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

实验题目： 逻辑回归

学号：1190301610
姓名：王家琪

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。

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。

三、设计思想（本程序中的用到的主要算法及数据结构）

逻辑回归（logistics regression）是一种分类算法。它的本质是：假设数据服从某个分布，然后利用最大似然法做参数估计。

3.1 logistic 分布

Logistic 分布的概率密度函数和分布函数如下所示：

$$F(x)=P(X\leqslant x)=\frac{1}{1+e^{-(x-\mu)/\gamma}}$$

$$f(x)=F'(x)=\frac{e^{-(x-\mu)/\gamma}}{\gamma(1+e^{-(x-\mu)/\gamma})^{2}}$$

我们常用的 sigmoid 函数是$\mu=0$，$\gamma=1$的特殊形式

3.2 二项 logistic 回归

二项 logistic 回归是一种分类模型，用条件概率$P(Y|X)$表示，这里 X 为随机变量取实数，随机变量 Y 为类别取 0 或 1。它的条件概率分布为：

$$P(Y=1|x)=\frac{\exp(\omega x+b)}{1+\exp(\omega x+b)}$$

$$P(Y=0|x)=\frac{1}{1+\exp(wx+b)}$$

其中，ω为权值向量，b为偏置，ωx为ω和x的内积 。

我们可以这么理解 logistic 回归模型。对于一般的线性模型，我们得到的结果是一个连续的实数，无法准确的进行分类。为了把它确定为某个离散的类别，

我们希望他映射到 0，1 之间，如果这个数偏向 0（比如$y<0.5$），那就是标记为 0 的这类，如果偏向 1（比如$y\geqslant 0.5$），就是标记为 1 的这类。

0 到 1 的映射函数选取 sigmoid 函数：$y=\frac{1}{1+e^{-x}}$。

我们将原本的线性模型 $y=wx+b$，变成了 $y=\frac{1}{1+e^{-(wx+b)}}$，$y\in(0,1)$。整理一下得到 $\ln\left(\frac{y}{1-y}\right)=wx+b$。

如果一个事件发生的概率为P，那么$\frac{P}{1-P}$可以理解为这个事件发生的**对率**，也就是该事件发生与不发生的比值。对于 logistic 模型而言，有 $\ln\left(\frac{P(Y=1|x)}{1-P(Y=1|x)}\right)=wx+b$，也就是说 logistic 模型实际上是计算 Y=1 的对率的线性函数。

3.3 模型参数估计

3.3.1 损失函数

线性回归的代价函数是最小二乘法：

$$J(\theta)=\frac{1}{2}\sum_{i=0}^{n}\left(h_\theta(x^{(i)})-y^{(i)}\right)^2$$

对于 logistic 模型，我们可以用极大似然法估计模型参数。设$P(Y=1|x)=\pi(x)$，$P(Y=0|x)=1-\pi(x)$，

似然函数为：$\prod_{i=1}^{n}[\pi(x_i)]^{y_i}[1-\pi(x_i)]^{1-y_i}$

为了方便计算，对似然函数取对数，得到对数似然函数为：

$$\begin{aligned}L(\omega)&=\sum_{i=0}^{n}y_i\ln\pi(x_i)+(1-y_i)\ln(1-\pi(x_i))\\&=\sum_{i=0}^{n}[y_i(wx_i)-\ln(1+\exp(wx_i))]\end{aligned}$$

求对数似然函数的最大值等于求 $-L(\omega)$ 的最小值，得到 logistic 二分类模型的损失函数为：

$$J(\omega)=-L(\omega)=\sum_{i=0}^{n}-y_i\ln\pi(x_i)-(1-y_i)\ln(1-\pi(x_i))$$

由于 logistic 的损失函数比较复杂，无法直接求出解析解，所以我们只能通过优化算法来求 ω，使得 $J(\omega)$ 最小，这类似于我们实验一多项式拟合的任务。

