HOMEWork 3: LINEAR REGRESSION AND LOGISTIC REGRESSION

DATA MINING AND MACHINE LEARNING (MARch 2022)

Student Name:郝裕玮 Student ID:18329015

Lectured by: Shangsong Liang
Sun Yat-sen University
Deadline of your submission is: 10:10 AM, March 29, 2022
Do NOT Distributed This Document and the Associated Datasets

1 Exercise One: Linear Regression

In this homework, you will investigate multivariate linear regression using Gradient Descent and Stochastic Gradient Descent. 1 You will also examine the relationship between the cost function, the convergence of gradient descent, overfitting problem, and the learning rate.

在本次作业中，你将探讨使用梯度下降法和随机梯度下降法的多变量线性回归模型。你将探讨损失函数、梯度下降法的收敛、过拟合问题和学习率等之间的关系。

Download the file “dataForTrainingLinear.txt” in the attached files called “Assignment 3” . This is a training dataset of apartment prices in Haizhu District, Guangzhou, Guangdong, China, where there are 50 training instances, one line per one instance, formatted in three columns separated with each other by a whitespace. The data in the first and the second columns are sizes of the apartments in square meters and the distances to the Double-Duck-Mountain Vocational Technical College in kilometers, respectively, while the data in the third are the corresponding prices in billion RMB. Please build a multivariate linear regression model with the training instances by script in any programming languages to predict the prices of the apartments. For evaluation purpose, please also download the file “dataForTestingLinear.txt” (the same format as that in the file of training data) in the same folder.

请在文件夹“Assignment 3”中下载文件名为“dataForTrainingLinear.txt”的文件。该文件包含广东省广州市海珠区的房价信息，里面包含 50 个训练样本数据。文件有三列，第一列对应房的面积（单位：平方米），第二列对应房子距离双鸭山职业技术学院的距离（单位：千米），第三列对应房子的销售价格（单位：元）。每一行对应一个训练样本。请使用提供的 50 个训练样本来训练多变量回归模型以便进行房价预测，请用（随机）梯度下降法的多变量线性回归模型进行建模。为了评估训练效果，请文件夹中下载测试数据集“dataForTestingLinear.txt”(该测试文件里的数据跟训练样本具有相同的格式，即第一列对应房子面积，第二列对应距离，第三列对应房子总价)。

(a) How many parameters do you use to tune this linear regression model? Please use Gradient Descent to obtain the optimal parameters. Before you train the model, please set the number of iterations to be 1500000, the learning rate to 0.00015, the initial values of all the parameters to 0.0. During training, at every 100000 iterations, i.e., 100000 , 200000,… , 1500000, report the current training error and the testing error in a figure (you can draw it by hands or by any software). What can you find in the plots? Please analyze the plots.

你需要用多少个参数来训练该线性回归模型？请使用梯度下降方法训练。训练时，请把迭代次数设成 1500000，学习率设成 0.00015，参数都设成 0.0。在训练的过程中，每迭代 100000 步，计算训练样本对应的误差，和使用当前的参数得到的测试样本对应的误差。请画图显示迭代到达 100000 步、 200000 步、…… 1500000 时对应的训练样本的误差和测试样本对应的误差（图可以手画，或者用工具画图）。从画出的图中，你发现什么？请简单分析。

答：需要用3个参数，线性回归模型如下：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

损失函数为：

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

每次更新参数的操作为：

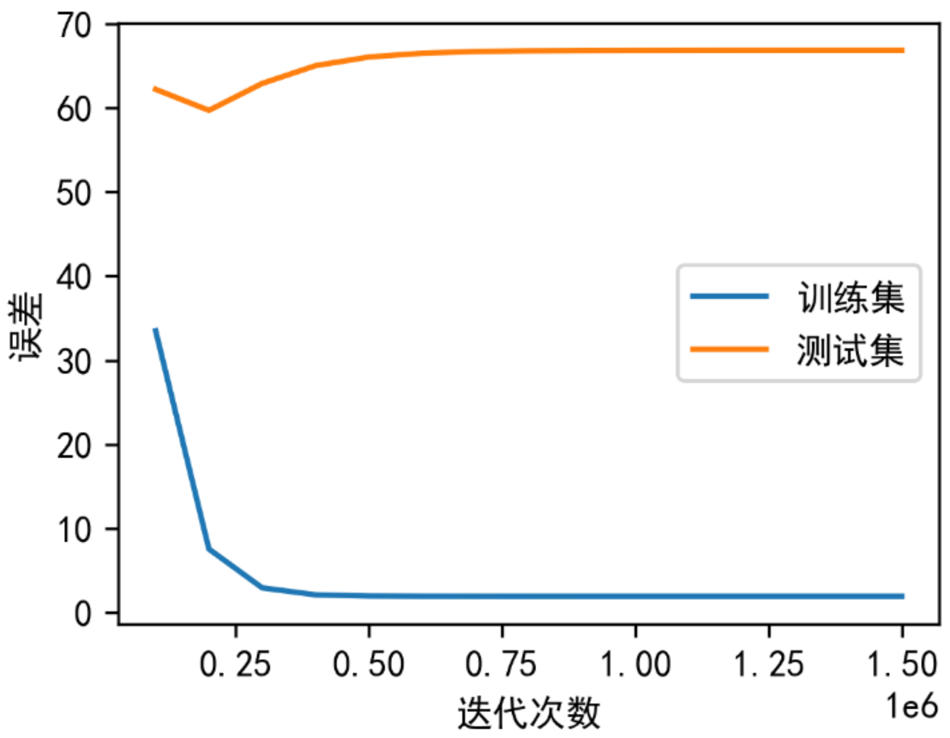
$$\theta_i = \theta_i - \alpha\frac{\partial J(\theta)}{\theta_i} = \theta_i - \alpha\frac{1}{m}\sum_{j=1}^{m}(h_\theta(x^{(j)}) - y^{(j)})x_i^{(j)} \quad (i = 0, 1, ..., n)$$

