**《机器学习与数据挖掘》实验三**

实验题目： 求解对数几率回归问题

实验目的： 掌握对数几率回归的基本原理与实现

实验环境（硬件和软件） Anaconda/Jupyter notebook/Pycharm

实验内容：

根据给定数据集（存放在data1.txt文件中，二分类数据），编码实现基于梯度下降的Logistic回归算法，并画出决策边界；

**一、**已经给定部分代码，补充完整的代码，需要补充代码的地方已经用红色字体标注，包括：

**（1）#补充计算代价的代码；**

def computeCost(X,y,theta):

    m = X.shape[0]

    #补充计算代价的代码；

    z = -y \* np.log(hypothesis(X,theta) + 1e-6) - (1 - *y*) \* np.log(1 - hypothesis(X,theta) + 1e-6)

    return np.sum(z)/m

**（2）#补充参数更新代码；**

def gradientDescent(X,y,theta,iterations,alpha):

    #取数据条数

    m = X.shape[0]

    #在x最前面插入全1的列

    X = np.hstack((np.ones((m, 1)), X))

    for i in range(iterations):

    #补充参数更新代码；

        theta\_temp = theta - alpha \* np.dot(X.*T*,hypothesis(X,theta) - y) / m

        theta = theta\_temp

        #每迭代1000次输出一次损失值

        if(i%10000==0):

            diff.append([*i*,computeCost(X,y,theta)])

            print('第',i,'次迭代，当前损失为：',computeCost(X,y,theta),'theta=',theta)

    return theta

**（3）#补充画决策边界的代码；**

def plotDescisionBoundary(X,y,theta):

    cm\_dark = mpl.colors.ListedColormap(['g', 'r'])

    plt.xlabel('Exam 1 score')

    plt.ylabel('Exam 2 score')

    plt.scatter(X[:,0],X[:,1],c=np.array(y).squeeze(),cmap=cm\_dark,s=30)

    #补充画决策边界代码；

    x1 = np.linspace(0,150,500)

    x2 = (-*theta*[0] - *theta*[1] \* *x1*) / theta[2]

    plt.plot(x1,x2)

    plt.show()

**二、**提交的实验内容：（1）补充完整的代码；（也可以自己重写这部分的代码提交）（2）数据散点图，以及得到的决策边界；（3）梯度下降过程中损失的变化图；（4）基于训练得到的参数，输入新的样本数据，输出预测值；

代码思路：

逻辑回归的原理是用逻辑函数把线性回归的结果(−∞,∞)映射到(0,1) ，

1. **线性回归函数**

线性回归函数的数学表达式，其中是自变量，y 是因变量，y的值域为， 是常数项， (i=1,2,...,n)是待求系数，不同的权重 反映了自变量对因变量不同的贡献程度。回归分析中包括两个或两个以上自变量的回归分析，称为多元线性回归分析。不管是一元线性回归分析还是多元线性回归分析，都是线性回归分析。

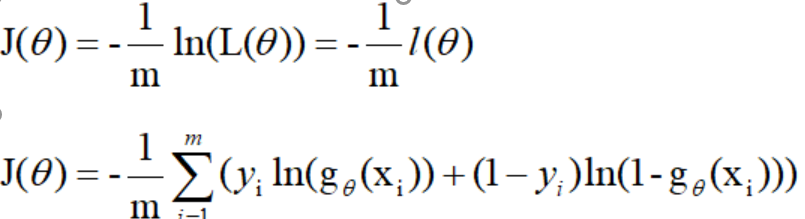
1. **逻辑函数—sigmoid函数**

函数的数学表达式： ,而逻辑回归函数的原理是用逻辑函数把线性回归的结果从映射到(0,1)。我们用公式描述上面这句话：,把线性回归函数的结果y，放到sigmod函数中去，就构造了逻辑回归函数。

