湖南大学

HUNAN UNIVERSITY

人工智能实验三报告

线性回归

目录

一、实验目的

本实验要求应用线性回归模型解决实际问题，通过实验加深对线性回归原理的理解。建议使用python编程实现，并在Mindspore框架下实现。

实验包括两部分：一元线性回归和多元线性回归。

二、实验描述

线性回归是一种预测模型，它试图通过拟合一个线性方程来描述自变量（解释变量）和因变量（响应变量）之间的关系。它可以用于预测和因果关系分析。

基本思想：

1.模型假设：线性回归假设自变量和因变量之间存在线性关系，即因变量可以表示为自变量的线性组合加上一个误差项。

2.损失函数：线性回归通过最小化损失函数（通常是均方误差，MSE）来寻找最佳拟合线。损失函数衡量了模型预测值与实际值之间的差异。

3.参数估计：通过最小化损失函数，可以估计线性回归模型的参数（系数）。这些参数表示了自变量对因变量的影响程度。

4.模型评估：使用诸如R^2（决定系数）、均方误差（MSE）或均方根误差（RMSE）等指标来评估模型的拟合优度和预测能力。

5.预测：一旦模型训练完成，就可以使用它来预测新数据的因变量值。

数据集处理：

使用 numpy 库的 loadtxt 方法来读取文件，该方法默认将数据加载为一个二维数组。

三、实验及结果分析

（1）开发语言及运行环境；

与实验一相同，不再赘述。

（2）实验的具体步骤；

①一元线性回归：

题目：

应用一元线性回归预测移动餐车的利润。假设你是一家餐饮连锁店的CEO，考虑在不同的城市开辟新店。该餐饮店已在许多城市拥有移动餐车，现有各个城

市移动餐车的利润和城市人口的数据。这些数据将帮助你选择在哪个城市进行新店扩张。请按要求完成实验。

数据集：

文件ex1data1.txt为该实验部分的数据集，第一列表示城市人口（单位为万人），第二列表示该城市的移动餐车的利润（单位为万美元，若利润为负值，表示损失）。

```
ex1data1.txt
文件    编辑    查看

6.1101,17.592
5.5277,9.1302
8.5186,13.662
```

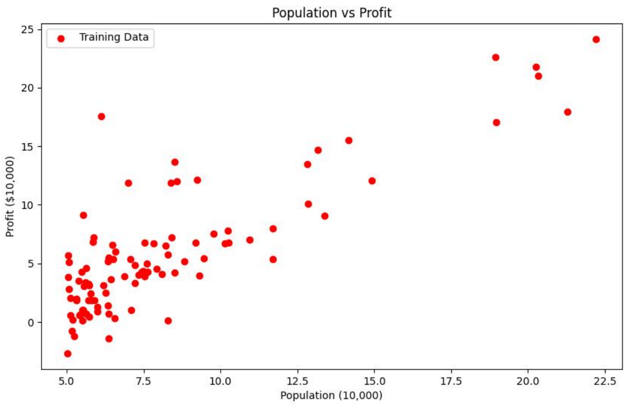
步骤与要求：

1）在开始任务之前，进行数据的可视化对于了解数据特征是很有帮助的。请你导入数据并以人口为横坐标，利润为纵坐标画出散点图并观察数据分布特征。（建议：用python 的matplotlib）

```python
# 1. 数据加载与可视化
1 usage
def load_data(filename):
    data = np.loadtxt(filename, delimiter=',')
    X = data[:, 0].reshape(-1, 1)  # 人口数据
    y = data[:, 1].reshape(-1, 1)  # 利润数据
    return X, y


1 usage
def visualize_data(X, y):
    plt.figure(figsize=(10, 6))
    plt.scatter(X, y, color='red', label='Training Data')
    plt.title('Population vs Profit')
    plt.xlabel('Population (10,000)')
    plt.ylabel('Profit ($10,000)')
    plt.legend()
    plt.show()
```