3.3.2 梯度下降法

我们利用梯度下降法优化参数。

在梯度下降开始之前，sigmoid 函数有一个很好的性质：$h'(x)=h(x)(1-h(x))$。

对 $J(w)$ 求导得到：$\dfrac{\partial J(w)}{w_j}=-\sum_{i=0}^{n}(y^{(i)}-h(x^{(i)}))x_j^{(i)}$

于是梯度下降法更新权重只要根据以下公式即可：

$$w_j=w_j-\eta\frac{\partial J(w)}{w_j}=w_j+\eta\sum_{i=0}^{n}(y^{(i)}-h(x^{(i)}))x_j^{(i)},(j=1,2,\ldots\ldots n)$$

3.3.3 惩罚项

为了更好的拟合数据，可以在损失函数里面加入对参数的惩罚项：

$$J(\omega)=\sum_{i=0}^{n}-y_i\ln\pi(x_i)-(1-y_i)\ln(1-\pi(x_i))+\lambda\|w\|^2$$

损失函数求导：

$$\frac{\partial J(w)}{w_j}=-\sum_{i=0}^{n}(y^{(i)}-h(x^{(i)}))x_j^{(i)}+\frac{\lambda}{2}\omega_j$$

梯度更新公式为：

$$w_j=w_j+\eta\sum_{i=0}^{n}(y^{(i)}-h(x^{(i)}))x_j^{(i)}-\eta\frac{\lambda}{2}w_j,(j=1,2,\ldots\ldots n)$$

3.3.4 牛顿法

牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用了函数 $f(x)$ 的泰勒级数的前面几项来寻找方程 $f(x)=0$ 的跟。牛顿法最大的特点在于它的收敛速度很快。

我们知道梯度下降法主要考虑的是损失函数的 一阶导数，计算公式为：

$$\frac{\partial J(w)}{w_j}=-\sum_{i=0}^{n}(y^{(i)}-h(x^{(i)}))x_j^{(i)}$$

牛顿下降法则应用了二阶泰勒展开，目的是最小化损失函数。

将$J(\theta^{t+1})$在θ^t处泰勒展开为：$J(\theta^{t+1})=J(\theta^t)+J'(\theta^t)\Delta\theta+\frac{1}{2!}\Delta\theta^2 J''(\theta^t)$

可将一阶和二阶导数分别记为g和h，要使得$J(\theta^{t+1})$极小，也就是让$g\Delta\theta+\frac{\Delta\theta^2}{2}h$极小，可令$\frac{\partial\left(g\Delta\theta+\frac{\Delta\theta^2}{2}h\right)}{\partial\Delta\theta}=0$，求得$\Delta\theta=-\frac{g}{h}$，故让$\theta^{t+1}=\theta^t+\Delta\theta=\theta^t-\frac{g}{h}$，推广到向量的形式得到的迭代公式为：

$\theta^{t+1}=\theta^t-H^{-1}g$，$H$为海森矩阵，$g$为雅各比矩阵。

具体的，损失函数的二阶导数公式为：

$$\frac{\partial^2 J(w)}{w_i w_j}=\sum_{k=1}^{n} x_i x_j (h(x_k)(1-h(x_k)))$$

Hessian 矩阵为：

$$H(\boldsymbol{x})=\begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n^2} \end{bmatrix}$$