所以代码及其注释如下所示：

```python
import numpy as np
import matplotlib.pyplot as plt

# 损失函数：均方误差
def loss(train_data, weight, train_real, num):
    # 将数据和权重两个矩阵相乘得到预测结果矩阵
    # matmul：将两个矩阵进行矩阵乘法
    result = np.matmul(train_data, weight.T)
    # 计算均方误差
    loss = train_real - result
    losses = pow(loss, 2)
    losses = losses.T
    final_loss = losses.sum()/(2*num)
    return final_loss

# 梯度下降
def gradient(train_data, weight, train_real, num):
    result = np.matmul(train_data, weight.T)
    loss = result - train_real
    x1 = train_data.T[0]
    x1 = x1.T
    theta_1 = loss * x1

    x2 = train_data.T[1]
    x2 = x2.T
    theta_2 = loss * x2

    x3 = train_data.T[2]
    x3 = x3.T
    theta_3 = loss * x3
    return theta_1.T.sum() / num, theta_2.T.sum() / num, theta_3.T.sum() / num

# 读取训练集和测试集数据
filename1 = "./dataForTrainingLinear.txt"
filename2 = "./dataForTestingLinear.txt"

# 训练集数据预处理
# train_num: 训练集数据数量
train_num = len(open(filename1,'r').readlines())
read_file = open(filename1)
train_lines = read_file.readlines()

list_all_train = []
list_element_train  = []
list_real_train  = []
# 遍历每行数据
for train_line in train_lines:
    list1 = train_line.split()
    for one in list1:
        # 将字符串转为浮点数
        list_element_train.append(float(one))
    # list_all_train 存储自变量（面积和距离，即 X 矩阵）
    list_all_train.append(list_element_train[0:2])
    # list_real_train 存储因变量（房价，即结果矩阵）
    list_real_train.append(list_element_train[2])
    list_element_train = []
# 给 list_all_train 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是偏差 b)
for one in list_all_train:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
train_data = np.array(list_all_train)
train_real = np.array(list_real_train)
# 矩阵转置
train_real = train_real.T

# 测试集数据预处理
# test_num: 训练集数据数量
test_num = len(open(filename2,'r').readlines())

read_test = open(filename2)
test_lines = read_test.readlines()

list_all_test = []
list_element_test = []
list_real_test = []
# 遍历每行数据
for test_line in test_lines:
    list2 = test_line.split()
    for one in list2:
        # 将字符串转为浮点数
        list_element_test.append(float(one))
    # list_all_test 存储自变量（面积和距离，即 X 矩阵）
    list_all_test.append(list_element_test[0:2])
    # list_real_test 存储因变量（房价，即结果矩阵）
    list_real_test.append(list_element_test[2])
    list_element_test = []
# 给 list_all_test 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_test:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
test_data = np.array(list_all_test)
test_real = np.array(list_real_test)
# 矩阵转置
test_real = test_real.T


'''
    参数初始化：
    weight：权重因子
    learn_rate：学习率
    x：自变量矩阵（面积和距离）
    train_loss：训练集误差
    test_loss：测试集误差
'''
weight = np.array([0.0, 0.0, 0.0])
learn_rate = 0.00015
x = []
train_loss = []
test_loss = []

# 迭代 1500000 次
for num in range(1, 1500001):
    train_losses = loss(train_data, weight, train_real, train_num)
    test_losses = loss(test_data, weight, test_real, test_num)

    theta_1, theta_2, theta_3 = gradient(train_data, weight, train_real, train_num)
    theta = np.array([theta_1, theta_2, theta_3])
    # weight 更新公式
    weight = weight - (theta * learn_rate)

    # 只打印每 100000 次的相关数据
    if num % 100000 == 0:
        x.append(num)
        train_loss.append(train_losses)
        test_loss.append(test_losses)
        print ("当前迭代次数为%d 次，训练集误差为：%f，测试集误差为：%f"%(num,
train_losses, test_losses))

# 画图
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换 sans-serif 字体
plt.figure(figsize=(4,3),dpi=300)
plt.xlabel('迭代次数')
plt.ylabel('误差')
plt.plot(x, train_loss,label='训练集')
plt.plot(x, test_loss,label='测试集')
plt.legend()
plt.show()
```

代码运行结果如下：

```
当前迭代次数为100000次，训练集误差为：33.504398，测试集误差为：62.231793
当前迭代次数为200000次，训练集误差为：7.529742，测试集误差为：59.727617
当前迭代次数为300000次，训练集误差为：2.908067，测试集误差为：62.936031
当前迭代次数为400000次，训练集误差为：2.085732，测试集误差为：65.048220
当前迭代次数为500000次，训练集误差为：1.939413，测试集误差为：66.074195
当前迭代次数为600000次，训练集误差为：1.913379，测试集误差为：66.530994
当前迭代次数为700000次，训练集误差为：1.908747，测试集误差为：66.727954
当前迭代次数为800000次，训练集误差为：1.907922，测试集误差为：66.811796
当前迭代次数为900000次，训练集误差为：1.907776，测试集误差为：66.847297
当前迭代次数为1000000次，训练集误差为：1.907750，测试集误差为：66.862296
当前迭代次数为1100000次，训练集误差为：1.907745，测试集误差为：66.868627
当前迭代次数为1200000次，训练集误差为：1.907744，测试集误差为：66.871299
当前迭代次数为1300000次，训练集误差为：1.907744，测试集误差为：66.872426
当前迭代次数为1400000次，训练集误差为：1.907744，测试集误差为：66.872901
当前迭代次数为1500000次，训练集误差为：1.907744，测试集误差为：66.873102
```

