中山大学计算机学院

机器学习与数据挖掘

本科生实验报告

（2023学年秋季学期）

课程名称：Machine Learning & Data Mining

教学班级	专业(方向)	学号	姓名
计科二班	计算机科学与技术	21307185	张礼贤

EX1:Linear Regression

Task1

问题表述

- 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

问题解答

- **迭代步骤及公式推导** 使用多变量线性回归模型进行求解，其公式如下所示：

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

即总共需要三个参数，在初始条件下，将三个参数都设置为0，学习率设置为0.00015，使用随机梯度下降法进行训练，迭代的规则与步骤如下：

1. 计算预测值 $h(x)$：

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

2. 随机选择一个样本 $(x^{(i)}, y^{(i)})$：

3. 计算预测值 $h(x^{(i)})$：

$$h(x^{(i)}) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_0$$

4. 计算误差（第 i 个样本的损失）：

$$\text{error}^{(i)} = h(x^{(i)}) - y^{(i)}$$

5. 更新模型参数：

$$\theta_1 := \theta_1 - \alpha \times \text{error}^{(i)} \times x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \times \text{error}^{(i)} \times x_2^{(i)}$$

$$\theta_0 := \theta_0 - \alpha \times \text{error}^{(i)}$$

其中，α 是学习率（learning rate），用于控制参数更新的步长。

6. 重复步骤 3 到步骤 6，直到达到设定的迭代次数或损失函数收敛到一个满意的程度。

- **代码实现:**

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

# 读取训练数据
train_data =
np.loadtxt('homework3\dataForTrainingLinear.txt')
X_train = train_data[:, :-1]  # 特征
y_train = train_data[:, -1]    # 目标

# 添加偏置项到特征矩阵
X_train = np.c_[np.ones(X_train.shape[0]), X_train]

# 读取测试数据
test_data =
np.loadtxt('homework3\dataForTestingLinear.txt')
X_test = test_data[:, :-1]
y_test = test_data[:, -1]
X_test = np.c_[np.ones(X_test.shape[0]), X_test]

# 定义随机梯度下降函数
def gradient_descent(X, y, learning_rate=0.00015,
epochs=1500000):
    theta = np.zeros(X.shape[1])  # 初始化参数
    training_errors = []  # 用于保存每迭代100000步的训
练误差
    test_errors = []  # 用于保存每迭代100000步的测试误差
    predictions = []  # 用于保存每迭代100000步的预测值
    for epoch in range(1, epochs + 1):
        error = y - np.dot(X, theta)
        theta += learning_rate * (X.T.dot(error) /
X.shape[0])
        # 计算每迭代100000步的训练误差、测试误差和预测值
        if epoch % 100000 == 0:
            training_errors.append(np.mean(error **
2))
            test_error = y_test - np.dot(X_test,
theta)
            test_errors.append(np.mean(test_error **
2))
            predictions.append(np.dot(X_test, theta))
            print(f"--------epoch:{epoch / 100000}---
-----")
            print(f"training_error:
{training_errors[-1]}")
            print(f"testing_error:{test_errors[-1]}")
            print(f"Parameters (theta):[{theta}]")
```

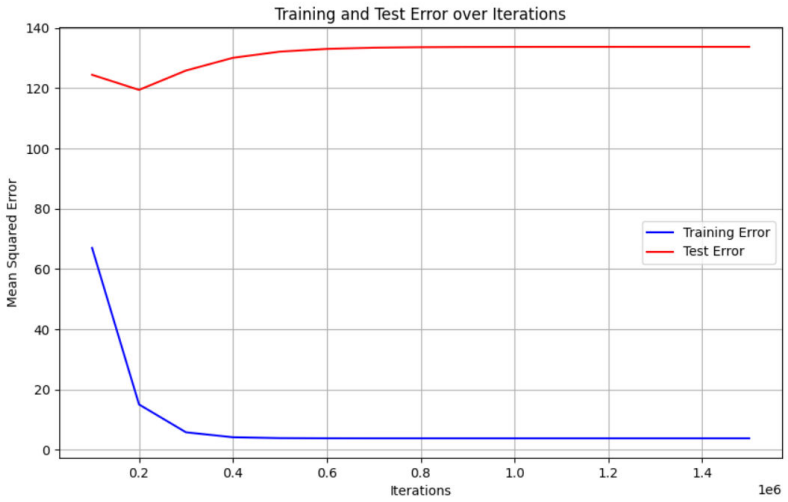
```
    return theta, training_errors, test_errors,
predictions

