实验一：线性回归

介绍：

在本部分的实验中，需要你实现单变量线性回归代码和多变量线性回归代码。
单变量线性回归：预测小吃店的利润。假设你是一家小吃店的老板，正在考虑不同的城市开设一个新的分店。该连锁店已经在各个城市拥有小吃店，而且你有来自城市的利润和人口数据。你希望使用这些数据来帮助你决定将业务扩展到那一个城市。数据中 x 表示人口数据，y 表示利润，一共 97 行数据。
多变量线性回归：需要预测房价，输入变量有两个特征，一是房子的面积，二是房子卧室的数量；输出变量是房子的价格。
实验文件说明：
ex1.py –单变量线性回归主文件
[*]ex1 multi.py -多变量线性回归主文件
ex1data1.txt –单变量线性回归数据
ex1data2.txt –多变量线性回归数据
[*] warmUpExercise.py -生成 one-hot 矩阵的函数
[*] plotData.py -用来显示数据集的函数
[*] computeCost.py -线性回归代价函数
[*] gradientDescent.py - 梯度下降函数
[ƒ] computeCostMulti.py –多变量线性回归代价函数
[ƒ] gradientDescentMulti.py -多变量线性回归梯度下降函数
[ƒ] featureNormalize.py -标准化函数
[ƒ] normalEqn.py -计算正规方程组的函数
带*和ƒ的函数有部分代码缺少需要你按照提示将代码补充完整。

1 Python 函数填写示例

在文件 warmUpExercise.py 中，你可以通过填写以下代码，以返回一个 5 x 5 的单位矩阵并打印输出:

```
A = np.eye(5)
print(A)
```

完成后，运行 ex1.py(假设你在正确的目录下，在八度提示符下输入″ ex1 ″)，你应该会看到类似如下的输出:

```
Running warmUpExercise ...
5x5 Identity Matrix:
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

2 单变量线性回归

在本实验的这一部分中，你将实现带有一个变量的线性回归来预测小吃店的利润。假设你是一家餐厅特许经营的首席执行官，正在考虑在不同的城市开设一家新的小吃店。该连锁店已经在多个城市拥有小吃店，你可以获得这些城市的利润和人口数据。你希望使用这些数据来帮助您选择下一步要扩展到哪个城市

ex1data1.txt 文件包含线性回归问题的数据集。第一列是一个城市的人口第二列是这个城市的小吃店的利润。利润为负值表示亏损。

ex1.py 已经设置了 python 脚本来为你加载这些数据。

2.1 数据可视化

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，您可以使用散点图来可视化数据，因为它只有两个属性要绘制(利润和人口)。(你在现实生活中遇到的许多其他问题都是多维度的，无法在二维图上表示出来。)

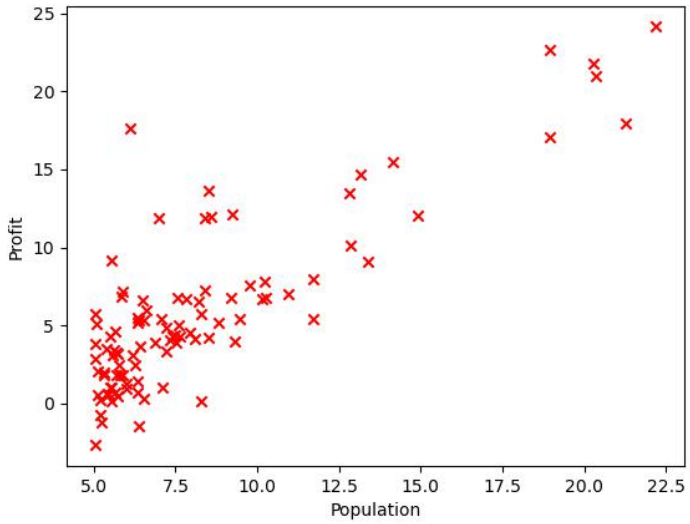
在 ex1.py 中，数据集从数据文件加载到变量 X 和 y:

```
path = 'ex1data1.txt'
data = pd.read_csv('ex1data1.txt', header=None)  # 无 header
X = data.iloc[:, 0]
y = data.iloc[:, 1]
m = len(y)#number of training examples
```

接下来，调用 plotData 函数来创建数据的散点图。你的工作是完成 plotData.py 表示画图;修改文件并填写如下代码:

```
plt.scatter(x=x, y=y, c='red', marker='x')
plt.xlabel('Population')
plt.ylabel('Profit')
plt.show()
```

现在，当你继续运行 ex1.py，我们的最终结果应该如图 1 所示，带有相同的红色“x”标记和轴标签。



图一：训练数据的散点图

2.2 梯度下降

在这一部分中，你将使用梯度下降来拟合线性回归参数 θ 到数据集。

2.2.1 Update Equations

线性回归的目标是最小化代价函数

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

假设$h_\theta(x)$是由线性模型给出的

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

模型的参数是θ_j值。这些是你需要调整的值，以使成本$J(\theta)$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

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \text{(simultaneously update } \theta_j \text{ for all } j).$$

随着梯度下降的每一步，你的参数θ_j越来越接近将达到最低代价$J(\theta)$的最优值。

2.2.2 实现

在 ex1.py 中，我们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θ_0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。

```
X = np.array(X).reshape(m, 1)
X = np.insert(X, 0, np.ones(m), axis=1)  # Add a column of ones to
y = np.array(y).reshape(m, 1)
theta = np.zeros((2, 1))  # initialize fitting parameters
# Some gradient descent settings
iterations = 1500
alpha = 0.01
```

2.2.3 计算代价$J(\theta)$

当你执行梯度下降学习最小化代价函数$J(\theta)$时，通过计算代价来监测收敛是有帮助的。在本节中，你将实现一个函数来计算$J(\theta)$，这样你就可以检查你的梯度下降实现的收敛性。

你的下一个任务是完成文件 computeCost.py 中的代码，它是一个计算$J(\theta)$的函数。注意变量 X 和 y 不是标量值，而是矩阵。

一旦你完成了函数，在 ex1.py 中的下一步将运行 computeCost.py 一旦使用 θ 初始化为零， cost 会输出到输出栏。

你应该会看到 Cost computed = 32.07。

2.2.4 梯度下降

接下来，你将在文件 gradientDescent.py 中实现梯度下降。循环结构已经为你写好了，你只需要在每次迭代中更新θ。

在你编程的时候，确保你明白你要优化的是什么，要更新的是什么。记住$J(\theta)$由向量θ是参数化的，而不是 X 和 y。也就是说，我们最小化$J(\theta)$的值通过改变的值向量θ，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$J(\theta)$的值，并检查它是否随着每一步而减小。gradientDescent.py 的起始代码，在每次迭代时调用 computeCost.py 并打印成本。假设你正确地实现了梯度下降和 computeCost，$J(\theta)$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，ex1.py 将使用你的最终参数来绘制线性拟合图像。结果应该如图 2 所示:

θ的最终值也将用于预测 35000 人和 70000 人的利润。请注意 ex1.py 中的下面几行代码。使用矩阵乘法来计算预测，而不是显式的求和或循环。

```
# Predict values for population sizes of 35,000 and 70,000
predict1 = np.array([[1, 3.5]]) @ theta.T
print('For population = 35,000, we predict a profit of %.2f' % (predict1
* 10000))
predict2 = np.array([[1, 7]]) @ theta.T
print('For population = 70,000, we predict a profit of %.2f\n' %
(predict2 * 10000))
```

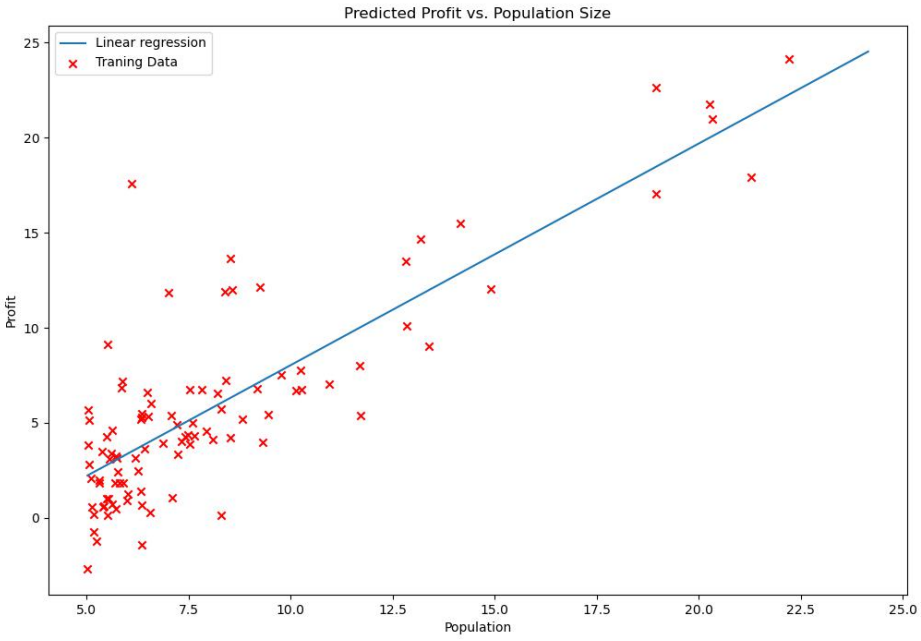


图 2:线性回归拟合的训练数据

2.3 调试

以下是在你执行梯度下降时需要记住的一些事情:

•python 数组的下标从 0 开始，而不是 1。如果将θ_0和θ_1存储在一个叫做 theta 的向量中，其值将是 theta(1)和 theta(2)。

•如果你在运行时看到很多错误，检查你的矩阵操作，以确保你添加和乘法矩阵的兼容维数。使用 data.shape 命令打印变量的形状将有助于调试。

•默认情况下，python 将数学运算符解释为矩阵运算符。例如，A*B 执行矩阵乘法，而 A@B 执行矩阵叉乘。

2.4 可视化$J(\theta)$

为了更好地理解代价函数$J(\theta)$，现在将在θ_0和θ_1值的二维网格上绘制代价。你不需要为这部分编写任何新的代码，但你应该了解你已经编写的代码是如何创建这些图像的。在 ex1.py 的下一步，使用 computeCost 函数计算$J(\theta)$。

