统计学习与模式识别实验二：梯度下降线性回归

蔡与望　2020010801024

一　实验原理

1.1　线性回归

线性回归的目标是，预测任意输入向量$x \in R^n$的输出值y。例如，在预测房价时，x是房子的各种特征（大小、房间数量等），y则是房价。

我们的目标是找到一个函数$y = h(x)$，它对于每条已知数据$\left(x^{(i)}, y^{(i)}\right)$，都有$y^{(i)} \approx h\left(x^{(i)}\right)$。如果我们能够找到这样的函数，并且已知的数据足够多，那么即使碰到了新的房子，我们也相信这个函数能够预测出它的房价。

为了找到这样的函数，我们先要确定$h(x)$的表达形式。在线性回归中，我们假定它是一个线性函数：

$$h_\theta(x) = \Sigma_j \theta_j x_j = \theta^T x$$

在刻画$h_\theta\left(x^{(i)}\right)$与$y^{(i)}$的相近程度时，使用下面的损失函数：

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

1.2　梯度下降

现在，目标就转化为找到一个θ，使得$J(\theta)$最小。梯度下降法能够很好的解决这一问题。我们首先计算出$J(\theta)$在当前θ值下的梯度：

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

其中，

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

由于梯度的方向是函数值增加最快的方向，所以只要沿着梯度的反方向步进，就能够迅速地找到该函数的极小值。也即，每次迭代都有：

$$\theta' = \theta - \alpha \nabla_\theta J(\theta), \alpha > 0$$

其中的α是每次递进的步长。如果步长太长，就有可能在极小值周围“徘徊”；如果步长太短，则迭代次数可能过多。

二　代码实现

读取房价数据集，随机打乱记录顺序。

```
housing = np.loadtxt("housing.data").T
housing = housing[:, np.random.permutation(housing.shape[1])]
```

划分 400 条数据作为训练集，剩下的作为测试集。经过观察发现，训练集量纲差距较大，所以需要做归一化预处理。另外，还需要加一个全为 1 的偏置。

```
X = housing[:-1, :]
X = (X - X.mean(axis=1, keepdims=True)) / X.std(axis=1,
keepdims=True)
X = np.vstack((np.ones(X.shape[1]), X))
y = housing[-1, :]

TRAINSET_SIZE = 400
train_X = X[:, :TRAINSET_SIZE]
train_y = y[:TRAINSET_SIZE]
test_X = X[:, TRAINSET_SIZE:]
test_y = y[TRAINSET_SIZE:]
```

使用梯度下降法训练权重向量。学习率为 0.01，迭代 1000 次。

```
m, n = train_X.shape
theta = np.random.rand(m, 1)

LEARNING_RATE = 0.01
ITERATION_NUM = 1000

costs = []

for iteration in range(ITERATION_NUM):
    difference = np.dot(theta.T, train_X) - train_y
    gradient = np.dot(train_X, difference.T) / n
    theta -= LEARNING_RATE * gradient

    cost = np.sum(difference ** 2) / (2 * n)
    if iteration % 100 == 0:
        costs.append(cost)
```

绘制预测值与真实值对比的散点图，和损失值变化的折线图。

```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

sort_index = test_y.argsort()
test_X = test_X[:, sort_index]
test_y = test_y[sort_index]

x_axis = np.arange(len(test_y))
predictions = [np.dot(theta.T, x)[0] for x in test_X.T]
axes[0].scatter(x_axis, predictions, s=5, c="blue", label="预测值
```

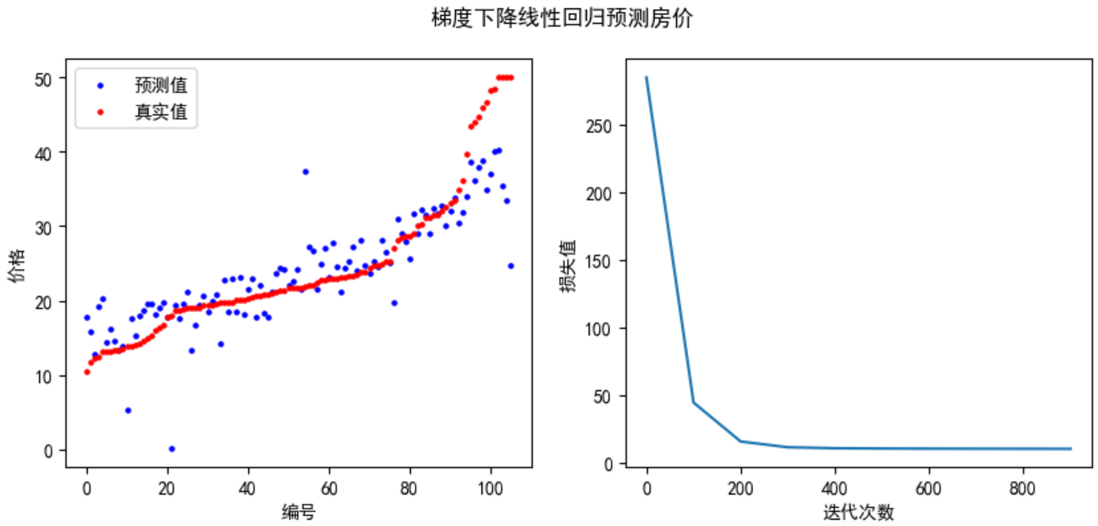
```
")
axes[0].scatter(x_axis, test_y, s=5, c="red", label="真实值")
axes[0].set_xlabel("编号")
axes[0].set_ylabel("价格")
axes[0].legend()

axes[1].plot(np.arange(len(costs)) * 100, costs)
axes[1].set_xlabel("迭代次数")
axes[1].set_ylabel("损失值")

fig.suptitle("梯度下降线性回归预测房价")
plt.show()
```

三 结果分析

绘制出的散点图与折线图如下。



左侧是预测值（蓝）与真实值（红）的对比。可以看到，预测值基本在真实值附近波动；偶尔有较大误差，但数量在可接受范围内。这说明，线性回归的代码在逻辑上是正确的，并且预测效果也较好。

右侧是损失值（loss function）随迭代次数增加的变化。可以看到，损失值在 0-100 次迭代中迅速下降，在 100-200 次迭代中逐渐平缓，在 200-1000 次迭代中逐渐收敛到一个固定值。通过 print 函数打印可知，这个固定值大约在 10 左右。这正是预测值波动的原因。

四 总结体会

在本次实验前，我只会调用 scikit-learn 等第三方库中实现的 LinearRegression 类来实现线性回归任务；但本次实验中，我通过理解它的底层数学原理，编写了梯度下降法的核心代码，从而实现了一个简单的、多维的线性回归。这让我对梯度下降有了更深刻的认识。

同时，本次实验拟合的效果还不太如人意。也许 400 条数据作为训练集还是比较少；如果有更多的数据，拟合的效果会更好。