# 训练模型并获取参数、训练误差列表、测试误差列表和预测值列
表
theta, training_errors, test_errors, predictions =
gradient_descent(X_train, y_train)

# 绘制训练误差和测试误差随迭代次数的变化图
plt.figure(figsize=(10, 6))
plt.plot(range(100000, 1500001, 100000),
training_errors, color='blue', label='Training
Error')
plt.plot(range(100000, 1500001, 100000), test_errors,
color='red', label='Test Error')
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Training and Test Error over Iterations')
plt.legend()
plt.grid(True)
plt.show()

# 绘制预测值和实际值的可视化
plt.figure(figsize=(10, 6))
for i in range(len(predictions)):
    plt.plot(y_test, predictions[i], marker='o',
linestyle='-', label=f'Iteration {i+1}')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual Prices vs. Predicted Prices')
plt.legend()
plt.grid(True)
plt.show()

print("The final Parameters (theta):", theta)
```

- **实验结果分析：**
  - **迭代结果：**

Epoch	Training Error	Testing Error	Parameters
1.0	67.0088	124.4633	[46.3296, 7.0894, -72.7599]
2.0	15.0595	119.4553	[65.4873, 6.9001, -72.5408]
3.0	5.8161	125.8721	[73.5683, 6.8202, -72.4483]
4.0	4.1715	130.0965	[76.9770, 6.7865, -72.4093]
5.0	3.8788	132.1484	[78.4149, 6.7723, -72.3929]
6.0	3.8268	133.0620	[79.0214, 6.7663, -72.3859]
7.0	3.8175	133.4559	[79.2772, 6.7638, -72.3830]
8.0	3.8158	133.6236	[79.3852, 6.7627, -72.3818]
9.0	3.8156	133.6946	[79.4307, 6.7623, -72.3813]
10.0	3.8155	133.7246	[79.4499, 6.7621, -72.3810]
11.0	3.8155	133.7373	[79.4580, 6.7620, -72.3810]
12.0	3.8155	133.7426	[79.4614, 6.7620,

Epoch	Training Error	Testing Error	Parameters
			-72.3809]
13.0	3.8155	133.7449	[79.4628, 6.7619, -72.3809]
14.0	3.8155	133.7458	[79.4634, 6.7619, -72.3809]
15.0	3.8155	133.7462	[79.4637, 6.7619, -72.3809]

- **误差曲线：**



- **预测值与实际值比较：**

- **结论：**

- 从图表中，我们可以看到随着迭代次数的增加，训练样本的误差逐渐减小，而测试样本的误差在一定程度上也减小。
- 然而，随着迭代次数的继续增加，测试样本的误差开始趋于稳定，甚至在某个点之后可能略微增加。这可能表明模型在一定程度上过拟合了训练数据，导致在测试数据上的性能没有明显提升，甚至出现了轻微的退化。
- 因此，在1500000次迭代后，模型的性能没有进一步改善，甚至可能变得更差。

Task2

问题表述

改变学习率，比如把学习率改成 0.0002，然后训练该回归模型。你有什么发现？请简单分析

问题解答

当把学习率设置为0.0002时，出现以下报错：