读取数据集，绘出散点图，并使用 plt.show() 显示图表。



2）将线性回归参数初始化为0，然后计算代价函数（cost function）并求出初始值。

```python
# 2. 线性回归模型
2 usages
class LinearRegressionModel(nn.Cell):
    def __init__(self, input_dim):
        super(LinearRegressionModel, self).__init__()
        # 初始化权重和偏置，均设为0
        self.weights = mindspore.Parameter(
            mindspore.Tensor(np.zeros((input_dim, 1)), mindspore.float32),
            name='weights'
        )
        self.bias = mindspore.Parameter(
            mindspore.Tensor(np.zeros((1, 1)), mindspore.float32),
            name='bias'
        )

    def construct(self, x):
        # 线性回归预测函数 y = wx + b
        return ops.matmul(x, self.weights) + self.bias
```

使用MindSpore中定义神经网络模型的基础类LinearRegressionModel，构造出一个线性回归模型，并定义线性回归预测函数。

3）使用线性回归的方法对数据集进行拟合，并用梯度下降法求解线性回归参数。（eg：迭代次数=1500，alpha=0.01）

代码中的3-4部分。（太长了，不再截图，实验报告最后会给出总代码）

compute_cost是代价函数，用于计算模型的均方误差。

gradient_descent是梯度下降算法，用于训练线性回归模型：将数据转换为MindSpore的Tensor格式，在每次迭代中，进行前向传播，计算预测值。计算梯度，使用矩阵乘法和求和操作。更新模型的权重和偏置。记录并每100次打印一

次迭代的代价。

4）画出数据的拟合图形。

代码中的第5部分.

plot_regression_line函数用于可视化模型的拟合结果，绘制线性回归之后的结果（回归线）。

5）预测人口数量为35000和70000时，利润为多少。

这部分在Main函数中，main函数先分别调用前五部分的代码，然后在第六步mindspore.Tensor预测利润大小。

```
    # 6. 预测
    def predict(population):
        pop_tensor = mindspore.Tensor([[population]], mindspore.float32)
        prediction = model(pop_tensor).asnumpy()
        return prediction[0][0]

    print(f'Prediction for 35000 population: ${predict(3.5):.2f} thousand')
    print(f'Prediction for 70000 population: ${predict(7.0):.2f} thousand')

    # 打印最终参数
    print(f'Final Weights: {model.weights.asnumpy()}')
    print(f'Final Bias: {model.bias.asnumpy()}')
```

②多元线性回归

题目：

应用多元线性回归预测房价。假设你打算出售你的房子，你想知道房子的市场价应该设为多少比较合适。一种方法就是收集最近的房屋销售信息并设计一个

房屋价格模型。请按要求完成实验。

数据集：

文件ex1data2.txt为该实验部分的数据集，第一列表示房屋的面积（平方英尺），第二列表示房间数目，第三列表示房屋价格。

ex1data1.txt　ex1data2.txt

文件　编辑　查看

```
2104,3,399900
1600,3,329900
2400,3,369000
1416,2,232000
```

步骤与要求：

1）导入数据，通过观察，容易发现房屋面积的大小约是房间数量的1000倍。当特征数量级不同时，对进行特征缩放能够使梯度下降更快地收敛。请对这两个特征进行归一化处理。

```python
def load_data(filename):
    """加载数据并返回特征矩阵X和目标值y"""
    data = np.loadtxt(filename, delimiter=',')
    X = data[:, 0:2]  # 前两列为特征
    y = data[:, 2]  # 第三列为房价
    return X, y


def feature_normalize(X):
    """特征归一化处理"""
    mu = np.mean(X, axis=0)
    sigma = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
    return X_norm, mu, sigma
```

load_data函数从文件中加载数据，并返回特征矩阵X和目标值y，使用np.loadtxt读取CSV文件，并将前两列作为特征（X），第三列作为目标值（y）。

feature_normalize函数通过公式(X - mu) / sigma对特征矩阵X进行归一化，使得每个特征的均值为0，标准差为1。函数返回归一化后的特征矩阵X_norm，以及用于归一化的均值mu和标准差sigma。