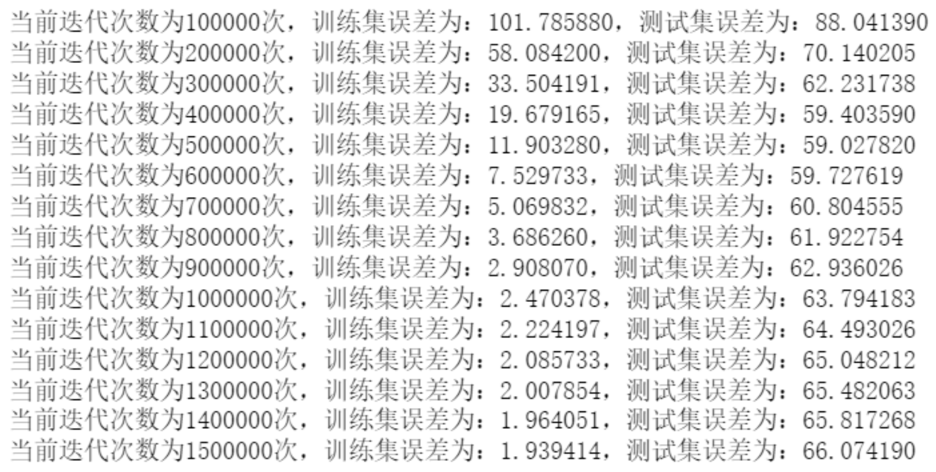


可见在训练集上误差随着迭代次数的增加而下降收敛。而在测试集上也随着迭代次数的增加而收敛，但误差并没有下降。

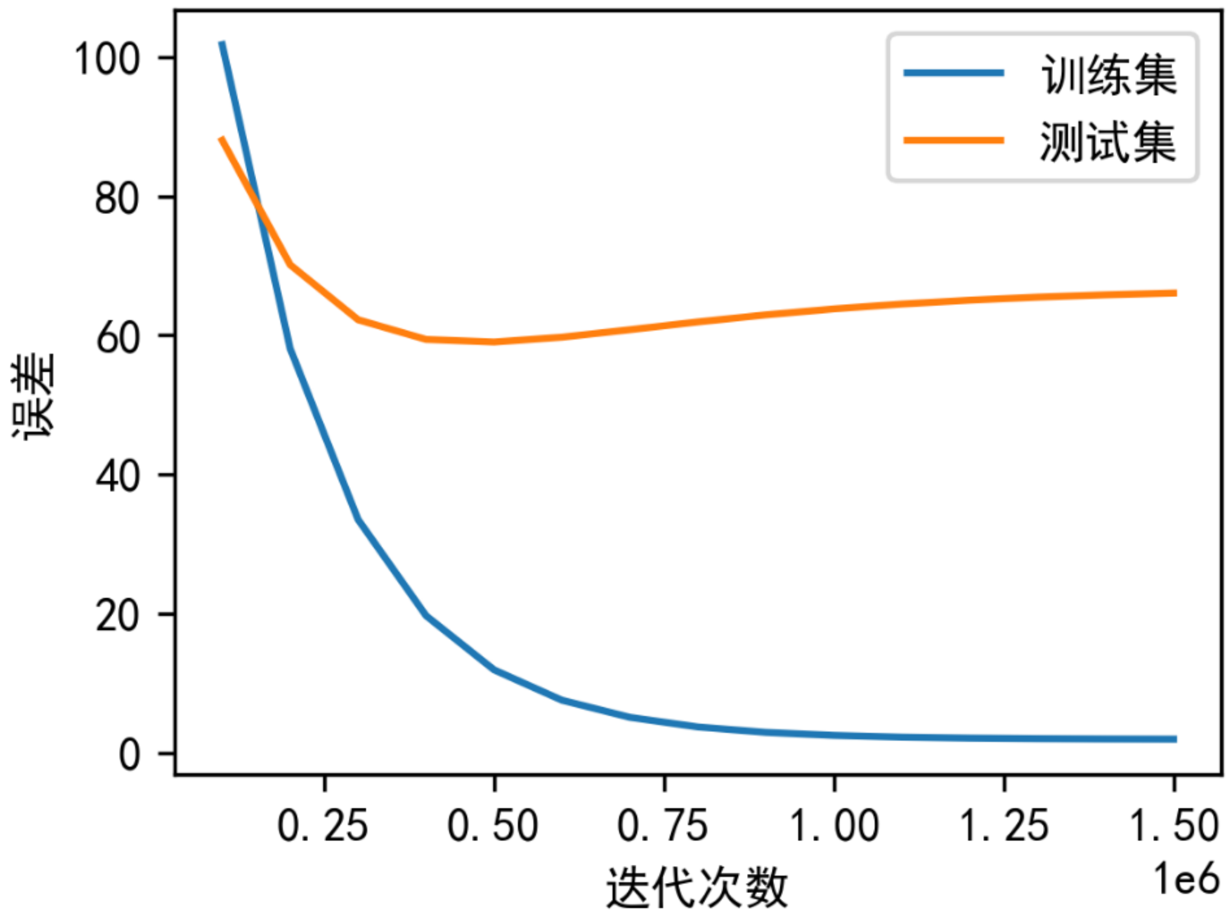
(b) Now, you change the learning rate to a number of different values, for instance, to 0.0002 (you may also change the number of iterations as well) and then train the model again. What can you find? Please conclude your findings.

现在，你改变学习率，比如把学习率改成 0.0002（此时，你可以保持相同的迭代次数也可以改变迭代次数），然后训练该回归模型。你有什么发现？请简单分析。

答：修改为0.002后，数组越界，无法存储结果。从（a）问也可看出0.002学习率过大，在一定迭代次数后会出现过拟合的现象。所以我们需要降低学习率，改为0.00005。迭代次数仍为1500000次，结果如下所示：



```
当前迭代次数为100000次，训练集误差为：101.785880，测试集误差为：88.041390
当前迭代次数为200000次，训练集误差为：58.084200，测试集误差为：70.140205
当前迭代次数为300000次，训练集误差为：33.504191，测试集误差为：62.231738
当前迭代次数为400000次，训练集误差为：19.679165，测试集误差为：59.403590
当前迭代次数为500000次，训练集误差为：11.903280，测试集误差为：59.027820
当前迭代次数为600000次，训练集误差为：7.529733，测试集误差为：59.727619
当前迭代次数为700000次，训练集误差为：5.069832，测试集误差为：60.804555
当前迭代次数为800000次，训练集误差为：3.686260，测试集误差为：61.922754
当前迭代次数为900000次，训练集误差为：2.908070，测试集误差为：62.936026
当前迭代次数为1000000次，训练集误差为：2.470378，测试集误差为：63.794183
当前迭代次数为1100000次，训练集误差为：2.224197，测试集误差为：64.493026
当前迭代次数为1200000次，训练集误差为：2.085733，测试集误差为：65.048212
当前迭代次数为1300000次，训练集误差为：2.007854，测试集误差为：65.482063
当前迭代次数为1400000次，训练集误差为：1.964051，测试集误差为：65.817268
当前迭代次数为1500000次，训练集误差为：1.939414，测试集误差为：66.074190
```



发现降低学习率后，测试集的误差也开始下降收敛，说明仍需要降低学习率来加快收敛速度。

(c) Now, we turn to use other optimization methods to get the optimal parameters. Can you use Stochastic Gradient Descent to get the optimal parameters? Plots the training error and the testing error at each K-step iterations (the size of K is set by yourself). Can you analyze the plots and make comparisons to those findings in Exercise 1?

现在，我们使用其他方法来获得最优的参数。你是否可以用随机梯度下降法获得最优的参数？请使用随机梯度下降法画出迭代次数（每 K 次，这里的 K 你自己设定）与训练样本和测试样本对应的误差的图。比较 Exercise 1 中的实验图，请总结你的发现。

答：随机梯度下降法每次只用一个样本数据更新weight，如下所示：

```
Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha \left( y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$        (for every $j$)
    }
}
```

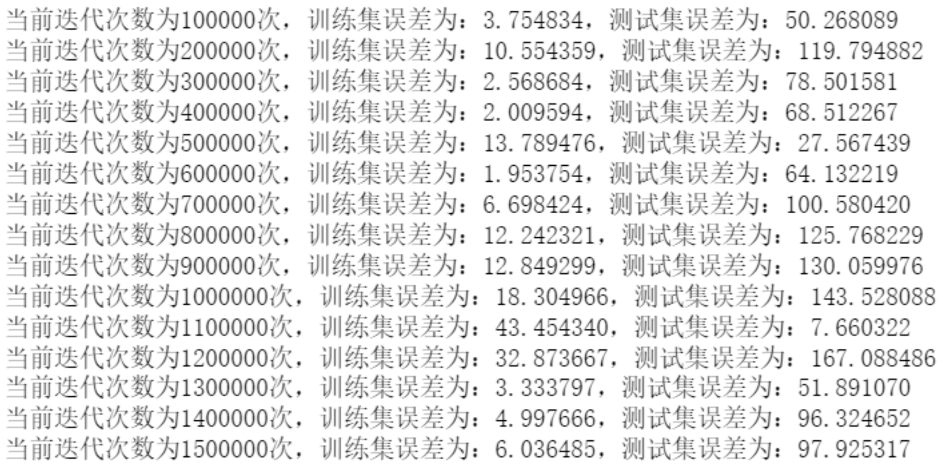
代码及其注释如下所示：

```python
import numpy as np
import math
import matplotlib.pyplot as plt


# 损失函数：均方误差
def loss(train_data, weight, train_real, num):
    # 将数据和权重两个矩阵相乘得到预测结果矩阵
    # matmul：将两个矩阵进行矩阵乘法
    result = np.matmul(train_data, weight.T)
    # 计算均方误差
    loss = train_real - result
    losses = pow(loss, 2)
    losses = losses.T
    final_loss = losses.sum()/(2*num)
    return final_loss

# 随机梯度下降
def random_Gradient(train_data, weight, train_real):
    # 随机选取 1 个点计算梯度和误差
    num = np.random.randint(0, 50)

    one_train_data = train_data[num]
    one_train_real = train_real[num]

    result = np.matmul(one_train_data, weight.T)
    loss = result - one_train_real

    x1 = one_train_data[0]
    theta_1 = loss * x1

    x2 = one_train_data[1]
    theta_2 = loss * x2

    x3 = train_data.T[2]
    x3 = x3.T
    theta_3 = loss * x3

    return theta_1.T.sum(), theta_2.T.sum(), theta_3.T.sum()


# 读取训练集和测试集数据
filename1 = "./dataForTrainingLinear.txt"
filename2 = "./dataForTestingLinear.txt"

# 训练集数据预处理
# train_num: 训练集数据数量
train_num = len(open(filename1,'r').readlines())
read_file = open(filename1)
train_lines = read_file.readlines()

list_all_train = []
list_element_train  = []
list_real_train  = []
# 遍历每行数据
for train_line in train_lines:
    list1 = train_line.split()
    for one in list1:
        # 将字符串转为浮点数
        list_element_train.append(float(one))
    # list_all_train 存储自变量（面积和距离，即 X 矩阵）
    list_all_train.append(list_element_train[0:2])
    # list_real_train 存储因变量（房价，即结果矩阵）
    list_real_train.append(list_element_train[2])
    list_element_train = []
# 给 list_all_train 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_train:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
train_data = np.array(list_all_train)
train_real = np.array(list_real_train)
```