```
d:\桌面\机器学习\homework3\hw3.py:26: RuntimeWarning: invalid value encountered in add
  theta += learning_rate * (X.T.dot(error) / X.shape[0])
--------epoch:1.0--------
training_error:nan testing_error:nan
Parameters (theta):[[nan nan nan]]
```

出现上面的问题是因为学习率调整的过大，导致最后无法进行拟合

Task3

问题描述

- 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

问题解答

- 不可以使用随机梯度下降法获得最优的参数，由于随机性，SGD并不保证在有限的迭代次数内收敛到全局最优解。它可能在局部最优解附近波动，而不是稳定地向全局最优解收敛

- 使用随机梯度下降法的实验结果如上图所示，无论是训练误差还是测试误差都是震荡下降，虽然有递减的趋势，但是只是在局部最优解附近波动，未必能按照最优的结果收敛

EX2：Logistic Regression

Task1

问题描述

- 什么是条件对数似然
- 如何用公式表示条件对数似然

问题解决

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。假设我们有一个包含 n 个训练样本和 p 个特征的数据集

1. **条件对数似然**
    - 条件对数似然是指，在给定输入特征 x 的情况下，我们预测观测到特定类别 y 的对数似然。也就是说，它表示了在已知输入特征的情况下，我们所预测的类别的概率的对数。
2. **用公式表示条件对数似然**
    - 假设我们的样本标签（类别）为 $y(i)$，对应的特征为 $x(1i), x(2i), \ldots, x(pi)$，参数为 $w_0, w_1, \ldots, w_p$。那么，条件对数似然可以用如下公式表示：

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

其中，$P(y(i) = 1|x(i); w)$ 表示在给定输入特征 $x(i)$ 和参数 w 的情况下，样本 i 属于类别 1 的概率。而 $P(y(i) = 0|x(i); w)$ 则表示样本 i 属于类别 0 的概率。

Task2

问题描述

计算逻辑回归目标函数对 w_0 和任意特定 w_j 的偏导数，其中 f 是上面提供的目标函数。同时，将这些偏导数表达式写成有限和的形式

问题解决

逻辑回归的目标函数为：

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

即上面第一问的对数似然函数

其中，$h_\theta(x^{(i)}) = \frac{1}{1+e^{-(w_0 + w_1 x_1^{(i)} + \ldots + w_j x_j^{(i)} + \ldots + w_p x_p^{(i)})}}$ 是表示假设函数的 sigmoid 函数。

现在，我们分别计算对 w_0 和 w_j 的偏导数。

1. 对 w_0 的偏导数：

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

   我们知道 sigmoid 函数的导数为：

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

   代入上式，得到对 w_0 的偏导数为：

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

2. 对 w_j（其中 j 为任意特征索引）的偏导数：

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

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

代入上式，得到对 w_j 的偏导数为：

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

Task3

问题描述

- 如何设计和训练逻辑回归分类器？
- 在你的逻辑回归分类器中的最优的估计参数是什么？
- 使用你估计的参数来计算测试数据集“dataForTestingLogistic。txt”中数据的预测标签

问题解决

选取逻辑回归分类器进行问题的求解，下面是代码及相应的解释

1. 数据加载和预处理：

```
import numpy as np

# 读取训练数据
train_data =
np.loadtxt('homework3\dataForTrainingLogistic.txt')
X_train = train_data[:, :-1]  # 特征
y_train = train_data[:, -1]    # 标签

# 添加偏置项到特征矩阵
X_train = np.c_[np.ones(X_train.shape[0]), X_train]

# 读取测试数据
test_data =
np.loadtxt('homework3\dataForTestingLogistic.txt')
X_test = test_data[:, :-1]
y_test = test_data[:, -1]
X_test = np.c_[np.ones(X_test.shape[0]), X_test]
```

这部分代码负责从文件中加载训练和测试数据，并为特征矩阵添加了一列偏置项，以便在逻辑回归中使用。

2. 定义Sigmoid函数和损失函数：

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

# 定义损失函数（条件对数似然）
def compute_loss(X, y, theta):
    length = len(y)
    h = sigmoid(X.dot(theta))
    loss = -1/length * (y.dot(np.log(h)) + (1 -
y).dot(np.log(1 - h)))
    return loss
```

在逻辑回归中，我们使用Sigmoid函数将预测值映射到0和1之间。对数似然函数用于定义损失，表示观测到标签的概率。

3. 梯度上升算法：

```
# 梯度上升算法
def gradient_ascent(X, y, learning_rate=0.001,
epochs=10000):
    theta = np.zeros(X.shape[1])
    length = len(y)
    for epoch in range(epochs):
        h = sigmoid(X.dot(theta))                # 计算
预测值的sigmoid
        gradient = 1/length * X.T.dot(h - y)     # 矩阵
求导
        theta -= learning_rate * gradient        # 更新
theta
        # 计算损失
        loss = compute_loss(X, y, theta)
        if epoch % 1000 == 0:
            print(f'Epoch: {epoch}, Loss: {loss}')
    return theta
```

这段代码实现了梯度上升算法，用于最大化对数似然函数。在每个迭代步骤中，计算梯度并更新参数。我们可以观察到损失函数随着迭代次数的增加而减小。

4. 训练模型并评估性能：

```
# 使用梯度上升算法获得最优参数
optimal_theta = gradient_ascent(X_train, y_train)

# 在测试集上评估模型
predictions = sigmoid(X_test.dot(optimal_theta))
predictions[predictions >= 0.5] = 1
predictions[predictions < 0.5] = 0

accuracy = np.mean(predictions == y_test)
print(f'Accuracy on test set: {accuracy * 100:.2f}%')
```