我们更新梯度的时候，需要求得 Hessian 矩阵的逆，并逐步迭代：

$w=w-\alpha H(w)^{-1}\nabla J(w)$，　α为学习率，一般取 0.01 左右

3.4 具体算法实现

3.4.1 高斯分布生成数据

首先利用高斯分布生成两个类别的数据，两组数据均不符合贝叶斯假设，即随机变量相互独立的假设。一组数据的均值为[2,3],另一组为[7,8],两组数据的协方差为[[2,1],[1,2]]，将生成的数据存储到‘exp.txt’文件里面备用，生成代码如下：

```
def generateData():
    f=open('exp.txt','w')
    mean=[2,3]#均值 1
    cov=np.mat([[2,1],[1,2]])
    x=np.random.multivariate_normal(mean,cov,100)
    for i in range(len(x)):
```

```python
        line = []
        line.append(x[i][0])
        line.append(x[i][1])
        line.append(1)
        line = ",".join(str(i) for i in line)
        line = line + "\n"
        f.write(line)
    mean1 = [7, 8]#均值 2
    X = np.random.multivariate_normal(mean1, cov, 100)
    for i in range(len(X)):
        line = []
        line.append(X[i][0])
        line.append(X[i][1])
        line.append(0)
        line = ",".join(str(i) for i in line)
        line = line + "\n"
        f.write(line)
    f.close()
```

3.4.2 pandas 读取文件

利用 read_csv 函数读取 txt 文件，注意该函数读取文件是用逗号分割的。

```python
def datadeal(path):
    data=pd.read_csv(path,header=None,names=['Exam 1','Exam 2','Admitted'])
    return data
```

3.4.3 可视化结果

利用 matplotlib.pyplot 的 subplots 函数将数据和优化得到的参数结果可视化处理。注意设置 x 和 y 坐标的范围便于观察。

```python
def drawline(data,result):
    plotting_x1 = np.linspace(0, 100, 100)
    #画出预测直线
    plotting_h1 = (- result[0] - result[1] * plotting_x1) / result[2]
    #画出正反类的散点图
    positive = data[data['Admitted'].isin([1])]
    negative = data[data['Admitted'].isin([0])]
    fig, ax = plt.subplots()
    ax.plot(plotting_x1, plotting_h1, 'y', label='Prediction')
```

```
ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='o', label='Admitted')
ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x', label='Not Admitted')
ax.legend()
ax.set_xlabel('Exam 1 Score')
ax.set_ylabel('Exam 2 Score')
plt.show()
```

3.4.4 梯度下降

梯度下降的原理已经在上文详细解释过了，代码见附录。需要指出的是本人在做实验的过程中，通过参数调试，发现梯度下降法有很多问题：

1、梯度下降在靠近最优解的时候**更新缓慢**。

2、参数设置不当容易成**“之”字型下降**，下降效果不好。

3、在迭代次数较少的时候（比如几千次），很难看到效果，通常要迭代几万次、几十万次才可以有比较漂亮的结果。

3.4.5 牛顿法

牛顿法的原理已经在上文详细解释过了，代码见附录。需要指出的是本人在做实验的过程中，通过参数调试，发现牛顿法有很多由于梯度下降的地方：

1、牛顿法**收敛速度很快**，在几千次就可以得到很好效果。

2、参数设置范围比较大。

3、牛顿法在更新参数的时候，由于要**计算 hessian 矩阵的逆**，所以运算速度比梯度下降法要慢。

3.5 梯度下降法和牛顿法优缺点对比

从本质上看，梯度下降法是一阶收敛的，牛顿法是二阶收敛的，所以牛顿法互殴更快。根据 wiki 百科上的解释，从几何上说，牛顿法就是用一个二次曲面去拟合你当前所处位置的局部曲面，而梯度下降法是用一个平面去拟合当前的局部曲面，通常情况下，二次曲面的拟合会比平面更好，所以牛顿法选择的下降路径会更符合真实的最优下降路径。

但是牛顿法除了收敛速度快的有点之外，还有一些缺点，最大的缺点就是每一次迭代都要求目标函数的 **Hessian 矩阵的逆矩阵，计算比较复杂**。

总的来说，这两种优化方法都**只能找到局部最优解**，容易陷入局部最优；这两种算法都需要给一个初始点才可以开始优化迭代。牛顿法对于局部凸的函数可以找到极小值，对于局部凹的函数可以找到极大值，对于局部不凹不凸的函数可能找到鞍点，而梯度下降法只能找到最小值，不能找到最大值，但可能同样会找到鞍点。

