# 实验一 线性回归

1. **实验目的**

熟悉线性回归的整个算法流程，学习使用sklearn库进行模型搭建。

1. **实验内容**
2. **一元线性回归**

数据来源于斯坦福大学的机器学习课程， 在本练习的这一部分中，你将实现带有一个变量的线性回归来预测流动餐车的利润。假设你是一家餐厅特许经营的首席执行官，正在考虑在其他城市开设一家新流动快餐店。该连锁店已经在多个城市开设了分店，你有来自城市的数据利润和人口。希望使用这些数据来帮助你选择下一步要扩展到哪个城市。

ex1data1.txt文件是本次线性回归问题的数据集，第一列是一个城市的人口数，第二列是这个城市流动餐车的利润。利润为负值表示亏损。

**1.1 数据分析和表示**

在开始任何任务之前，通过可视化来理解数据通常是有用的。对于这个数据集，可以使用散点图可视化数据，因为它只有两个属性要绘制(利润和人口)。数据集从数据文件加载到变量X和y:

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

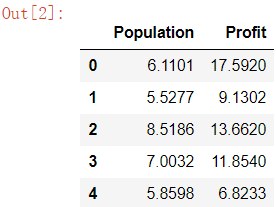
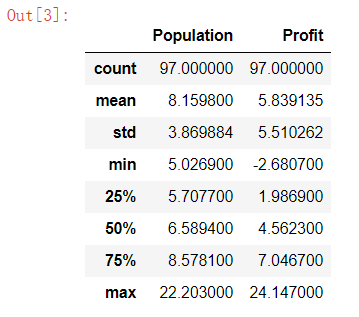
import os

path = 'ex1data1.txt'

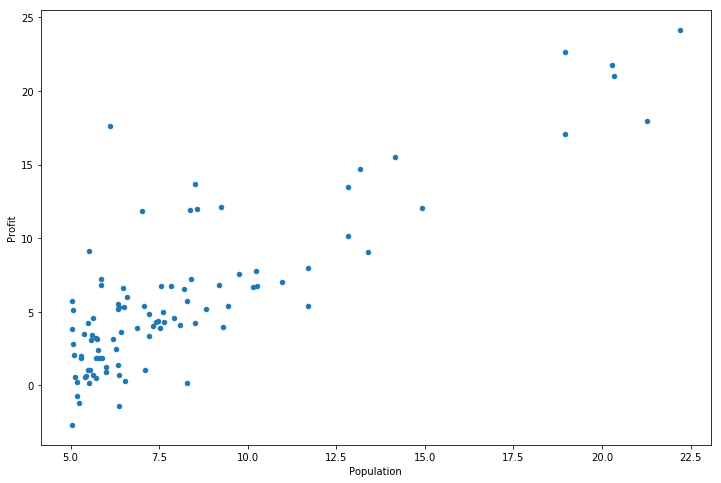
data = pd.read\_csv(path, header=None, names=['Population', 'Profit'])

data.head()

data.describe()

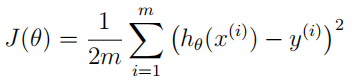
data.plot(kind='scatter',x='Population',y='Profit',figsize=(12,8))

****

**1.2 梯度下降法**

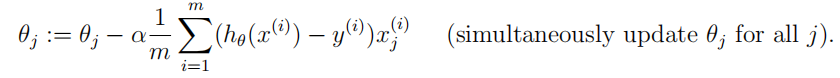
在这一部分中，你将使用梯度下降来拟合线性回归参数θ到我们的数据集。

线性回归的目标是最小化代价函数：





回想一下，模型的参数是θj值。这些是你需要调整的值，以使成本J(θ)最小。一种方法是使用批量梯度下降算法。在批量梯度下降中，每次迭代执行更新。



随着梯度下降的每一步，你的参数θ J越来越接近于使代价最低的J(θ)的最优值。

我们将每个样本存储为X矩阵中的一行。为了考虑截距项(θ0)，我们在X上增加一列，并使其均为1。这使得我们可以将θ0简单地视为另一个“特征”。

当前已经建立了线性回归的数据。在下一行中，我们向数据添加另一个维度以适应θ0截距项。我们还将初始参数初始化为0，将学习速率alpha初始化为0.01。

data.insert(0, 'Ones', 1)

cols = data.shape[1]

X = data.iloc[:,0:cols-1]

y = data.iloc[:,cols-1:cols]

X = np.matrix(X.values)

y = np.matrix(y.values)

theta = np.matrix(np.array([0,0]))