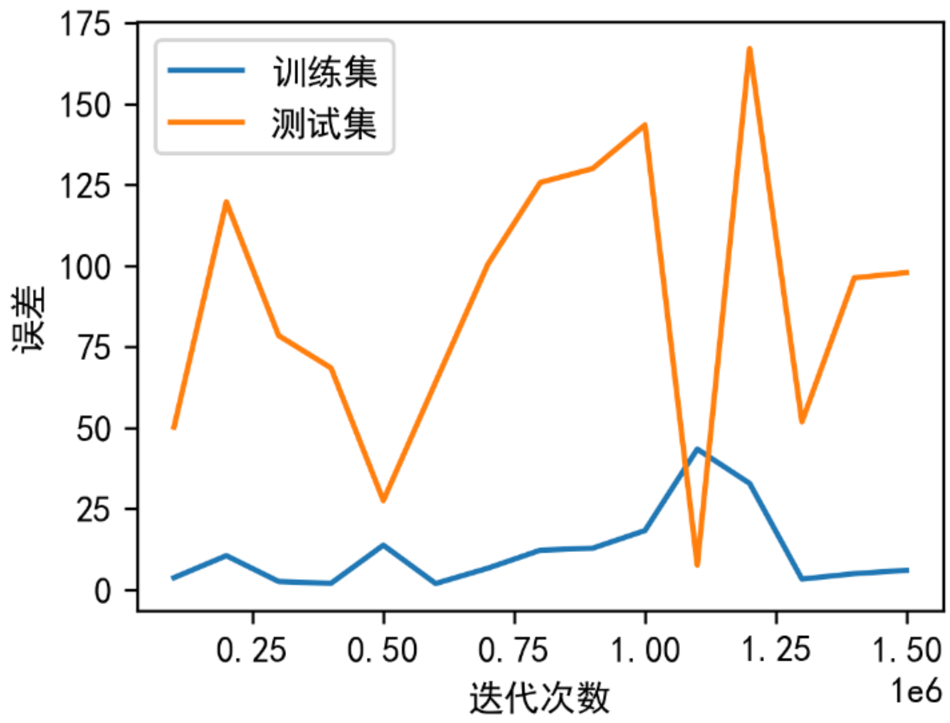
```
# 矩阵转置
train_real = train_real.T

# 测试集数据预处理
# test_num: 训练集数据数量
test_num = len(open(filename2,'r').readlines())

read_test = open(filename2)
test_lines = read_test.readlines()

list_all_test = []
list_element_test = []
list_real_test = []
# 遍历每行数据
for test_line in test_lines:
    list2 = test_line.split()
    for one in list2:
        # 将字符串转为浮点数
        list_element_test.append(float(one))
    # list_all_test 存储自变量（面积和距离，即 X 矩阵）
    list_all_test.append(list_element_test[0:2])
    # list_real_test 存储因变量（房价，即结果矩阵）
    list_real_test.append(list_element_test[2])
    list_element_test = []
# 给 list_all_test 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_test:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
test_data = np.array(list_all_test)
test_real = np.array(list_real_test)
# 矩阵转置
test_real = test_real.T


'''
    参数初始化：
    weight：权重因子
    learn_rate：学习率
    x：自变量矩阵（面积和距离）
    train_loss：训练集误差
    test_loss：测试集误差
'''
weight = np.array([0.0, 0.0, 0.0])
learn_rate = 0.00015
x = []
train_loss = []
test_loss = []
```

```
# 迭代 1500000 次
for num in range(1, 1500001):
    train_losses = loss(train_data, weight, train_real, train_num)
    test_losses = loss(test_data, weight, test_real, test_num)

    theta_1, theta_2, theta_3 = random_Gradient(train_data, weight, train_real)
    theta = np.array([theta_1, theta_2, theta_3])
    # weight 更新公式
    weight = weight - (theta * learn_rate)

    # 只打印每 100000 次的相关数据
    if num % 100000 == 0:
        x.append(num)
        train_loss.append(train_losses)
        test_loss.append(test_losses)
        print ("当前迭代次数为%d 次，训练集误差为：%f，测试集误差为：%f"%(num,
train_losses, test_losses))

# 画图
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换 sans-serif 字体
plt.figure(figsize=(4,3),dpi=300)
plt.xlabel('迭代次数')
plt.ylabel('误差')
plt.plot(x, train_loss,label='训练集')
plt.plot(x, test_loss,label='测试集')
plt.legend()
plt.show()
```

代码运行结果如下（学习率为0.00015，迭代次数为1500000次）：



```
当前迭代次数为100000次，训练集误差为：3.754834，测试集误差为：50.268089
当前迭代次数为200000次，训练集误差为：10.554359，测试集误差为：119.794882
当前迭代次数为300000次，训练集误差为：2.568684，测试集误差为：78.501581
当前迭代次数为400000次，训练集误差为：2.009594，测试集误差为：68.512267
当前迭代次数为500000次，训练集误差为：13.789476，测试集误差为：27.567439
当前迭代次数为600000次，训练集误差为：1.953754，测试集误差为：64.132219
当前迭代次数为700000次，训练集误差为：6.698424，测试集误差为：100.580420
当前迭代次数为800000次，训练集误差为：12.242321，测试集误差为：125.768229
当前迭代次数为900000次，训练集误差为：12.849299，测试集误差为：130.059976
当前迭代次数为1000000次，训练集误差为：18.304966，测试集误差为：143.528088
当前迭代次数为1100000次，训练集误差为：43.454340，测试集误差为：7.660322
当前迭代次数为1200000次，训练集误差为：32.873667，测试集误差为：167.088486
当前迭代次数为1300000次，训练集误差为：3.333797，测试集误差为：51.891070
当前迭代次数为1400000次，训练集误差为：4.997666，测试集误差为：96.324652
当前迭代次数为1500000次，训练集误差为：6.036485，测试集误差为：97.925317
```



由于每次迭代训练是随机选取一个数据，所以误差波动较大，但训练时间大幅度减少，相较于梯度下降，效率有所提高。

2 Exercise Two: Logistic Regression

You will implement a logistic regression classifier and apply it to a two-class classification problem. To get started, download the two datasets, "dataForTrainingLogistic.txt" and "dataForTestingLogistic.txt" from the folder called "Assignment 3". In both of these two datasets, each instance is put per line with the first to the six columns being the features of the instance and the last column being the ground-truth label of the category (either "1" or "0") that the instance should be classified into. Each column per line is separated by a whitespace.

