实验2 线性回归

实验目的

1. 实现单变量和多变量线性回归算法。
2. 初次应用Numpy和Matplotlib类库，以及使用jupyter notebook。
3. 初步体验机器学习的数据、模型、算法，体验机器学习模型的训练与预测。
4. 能够在VS Code中使用jupyter notepad进行开发

实验数据

1. ex1data1.txt-用于单变量线性回归的快餐车收益数据集
2. ex1data2.txt-用于多变量线性回归的波特兰房价数据集

实验步骤

1. 单变量线性回归

假设你是一家连锁快餐车的CEO，并且正在考虑在新的城市布设新的快餐车。其主要判定依据是根据城市人口估算未来快餐车的收益。CEO手中有50个城市快餐车的收益数据，城市的人口数和快餐车的收益大致呈线性关系，请据此使用线性回归实现一个收益模型。

文件 ex1data1.txt中存储了本次线性回归实验的快餐车收益数据集，第一列为一个城市的人口数量，第二列为餐车在对应城市获得的收益。

1.1 读取数据

首先你需要做的是将 ex1data1.txt 文件中的数据进行读取，使用的方法是 numpy.loadtxt()，读取完成后将所读数据的规模和维数进行打印。

```python
ex1data1 = './ex1data1.txt'
data1 = np.loadtxt(ex1data1,delimiter=',')
print(data1.shape, data1.ndim)
```

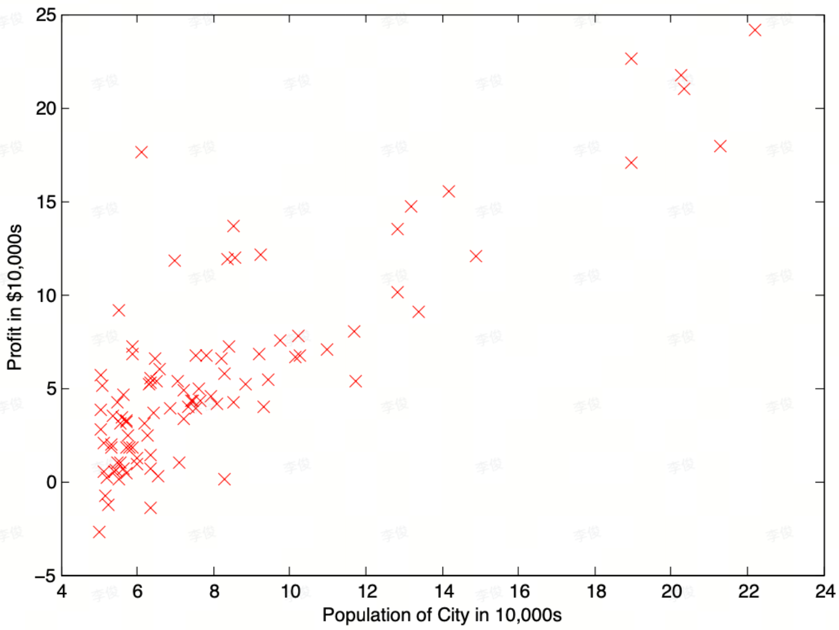
1.2 可视化数据

对数据进行可视化有助于更好的理解数据集的分布，对于本次实验的数据， 可以通过绘制散点图进行可视化，因为数据只有两个特征（人口数量、利润）。 画图需要使用 matplotlib 库中的相关函数。

```python
x=data1[:,0]
y=data1[:,1]
print(x.shape, x.ndim)

#使用plt.xlabel plt.ylabel plt.plot函数来进行数据可视化
plt.xlabel('Population of City in 10000s')
plt.ylabel('Profit in $10000s')
plt.plot(x, y, 'xr')
```

数据可视化的结果如下图所示。



1.3 训练线性回归模型

该部分你要完成输入数据的预处理、损失函数的编写、梯度下降法求模型参数、 解析法直接计算最优解。

1.3.1 数据预处理

在线性回归的模型参数中会存在一个w0，这是参数w被声明为n+1维的原因。为对应模型中的w0，为输入数据X增加一列全1列，便可以将f(x)统一表示成w^T*x。

```python
m=data1.shape[0] # m为是数据样本个数
x0 = np.ones((m,1))
x1 = data1[:, [0]] #在本实验中，data1[:,0]返回的是一维数组，使用data1[:, [0]]会返回m*1的二维数组
X = np.hstack((x0, x1)).T #hstack为水平拼接函数
y = data1[:, [1]].T
```