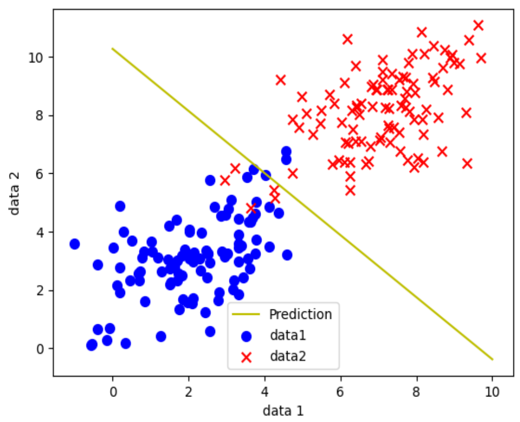
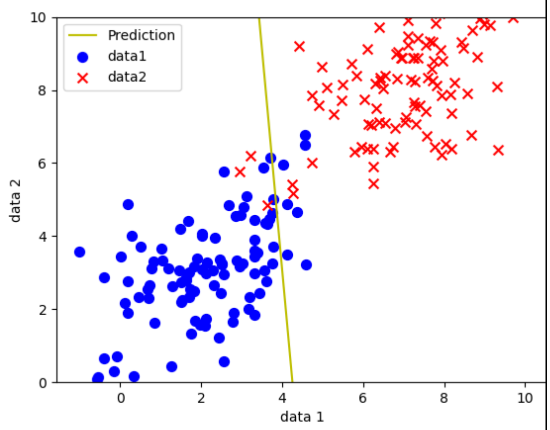
四、实验结果与分析

4.1 手工生成高斯数据验证

我们首先手工生成了两组高斯数据，并且他们是不符合贝叶斯分布假设的，也就是两个维度的数据不是相互独立的。在实验中，我们分别用有惩罚项、无惩罚项的梯度下降法进行线性逻辑回归，设置不同的学习率进行对比。

效果如下图所示，实验中发现学习率的设置对逻辑回归结果有较大的影响。虽然学习率设置过大容易造成梯度爆炸（实验一多项式拟合就是这样），但是学习率设置过小也容易造成参数收敛缓慢的情况。在实验中，我们首先选取 1 为学习率，迭代 5000 次，得到了较好的拟合结果（准确率为 0.965），但是如果调小学习率，比如 0.0001,进行同样的迭代次数，准确率只有 0.95。

实验发现可能需要需要选取合适的学习率，比如 0.001 来进行梯度下降。

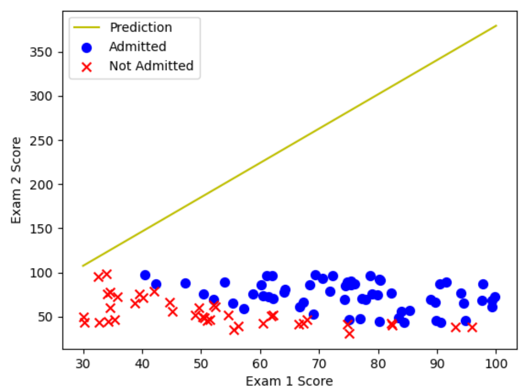
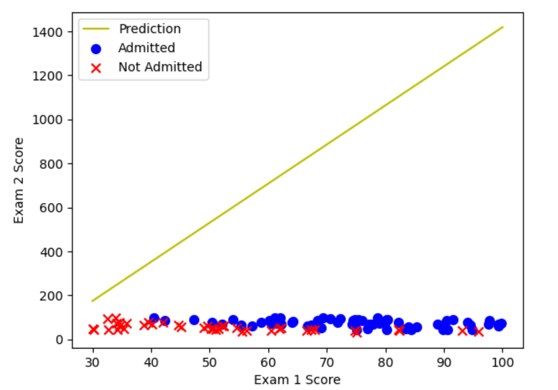
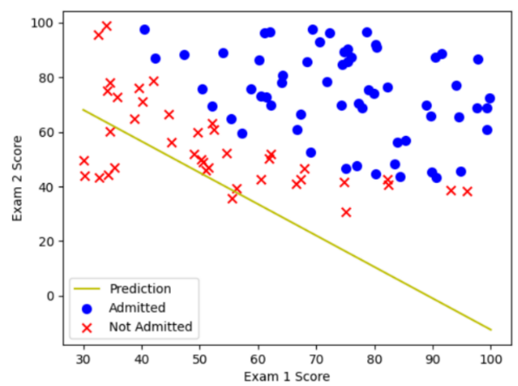
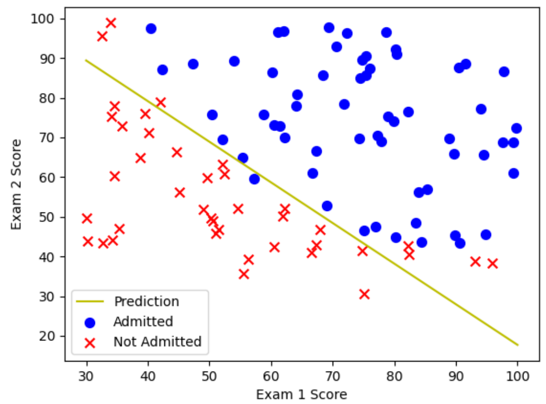
迭代次数均为 5000 次	
	
