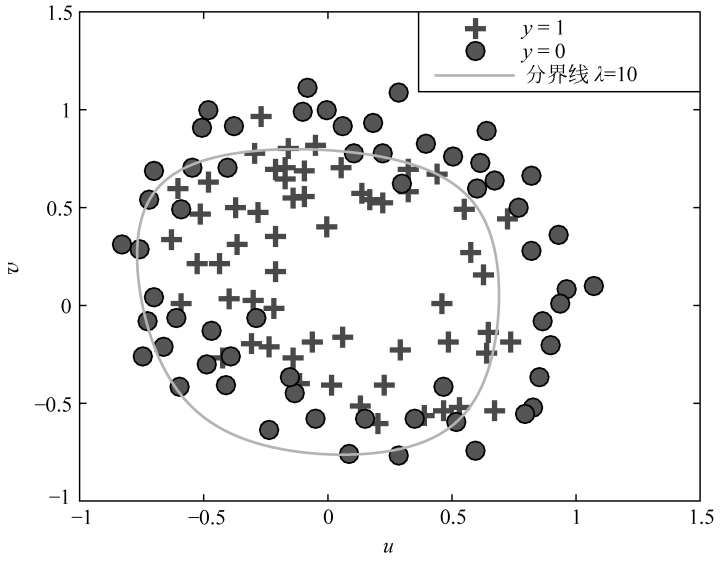



(a) $\lambda=0$




(b) $\lambda=1$

**图 3.8　不同的正则化参数模型对比**




(c) $\lambda$=10

**图 3.8** （续）

调用 map_feature 函数，具体实现代码为：

```
u = linspace( - 1, 1.5, 200);
v = linspace( - 1, 1.5, 200);
z = zeros(length(u), length(v));
for i = 1:length(u)
    for j = 1:length(v)
        z(i,j) = map_feature(u(i), v(j)) * theta;
    end
end
```

画分界线：

```
% 在调用 contour 命令显示图形前，对 z 执行转置操作
z = z';
contour(u, v, z, [0, 0], 'LineWidth', 1)
```

调整 λ 的值，运行代码，对于 3 个不同的 λ 值可以得到如图 3.8 所示的结果。

最后，因为 $\theta$ 有 28 个元素，解决方案中将不提供每个元素的一一对应的对比结果。MATLAB/Octave 中，可以采用 norm(theta)命令计算 $\theta$ 的 L2 正则化结果，这样就可以参考解决方案中的标准来对比检测自己的实现结果是否正确。

### 3.6.3 参考解决方案

在完成以上实例分析后，请参考以下解决方案，检查执行结果是否正确。如果在选择与下述相同的参数/函数的基础上，得到不同的结果，调试你的方案直到得到解决方案中给出的结果。正则化的线性回归和逻辑回归实现代码，源码 ex361.m 和 ex362.m 文件见本书配套的代码清单。

**1. 实例一：最小二乘正规方程法优化正则化线性回归模型的运行结果（见表 3.3）**

**表 3.3　最小二乘正规方程法优化正则化线性回归模型的运行结果**

| 参　　数 | $\lambda=0$ | $\lambda=1$ | $\lambda=10$ |
|---|---|---|---|
| $\theta_0$ | 0.4725 | 0.3976 | 0.5205 |
| $\theta_1$ | 0.6814 | −0.4207 | −0.1825 |
| $\theta_2$ | −1.3801 | 0.1296 | 0.0606 |
| $\theta_3$ | −5.9777 | −0.3975 | −0.1482 |
| $\theta_4$ | 2.4417 | 0.1753 | 0.0743 |
| $\theta_5$ | 4.7371 | −0.3394 | −0.1280 |
| norm $\boldsymbol{\theta}$ | 8.1687 | 0.8098 | 0.5931 |

注意：当 $\lambda$ 增大时，$\boldsymbol{\theta}$ 的标准值将减小。这是因为一个较高的 $\lambda$ 意味着一个较大的拟合参数。通过调整 $\lambda$，可以更好地控制对数据的拟合程度。