此外有两点需要注意：

1. 在文件中存储的数据以行为单位来存储数据样本，但在教材和论文中则以列向量的方式存储。如果想跟教材保持一致，需要在数据装载后对数据进行转置。
2. numpy.loadtxt()所读取的单列数据会存至一维数组，但不利于后续的转置或者矩阵乘等操作。因此，在读取的维度上再加上方括号便会实现扩增一维的操作。如在ex1data1.txt中读取的data1[:,0]是长度为m一维数组，使用data1[:, [0]]返回的是尺寸为m*1的二维数组。

1.3.2 代价函数

线性回归的代价函数如下。

$$L(\boldsymbol{w})=\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})-y^{(i)}\right)^2=\frac{1}{2m}\sum_{i=1}^{m}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}-y^{(i)}\right)^2$$

此处需要再次强调代价函数是关于模型参数w的函数，而不是X和y。在最小化代价函数的过程中，调整的是参数w，而X和y是给定的、不发生变化的。

请在computeCost()函数中实现代价函数的计算。其中会使用data1数据作为代价函数L(w)的输入，如果w被初始化为0，那么函数返回值应为32.07。

```python
def computeCost(X, y, w):
    L = 0
    # Your code here
    return L
```

📌 实验中会大量使用numpy.dot(,)进行向量和矩阵乘，需要确认输入和结果的矩阵（多维数组）维度。根据前几届的经验，这是出问题的重灾区。

1.3.3 用于单变量线性回归的梯度下降法算法

为最小化代价函数并获得对应的模型参数w，采用梯度下降法进行优化。通过迭代不断调整参数w，使得代价函数L(w)不断变小，多次迭代收敛后便可便可获得最小的L(w)和对应的模型参数w。下面是用于单变量线性回归的梯度下降法：

Algorithm 1: 用于单变量线性回归的梯度下降法

Input: 样本数据集 $\boldsymbol{X}_{(2,m)}$，$\boldsymbol{y}_{(1,m)}$，最大迭代次数 T

Output: $\boldsymbol{w}_{(2,1)}$

初始化 $\boldsymbol{w}_0$ 和 $\boldsymbol{w}_1$

for $t = 1, \cdots, T$ **do**

  $\boldsymbol{w}_0 := \boldsymbol{w}_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)$

  $\boldsymbol{w}_1 := \boldsymbol{w}_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right) \boldsymbol{x}^{(i)}$

  if 迭代前后 $\boldsymbol{w}$ 变化小于 ε **then**

    break

  end

end

其中代价函数关于模型参数的梯度如下，这也是梯度下降法的重要组成部分。

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}_j} = \frac{1}{m} \sum_{i=1}^{m} (f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}) \cdot \boldsymbol{x}_j^{(i)}$$

然后，根据上面给出的算法，实现gradientDescent(X, y, w, alpha, num_iters)中使用梯度更新模型参数的部分。

```python
def gradientDescent(X, y, w, alpha, num_iters):
    cost_history = numpy.zeros(num_iters)
    for i in range(num_iters):
        # your code here

        cost_history[i] = computeCost(X, y, w)

    return w, cost_history
```

如果模型参数w被初始化为0，那么，gradientDescent()函数将输出：

array([[-3.63029144],
    [1.16636235]])

1.4 使用训练得到的模型进行预测并可视化结果

当线性模型的参数w被学到后，便可以使用f(x)=w0 + w1x1进行预测。假设目标城市有5万和25万人，则可以使用学到的w对其进行预测。

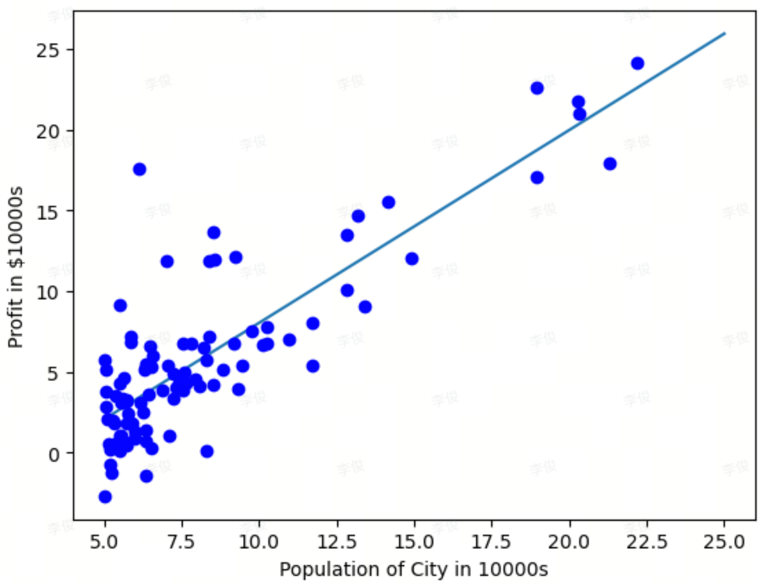
```python
x0 = 5.0
y0 = w[0] + x0*w[1]
```

```
x1 = 25.0
y1 = w[0] + x1*w[1]
print(y0, y1)
```

cell正确结果： [2.06938734] [25.93006023]

