Homework 3:
Linear Regression and Logistic Regression

Machine Learning and Data Mining (Fall 2023)

Student Name: 刘俊杰 Student ID: 21307174

Lectured by: Shangsong Liang
Sun Yat-sen University

Your assignment should be submitted to the email that will be provided by the TA
Deadline of your submission is: 23:59PM, November 20, 2023
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 “Homework 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“Homework 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）。

(1)分析:

首先写出线性回归的模型:

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

任取一个x^i,通过线性回归模型预测的值为:

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

此时的的损失值为:Loss=$h_\theta(x^i)$-y^i

更新权值 θ 为:

$$\theta_0\text{-=}\alpha h_\theta(x^i)\text{-}y^i$$
$$\theta_1\text{-=}(\alpha h_\theta(x^i)\text{-}y^i)x_1^i$$
$$\theta_2\text{-=}\alpha(h_\theta(x^i)\text{-}y^i)x_2^i$$

(其中α为学习率)

(2)代码实现:

①数据预处理:

```
# 从文件中加载数据
def load_data(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
    data = []
    for line in lines:
        values = line.strip().split(' ')
        data.append(list(map(float, values)))
    data = np.array(data)
    X = data[:, :2]  # 特征：面积和距离
    y = data[:, 2]   # 目标：房价
    return X, y
```

②利用测试集计算平均误差:

```
# 用测试数据集评估模型
def evaluate_model(X_test, y_test, theta):
    m = X_test.shape[0]
    X_bias = np.c_[np.ones((m, 1)), X_test]
    y_pred = np.dot(X_bias, theta)
    mse = ((y_test - y_pred) ** 2).mean()
    return mse
```

③梯度下降法更新权值:

```
def gradient_descent(X, y, learning_rate, epochs):
    m, n = X.shape
    theta = np.zeros(n + 1)  # 初始化参数向量（包括截距项）
    errors_train = []  # 保存训练误差
    errors_test = []   # 保存测试误差
    for epoch in range(epochs):
        X_bias = np.c_[np.ones((m, 1)), X]  # 添加截距项
        y_pred = np.dot(X_bias, theta)
        error = y_pred - y
        gradient = (1/m) * np.dot(X_bias.T, error)
        theta -= learning_rate * gradient
        if (epoch + 1) % 100000 == 0:
            mse_train = ((y - y_pred) ** 2).mean()
            mse_test = evaluate_model(X_test, y_test, theta)
            errors_train.append(mse_train)
            errors_test.append(mse_test)
            print("epoch:",epoch+1 ," train error:",mse_train," test error:",mse_test,"theta:",theta )
    return theta, errors_train, errors_test
```

完整代码:

```
import numpy as np
import matplotlib.pyplot as plt
# 从文件中加载数据
def load_data(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
    data = []
    for line in lines:
```

```
        values = line.strip().split(' ')
        data.append(list(map(float, values)))
    data = np.array(data)
    X = data[:, :2]  # 特征：面积和距离
    y = data[:, 2]   # 目标：房价
    return X, y
# 梯度下降法
def gradient_descent(X, y, learning_rate, epochs):
    m, n = X.shape
    theta = np.zeros(n + 1)  # 初始化参数向量（包括截距项）
    errors_train = []  # 保存训练误差
    errors_test = []   # 保存测试误差
    for epoch in range(epochs):
        X_bias = np.c_[np.ones((m, 1)), X]  # 添加截距项
        y_pred = np.dot(X_bias, theta)
        error = y_pred - y
        gradient = (1/m) * np.dot(X_bias.T, error)
        theta -= learning_rate * gradient
        if (epoch + 1) % 100000 == 0:
            mse_train = ((y - y_pred) ** 2).mean()
            mse_test = evaluate_model(X_test, y_test, theta)
            errors_train.append(mse_train)
            errors_test.append(mse_test)
            print("epoch:",epoch+1 ," train error:",mse_train," test error:",mse_test,"theta:",theta )
    return theta, errors_train, errors_test
# 用测试数据集评估模型
def evaluate_model(X_test, y_test, theta):
    m = X_test.shape[0]
    X_bias = np.c_[np.ones((m, 1)), X_test]
    y_pred = np.dot(X_bias, theta)
    mse = ((y_test - y_pred) ** 2).mean()
    return mse
# 加载训练数据和测试数据
X_train, y_train = load_data('D:\\d_code\\MLDM\\Lab3\\assignment 3\\dataForTrainingLinear.txt')
X_test, y_test = load_data('D:\\d_code\\MLDM\\Lab3\\assignment 3\\dataForTestingLinear.txt')
# 设置超参数
learning_rate = 0.00015
epochs = 1500000
# 训练模型
theta, errors_train, errors_test = gradient_descent(X_train, y_train, learning_rate, epochs)
# 绘制训练误差和测试误差的图表
iterations = np.arange(100000, epochs + 1, 100000)
plt.plot(iterations, errors_train, label='Training Error')
plt.plot(iterations, errors_test, label='Testing Error')
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.title('Training and Testing Error vs. Iterations')
plt.show()
```

(a) How many parameters do you use to tune this linear regression model? Please use Gradient De- scent 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 train- ing, 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、

[1] To see what are “Gradient Descent” and “Stochastic Gradient Descent”, please refer to pages 38 and 39 of the slides “linear__model.pptx” , respectively.