您将实现逻辑回归分类器并将其应用于二分类问题。 首先，从名为"Assignment 3"的文件夹中下载两个数据集"dataForTrainingLogistic.txt"和"dataForTestingLogistic.txt"。 在这两个数据集中，每个实例都按行放置，第一到六列是实例的特征，最后一列是类别的真实标签（"1"或"0"） 实例应分类为。 每行的每一列由一个空格分隔。

(a) In logistic regression, our goal is to learn a set of parameters by maximizing the conditional log likelihood of the data. Assuming you are given a dataset with n training examples and p features, write down a formula for the conditional log likelihood of the training data in terms of the the class labels $y^{(i)}$, the features $x_1^{(i)},\dots, x_p^{(i)}$, and the parameters $w_0; w_1,\dots, w_p$, where the superscript (i) denotes the sample index. This will be your objective function for gradient ascent.

(a) 在逻辑回归中，我们的目标是通过最大化数据的条件对数似然来学习一组参数。 假设您有一个包含 n 个训练示例和 p 个特征的数据集，请根据类标签 $y^{(i)}$、特征 $x_1^{(i)},\dots, x_p^{(i)}$,以及参数$w_0; w_1,\dots, w_p$ 写下训练数据的条件对数似然的公式。其中上标 (i) 表示样本索引。这将是梯度上升的目标函数。

答：公式如下：
对数似然函数为：

$$L(w)=\sum[y_i \ln p(x_i)+(1-y_i)\ln(1-p(x_i))]$$

其中:

$$p(x)=\frac{1}{1+e^{-(w^Tx+b)}}$$

$$w=[w_1,w_2,\ldots,w_p]$$

$$b=w_0$$

(b) Compute the partial derivative of the objective function with respect to w_0 and with respect to an arbitrary w_j, i.e. derive $\partial f/\partial w_0$ and $\partial f/\partial w_j$, where f is the objective that you provided above. Please show all derivatives can be written in a finite sum form.
(b) 计算目标函数关于w_0和关于任意w_j的偏导数，即导出 $\partial f/\partial w_0$ 和 $\partial f/\partial w_j$，其中 f 是您在上面提供的目标。 请证明所有导数都可以写成有限和形式。

答：

$$\frac{\partial f}{\partial w_0}=\sum(y^{(i)}-p^{(i)})$$

$$\frac{\partial f}{\partial w_j}=\sum(y^{(i)}-p^{(i)})x_j^{(i)},\ j\in[1,6],j\text{为整数}$$

(c)Train your logistic regression classifier on the data provided in the training dataset “dataForTrainingLogistic.txt”. How do you design and train your logistic regression classifier? What are your optimal estimated parameters in your logistic regression classifier? Use your estimated parameters to calculate predicted labels for the data in the testing dataset “dataForTestingLogistic.txt” (Do not use the label information (the last column in the file) for testing).

(c) 根据训练数据集“dataForTrainingLogistic.txt”中提供的数据训练逻辑回归分类器。你如何设计和训练你的逻辑回归分类器？ 您的逻辑回归分类器中的最佳估计参数是什么？ 使用您的估计参数计算测试数据集“dataForTestingLogistic.txt”中数据的预测标签（不要使用标签信息（文件中的最后一列）进行测试）。

答：我们选择使用随机梯度下降来进行训练，迭代次数为500次，学习率为0.00015

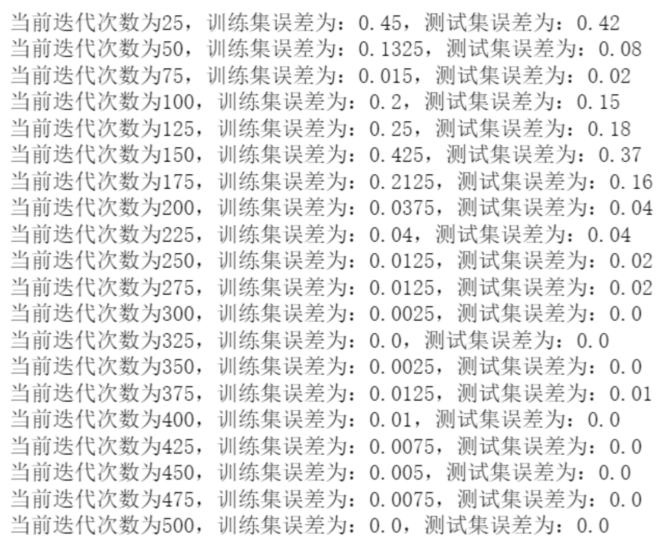
代码及其注释如下所示：

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
import random
import math
from matplotlib.pyplot import MultipleLocator

# 逻辑回归公式
def logistic(w, x):
    # dot: 矩阵乘法
    # 因为 w 和 x 都是 a 一维数组，所以返回的是向量的内积（一个数）
    temp = w.dot(x)
    temp = 1.0 / (1 + np.exp(-1 * temp))
    return temp

# 随机梯度下降
def random_Gradient(data, label, rate, weight):
    sum = 0
    log_w = 0

    i = random.randint(0, len(data) - 1)
    sum += (label[i] - logistic(weight, data[i])) * data[i]
    weight = weight + rate * sum

    for i in range(len(data)):
        log_w += label[i] * math.log(logistic(weight, data[i])) + (1-label[i]) *
math.log(1 - logistic(weight, data[i]))

    return weight

# 利用训练后的 weight 对训练集和测试集计算错误率和最大似然函数值
def predict(data, label, weight):
    error = 0
    log_w = 0
    # 计算最大似然函数值
    for i in range(len(data)):
        temp = logistic(weight, data[i])
        log_w += label[i] * math.log(logistic(weight, data[i])) + (1-label[i]) *
math.log(1 - logistic(weight, data[i]))
        # 预测错误则错误数+1
        if (temp >= 0.5 and label[i] == 0) or (temp < 0.5 and label[i] == 1):
            error = error + 1
    error /= len(data)
    return error, log_w


filename1 = "./dataForTrainingLogistic.txt"
filename2 = "./dataForTestingLogistic.txt"