**1.3 计算损失函数**

当你执行梯度下降学习最小化代价函数J(θ)时，通过计算损失函数监测收敛是有帮助的。在本节中，你将实现一个函数来计算J(θ)，这样你就可以检查你的梯度下降实现的收敛性。任务是完成文件函数中的代码，它是一个计算J(θ)的函数。当你这样做的时候，记住变量X和y不是标量值，而是矩阵，它们的行代表了来自训练集的例子。一旦你完成了函数，在ex1中的下一步。m将运行computeCost一旦使用θ初始化为零，你会看到损失值32.07打印到屏幕上。

def computeCost(X, y, theta):

m = len(y)

J = 0

h = X \* theta.T

J = (1 / (2 \* m)) \* np.sum(np.square(h - y))

return J



**1.4 梯度下降**

接下来，你将在函数gradientDescent中实现梯度下降。循环结构已经为你写好，你只需要在每次迭代中对θ进行更新。注意J(θ)是向量θ的函数,而不是X和y。也就是说,我们通过改变的向量θ来最小化J(θ)的值，而不是通过改变X或y。验证梯度下降是否正确的一个好方法是观察J(θ)的值，并检查它是否随着每一步迭代而减小。gradientDescent在每次迭代时调用computeCost，然后打印损失函数的值。假设你正确地实现了梯度下降和computeCost，你的J(θ)的值永远不会增加，并且应该在算法结束时收敛到一个稳定的值。在你完成之后，使用你的最终参数来绘制线性拟合。

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

temp = np.matrix(np.zeros(theta.shape))

parameters = int(theta.ravel().shape[1])

cost = np.zeros(iters)

m = X.shape[0]

for i in range(iters):

error = (X \* theta.T) - y

for j in range(parameters):

term = np.multiply(error, X[:, j])

temp[0, j] = theta[0, j] - ((alpha / m) \* np.sum(term))

theta = temp

cost[i] = computeCost(X, y, theta)

return theta, cost

alpha = 0.01

iters = 1000

g, cost = gradientDescent(X, y, theta, alpha, iters)

computeCost(X, y, g)



x = np.linspace(data.Population.min(), data.Population.max(), 100)

f = g[0, 0] + (g[0, 1] \* x)

fig, ax = plt.subplots(figsize=(12,8))

ax.plot(x, f, 'r', label='Prediction')

ax.scatter(data.Population, data.Profit, label='Traning Data')

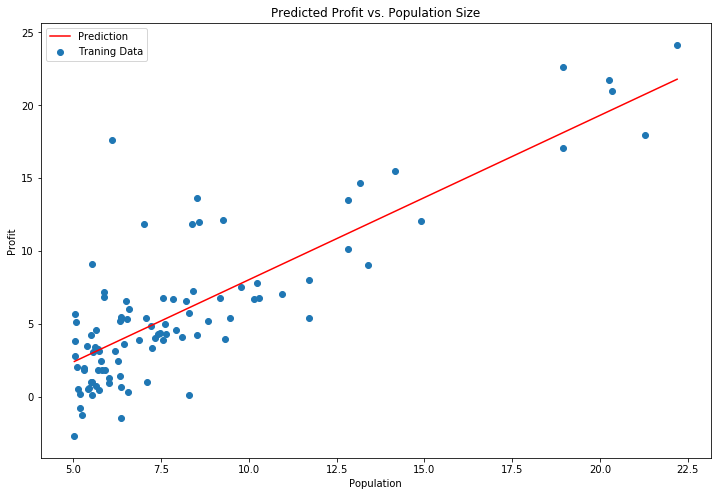
ax.legend(loc=2)

ax.set\_xlabel('Population')

ax.set\_ylabel('Profit')

ax.set\_title('Predicted Profit vs. Population Size')

plt.show()



**1.5 使用scikit-learn工具库**

除了从头开始实现这些算法，我们还可以使用scikit-learn库中的线性回归函数。我们将scikit-learn 的线性回归算法应用于前面的数据。首先导入sklearn中的linear\_model模块，对于线性回归所要使用的是LinearRegression函数，之后调用fit函数进行模型训练。

from sklearn import linear\_model

model = linear\_model.LinearRegression()

model.fit(X, y)

模型训练好后，你可以使用predict函数进行预测，最终预测结果如下。

x = np.array(X[:, 1].A1)

f = model.predict(X).flatten()

fig, ax = plt.subplots(figsize=(12,8))

ax.plot(x, f, 'r', label='Prediction')