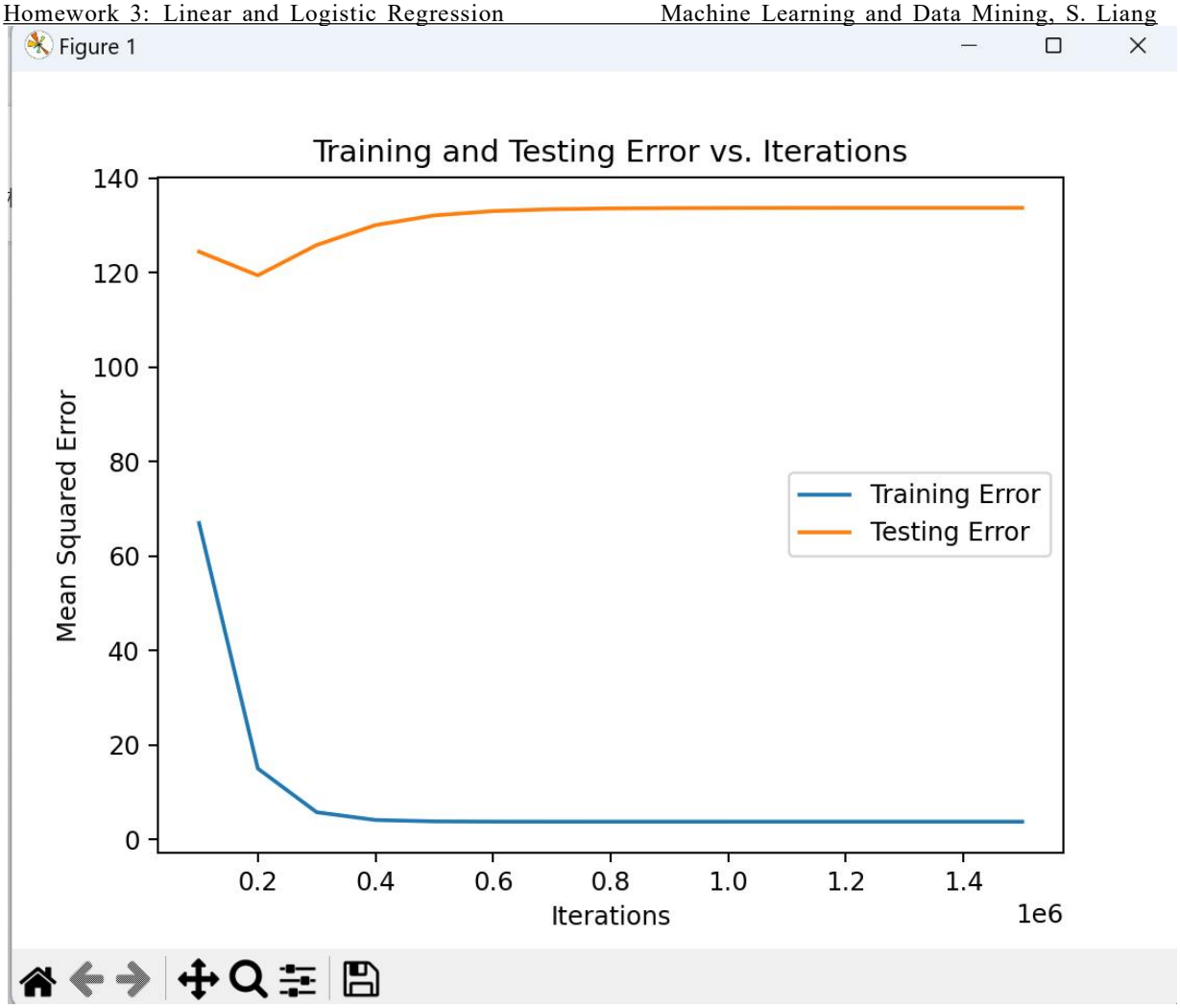
……1500000 时对应的训练样本的误差和测试样本对应的误差（图可以手画， 或者用工具画图）。从画出 的图中，你发现什么？请简单分析。

学习率为0.00015，迭代1500000:次

不同迭代次数后的训练集平均误差、测试集平均误差和权值:

```
PS D:\d_code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "d:\d_code\MLDM\Lab3\assignment 3\ex1.py"
epoch: 100000  train error: 67.00879527061903  test error: 124.46329858931401 theta: [ 46.32960907   7.089390
25 -72.7598862 ]
epoch: 200000  train error: 15.059483400505997  test error: 119.45529167924049 theta: [ 65.48726377   6.90006
303 -72.54075178]
epoch: 300000  train error: 5.816134008292042  test error: 125.87211889837913 theta: [ 73.56830135   6.820201
46 -72.448317  ]
epoch: 400000  train error: 4.171463368570337  test error: 130.09646859880718 theta: [ 76.97702593   6.786514
44 -72.40932638]
epoch: 500000  train error: 3.8788268578872063  test error: 132.14840411638548 theta: [ 78.41488624   6.77230
465 -72.39287945]
epoch: 600000  train error: 3.82675799661099  test error: 133.06199330784196 theta: [ 79.02140115   6.7663107
2 -72.38594184]
epoch: 700000  train error: 3.8174933757736924  test error: 133.45591030800296 theta: [ 79.27723986   6.76378
237 -72.38301543]
epoch: 800000  train error: 3.81584492030241  test error: 133.6235925765032 theta: [ 79.38515716   6.76271587
 -72.38178102]
epoch: 900000  train error: 3.815551610356032  test error: 133.69459457452726 theta: [ 79.43067858   6.762266
   -72.38126033]
epoch: 1000000  train error: 3.8154994216702454  test error: 133.72459263011538 theta: [ 79.44988032   6.7620
7623 -72.38104069]
epoch: 1100000  train error: 3.815490135729033  test error: 133.73725490910758 theta: [ 79.45797996   6.76199
619 -72.38094804]
epoch: 1200000  train error: 3.8154884834800264  test error: 133.74259760694827 theta: [ 79.46139653   6.7619
6242 -72.38090896]
epoch: 1300000  train error: 3.8154881894950945  test error: 133.74485152266237 theta: [ 79.4628377    6.7619
4818 -72.38089248]
epoch: 1400000  train error: 3.8154881371863105  test error: 133.74580231243954 theta: [ 79.46344561   6.7619
4217 -72.38088552]
epoch: 1500000  train error: 3.815488127878956  test error: 133.74620338098063 theta: [ 79.46370204   6.76193
964 -72.38088259]
```