# 训练集数据预处理
# train_num: 训练集数据数量
train_num = len(open(filename1,'r').readlines())
read_file = open(filename1)
```

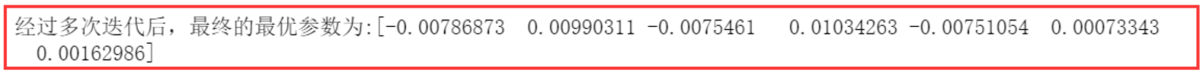
```
train_lines = read_file.readlines()

list_all_train = []
list_element_train  = []
list_label_train  = []
# 遍历每行数据
for train_line in train_lines:
    list1 = train_line.split()
    for one in list1:
        # 将字符串转为浮点数
        list_element_train.append(float(one))
    # list_all_train 存储自变量（面积和距离，即 X 矩阵）
    list_all_train.append(list_element_train[0:6])
    # list_label_train 存储因变量（房价，即结果矩阵）
    list_label_train.append(list_element_train[6])
    list_element_train = []
# 给 list_all_train 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_train:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
train_data = np.array(list_all_train)
train_label = np.array(list_label_train)

# 测试集数据预处理
# test_num: 训练集数据数量
test_num = len(open(filename2,'r').readlines())

read_test = open(filename2)
test_lines = read_test.readlines()

list_all_test = []
list_element_test = []
list_label_test = []
# 遍历每行数据
for test_line in test_lines:
    list2 = test_line.split()
    for one in list2:
        # 将字符串转为浮点数
        list_element_test.append(float(one))
    # list_all_test 存储自变量（面积和距离，即 X 矩阵）
    list_all_test.append(list_element_test[0:6])
    # list_label_test 存储因变量（房价，即结果矩阵）
    list_label_test.append(list_element_test[6])
    list_element_test = []
# 给 list_all_test 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_test:
    one.append(1.0)
```

```
# 均转为 numpy 数组类型，便于后续操作
test_data = np.array(list_all_test)
test_label = np.array(list_label_test)


'''
    参数初始化：
    weight：权重因子
    learn_rate：学习率
    x：自变量矩阵
    train_loss：训练集误差
    test_loss：测试集误差
'''
weight = np.zeros(7)
# 学习率
learn_rate = 0.00015
x = []
# 最大化似然函数值
train_log_w = []
test_log_w = []
# 误差（这里指错误率）
train_error = []
test_error = []

# 开始迭代学习
for i in range(1, 501):
    weight = random_Gradient(train_data, train_label, 0.00015, weight)
    train_error1, train_log_w1 = predict(train_data, train_label, weight)
    test_error1, test_log_w1 = predict(test_data, test_label, weight)

    if (i % 25 == 0):
        print("当前迭代次数为%s，train_error：%s，test_error：%s" % (i, train_error1,
test_error1))
    x.append(i)
    train_log_w.append(train_log_w1)
    test_log_w.append(test_log_w1)
    train_error.append(train_error1)
    test_error.append(test_error1)

print("\n\n 经过多次迭代后，最终的最优参数为:%s" %weight)

# 画图
plt.figure(1)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换 sans-serif 字体
plt.rcParams['axes.unicode_minus'] = False  # 解决坐标轴负数的负号显示问题
plt.figure(figsize=(4,3),dpi=300)
plt.xlabel('迭代次数')
plt.ylabel('最大似然函数值')
plt.plot(x, train_log_w,label='训练集')
plt.legend()

plt.figure(1)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换 sans-serif 字体
plt.rcParams['axes.unicode_minus'] = False  # 解决坐标轴负数的负号显示问题
plt.figure(figsize=(4,3),dpi=300)
plt.xlabel('迭代次数')
plt.ylabel('最大似然函数值')
plt.plot(x, test_log_w,label='测试集')
plt.legend()

plt.figure(2)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换 sans-serif 字体
plt.rcParams['axes.unicode_minus'] = False  # 解决坐标轴负数的负号显示问题
plt.figure(figsize=(4,3),dpi=300)
plt.xlabel('迭代次数')
plt.ylabel('误差')
plt.plot(x, train_error,label='测试集')
plt.plot(x, test_error,label='训练集')
plt.legend()

plt.show()
```

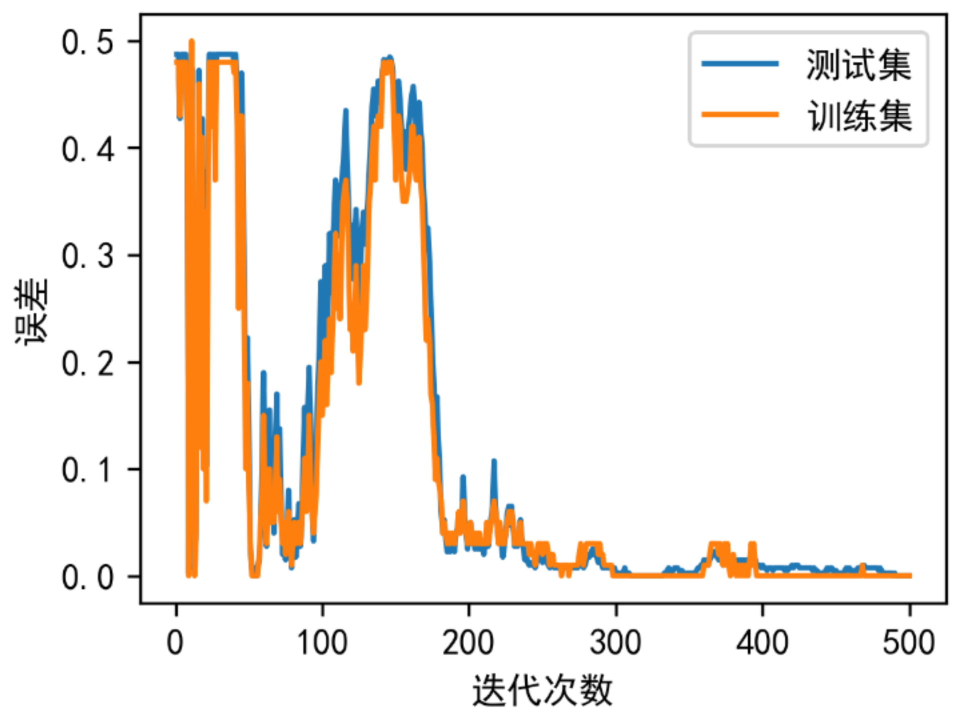
代码运行结果如下所示（学习率为0.00015，迭代次数为500次）：



```
当前迭代次数为25，训练集误差为：0.45，测试集误差为：0.42
当前迭代次数为50，训练集误差为：0.1325，测试集误差为：0.08
当前迭代次数为75，训练集误差为：0.015，测试集误差为：0.02
当前迭代次数为100，训练集误差为：0.2，测试集误差为：0.15
当前迭代次数为125，训练集误差为：0.25，测试集误差为：0.18
当前迭代次数为150，训练集误差为：0.425，测试集误差为：0.37
当前迭代次数为175，训练集误差为：0.2125，测试集误差为：0.16
当前迭代次数为200，训练集误差为：0.0375，测试集误差为：0.04
当前迭代次数为225，训练集误差为：0.04，测试集误差为：0.04
当前迭代次数为250，训练集误差为：0.0125，测试集误差为：0.02
当前迭代次数为275，训练集误差为：0.0125，测试集误差为：0.02
当前迭代次数为300，训练集误差为：0.0025，测试集误差为：0.0
当前迭代次数为325，训练集误差为：0.0，测试集误差为：0.0
当前迭代次数为350，训练集误差为：0.0025，测试集误差为：0.0
当前迭代次数为375，训练集误差为：0.0125，测试集误差为：0.01
当前迭代次数为400，训练集误差为：0.01，测试集误差为：0.0
当前迭代次数为425，训练集误差为：0.0075，测试集误差为：0.0
当前迭代次数为450，训练集误差为：0.005，测试集误差为：0.0
当前迭代次数为475，训练集误差为：0.0075，测试集误差为：0.0
当前迭代次数为500，训练集误差为：0.0，测试集误差为：0.0
```