此外，还可以在训练数据上绘制学到的线性回归模型，通过上一步计算的(x0, y0)和(x1, y1)来绘制。

```
plt.xlabel('Population of City in 10000s')
plt.ylabel('Profit in $10000s')
plt.plot((x0, x1), (y0, y1))
plt.plot(data1[:,0], y, "ob")
```



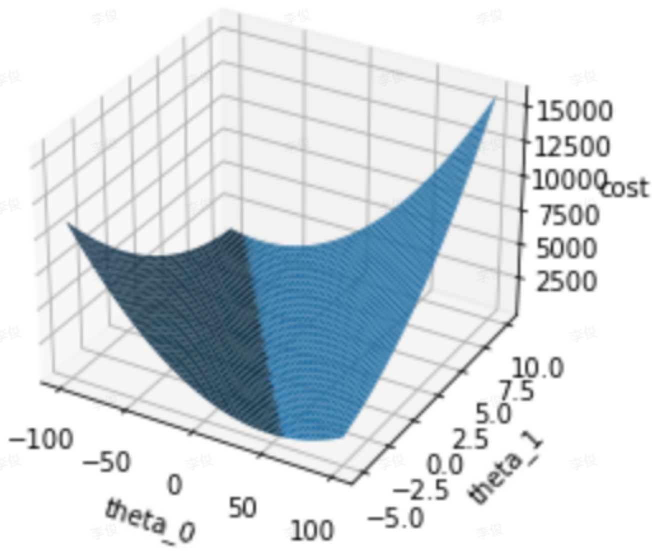
1.5 可视化线性回归的代价函数

因为单变量线性回归的代价函数只有w0和w1两个参数，可以通过plot_surface()函数和contour()来绘制代价函数。从生成的图可看出线性回归的代价函数是一个碗型函数，具有全局最小值，梯度下降法的每一次迭代都能够让(w0, w1)向着全局最小值移动。

```
print('Visualizing L(w_0, w_1) ...')
w0_vals = np.linspace(-100, 100, 100)
w1_vals = np.linspace(-5, 10, 100)
cost_vals = np.zeros((len(w0_val), len(w1_val)))
for i in range(len(w0_vals)):
    for j in range(len(w1_vals)):
        cost_vals[i, j] = computeCost(X, y, np.array([[w0_vals[i]],
[w1_val[j]]]))
```

```python
8   cost_vals = cost_vals.T
9
10  w0_vals_m, w1_vals_m = np.meshgrid(w0_vals, w1_vals)
11
12  fig = plt.figure()
13  ax = plt.axes(projection = '3d')
14  ax.plot_surface(w0_vals_m, w1_vals_m, cost_vals, cmap='rainbow')
15  ax.set_xlabel('w_0')
16  ax.set_ylabel('w_1')
17  ax.set_zlabel('cost')
18  plt.show()
```

代码1~8行负责在指定值域内准备w0和w1的各种组合，然后计算各参数组合下的代价值cost_vals。之后使用matplotlib的plot_surface绘制代价函数L(w0, w1)的图像，如下图所示。



2. 多变量线性回归

在这部分中，你需要使用多元变量的线性回归来预测房价。根据常识，房价大致同房屋面积、房屋数量、楼层成线性关系。为了能够较为准确的预测房价，可以收集目标区域最近出售房屋的相关新信息，使用多变量线性回归建立一个房价预测模型。

ex1data2.txt 文件是俄勒冈州波特兰市的房价数据。第一列是房屋尺寸（以平方英尺），第二列是卧室的数量，第三列是房屋价格。

2.1 特征规范化（Feature Normalization）

在波特兰房价数据中，房屋尺寸的数值和卧室数差别千余倍，会对参数w的优化范围和速度产生很大影响。此类情况使用特征规范化（feature normalization）对各维特征进行处理。此处选择z-score规范化。

$$z = \frac{x - \mu}{\sigma}$$

其中，均值计算使用numpy.mean()，标准差使用numpy.std()

```
def featureNormalize(x):
    mu = np.mean(x, axis = 0)
    std_sigma = np.std(x, axis = 0, ddof = 1) # ddof = k,最后平方和除以 n - k
    return (x - mu) / sigma, mu, std_sigma
```