```
# initialize J_vals to a matrix of 0's
J_vals = np.zeros((len(theta0_vals), len(theta1_vals)))

# Fill out J_vals
for i in range(0, theta0_vals.shape[0]):
    for j in range(0, theta1_vals.shape[0]):
        t = np.array([theta0_vals[i], theta1_vals[j]])
        J_vals[i, j] = computeCost(X, y, t)
```

在执行以上代码之后，你将得到一个$J(\theta)$值的二维数组。ex1.py 将使用这些值使用 meshgrid 和 contourf 命令生成$J(\theta)$的曲面和等高线图如图 3 所示：

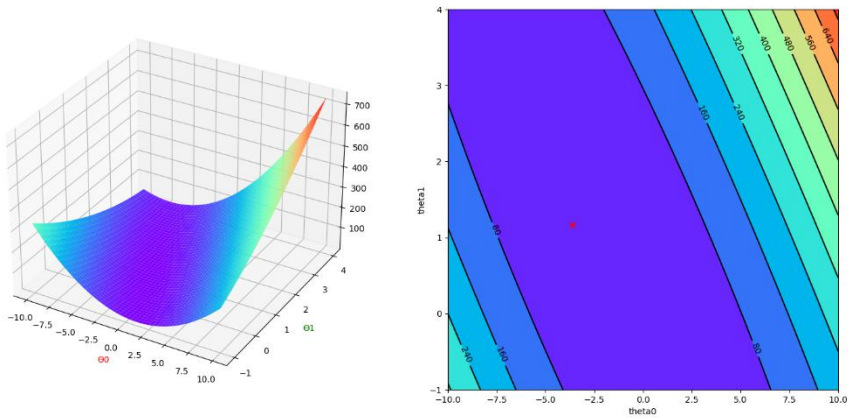


图 3:代价函数$J(\theta)$

这些图的目的是向你们展示$J(\theta)$如何随θ_0和θ_1的变化而变化。代价函数$J(\theta)$是碗形的，具有全局极小值。(这在等高线图中比在三维曲面图中更容易看到)。该最小值为θ_0和θ_1的最优点，梯度下降的每一步都向该点靠近。

3 多元线性回归

在这一部分中，你将实现多变量线性回归来预测房价。假设你要卖房子，你想知道一个好的市场价格是多少。一种方法是首先收集最近售出的房屋的信息，并制作一个房屋价格模型。

ex1data2.txt 文件包含俄勒冈州波特兰市的房价数据集。第一列是房子的大小(平方英尺)，第二列是卧室的数量，第三列是房子的价格。

ex1multi.py 是用来帮助你一步步完成这个练习的。

3.1 标准化特征

ex1multi.py 加载数据集和显示一些值开始。通过查看这些数值，你会发现房子的大小大约是卧室数量的 1000 倍。当特征相差数量级时，首先对特征进行标准化可以使梯度下降更快地收敛。

这里的任务是完成 featurennormalize.py 中的代码。

•从数据集中减去每个特征的平均值。

•在减去平均值之后，再将特征值除以它们各自的“标准偏差”。

标准差是一种测量某一特征值范围内变化程度的方法(大多数数据点位于均值±2 个标准差范围内);这是取值范围(max-min)的另一种方法。在 Python 中，可以使用“std”函数来计算标准偏差。例如，在 featurennormalize.py 中， X 包含了训练集中房子尺寸 x1 和卧室的数量 x2，np.std(X)计算房子尺寸的标准差。

对所有的特性都这样做，这样代码能够处理任何大小的数据集(任何数量的特性/示例)。注意，矩阵 X 的每一列对应一个特征。

注意:在对特征进行归一化时，重要的是要存储用于归一化的值——用于计算的平均值和标准差。在从模型中学习参数后，我们常常想要预测我们以前没有见过的房子的价格。给定一个新的 x 值(客厅面积和卧室数量)，我们必须首先使用之前从训练集中计算的均值和标准差对 x 进行标准化。