```
经过多次迭代后，最终的最优参数为:[-0.00786873  0.00990311 -0.0075461   0.01034263 -0.00751054  0.00073343
  0.00162986]
```



(d) Report the number of misclassified examples in the testing dataset.

(d)报告测试数据集中错误分类示例的数量。

答：由（c）题可知，测试集误差为0，所以错误分类示例数量为0。

(e) Plot the value of the objective function on each iteration of stochastic gradient ascent, with the iteration number on the horizontal axis and the objective value on the vertical axis. Make sure to include axis labels and a title for your plot. Report the number of iterations that are required for the algorithm to converge.

(e) 绘制随机梯度上升每次迭代的目标函数值，水平轴为迭代次数，垂直轴为目标值。 确保包含轴标签和绘图的标题。 报告算法收敛所需的迭代次数。

答：如下所示：

在（c）题代码中添加以下部分：

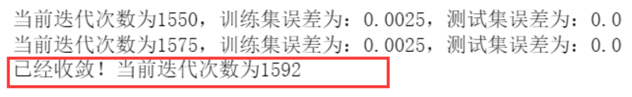
（1）设定当相邻两次weight权重更新后的欧氏距离小于$1 * 10^{-4}$时停止迭代：

```
# 随机梯度下降
def random_Gradient(data, label, rate, weight):
    sum = 0
    log_w = 0
    flag = False
    dis = 1e-4

    i = random.randint(0, len(data) - 1)
    sum += (label[i] - logistic(weight, data[i])) * data[i]
    #判断是否已经收敛
    temp = weight
    weight = weight + rate * sum
    dis1 = np.linalg.norm(weight - temp)
    if (dis1 <= dis):
        flag = True
    for i in range(len(data)):
        log_w += label[i] * math.log(logistic(weight, data[i])) + (1-label[i]) *
math.log(1 - logistic(weight, data[i]))

    return weight,flag
```

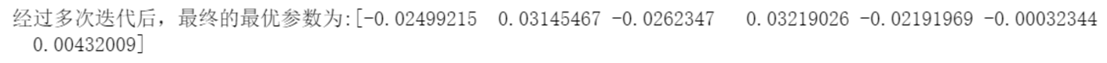
（2）添加flag判断跳出循环部分（下方代码中的第4-6行）

```
# 开始迭代学习
for i in range(1, 100001):
    weight, flag = random_Gradient(train_data, train_label, 0.00015, weight)
    if flag == True:
        print("已经收敛！当前迭代次数为%d\n" %i)
        break
    train_error1, train_log_w1 = predict(train_data, train_label, weight)
    test_error1, test_log_w1 = predict(test_data, test_label, weight)
```

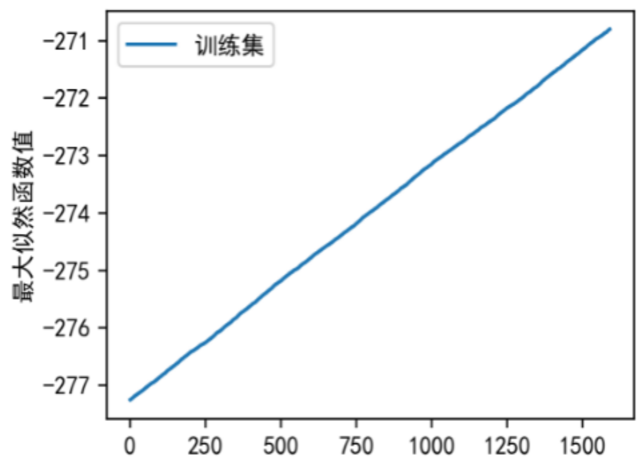
代码运行结果如下（学习率为0.00015）：

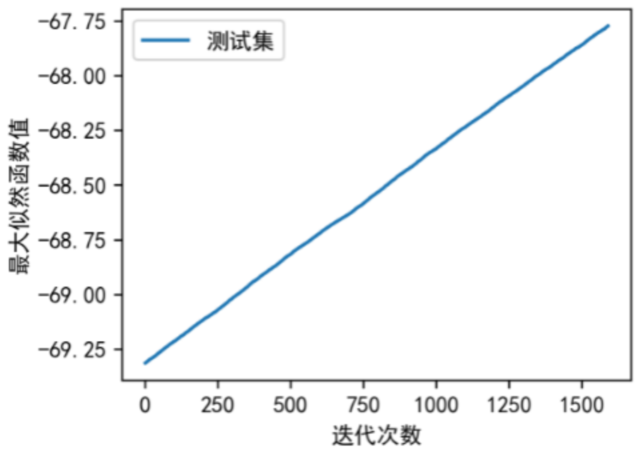


```
当前迭代次数为1550, 训练集误差为: 0.0025, 测试集误差为: 0.0
当前迭代次数为1575, 训练集误差为: 0.0025, 测试集误差为: 0.0
已经收敛！当前迭代次数为1592
```