2.2 使用向量法实现多变量的线性回归的训练和预测

2.2.1 向量化代价函数

在1.3.2中给出的代价函数可以向量化成下式，由于numpy的n维数组间的运算经过优化，使用向量化或矩阵化的方式进行计算，其性能和代码书写效率都得到很大提升。

$$L(\boldsymbol{w}) = \frac{1}{2m}(\boldsymbol{w}^T\boldsymbol{X} - \boldsymbol{y})^T(\boldsymbol{w}^T\boldsymbol{X} - \boldsymbol{y})$$

请根据这个向量化的代价函数升级原先的computeCost()到computeCostMulti()函数。

2.2.2 向量化多变量梯度下降法

从1.3节的单变量梯度下降法可以看出，当x0均被设为1之后，所有n+1维梯度便可统一表示为

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}_j} = \frac{1}{m}\sum_{i=1}^{m}(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}) \cdot \boldsymbol{x}_j^{(i)}$$

向量化后可表示为

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{m}(\boldsymbol{w}^T\boldsymbol{X} - \boldsymbol{y})\boldsymbol{X}^T$$

请根据这个向量化的梯度升级gradientDescent()到gradientDescentMulti()函数。

2.2.3 房价预测

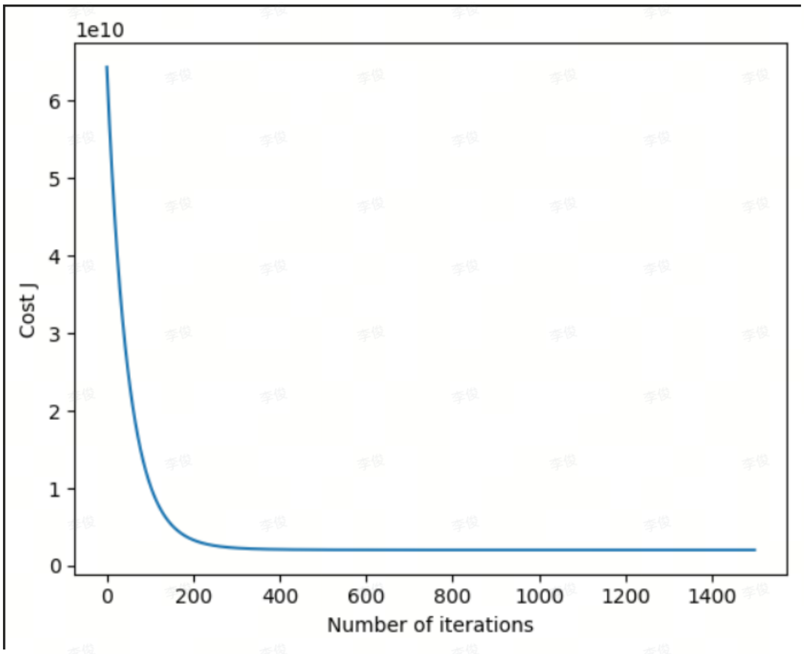
使用学习到的线性回归模型对面积1650平方英尺、3个卧室的房子价格。首先，需要对输入数据进行规范化（feature normalization），均值和方差使用2.1节中统计出来的值。然后，使用上一节

2.2.4 通过学习曲线（Learning Curve）选择合适的学习率（Learning Rate）

对于梯度下降，学习率alpha是一个重要的需要人为设定的超参数（hyper-parameter）。如果它过小，则迭代收敛速度过慢；如果过大，则容易在最小值附近振荡而不收敛。

为更好地选择梯度下降的学习率，可以通过预先存储的cost_history绘制学习曲线（Learning Curve）来了解迭代过程中代价函数的取值变化。

之后，可以为学习率设置一定范围如逐个枚举0.3, 0.1, 0.03, 0.01, 0.003。之后通过绘制的学习曲线来观看收敛效果，进而选择收敛快不振荡的学习率。



2.3 使用解析法最小化代价函数

由于线性代数的代价函数是关于w的二次函数，因此，它一定存在最小解，且在X^T*X可逆的情况下，可通过解析法求解，并获得如下的结果

$$\boldsymbol{w}^* = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\boldsymbol{y}$$

利用numpy适合进行矩阵运算的优势，使用上式可直接计算线性回归的模型参数w。其中，计算矩阵的逆可以使用numpy.linalg.inv()函数。