可视化后：



分析:

可以看到在大约迭代30000次之前，训练集和测试集上的误差都一直在减小，而在30000次之后训练集的误差基本不变、的测试集误差变大，说明300000次之后出现了过拟合。

(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。

学习率改为0.0002,出现问题:

```
PS D:\d_code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "d:\d_code\MLDM\Lab3\assignment 3\ex1.py"
d:\d_code\MLDM\Lab3\assignment 3\ex1.py:26: RuntimeWarning: invalid value encountered in subtract
  theta -= learning_rate * gradient
epoch: 100000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 200000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 300000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 400000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 500000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 600000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 700000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 800000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 900000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 1000000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 1100000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 1200000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 1300000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 1400000  train error: nan  test error: nan theta: [nan nan nan]
epoch: 1500000  train error: nan  test error: nan theta: [nan nan nan]
PS D:\d_code>
```

提高学习率可能会导致这类问题，这是因为较高的学习率可能导致模型在训练过程中过快地跳过局部最优解，计算出的数字太大, 导致模型无法充分拟合训练数据。

(c) Now, we turn to use other optimization methods to get the optimal parameters. Can you use Stochas- tic 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?

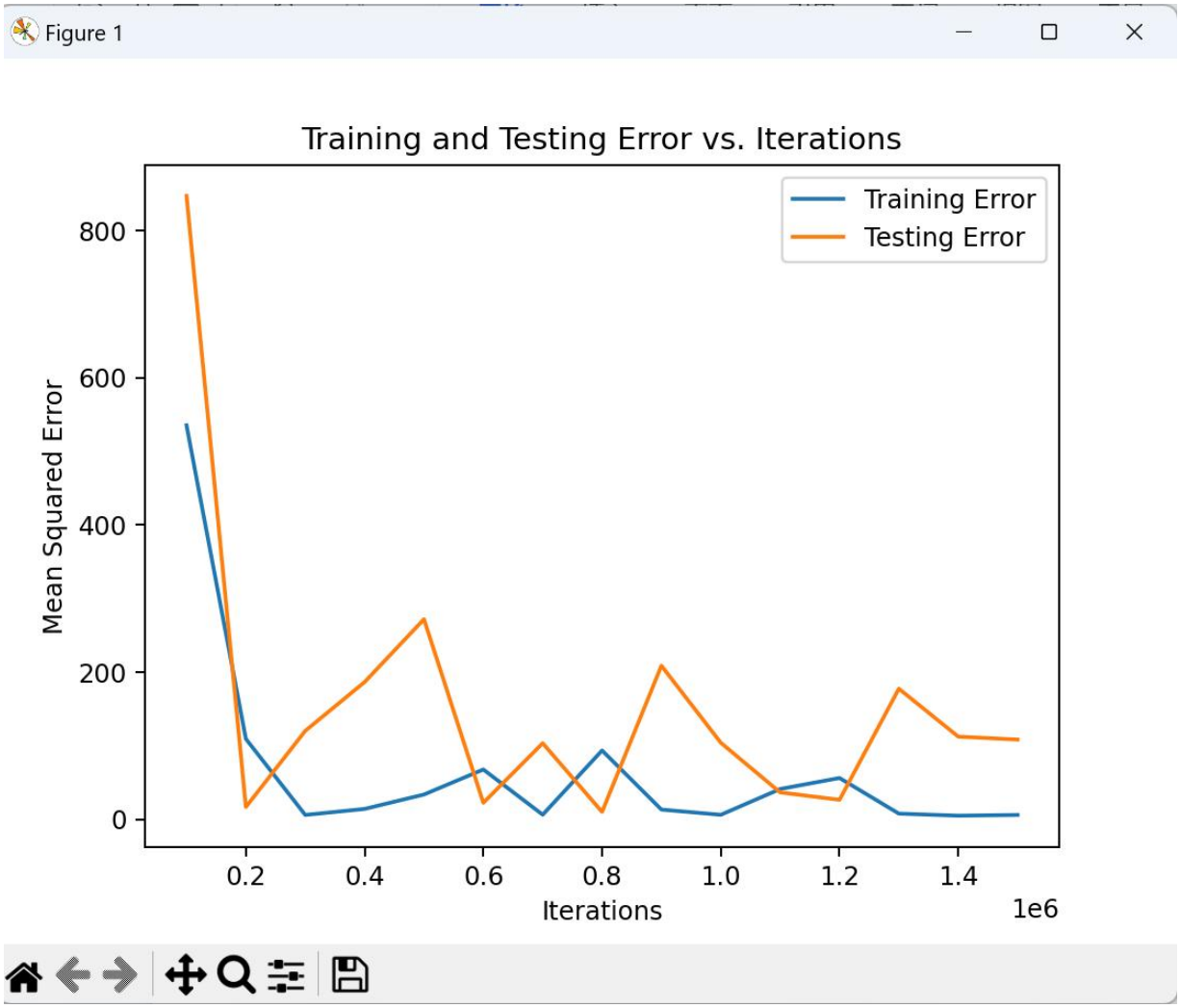
现在， 我们使用其他方法来获得最优的参数。你是否可以用随机梯度下降法获得最优的参数？ 请使用随 机梯度下降法画出迭代次数（每 K 次， 这里的 K 你自己设定） 与训练样本和测试样本对应的误差的图。 比较 Exercise 1 中的实验图，请总结你的发现。