```
经过多次迭代后，最终的最优参数为:[-0.02499215  0.03145467 -0.0262347   0.03219026 -0.02191969 -0.00032344
  0.00432009]
```





(f) Next, you will evaluate how the training and test error change as the training set size increases. For each value of *k* in the set {10,20,30,…,380,390,400}, first choose a random subset of the training data of size *k*. Then re-train your logistic regression classifier using the *k* random subset of the training data you just chose, and use the estimated parameters to calculate the number of misclassified examples on both the current training set (*k* random instances) and on the original test set “dataForTestingLogistic.txt”. Finally, generate a plot with two lines: in blue, plot the value of the training error against *k*, and in red, plot the value of the test error against *k*, where the error should be on the vertical axis and training set size should be on the horizontal axis. Make sure to include a legend in your plot to label the two lines.Describe what happens to the training and test error as the training set size increases, and provide an explanation for why this behavior occurs.

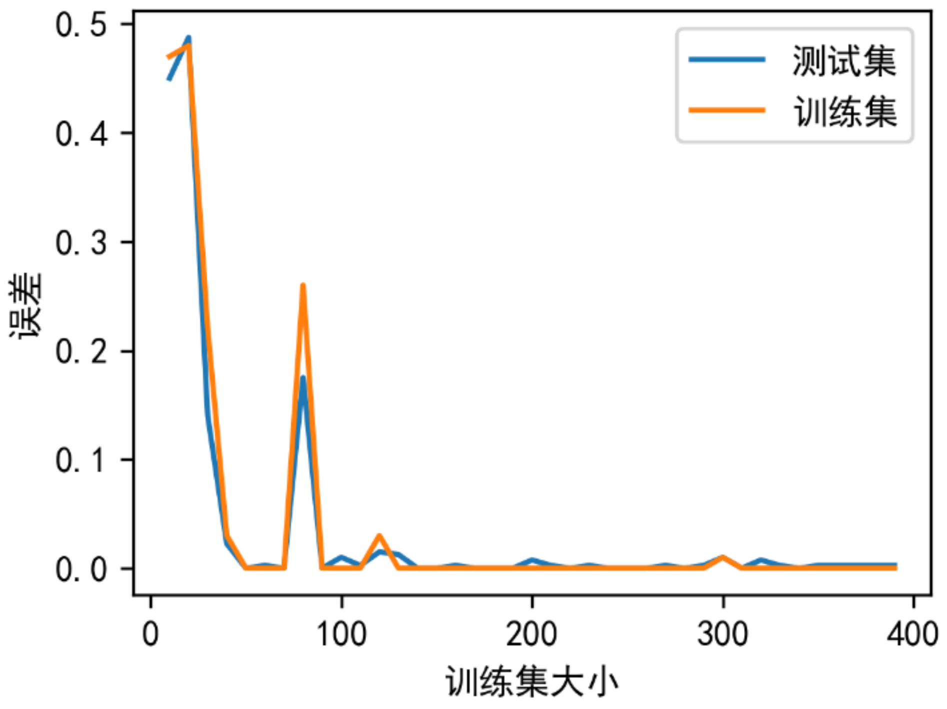
(f) 接下来，您将评估训练和测试误差如何随着训练集大小的增加而变化。对于集合{10,20,30,…,380,390,400} 中的每个 k 值，首先选择大小为 k 的训练数据的随机子集。然后使用您刚刚选择的训练数据的 k个随机子集重新训练您的逻辑回归分类器，并使用估计的参数来计算当前训练集（k 个随机实例）和原始测试集上的错误分类示例数“dataForTestingLogistic.txt”。最后，生成带有两条线的图：蓝色表示训练误差值与 k 的关系，红色表示测试误差值与 k 的关系，其中误差应位于纵轴上，训练集大小应位于位于水平轴上。确保在绘图中包含一个图例来标记两条线。描述随着训练集大小的加，训练和测试错误会发生什么变化，并解释为什么会发生这种行为。

答：代码及其注释如下所示（见下页）：

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
import random
import math
from matplotlib.pyplot import MultipleLocator

# 逻辑回归公式
def logistic(w, x):
    # dot: 矩阵乘法
    # 因为 w 和 x 都是 a 一维数组，所以返回的是向量的内积（一个数）
    temp = w.dot(x)
    temp = 1.0 / (1 + np.exp(-1 * temp))
    return temp

# 梯度下降
def gradient(train_data, label, rate, weight):
    sum = 0
    log_w = 0
    # 更新 weight
    for i in range(len(train_data)):
        sum += (label[i] - logistic(weight, train_data[i])) * train_data[i]
    weight = weight + rate * sum

    return weight

# 利用训练后的 weight 对训练集和测试集计算错误率和最大似然函数值
def predict(data, label, weight):
    error = 0
    log_w = 0
    # 计算最大似然函数值
    for i in range(len(data)):
        temp = logistic(weight, data[i])
        log_w += label[i] * math.log(logistic(weight, data[i])) + (1-label[i]) *
math.log(1 - logistic(weight, data[i]))
        # 预测错误则错误数+1
        if (temp >= 0.5 and label[i] == 0) or (temp < 0.5 and label[i] == 1):
            error = error + 1
    error /= len(data)
    return error, log_w


filename1 = "./dataForTrainingLogistic.txt"
filename2 = "./dataForTestingLogistic.txt"

# 训练集数据预处理
# train_num: 训练集数据数量
train_num = len(open(filename1,'r').readlines())
read_file = open(filename1)
train_lines = read_file.readlines()

list_all_train = []
list_element_train  = []
list_label_train  = []
# 遍历每行数据
for train_line in train_lines:
    list1 = train_line.split()
    for one in list1:
        # 将字符串转为浮点数
        list_element_train.append(float(one))
    # list_all_train 存储自变量（面积和距离，即 X 矩阵）
    list_all_train.append(list_element_train[0:6])
    # list_label_train 存储因变量（房价，即结果矩阵）
    list_label_train.append(list_element_train[6])
    list_element_train = []
# 给 list_all_train 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_train:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
train_data = np.array(list_all_train)
train_label = np.array(list_label_train)

# 测试集数据预处理
# test_num：训练集数据数量
test_num = len(open(filename2,'r').readlines())

read_test = open(filename2)
test_lines = read_test.readlines()

list_all_test = []
list_element_test = []
list_label_test = []
# 遍历每行数据
for test_line in test_lines:
    list2 = test_line.split()
    for one in list2:
        # 将字符串转为浮点数
        list_element_test.append(float(one))
    # list_all_test 存储自变量（面积和距离，即 X 矩阵）
    list_all_test.append(list_element_test[0:6])
    # list_label_test 存储因变量（房价，即结果矩阵）
    list_label_test.append(list_element_test[6])
    list_element_test = []
# 给 list_all_test 矩阵增加一个系数 theta0，初始值设为 1.0(因为对应着 x0=1 的系数，也就是
偏差 b)
for one in list_all_test:
    one.append(1.0)

# 均转为 numpy 数组类型，便于后续操作
test_data = np.array(list_all_test)
test_label = np.array(list_label_test)


'''
    参数初始化：
    weight：权重因子
    learn_rate：学习率
    x：自变量矩阵
    train_loss：训练集误差
    test_loss：测试集误差
```

```
'''
weight = np.zeros(7)
# 学习率
learn_rate = 0.00015
x = []
# 最大化似然函数值
train_log_w = []
test_log_w = []
# 误差（这里指错误率）
train_error = []
test_error = []

# 开始迭代学习
for i in range(10,400,10):
    weight = np.zeros(7)
    sample = []
    sample_label = []
    print("\n\n 当前训练集大小为：%d\n\n" % i)
    for j in range(0,i):
        m = random.randint(0, len(train_data) - 1)
        sample.append(train_data[m])
        sample_label.append(train_label[m])
        sample1 = np.array(sample)
        sample_label1 = np.array(sample_label)
    for k in range(1, 201):
        weight = gradient(sample1, sample_label1, 0.00015, weight)
        train_error1, train_log_w1 = predict(train_data, train_label, weight)
        test_error1, test_log_w1 = predict(test_data, test_label, weight)
        if (k % 20 == 0):
            print("当前迭代次数为：%d, 训练集误差为：%f, 测试集误差为：%f" % (k,
train_error1, test_error1))
    train_error.append(train_error1)
    test_error.append(test_error1)
    x.append(i)


# 画图
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换 sans-serif 字体
plt.rcParams['axes.unicode_minus'] = False  # 解决坐标轴负数的负号显示问题
plt.figure(figsize=(4,3),dpi=300)
plt.xlabel('训练集大小')
plt.ylabel('误差')
plt.plot(x, train_error,label='测试集')
plt.plot(x, test_error,label='训练集')
plt.legend()

plt.show()
```

代码运行结果如下（学习率为0.00015，不同大小的训练集的迭代次数均为200次）：



随着训练集大小的增加，误差（错误率）整体逐渐降低。但误差仍存在波动，原因是我们每次对于不同大小训练集的样本抽取是随机的，所以会导致误差波动。