在图 3.6(a)中，$\lambda=0$，意味着这个拟合是和未正则化的线性回归相同的。优化的目的是寻找最小的平方误差，这个曲线对这个数据集是有效的，但是也许曲线不能很好地表现数据的变化趋势，这就是过拟合。

在图 3.6(b)中，通过引入增加的正则化参数 $\lambda=1$，过拟合被很好地削弱了。虽然这个拟合函数依然是五阶多项式，但是与图 3.6(a)中的图形比较，曲线显得更简单。

在图 3.6(c)中，当 $\lambda$ 太大时，欠拟合出现，同时曲线也不能像此前一样很好地跟随数据点的变化趋势。

**2. 实例二：牛顿法优化正则化逻辑回归模型的运行结果（见表 3.4）**

以下为牛顿法收敛后的 $\boldsymbol{\theta}$ 的标准值。对于 $\lambda=0$，收敛需要 15 次迭代，对于 $\lambda=1$ 和 $\lambda=10$ 需要 5 次或者更少的迭代次数。

**表 3.4　牛顿法优化正则化逻辑回归模型的运行结果**

| 参　　数 | $\lambda=0$ | $\lambda=1$ | $\lambda=10$ |
|---|---|---|---|
| norm $\boldsymbol{\theta}$ | 7.1727e+03 | 4.2400 | 0.9384 |

注意：当 $\lambda$ 增大时，$\boldsymbol{\theta}$ 的标准值减小，在对应的图中可以看到很明显的拟合变化。

在图 3.8(a)中，算法试图找到一个在正和负样本之间非常精确的分界线。所以在大范围的 $y=1$ 区域中，出现了一个 $y=0$ 的孤岛区域。这样的分类结果过于精确，出现过拟合，并不是模型致力于找寻的有泛化能力的分类趋势。

对于图 3.8(b)中 $\lambda=1$ 的图形，显示了一个简单的分界线，相当好地分离了正点和负点。

而对于图 3.8(c)中 $\lambda=10$ 的图形，此时的 $\lambda$ 已经是一个相当大的正则化参数，所以分界线不能很好地跟随数据的变化趋势，尤其明显地表现在图中的左下角部分，出现了欠拟合。

总而言之，正则化是保证机器学习模型泛化能力的有效技术，目前有多种正则化方法，

如数据增强，L0、L1 正则化，L2 正则化，Dropout 参数和提前停止等。其中，数据增强例如可以通过对原始图像的旋转、裁剪、色彩空间变换等获得扩展训练数据集。而 L1 和 L2 正则化是最常用的方法，L1 用参数的绝对值总和来构建正则化项，L2 采用参数的平方总和来构建正则化项。Dropout 常见于神经网络模型训练中，暂时丢弃一部分神经元以及连接，通过随机概率丢弃神经元可以有效地防止过拟合。提前停止法，可以限制最小化代价函数所需要的迭代次数。一般迭代次数太少，容易欠拟合；而迭代次数太多，容易过拟合。提前停止法可以通过确定迭代次数解决这个问题。

## 3.7 习题

1. 正则化(Regularization)是一种行之有效的避免模型过拟合的方法，正则化作为模型代价函数的惩罚项，可以增强模型的泛化能力。正则化函数可以有多种选择，一般是模型复杂度的单调递增函数，模型越复杂正则化的值就越大，例如正则化项，可以是模型参数向量的范数，请简述 L0、L1 和 L2 范数正则化项，对逻辑回归模型的具体影响。

2. 逻辑回归是一种简单有效的分类模型，可以用于简单的二分类问题，也可扩展到多分类问题的求解中，请参考本章节学习中的逻辑回归 MATLAB/Octave 仿真代码，采用 C/C++或者 Python 语言实现逻辑回归模型，并可视无正则化和加入 L2 正则化后，逻辑回归模型的训练迭代曲线和测试曲线，讨论 L2 正则化对模型性能的影响。