3.2 梯度下降

多变量梯度下降与单变量梯度下降唯一的区别是矩阵 x 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。你需要完善 computeCostMult.py 和 gradientDescentMulti.py 中的代码以实现多变量线性回归的代价函数和梯度下降。如果前一部分(单变量)中的代码已经支持多个变量，那么在这里也可以使用它。

请确保你的代码支持任意数量的特性，并具有良好的鲁棒性。你可以使用X.shape[1]来找出有多少特征出现在数据集中。

注:在多元情况下，代价函数也可以写成如下矢量化形式:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T (X\theta - \vec{y})$$

其中

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

3.2.1 选择学习速率

在本部分的练习中，你可以尝试数据集的不同学习率，并找到快速收敛的学习速率。通过修改 ex1multi.py 中设置学习速率的部分改变学习率。如果你在一个合适的范围内选择一个学习率，那么您的图表与图 4 类似。如果你的曲线看起来很不一样，特别是当你的$J(\theta)$值增加甚至爆炸时，调整你的学习速率并再次尝试。我们建议在对数尺度上尝试学习速率 α 的值，用 3 倍于之前值的乘法步骤(例如，0.3,0.1,0.03,0.01 等等)。你还可以调整迭代次数，这将帮助你看到曲线的总体趋势。

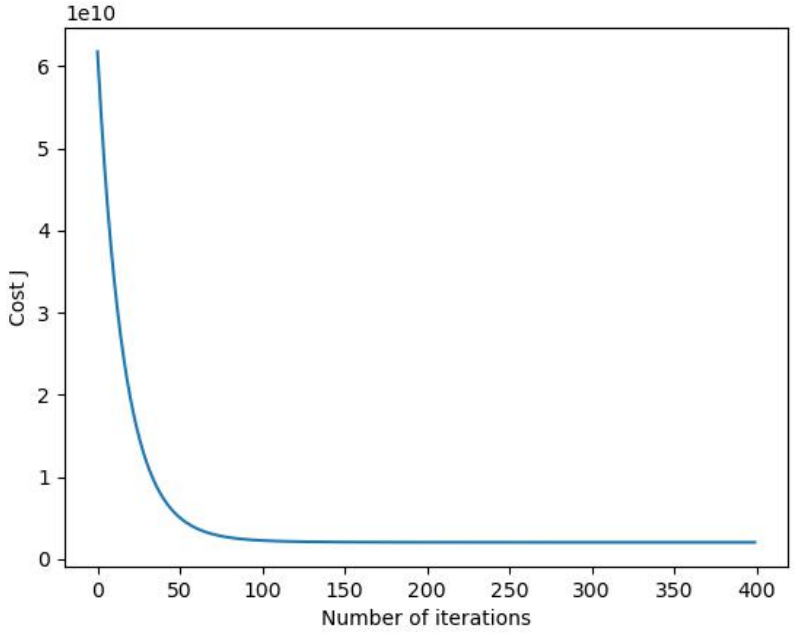


图 4:在适当的学习速率下梯度下降的收敛曲线

注意随着学习速率的变化收敛曲线的变化。在学习速率很小的情况下，你会发现梯度下降需要很长时间才能收敛到最优值。相反，如果学习速率很大，梯度下降可能不会收敛，甚至可能发散!使用你找到的最佳学习率，运行 exlmulti.py 梯度下降直到收敛，找到θ的最终值。接下来，利用$J(\theta)$的这个值来预测一个 1650 平方英尺和 3 间卧室的房子的价格。

3.3 正规方程

线性回归的闭式解是:

$$\theta = \left(X^T X\right)^{-1} X^T \vec{y}.$$

使用这个公式不需要任何特征缩放，你将在一次计算中得到一个精确的解方:不存在像梯度下降那样的“直到收敛为止的循环”。

你需要完善 normalEqn.py 中的代码。用上面的公式计算θ。请记住，虽然不需要缩放特征，但我们仍然需要在 X 矩阵中添加一列 1，以得到截距项(θ_0)。exlmulti.py 中的代码会把 1 的列加到 X 上。最后你需要完善 exlmulti.py 中的代码，使用正规方程计算出的θ，来预测一个 1650 平方英尺和 3 间卧室的房子的价格。