①每次下降随机选择一个训练集样本:

```python
# 随机梯度下降法
def stochastic_gradient_descent(X, y, learning_rate, epochs):
    m, n = X.shape
    theta = np.zeros(n + 1)  # 初始化参数向量（包括截距项）
    errors_train = []  # 保存训练误差
    errors_test = []   # 保存测试误差
    for epoch in range(epochs):
        random_idx=[random.randint(0,m-1)]
        for i in random_idx:
            xi = X[i]
            yi = y[i]
            X_bias = np.insert(xi, 0, 1)
            y_pred = np.dot(X_bias, theta)
            error = y_pred - yi
            gradient =  error * X_bias
            theta -= learning_rate * gradient
        if (epoch + 1) % 100000 == 0:
            mse_train = evaluate_model(X_train, y_train, theta)
            mse_test = evaluate_model(X_test, y_test, theta)
            errors_train.append(mse_train)
            errors_test.append(mse_test)
            print("epoch:", epoch + 1, " train error:", mse_train, " test error:", mse_test, "theta:", theta)
    return theta, errors_train, errors_test
```

迭代**1500000**次结果:

```
epoch: 100000  train error: 535.5495550588028  test error: 847.4452386556735 theta: [ 45.6812091    7.29255346 -72.56262685]
epoch: 200000  train error: 109.31118928034404  test error: 16.733616452746467 theta: [ 65.04197743   6.81421567 -72.59590234]
epoch: 300000  train error: 5.919706606770924  test error: 120.23995597594899 theta: [ 73.56050904   6.81379978 -72.32204597]
epoch: 400000  train error: 14.089901429508597  test error: 186.58443355347836 theta: [ 76.99780577   6.80186179 -72.13151329]
epoch: 500000  train error: 33.68630541424996  test error: 272.0272307958703 theta: [ 78.53542969   6.82504717 -72.42249015]
epoch: 600000  train error: 67.70660545908149  test error: 22.41058926735071 theta: [ 79.06752935   6.69140941 -72.42959049]
epoch: 700000  train error: 6.246044558674272  test error: 103.45451499635048 theta: [ 79.16206843   6.75048506 -72.39432534]
epoch: 800000  train error: 93.60592061179476  test error: 10.072090347502135 theta: [ 79.21381224   6.66488609 -72.20993521]
epoch: 900000  train error: 13.211872220946951  test error: 208.7811509289583 theta: [ 79.13700259   6.79851708 -72.46464419]
epoch: 1000000  train error: 6.123393986087086  test error: 104.10063644562663 theta: [ 79.19149761   6.75043431 -72.3912272 ]
epoch: 1100000  train error: 41.171496144557075  test error: 36.56624882480759 theta: [ 79.23931255   6.70233853 -72.31945832]
epoch: 1200000  train error: 56.152551208701006  test error: 26.474750117608732 theta: [ 79.28981745   6.6917788  -72.33318221
]
epoch: 1300000  train error: 7.725063231679923  test error: 177.50827607392077 theta: [ 79.36799313   6.78144588 -72.37391986]
epoch: 1400000  train error: 4.918179443655794  test error: 112.41318598309215 theta: [ 79.42695566   6.75172603 -72.37120191]
epoch: 1500000  train error: 5.93105851823598  test error: 108.35963237430936 theta: [ 79.46841049   6.75155209 -72.45078315]
```