模型评估后显示的结果如下，再迭代训练了10000次后，显示出了非常好的结果，达到了100%的正确率

Task4

问题描述

报告在测试集中被错误分类的案例

问题解决

在训练迭代次数足够多的情况下，似乎不会出现被错误分类的案例

Task5

问题描述

绘制出在随机梯度上升的每次迭代中目标函数的值

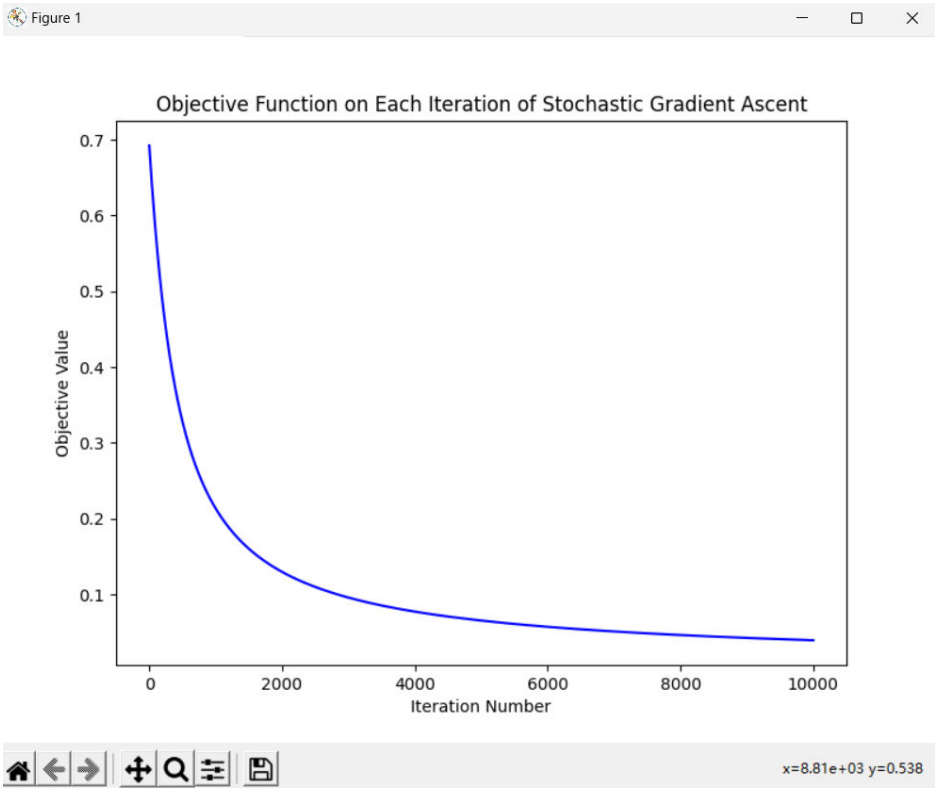
问题解决

- 与前面的问题稍有不同的是需要更改梯度上升算法的实现，需要更改成随机梯度上升法，随机选取其中的样本进行梯度上升，计算loss并且进行可视化

```
# 随机梯度上升算法
def stochastic_gradient_ascent(X, y,
learning_rate=0.00005, epochs=10000):
    theta = np.zeros(X.shape[1])
    length = len(y)
    losses = []  # 用于存储每次迭代的损失值
    for epoch in range(epochs):
        total_loss = 0
        for i in range(length):
            rand_index = np.random.randint(0, length)
# 随机选择一个样本
            h = sigmoid(X[rand_index].dot(theta))
# 计算预测值的sigmoid
            gradient = X[rand_index].T * (h -
y[rand_index])  # 计算梯度
            theta -= learning_rate * gradient
# 更新theta
            total_loss += compute_loss(X, y, theta)
# 计算总损失
        avg_loss = total_loss / length
        losses.append(avg_loss)
        if epoch % 1000 == 0:
            print(f'Epoch: {epoch}, Loss:
{avg_loss}')
    return theta, losses

# 使用随机梯度上升算法获得最优参数和损失值列表
optimal_theta, losses =
stochastic_gradient_ascent(X_train, y_train)

# 绘制损失值随迭代次数的变化图表
plt.figure(figsize=(8, 6))
plt.plot(range(len(losses)), losses, color='b')
plt.xlabel('Iteration Number')
plt.ylabel('Objective Value')
plt.title('Objective Function on Each Iteration of
Stochastic Gradient Ascent')
plt.show()

# 在测试集上评估模型
predictions = sigmoid(X_test.dot(optimal_theta))
predictions[predictions >= 0.5] = 1
predictions[predictions < 0.5] = 0

accuracy = np.mean(predictions == y_test)
```

```
print(f'Accuracy on test set: {accuracy * 100:.2f}%')
```