2）使用梯度下降法求解线性回归参数。尝试使用不同的alpha（学习率）进行实验，找到一个合适的alpha使算法快速收敛。思考alpha的大小对于算法性能

的影响。

```python
def gradient_descent(model, X, y, alpha, num_iters):
    """梯度下降优化"""
    m = len(y)
    costs = []

    optimizer = nn.SGD(model.trainable_params(), learning_rate=alpha)

    def forward_fn(X, y):
        cost = compute_cost(model, X, y)
        return cost

    grad_fn = ops.value_and_grad(forward_fn, None, model.trainable_params())

    for i in range(num_iters):
        cost, grads = grad_fn(X, y)
        optimizer(grads)
        costs.append(float(cost))

        if i % 100 == 0:
            print(f'Iteration {i}: Cost = {float(cost)}')

    return costs
```

gradient_descent使用梯度下降法优化线性回归模型。

初始化：计算样本数量 m，初始化一个空列表 costs 用于存储每次迭代的代价。定义优化器：使用MindSpore的随机梯度下降（SGD）优化器，设置学习率为alpha。定义前向传播函数：forward_fn是一个内部函数，用于计算给定X和y下模型的代价。自动微分：使用ops.value_and_grad来获取代价函数的值和梯度。这个函数返回一个新函数grad_fn，它在调用时会计算代价和梯度。

迭代训练：在num_iters次迭代中，执行以下步骤：调用grad_fn计算当前参数下的代价和梯度。使用优化器更新模型参数。将代价添加到costs列表中。每100次迭代打印一次当前的代价。

返回代价列表：函数返回包含每次迭代代价的列表。

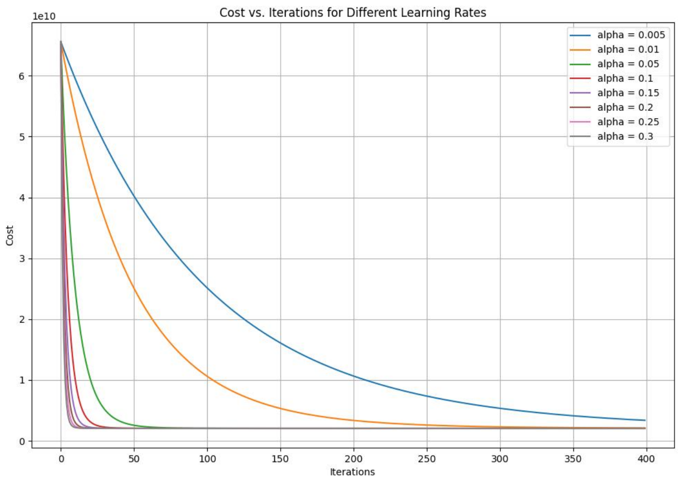
```python
    # 尝试不同的学习率
    alphas = [0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3]
```

在main中指定学习率大小。

```python
    # 4. 使用最佳学习率重新训练模型
    best_alpha = 0.1  # 手动设置最佳学习率
```

3）使用你认为最佳的alpha运行梯度下降法求出线性回归参数，然后预测房屋面积为1650平方英尺，房间数量为3时，房屋的价格。

代价函数是衡量模型预测值与实际值之间差异的函数。它用于指导模型训练过程中的参数优化，在这里损失函数是均方误差（Mean Squared Error, MSE）。



根据这张损失函数图和输出结果可以知道，0.1是最好的alpha。

```
训练模型 (alpha = 0.005):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 25120200704.0
Iteration 200: Cost = 10639077376.0
Iteration 300: Cost = 5338933760.0

训练模型 (alpha = 0.01):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 10596966400.0
Iteration 200: Cost = 3344768768.0
Iteration 300: Cost = 2288005120.0

训练模型 (alpha = 0.05):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 2062616064.0
Iteration 200: Cost = 2043482240.0
Iteration 300: Cost = 2043282432.0

训练模型 (alpha = 0.1):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 2043463040.0
Iteration 200: Cost = 2043280000.0
Iteration 300: Cost = 2043280000.0
```

```
训练模型 (alpha = 0.15):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 2043281792.0
Iteration 200: Cost = 2043280000.0
Iteration 300: Cost = 2043280000.0

训练模型 (alpha = 0.2):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 2043280000.0
Iteration 200: Cost = 2043280000.0
Iteration 300: Cost = 2043280000.0

训练模型 (alpha = 0.25):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 2043280000.0
Iteration 200: Cost = 2043280128.0
Iteration 300: Cost = 2043280128.0

训练模型 (alpha = 0.3):
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 2043280128.0
Iteration 200: Cost = 2043280128.0
Iteration 300: Cost = 2043280128.0
```