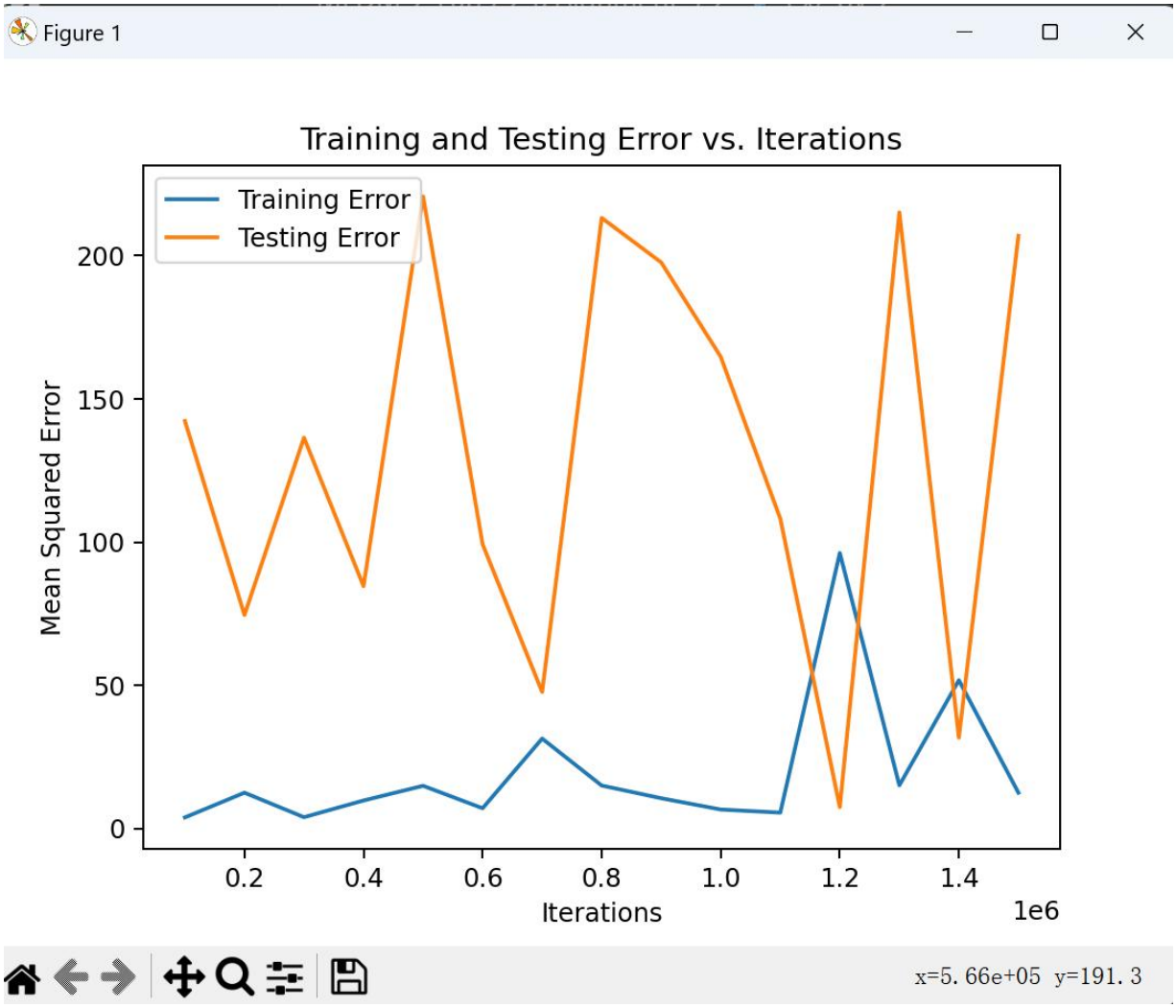
②每次下降随机选择一组训练集样本(10个)

```python
# 随机梯度下降法
def stochastic_gradient_descent(X, y, learning_rate, epochs):
    m, n = X.shape
    theta = np.zeros(n + 1)  # 初始化参数向量（包括截距项）
    errors_train = []  # 保存训练误差
    errors_test = []   # 保存测试误差
    for epoch in range(epochs):
        random_idx = np.random.permutation(m)
        random_idx = random_idx[0:10]
        for i in random_idx:
            xi = X[i]
            yi = y[i]
            X_bias = np.insert(xi, 0, 1)
            y_pred = np.dot(X_bias, theta)
            error = y_pred - yi
            gradient =  error * X_bias
            theta -= learning_rate * gradient
        if (epoch + 1) % 100000 == 0:
```

```
            mse_train = evaluate_model(X_train, y_train, theta)
            mse_test = evaluate_model(X_test, y_test, theta)
            errors_train.append(mse_train)
            errors_test.append(mse_test)
            print("epoch:", epoch + 1, " train error:", mse_train, " test error:", mse_test, "theta:", theta)
    return theta, errors_train, errors_test
```

迭代1500000次结果:

```
PS D:\d_code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "d:\d_code\MLDM\Lab3\assignment 3\ex1_2.py"
epoch: 100000  train error: 3.9519612894033402  test error: 142.29684918397083 theta: [ 79.47491751   6.76623313 -72.39840636]
epoch: 200000  train error: 12.579755689417084  test error: 74.59223782939104 theta: [ 79.35131246   6.72718255 -72.22856397]
epoch: 300000  train error: 4.023662598442559  test error: 136.48650906949032 theta: [ 79.48309886   6.75969618 -72.27022362]
epoch: 400000  train error: 9.851764526641228  test error: 84.65824247052589 theta: [ 79.51464991   6.7332898  -72.28695551]
epoch: 500000  train error: 14.97784987681724  test error: 220.73810970053387 theta: [ 79.45718944   6.80189842 -72.54189991]
epoch: 600000  train error: 7.14798600195353  test error: 99.2769935218166 theta: [ 79.6154612    6.74346075 -72.38964657]
epoch: 700000  train error: 31.4653081579019  test error: 47.727821826398156 theta: [ 79.45967284   6.71125455 -72.37526165]
epoch: 800000  train error: 15.07076503936787  test error: 213.1971790920098 theta: [ 79.471537     6.79431465 -72.38618702]
epoch: 900000  train error: 10.643439501447057  test error: 197.66060156525623 theta: [ 79.34397009   6.79225946 -72.46495193]
epoch: 1000000  train error: 6.706697980287138  test error: 164.71213794420797 theta: [ 79.38743558   6.77257528 -72.25897977]
epoch: 1100000  train error: 5.619374901554384  test error: 108.2426556634538 theta: [ 79.5318948    6.74931845 -72.3985907 ]
epoch: 1200000  train error: 96.24096438297296  test error: 7.528135907194789 theta: [ 79.35547796   6.65437163 -72.04811274]
epoch: 1300000  train error: 15.151375012097091  test error: 215.126905805514 theta: [ 79.40530842   6.7967084  -72.41956282]
epoch: 1400000  train error: 51.826907412452286  test error: 31.771846792582643 theta: [ 79.51465261   6.69780172 -72.43647129
]
epoch: 1500000  train error: 12.568466213871602  test error: 206.96659387423892 theta: [ 79.57408128   6.79326317 -72.46137006
]
```



可以从以上两次的结果看出，在迭代的过程中模型的效果出现了很大的波动，并没有收敛到全局最优解。这是因为随机梯度下降

因为其随机性，导致在一定的迭代过程中在局部最优解附近波动，没有稳定地逼近全局最优解。

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 "Homework 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.