学习率 1，准确率 0.965	学习率 0.0001，准确率 0.95

4.2 实际数据验证

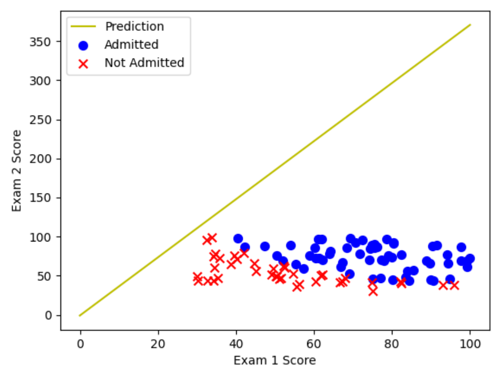
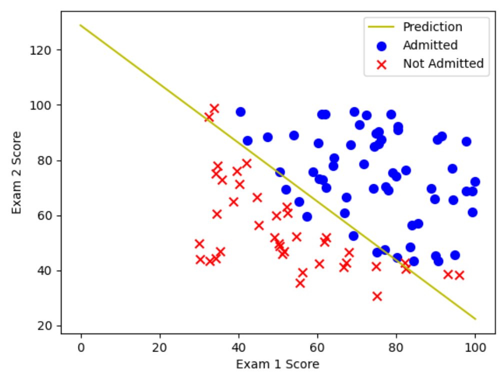
验证实验选取 100 组真实数据，每组包含考试成绩 1，考试成绩 2，以及是否被录取（被录取为 1，未被录取为 0）。数据来源：吴恩达-机器学习 experiment2。利用梯度下降法进行逻辑回归，固定学习率为 0.01，通过改变迭代的次数观察梯度下降法的效果。

logistic 二分类模型预测结果如下图所示。我们发现，固定合适的学习率，迭代的次数越多，效果越好。当迭代次数较少的时候，比如 1000 次、10000 次，此时所有数据全部在直线的同侧，但是此时测试的准确率仍然为较高的 0.6，这是因为数据本身分布约为 1:1,即使逻辑回归模型将所有的数据均归位 1 类，也可以得到大概 0.5 左右的准确率甚至更高。所以此时的 0.6 说明不了什么问题，可以当做的准确率的最低值。

当迭代的次数增多，比如 100000，300000 次的时候，我们发现逻辑回归的预测的效果明显增强，预测直线可以很好的分割两种类型的数据，准确率也提升，最高可到达 0.88。