1. **损失函数**

需要一个函数能够衡量模型拟合程度的好坏，也就是说当模型拟合误差越大的时候，函数值应该比较大，反之应该比较小，这就是损失函数。如果损失函数越小，说明模型预测越准。所以在函数比较复杂没有确定解(解析解)或很难求出确定解的情况下，一般求的是数值解(近似解)。一般模型求数值解可以求出使得损失函数最小对应的参数θ。结合逻辑回归中的极大似然函数，如果取整个数据集上的平均对数似然损失，我们可以得到:

=

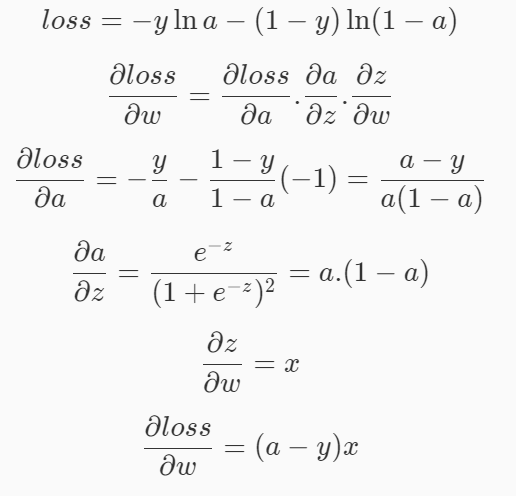
其中J(θ)为损失函数，由对数似然函数前面添加负号取平均得到。由损失函数求导知：

1. **梯度下降算法流程：**

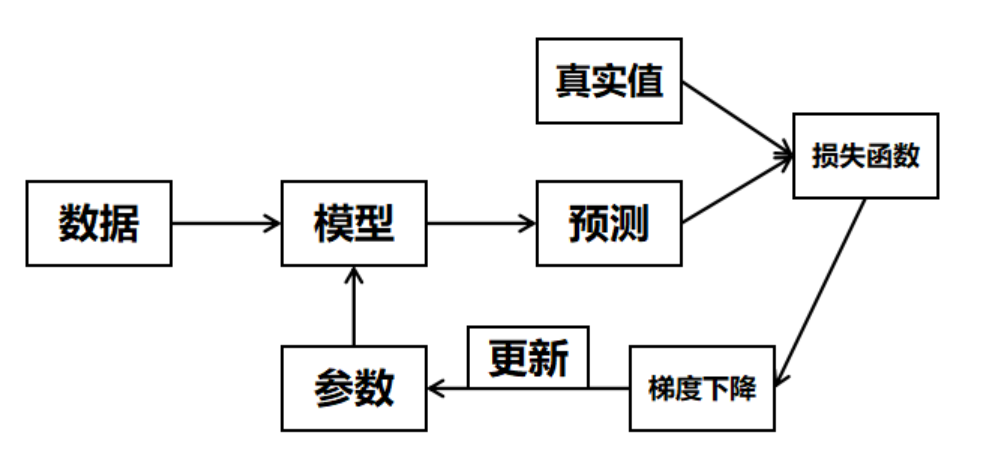
* 随机初始参数
* 确定学习率
* 求出损失函数对参数梯度
* 按照公式更新参数
* 重复3、4直到满足终止条件(如：损失函数或参数更新变化值小于某个阈值，或者训练次数达到设定阈值)

5. 训练逻辑回归模型

要使用梯度下降算法首先要知道损失函数对参数的梯度，即损失函数对每个参数的偏导，求解步骤如下



其中a为预测值，y为真实值。于是，在逻辑回归中的梯度下降公式如下：

*wi*​=*wi*​−*η*(*a*−*y*)*xi*​  


**完整代码:**

from matplotlib.font\_manager import FontProperties

import numpy as np

import matplotlib.pyplot as plt

import matplotlib as mpl

# from sklearn.metrics import accuracy\_score

def loaddata():

    data = np.loadtxt('data1.txt',delimiter=',')

    n = data.shape[1] - 1  # 特征数

    X = data[:, 0:n]

    y = data[:, -1].reshape(-1, 1)

    return X, y

def plot(X,y):

    pos = np.where(y==1)

    neg = np.where(y==0)

    plt.scatter(X[pos[0],0],X[pos[0],1],marker='x')

    plt.scatter(X[neg[0], 0], X[neg[0], 1], marker='o')

    plt.xlabel('Exam 1 score')

    plt.ylabel('Exam 2 score')

    plt.show()