(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 $x_1^{(i)}, \ldots, x_p^{(i)}$, and the parameters $w_0, w_1, \ldots, w_p$, where the superscript (i) denotes the sample index. This will be your objective function for gradient ascent.

$$\mathbf{L(w)=\sum_{i=1}^{n}\{y^i log[P(wx^i=1)|x^i,w]+(1-y^i)log[P(wx^i=0)|x^i,w]\}}$$

(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.

不妨设$\mathbf{sigmoid(wx^i)=h_\theta(x^i)}$，L(w)为f。

$$\mathbf{f=\sum_{i=1}^{n} y^i log h_\theta(x^i)+(1-y^i)log(1-h_\theta(x^i))}$$

①求$\frac{\partial f}{\partial w_0}$

$$\frac{\partial f}{\partial w_0}=\sum_{i=1}^{n} y^i \frac{1}{h_\theta(x^i)}\frac{\partial h_\theta(x^i)}{\partial w_0}-(1-y^i)\frac{1}{1-h_\theta(x^i)}\frac{\partial h_\theta(x^i)}{\partial w_0}$$

将$\frac{\partial h_\theta(x^i)}{\partial w_0}$=(1-$h_\theta(x^i)$)$h_\theta(x^i)$带入得到

$$\frac{\partial f}{\partial w_0}=\sum_{i=1}^{n}[y^i \frac{1}{h_\theta(x^i)}(1-h_\theta(x^i))h_\theta(x^i)-(1-y^i)\frac{1}{1-h_\theta(x^i)}(1-h_\theta(x^i))h_\theta(x^i)\ ]$$

化简后得到:$\frac{\partial f}{\partial w_0} = \sum_{i=1}^{n} y - h_\theta(x^i)$

②求$\frac{\partial f}{\partial w_j}$

$$\frac{\partial f}{\partial w_j} = \sum_{i=1}^{n} y^i \frac{1}{h_\theta(x^i)} \frac{\partial h_\theta(x^i)}{\partial w_j} - (1 - y^i) \frac{1}{1 - h_\theta(x^i)} \frac{\partial h_\theta(x^i)}{\partial w_j}$$

将$\frac{\partial h_\theta(x^i)}{\partial w_j} = (1-h_\theta(x^i))h_\theta(x^i)x_j^i$带入得到:

$$\frac{\partial f}{\partial w_j} = \sum_{i=1}^{n} y^i \frac{1}{h_\theta(x^i)} (1 - h_\theta(x^i))h_\theta(x^i)x_j^i - (1 - y^i) \frac{1}{1 - h_\theta(x^i)} (1 - h_\theta(x^i))h_\theta(x^i)x_j^i$$

化简后得到:$\frac{\partial f}{\partial w_j} = \sum_{i=1}^{n} (y^i - h_\theta(x^i))\, x_j^i$

(c) Train your logistic regression classifier on the data provided in the training dataset "dataForTrain- ingLogistic.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).

①加载数据

```
# 加载训练集和测试集
train_data=np.loadtxt('D:\\d_code\\MLDM\\Lab3\\assignment 3\\dataForTrainingLogistic.txt')
test_data=np.loadtxt('D:\\d_code\\MLDM\\Lab3\\assignment 3\\dataForTestingLogistic.txt')
# 提取特征和标签
X_train, y_train = train_data[:, :-1], train_data[:, -1]
X_test, y_test = test_data[:, :-1], test_data[:, -1]
```

②sigmoid函数

```
# Sigmoid 函数
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

③梯度上升

```
# 训练逻辑回归分类器
for iteration in range(iterations):
    objective = 0
    for i in range(len(X_train)):
        xi = np.insert(X_train[i], 0, 1)  # 增加偏置项
        zi = np.dot(w, xi)
        predicted = sigmoid(zi)
        gradient = xi * (predicted - y_train[i])
        w -= learning_rate * gradient
        objective += y_train[i] * np.log(predicted) + (1 -
y_train[i]) * np.log(1 - predicted)
    objective_values.append(objective)
```

④可视化

```
# 可视化结果
plt.plot(range(iterations), objective_values)
plt.xlabel('Iteration')
plt.ylabel('Objective Function Value')
plt.title('Objective Function Value vs. Iteration')
plt.show()
```

完整代码:

```
import numpy as np
import matplotlib.pyplot as plt
# 加载训练集和测试集
train_data=np.loadtxt('D:\\d_code\\MLDM\\Lab3\\assignment
3\\dataForTrainingLogistic.txt')
test_data=np.loadtxt('D:\\d_code\\MLDM\\Lab3\\assignment
3\\dataForTestingLogistic.txt')
# 提取特征和标签
X_train, y_train = train_data[:, :-1], train_data[:, -1]
X_test, y_test = test_data[:, :-1], test_data[:, -1]
# Sigmoid 函数
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
# 计算预测结果
def predict(X, w):
    X = np.column_stack((np.ones(X.shape[0]), X))  # 加偏置项
    scores = np.dot(X, w)
    predicted_labels = np.round(sigmoid(scores))
    return predicted_labels
# 初始化参数
p = X_train.shape[1]
```

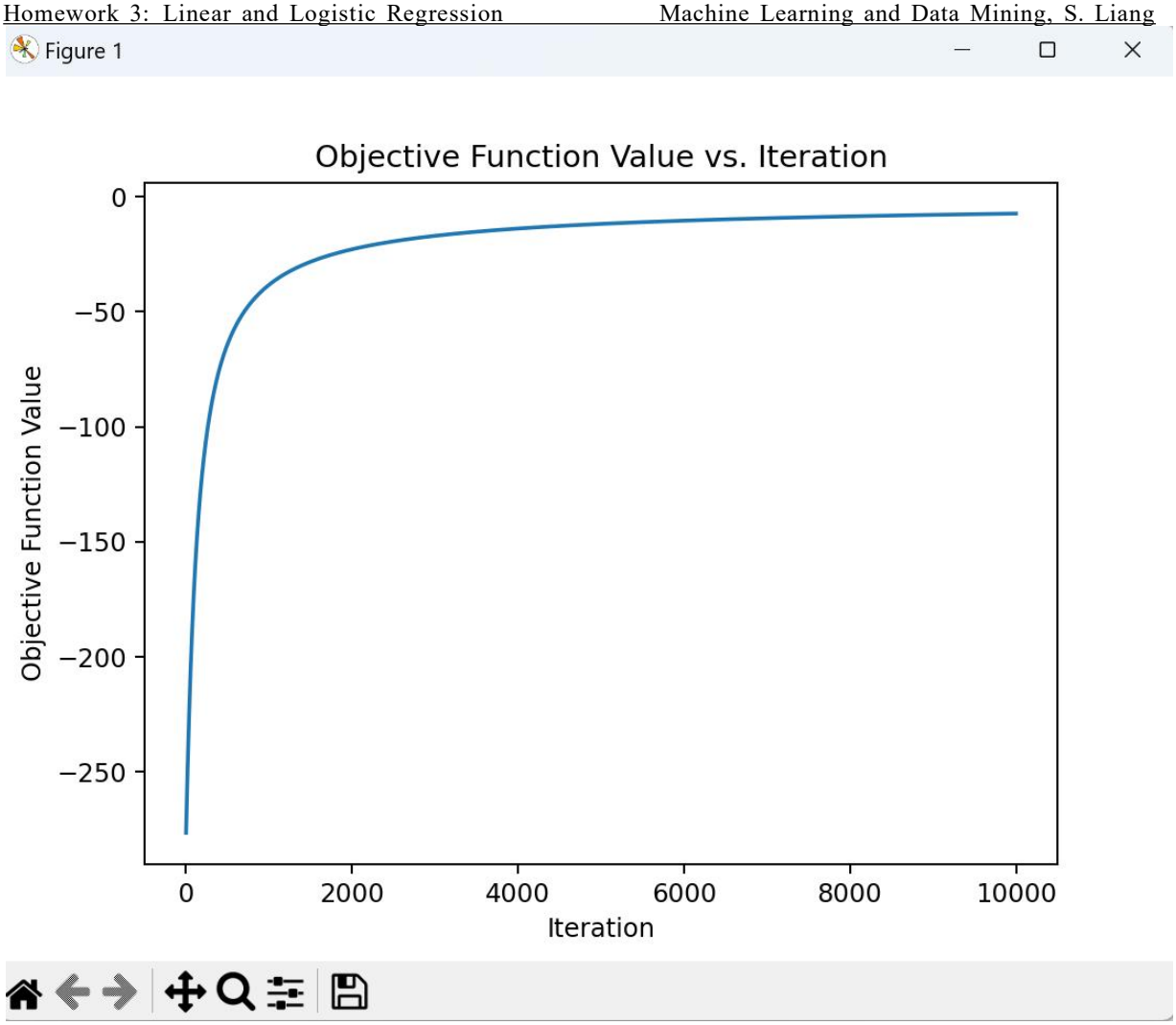
```
w = np.zeros(p + 1)  # 加截距项
# 超参数
learning_rate = 0.00015
iterations = 1000
# 存储目标函数
objective_values = []
# 训练逻辑回归分类器
for iteration in range(iterations):
    objective = 0
    for i in range(len(X_train)):
        xi = np.insert(X_train[i], 0, 1)  # 增加偏置项
        zi = np.dot(w, xi)
        predicted = sigmoid(zi)
        gradient = xi * (predicted - y_train[i])
        w -= learning_rate * gradient
        objective += y_train[i] * np.log(predicted) + (1 -
y_train[i]) * np.log(1 - predicted)
    objective_values.append(objective)
# 统计测试集错分数目
y_pred = predict(X_test, w)
misclassified = np.sum(y_pred != y_test)
print(f"Number of misclassified examples in the testing dataset:
{misclassified}")
accuracy = (p-misclassified)/p*100
print(f"Accuracy: {accuracy}")
# 可视化结果
plt.plot(range(iterations), objective_values)
plt.xlabel('Iteration')
plt.ylabel('Objective Function Value')
plt.title('Objective Function Value vs. Iteration')
plt.show()
```

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