学习率（alpha）= 0.005 到 0.01：代价下降非常缓慢，300次迭代后仍然很高，未达到最佳值。

学习率（alpha）= 0.1：代价下降迅速，且达到最佳值。

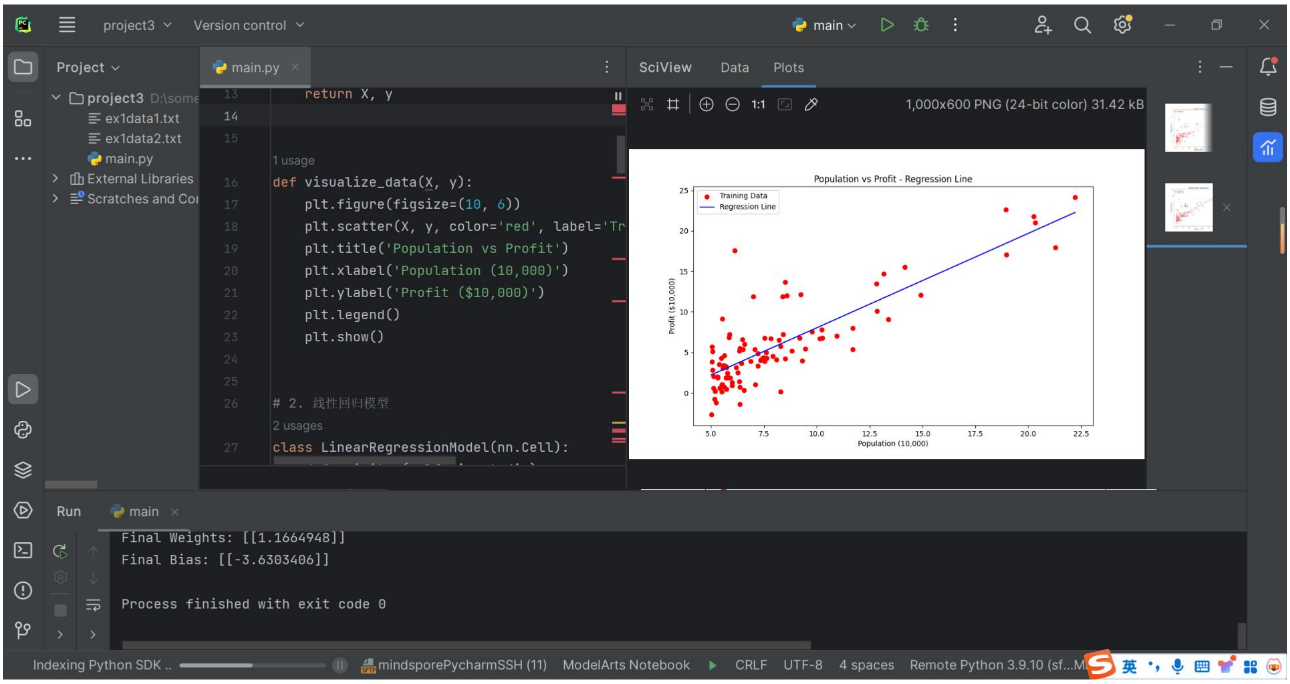
学习率（alpha）= 0.15 到 0.3：代价在100次迭代后几乎不再下降，这表明学习率可能过大，在200次和300次迭代时代价略有上升，可能是由于数值稳定性问题或学习率略大，导致模型在接近最优解时震荡。

预测结果：

```
预测结果：
面积为1650平方英尺，3个房间的房屋预测价格为：$293,081.34
```

（3）根据实验数据集，按实验要求给出相应的结果（截图）并对实验结果进行简要分析。

①一元线性回归：



这两张绘制的图已在前面给出，不再赘述。

```
Initial Cost: 0.3306467533111572
Iteration 0: Cost = 0.06946398317813873
Iteration 100: Cost = 0.05646922066807747
Iteration 200: Cost = 0.05334858223795891
Iteration 300: Cost = 0.05117318406701088
Iteration 400: Cost = 0.04965656250715256
Iteration 500: Cost = 0.0485994778573513
Iteration 600: Cost = 0.04786253347992897
Iteration 700: Cost = 0.04734862595796585
Iteration 800: Cost = 0.046990569680929184
Iteration 900: Cost = 0.04674077779054642
Iteration 1000: Cost = 0.046566836535930634
Iteration 1100: Cost = 0.04644554480910301
Iteration 1200: Cost = 0.046360768377780914
Iteration 1300: Cost = 0.04630196467041969
Iteration 1400: Cost = 0.04626072198152542
Prediction for 35000 population: $0.45 thousand
Prediction for 70000 population: $4.53 thousand
Final Weights: [[1.1664948]]
Final Bias: [[-3.6303406]]
```