梯度下降法，学习率均为 0.01	
	
迭代 1000 次，准确率 0.6	迭代 10000 次，准确率 0. 6
	
迭代 100000 次，准确率 0. 73	迭代 300000 次，准确率 0. 88

利用牛顿法进行逻辑回归，对比梯度下降法得到以下结果。可以看出梯度下降法收敛速度非常慢，而牛顿法在几十次左右就可以充分拟合数据了。

学习率为 0. 01	
	
梯度下降迭代 100 次，准确率为 0. 6	牛顿法迭代 100 次，准确率 0. 9

五、结论

1、关于惩罚项：对于逻辑回归而言，带正则项和不带正则项的差别没有多项式拟合函数那么大。但是使用牛顿法进行优化时，由于比较容易找到最小值，所

以如果不加正则项会发生过拟合。
2、关于牛顿法：牛顿法每次迭代的时间代价相比梯度下降法，每次的时间开销和空间占用会更大。但是牛顿法仅需大约几百次就能找到最小值，比梯度下降法快得多（几万次）。但是牛顿法的计算过程中涉及求黑塞矩阵的逆，如果矩阵奇异，则牛顿法不再适用。
3、关于精度：python 编译器默认浮点数为 float32，有时候精度丢失会比较严重，如果需要使用 float64 表示数据，需要自己手动设置。关于 sigmoid 函数，sigmoid 函数可能会发生溢出。

六、参考文献

[1]https://blog.csdn.net/lsgqjh/article/details/79168095
[2] https://www.jianshu.com/p/d892d0d13b6d
[3] https://www.coursera.org/learn/machine-learning(吴恩达机器学习)
[4]《统计学习方法》，李航
[5]《机器学习》，周志华

七、附录：源代码（带注释）

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def datadeal(path):

    data=pd.read_csv(path,header=None,names=['Exam 1','Exam 2','Admitted'])

    data.head()

    return data

#可视化考试录取数据的结果

def drawline(data,result):

    plotting_x1 = np.linspace(0, 100, 100)

    #画出预测直线

    plotting_h1 = (- result[0] - result[1] * plotting_x1) / result[2]

    #画出正反类的散点图

    positive = data[data['Admitted'].isin([1])]

    negative = data[data['Admitted'].isin([0])]

    fig, ax = plt.subplots()

    ax.plot(plotting_x1, plotting_h1, 'y', label='Prediction')
```

```
    ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='o', label='Admitted')

    ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x', label='Not Admitted')

    ax.legend()

    ax.set_xlabel('Exam 1 Score')

    ax.set_ylabel('Exam 2 Score')

    plt.show()

#可视化高斯分布数据的结果

def drawlineGauss(data,result):

    plotting_x1 = np.linspace(0, 10, 100)

    #画出预测直线

    #plotting_h1 = (- result[0][0] - result[0][1] * plotting_x1) / result[0][2]

    plotting_h1 = (- result[0] - result[1] * plotting_x1) / result[2]

    #画出正反类的散点图

    positive = data[data['Admitted'].isin([1])]

    negative = data[data['Admitted'].isin([0])]

    fig, ax = plt.subplots()

    ax.plot(plotting_x1, plotting_h1, 'y', label='Prediction')

    ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='o', label='data1')

    ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x', label='data2')

    ax.legend()

    ax.set_xlabel('data 1')

    ax.set_ylabel('data 2')

    ax.set_ylim([0,10])

    plt.show()


def sigmoid(z):
```

```python
    return 1/(1+np.exp(-z))
#损失函数
def cost(theta,X,y):
    theta=np.matrix(theta)
    X=np.matrix(X)
    y=np.matrix(y)
    #multiply 对应元素相乘
    first=np.multiply(-y,np.log(sigmoid(X*theta.T)))
    second=np.multiply((1-y),np.log(1-sigmoid(X*theta.T)))
    return np.sum(first-second)/len(X)
#求梯度
def gradient(theta,X,y):
    theta=np.matrix(theta)
    X=np.matrix(X)
    y=np.matrix(y)
    parameters = int(theta.ravel().shape[1])
    grad = np.zeros(parameters)
    error = sigmoid(X * theta.T) - y
    for i in range(parameters):
        term = np.multiply(error, X[:, i])
       # grad[i] = np.sum(term) / len(X)
        grad[i] = np.sum(term)/(2*len(X))
    return grad
#梯度下降法
def graidentRun(theta,X,y,iteranum=10000,learn_rate=0.001):
    for i in range(iteranum):
        grad = gradient(theta, X, y)
        theta=theta-learn_rate*grad
        if(i %100==0):
            loss=cost(theta,X,y)
```