```
PS D:\d_code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "d:\d_code\MLDM\Lab3\assignme
nt 3\ex2.py"
Number of misclassified examples in the testing dataset: 0
Accuracy:  1.0
PS D:\d_code>
```

最后在测试集上分类错误的个数是0。

(e) Plot the value of the objective function on each iteration of stochastic gradient ascent, with the iter- ation 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.



可以看到在大约7000次左右达到收敛

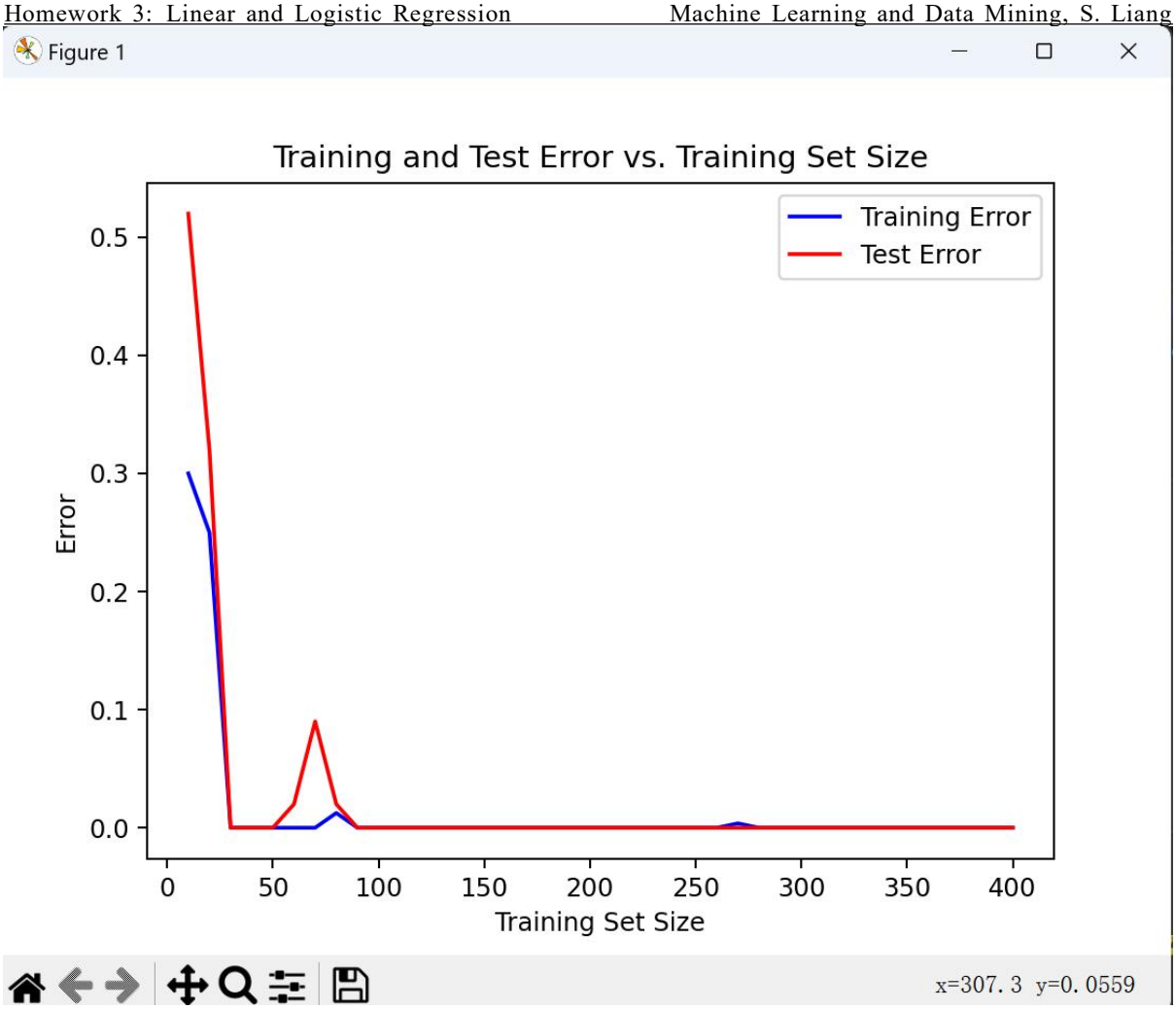
(f) Next, you will evaluate how the training and test error change as the training set size increases. For each value of k in the set $\{10, 20, 30, \ldots, 380, 390, 400\}$, first choose a random subset of the training data of size k. Then re-train your logistic regression classifier using the k random 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.

代码实现:

```
# 记录随着训练子集增大误差的变化
training_sizes = list(range(10, 401, 10))
train_errors = []
test_errors = []
for size in training_sizes:
    # 随机选择一组数据
    random_indices = np.random.choice(len(X_train), size,
replace=False)
    X_subset = X_train[random_indices]
    y_subset = y_train[random_indices]
```

```
    # 在随机选择数据上训练
    w_subset = np.zeros(p + 1)
    for iteration in range(iterations):
        objective = 0
        for i in range(size):
            xi = np.insert(X_subset[i], 0, 1)  # 加偏置项
            zi = np.dot(w_subset, xi)
            predicted = sigmoid(zi)
            gradient = xi * (predicted - y_subset[i])
            w_subset -= learning_rate * gradient
            objective += y_subset[i] * np.log(predicted) +
(1 - y_subset[i]) * np.log(1 - predicted)
    # 预测测试集
    y_pred_subset = predict(X_test, w_subset)
    # 统计训练集和测试集误差
    train_errors.append(np.sum(predict(X_subset, w_subset)
!= y_subset) / size)
    test_errors.append(np.sum(y_pred_subset != y_test) /
len(y_test))
# 可视化
plt.plot(training_sizes, train_errors, label='Training
Error', color='blue')
plt.plot(training_sizes, test_errors, label='Test Error',
color='red')
plt.xlabel('Training Set Size')
plt.ylabel('Error')
plt.title('Training and Test Error vs. Training Set Size')
plt.legend()
plt.show()
```

可视化结果：



分析:

使用不同大小的随机选择的训练子集训练逻辑回归模型，观察结果:

①可以看到当训练子集的规模较小时，测试误差和训练误差都较大这是因为训练的子集较小，每次训练对模型的波动大，训练出来的模型没有很好的泛化能力，对测试集的未知数据的的预测准确率低，而且对训练集也没有达到很好的拟合效果。

②当训练子集的规模变大时，测试误差和训练误差都减小甚至可以收敛到0，说明该模型的泛化能力得到了增强，对测试集未知数据的预测准确率提高。