X,y = loaddata()

plot(X,y)

def sigmoid(z):

    r = 1/(1+np.exp(-z))

    return r

def hypothesis(X,theta):

    z=np.dot(X,theta)

    return sigmoid(z)

def computeCost(X,y,theta):

    m = X.shape[0]

    #补充计算代价的代码；

    z = -y \* np.log(hypothesis(X,theta) + 1e-6) - (1 - *y*) \* np.log(1 - hypothesis(X,theta) + 1e-6)

    return np.sum(z)/m

diff = []

#

def gradientDescent(X,y,theta,iterations,alpha):

    #取数据条数

    m = X.shape[0]

    #在x最前面插入全1的列

    X = np.hstack((np.ones((m, 1)), X))

    for i in range(iterations):

    #补充参数更新代码；

        theta\_temp = theta - alpha \* np.dot(X.*T*,hypothesis(X,theta) - y) / m

        theta = theta\_temp

        #每迭代1000次输出一次损失值

        if(i%10000==0):

            diff.append([*i*,computeCost(X,y,theta)])

            print('第',i,'次迭代，当前损失为：',computeCost(X,y,theta),'theta=',theta)

    return theta

def predict(X):

    # 在x最前面插入全1的列

    c = np.ones(X.*shape*[0]).transpose()

    X = np.insert(X, 0, values=c, axis=1)

    #求解假设函数的值

    h = hypothesis(X,theta)

    #根据概率值决定最终的分类,>=0.5为1类，<0.5为0类

    h[h>=0.5]=1

    h[h<0.5]=0

    return h

X,y = loaddata()

n = X.shape[1]#特征数

theta = np.zeros(n+1).reshape(n+1, 1)

# theta是列向量,+1是因为求梯度时X前会增加一个全1列

theta\_temp = np.zeros(n+1).reshape(n+1, 1)

iterations = 250000

alpha = 0.008

theta = gradientDescent(X,y,theta,iterations,alpha)

print('theta=\n',theta)

def plotDescisionBoundary(X,y,theta):

    cm\_dark = mpl.colors.ListedColormap(['g', 'r'])

    plt.xlabel('Exam 1 score')

    plt.ylabel('Exam 2 score')

    plt.scatter(X[:,0],X[:,1],c=np.array(y).squeeze(),cmap=cm\_dark,s=30)

    #补充画决策边界代码；

    x1 = np.linspace(0,150,500)

    x2 = (-*theta*[0] - *theta*[1] \* *x1*) / theta[2]

    plt.plot(x1,x2)

    plt.show()

def plotLoss():

    d = np.array(diff)

    x = d[:,0]

    y = d[:,1]

    plt.plot(x,y)

    plt.title("损失函数变化图",fontsize = 20,fontproperties = "kaiti")

    plt.show()

def plotPred():

    test\_x = []

    for i in range(233):

        tx = np.random.uniform(0.0,100.0)

        ty = np.random.uniform(0.0,100.0)

        test\_x.append([*tx*,ty])

    test\_x = np.array(test\_x)

    test\_y = predict(test\_x)

    cm\_dark = mpl.colors.ListedColormap(['b', 'pink'])

    plt.scatter(test\_x[:, 0], test\_x[:, 1], c=np.array(test\_y).squeeze(), cmap=cm\_dark, s=30)

    x1 = np.linspace(0, 150, 500)

    x2 = (- theta[0] - *theta*[1] \* *x1*) / theta[2]

    plt.plot(x1,x2)

    plt.title("预测",fontproperties = 'kaiti',fontsize = 20)

    plt.show()

plotDescisionBoundary(X,y,theta)

plotLoss()

plotPred()

**实验图像和日志**