初始代价为0.3306，这表示在训练开始之前，模型的预测与实际数据之间的误差较大。随着梯度下降算法的迭代，代价逐渐降低，在1400次迭代后，代价降低到了0.0463。

模型对35000人口的预测利润为$0.45千，对70000人口的预测利润为$4.53千。

最终的权重为[1.1664948]，偏置为[-3.6303406]。这些参数定义了线性回归模型的决策边界，即预测利润的线性方程为 profit = 1.1665 * population - 3.6303。这个方程可以用来预测任意人口数量下的利润。

从代价的下降趋势来看，模型似乎已经收敛，因为代价的减少在逐渐变小，进一步的训练可能不会带来太大的改进。

②多元线性回归：

在前面的报告中已经作了全部的解释，不再赘述。

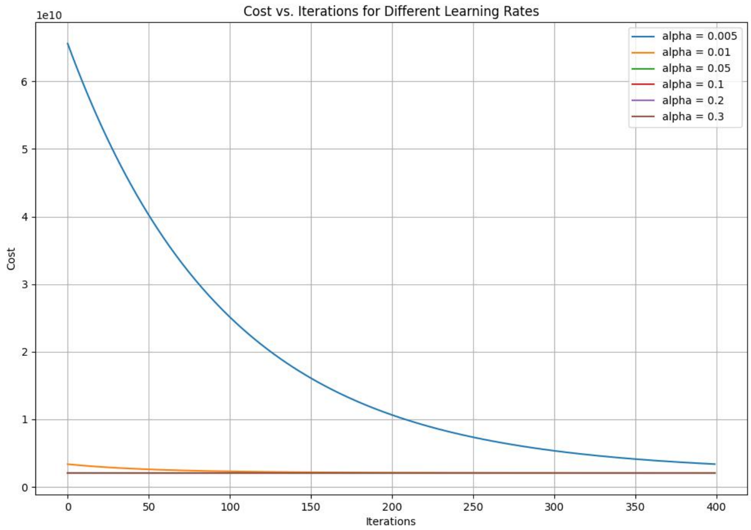
四、心得

对于多元线性回归：测试输出的时候发现一个问题，虽然我设置的最佳学习率是0.01（当时看起来是），但使用它重新训练输出时和之前的输出并不相同，后来发现原因是在测试不同学习率时使用的是同一个模型实例，模型参数在不同学习率的训练过程中被累积更新了。

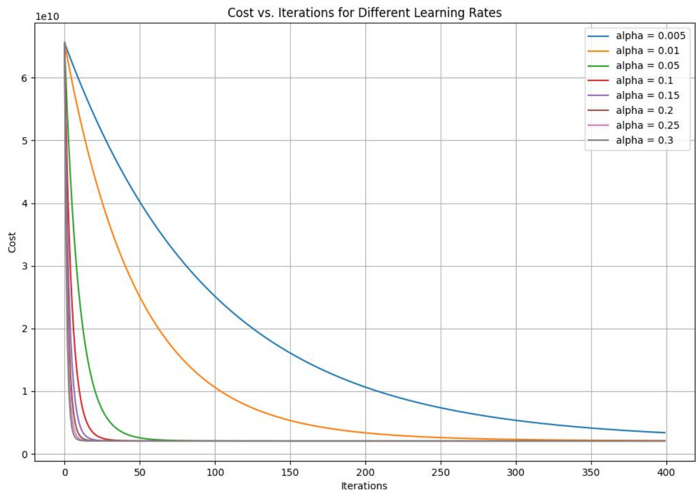
```
训练模型 (alpha = 0.01):
Iteration 0: Cost = 3356021248.0
Iteration 100: Cost = 2289630976.0
Iteration 200: Cost = 2105722752.0
Iteration 300: Cost = 2063842944.0
```

```
使用最佳学习率重新训练模型:
Iteration 0: Cost = 65591545856.0
Iteration 100: Cost = 10596966400.0
Iteration 200: Cost = 3344768768.0
Iteration 300: Cost = 2288005120.0
```