```python
            print('准确率:',rightrate(theta,X,y))
            print('梯度下降法：第{}次迭代，误差下降为：{},参数为：{}'.format
(i, loss,theta))
    return theta
#计算牛顿法方向向量
def Newton(theta,X,y):
    l=len(y)
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    parameters = int(theta.ravel().shape[1])
    grad = np.zeros(parameters)
    error = sigmoid(X * theta.T) - y
    h_value=sigmoid(X * theta.T)
    for i in range(parameters):
        term = np.multiply(error, X[:, i])
        grad[i] = np.sum(term) / ( len(X))
    grad=np.matrix(grad).T
    h=np.eye(l)
    for i in range(l):
        h[i,i]=(1-h_value[i])*h_value[i]
    h=np.dot(X.T,h)
    h=np.dot(h,X)
    result=np.dot(h.I,grad)
    r=np.zeros(parameters)
    for i in range(parameters):
        r[i]=result[i][0].item()
    return r
#牛顿法优化参数
def NewtonRun(theta,X,y,iteranum=10000,learn_rate=0.001):
```

```
    for i in range(iteranum):
        grad = Newton(theta, X, y)
        theta=theta-learn_rate*grad
        if(i %10==0):
            loss=cost(theta,X,y)
            print('准确率:',rightrate(theta,X,y))
            print('Newton 法：第{}次迭代，误差下降为：{},参数为：{}'.format
(i, loss,theta))
    return theta


#预测类别
def predict(theta,X):
    probability=sigmoid(np.dot(X,theta.T))
    pre=[1 if x >=0.5 else 0 for x in probability]
    return pre
#计算准确率
def rightrate(theta,X,y):
    predictions=predict(theta,X)
    correct=[1 if ((a==1 and b==1)or(a==0 and b==0)) else 0 for (a,b) i
n zip(predictions,y)]
    accuracy=sum(correct)/len(correct)
    return accuracy
#逻辑回归主函数
def logisticmain(path):
    data=datadeal(path)
    # 加一列常数列,为以后的偏置做准备
    data.insert(0, 'Ones', 1)
    # 初始化 X，y，θ
    cols = data.shape[1]
    X = data.iloc[:,0:cols-1]
    y = data.iloc[:,cols-1:cols]
```

```
    theta = np.zeros(3)
    # 转换 X, y 的类型
    X = np.array(X.values)
    y = np.array(y.values)
    result=graidentRun(theta,X,y,iteranum=300000,learn_rate=0.01)
    loss=cost(theta,X,y)
    accuracy=rightrate(theta,X,y)
    print('accuracy=',accuracy)
    drawline(data,result)
#生成高斯分布的两类数据
def generateData():
    f=open('exp.txt','w')
    mean=[2,3]
    cov=np.mat([[2,1],[1,2]])
    x=np.random.multivariate_normal(mean,cov,100)
    for i in range(len(x)):
        line = []
        line.append(x[i][0])
        line.append(x[i][1])
        line.append(1)
        line = ",".join(str(i) for i in line)
        line = line + "\n"
        f.write(line)
    mean1 = [7, 8]
    X = np.random.multivariate_normal(mean1, cov, 100)
    for i in range(len(X)):
        line = []
        line.append(X[i][0])
        line.append(X[i][1])
        line.append(0)
```

```
        line = ",".join(str(i) for i in line)

        line = line + "\n"

        f.write(line)

    f.close()


#generateData()

path='ex2data1.txt'

logisticmain(path)
```