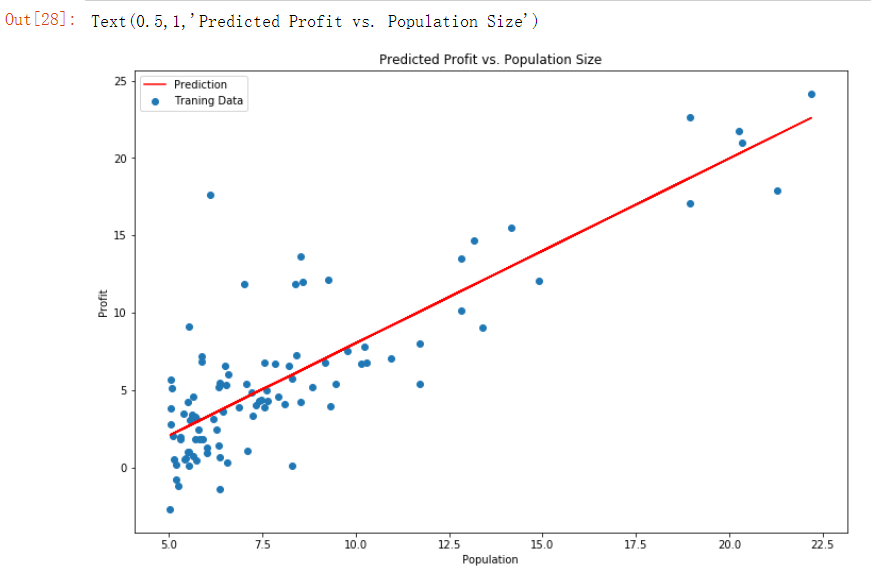
ax.scatter(data.Population, data.Profit, label='Traning Data')

ax.legend(loc=2)

ax.set\_xlabel('Population')

ax.set\_ylabel('Profit')

ax.set\_title('Predicted Profit vs. Population Size')



**2.多元线性回归**

如果你已经成功地完成了上述实验，祝贺你已经理解了线性回归，并且应该能够开始在你自己的数据集上使用它。这一部分的练习将帮助你对线性回归有更深入的理解，你将使用多元线性回归进行房价的预测。假设你要卖掉房子，你想要知道一个好的市场价格是多少。一种方法是先收集有关最近出售房屋的信息并制作房价模型。

数据来源于斯坦福大学的机器学习课程。文件ex1data2.txt包含美国俄勒冈州波特兰市房价的训练集。数据集中第一列为房屋大小（以平方英尺为单位），第二列为卧室的数量，第三列为房屋价格。

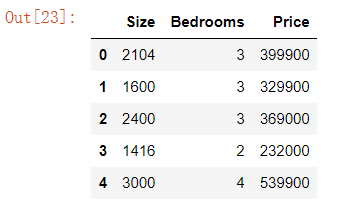
**2.1 数据分析**

与一元线性回归不同的是，该数据集中包含两个变量（房屋面积和卧室数量）以及目标值（房屋价格）。首先来分析该数据集：

path = 'ex1data2.txt'

data2 = pd.read\_csv(path, header=None, names=['Size', 'Bedrooms', 'Price'])

data2.head()

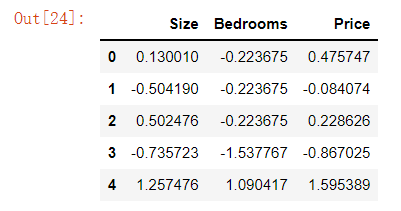


**2.2 特征规则化**

查看上述值可以发现房屋大小的数值约是房屋数量数值的1000倍。当特征相差几个数量级时，首先执行特征缩放可以使梯度下降更快的收敛。对应这个任务，我们添加了另一个预处理步骤——特征标准化。你的任务是根据下面描述将代码补充完整，首先从数据集中减去每个特征的平均值，然后将特征值除以各自的“标准差”（pandas中mean()函数可以求平均值，std()函数为计算标准差函数）。完成特征标准化后，将得到如下的输出结果。

data2 = (data2 - data2.mean()) / data2.std()

data2.head()



**2.3 梯度下降**

之前，你在一元线性回归问题上实现了梯度下降，现在唯一的区别是矩阵X中多了一个特征，假设函数和批量梯度下降更新规则保持不变，现在让我们重复上一实验中的预处理步骤，并在新数据集上运行线性回归过程。你需要将代码根据注释实现出来，最终输出模型的损失函数值。（由于实验1代码已支持多元，函数gradientDescent和computeCost可直接调用）

# 在训练集中添加一列ones

data2.insert(0, 'Ones', 1)

# 对变量X(训练数据)和y(目标变量)进行初始化

cols = data2.shape[1]

X2 = data2.iloc[:, 0:cols - 1]

y2 = data2.iloc[:, cols - 1:cols]

# 将X和y转换为矩阵并初始化theta

X2 = np.matrix(X2.values)

y2 = np.matrix(y2.values)

theta2 = np.matrix(np.array([0, 0, 0]))

# 对数据集执行线性回归

**g2, cost2 = gradientDescent(X2, y2, theta2, alpha, epoch)**

# 计算模型的误差

**print(computeCost(X2, y2, g2)), g2**