六、附录



修改成为每个学习率单独训练一个模型即可，并更新之前的实验报告。



五、程序文件名清单

main.py：一元线性回归的源代码

main.py.pdf：一元线性回归的源代码 pdf 版

main2.py：多元线性回归的源代码

main2.py.pdf：多元线性回归的源代码 pdf 版

六、附录

代码如下。

main.py

```python
import mindspore
import mindspore.nn as nn
import mindspore.ops as ops
import numpy as np
import matplotlib.pyplot as plt


# 1. 数据加载与可视化
def load_data(filename):
    data = np.loadtxt(filename, delimiter=',')
    X = data[:, 0].reshape(-1, 1)  # 人口数据
    y = data[:, 1].reshape(-1, 1)  # 利润数据
    return X, y


def visualize_data(X, y):
    plt.figure(figsize=(10, 6))
    plt.scatter(X, y, color='red', label='Training Data')
    plt.title('Population vs Profit')
    plt.xlabel('Population (10,000)')
    plt.ylabel('Profit ($10,000)')
    plt.legend()
    plt.show()


# 2. 线性回归模型
class LinearRegressionModel(nn.Cell):
    def __init__(self, input_dim):
        super(LinearRegressionModel, self).__init__()
        # 初始化权重和偏置，均设为0
        self.weights = mindspore.Parameter(
            mindspore.Tensor(np.zeros((input_dim, 1)), mindspore.float32),
            name='weights'
        )
        self.bias = mindspore.Parameter(
            mindspore.Tensor(np.zeros((1, 1)), mindspore.float32),
            name='bias'
        )

    def construct(self, x):
        # 线性回归预测函数 y = wx + b
        return ops.matmul(x, self.weights) + self.bias


# 3. 代价函数（均方误差）
def compute_cost(model, X, y):
    m = X.shape[0]
    predictions = model(X)
    cost = ops.reduce_mean((predictions - y) ** 2) / (2 * m)
    return cost


# 4. 梯度下降训练
def gradient_descent(X, y, model, learning_rate, iterations):
    m = X.shape[0]
    costs = []

    # 转换为Tensor
    X_tensor = mindspore.Tensor(X, mindspore.float32)
    y_tensor = mindspore.Tensor(y, mindspore.float32)

    for i in range(iterations):
        # 前向传播
        predictions = model(X_tensor)

        # 计算梯度
```

```python
        dw = ops.matmul(X_tensor.T, (predictions - y_tensor)) / m
        db = ops.reduce_sum(predictions - y_tensor) / m

        # 更新参数
        model.weights -= learning_rate * dw
        model.bias -= learning_rate * db

        # 记录代价
        cost = compute_cost(model, X_tensor, y_tensor)
        costs.append(cost.asnumpy())

        # 每100次迭代打印一次代价
        if i % 100 == 0:
            print(f'Iteration {i}: Cost = {cost.asnumpy()}')

    return costs


# 5. 可视化拟合结果
def plot_regression_line(X, y, model):
    plt.figure(figsize=(10, 6))
    plt.scatter(X, y, color='red', label='Training Data')

    # 绘制回归线
    X_sorted = np.sort(X, axis=0)
    y_pred = model(mindspore.Tensor(X_sorted, mindspore.float32)).asnumpy()

    plt.plot(X_sorted, y_pred, color='blue', label='Regression Line')
    plt.title('Population vs Profit - Regression Line')
    plt.xlabel('Population (10,000)')
    plt.ylabel('Profit ($10,000)')
    plt.legend()
    plt.show()


# 主函数
def main():
    # 加载数据
    X, y = load_data('ex1data1.txt')

    # 1. 数据可视化
    visualize_data(X, y)

    # 2. 初始化模型
    model = LinearRegressionModel(input_dim=1)

    # 3. 初始代价
    X_tensor = mindspore.Tensor(X, mindspore.float32)
    y_tensor = mindspore.Tensor(y, mindspore.float32)
    initial_cost = compute_cost(model, X_tensor, y_tensor)
    print(f'Initial Cost: {initial_cost.asnumpy()}')

    # 4. 梯度下降训练
    learning_rate = 0.01
    iterations = 1500
    costs = gradient_descent(X, y, model, learning_rate, iterations)

    # 5. 可视化拟合结果
    plot_regression_line(X, y, model)

    # 6. 预测
    def predict(population):
        pop_tensor = mindspore.Tensor([[population]], mindspore.float32)
        prediction = model(pop_tensor).asnumpy()
        return prediction[0][0]

    print(f'Prediction for 35000 population: ${predict(3.5):.2f} thousand')
    print(f'Prediction for 70000 population: ${predict(7.0):.2f} thousand')