- 上面的代码绘制出的可视化图表如下：



可见从epoch 7000 左右就开始收敛了

Task6

问题描述

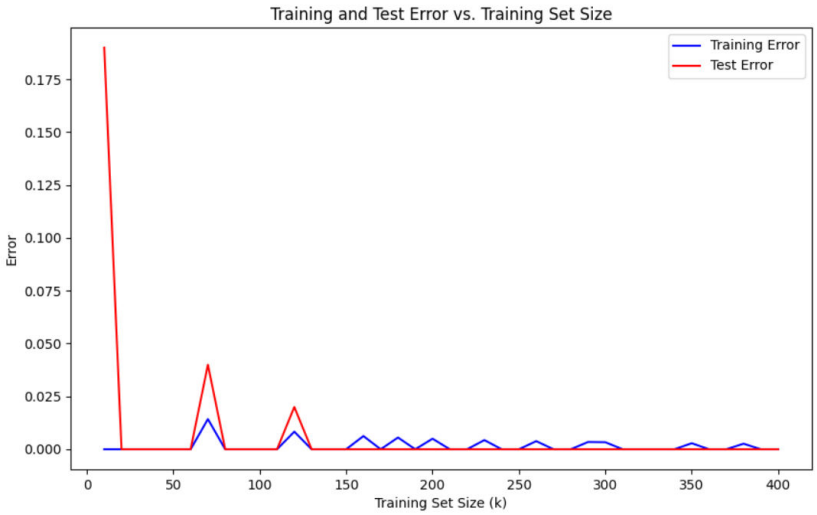
要求评估随着训练集大小的增加，训练误差和测试误差的变化情况

问题解决

- 与前面的梯度上升算法不同的是，这次需要对训练集的大小进行评估，于是要创建一个训练集的大小取值样本集合，再循环遍历计算，之后的计算方法与前面的无异，核心代码如下：

```
k_values = range(10, 401, 10)
train_errors = []
test_errors = []

for k in k_values:
    # 随机选择大小为k的训练子集
    random_indices = np.random.choice(len(X_train),
k, replace=False)
    X_train_subset = X_train[random_indices]
    y_train_subset = y_train[random_indices]

    # 使用梯度上升算法获得最优参数
    optimal_theta = gradient_ascent(X_train_subset,
y_train_subset)

    # 在当前训练子集上计算训练误差
    train_predictions =
sigmoid(X_train_subset.dot(optimal_theta))
    train_predictions[train_predictions >= 0.5] = 1
    train_predictions[train_predictions < 0.5] = 0
    train_error = np.mean(train_predictions !=
y_train_subset)
    train_errors.append(train_error)

    # 在测试集上计算测试误差
    test_predictions =
sigmoid(X_test.dot(optimal_theta))
    test_predictions[test_predictions >= 0.5] = 1
    test_predictions[test_predictions < 0.5] = 0
    test_error = np.mean(test_predictions != y_test)
    test_errors.append(test_error)

# 绘制图表
plt.figure(figsize=(10, 6))
plt.plot(k_values, train_errors, label='Training
Error', color='blue')
plt.plot(k_values, test_errors, label='Test Error',
color='red')
plt.xlabel('Training Set Size (k)')
plt.ylabel('Error')
plt.legend()
plt.title('Training and Test Error vs. Training Set
Size')
plt.show()
```

- 实验得到的结果如下：



上述代码中，我们随机选择不同大小的训练子集，然后使用这些子集训练逻辑回归模型，并在原始测试集上评估模型的性能。通过绘制训练误差和测试误差随训练集大小增加的变化趋势，我们可以得出以下分析：

1. **训练误差（Training Error）**：
    - 当训练集很小（k较小时），模型相对较容易拟合这个小规模的数据集，因此训练误差较低。
    - 随着训练集大小增加，模型需要拟合更多的数据，训练误差可能逐渐增加。这是因为较大的训练集可能包含更多噪声或难以拟合的样本。
2. **测试误差（Test Error）**：
    - 当训练集很小时，模型可能在训练集上表现得非常好，但在未见过的测试数据上表现较差，导致测试误差较高。
    - 随着训练集大小增加，模型的泛化能力可能提高，因此测试误差可能逐渐减小。较大的训练集有助于模型学习到更好的特征表示，提高对未知数据的预测准确性。

综上所述，随着训练集大小的增加，训练误差可能略微波动上升，但测试误差通常会下降。这是因为更多的训练数据可以帮助模型更好地泛化到未知数据，提高模型的预测性能。但是，当训练集达到一定规模后，进一步增加训练集的大小可能对模型的性能提升有限，而且可能会增加训练成本。因此，在选择训练集大小时，需要权衡模型性能和计算资源之间的关系，选择一个适当的训练集规模。