```

```python
    # 打印最终参数
    print(f'Final Weights: {model.weights.asnumpy()}')
    print(f'Final Bias: {model.bias.asnumpy()}')


if __name__ == "__main__":
    main()
```

main2.py

```python
import numpy as np
import mindspore as ms
from mindspore import nn, ops
from mindspore import context
import matplotlib.pyplot as plt

# 设置MindSpore运行环境
context.set_context(mode=context.GRAPH_MODE, device_target="CPU")


def load_data(filename):
    """加载数据并返回特征矩阵X和目标值y"""
    data = np.loadtxt(filename, delimiter=',')
    X = data[:, 0:2]  # 前两列为特征
    y = data[:, 2]  # 第三列为房价
    return X, y


def feature_normalize(X):
    """特征归一化处理"""
    mu = np.mean(X, axis=0)
    sigma = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
    return X_norm, mu, sigma


class LinearRegression(nn.Cell):
    """线性回归模型类"""

    def __init__(self, input_dim):
        super(LinearRegression, self).__init__()
        self.weights = ms.Parameter(ms.Tensor(np.zeros((input_dim, 1)), ms.float32))
        self.bias = ms.Parameter(ms.Tensor(np.zeros(1), ms.float32))

    def construct(self, x):
        return ops.matmul(x, self.weights) + self.bias


def compute_cost(model, X, y):
    """计算损失函数"""
    m = len(y)
    predictions = model(X)
    cost = ops.reduce_mean(ops.square(predictions - y.reshape(-1, 1))) / 2.0
    return cost


def gradient_descent(model, X, y, alpha, num_iters):
    """梯度下降优化"""
    m = len(y)
    costs = []

    optimizer = nn.SGD(model.trainable_params(), learning_rate=alpha)

    def forward_fn(X, y):
        cost = compute_cost(model, X, y)
        return cost

    grad_fn = ops.value_and_grad(forward_fn, None, model.trainable_params())

    for i in range(num_iters):
        cost, grads = grad_fn(X, y)
        optimizer(grads)
        costs.append(float(cost))

        if i % 100 == 0:
            print(f'Iteration {i}: Cost = {float(cost)}')
```

```python

    return costs


def main():
    # 1. 加载数据
    X, y = load_data('ex1data2.txt')

    # 2. 特征归一化
    X_norm, mu, sigma = feature_normalize(X)

    # 将数据转换为MindSpore张量
    X_ms = ms.Tensor(X_norm, ms.float32)
    y_ms = ms.Tensor(y, ms.float32)

    # 3. 创建和训练模型
    model = LinearRegression(input_dim=2)

    # 尝试不同的学习率
    alphas = [0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3]
    plt.figure(figsize=(12, 8))

    for alpha in alphas:
        print(f"\n训练模型 (alpha = {alpha}):")
        model = LinearRegression(input_dim=2)  # 为每个学习率创建新的模型
        costs = gradient_descent(model, X_ms, y_ms, alpha, num_iters=400)
        plt.plot(range(len(costs)), costs, label=f'alpha = {alpha}')

    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('Cost vs. Iterations for Different Learning Rates')
    plt.legend()
    plt.grid(True)
    plt.show()

    # 4. 使用最佳学习率重新训练模型
    best_alpha = 0.1  # 手动设置最佳学习率
    print("\n使用最佳学习率重新训练模型:")
    model = LinearRegression(input_dim=2)
    costs = gradient_descent(model, X_ms, y_ms, best_alpha, num_iters=400)

    # 5. 预测房价
    # 对新数据进行归一化
    test_X = np.array([[1650, 3]])
    test_X_norm = (test_X - mu) / sigma
    test_X_ms = ms.Tensor(test_X_norm, ms.float32)

    # 进行预测
    prediction = model(test_X_ms)
    predicted_price = float(prediction.asnumpy()[0][0])
    print(f"\n预测结果:")
    print(f"面积为1650平方英尺，3个房间的房屋预测价格为: ${predicted_price:,.2f}")


if __name__ == "__main__":
    main()
```