理解

1. 线性：
   - 数学上：满足齐次性、可加性的映射。
   - 机器学习中：一般形容输入特征（自变量）和输出值（因变量）之间的关系。“线性”意味着模型预测输出值作为输入特征的加权和，加上一个常数项（也称为截距）。
   - 通俗地说，若一个多元函数y对它的每个自变量求偏导，结果都是一个（与自变量无关的）常数，那么y与每个自变量之间存在线性关系。
2. 回归：
   - 一组连续随机变量和另一组变量之间关系的统计分析方法。
   - 通俗点说，$x_1, x_2, \ldots, x_d$是输入特征，y是输出特征，想要用一个函数，它的函数值$\hat{y}$（估计的y）与自变量$x_1, x_2, \ldots, x_d$所对应的y(真实的y)最为接近，找到这个函数的过程叫做回归。
3. 线性回归的作用：
   i. 建立的线性回归模型，去利用已经知道的自变量来预测未知的因变量，使得预测值与真实值尽可能接近。（主要）（本题实现此作用）
   ii. 确定$x_1, x_2, \ldots, x_d$(自变量,通常为定量数据)对y(因变量,定量数据)的影响关系情况（线性？非线性？），或者有没有影响。

学习笔记

1. 术语
   - *标签*（label）
     - 是什么：试图预测的目标（比如要预测的房屋价格）。每个标签值是一个标量。
     - 数学表示：第i个：$y^{(i)}$
     - python中表示：第i个： `y[i]`
   - *特征*（feature）
     - 是什么：预测所依据的自变量（面积和房龄）。每组特征值是一个行向量。
     - 数学表示：第i组：$\mathbf{x}^{(i)}$
     - python中表示：第i组： `X[i]`
   - *样本*（sample）
     - 一行数据，包括一组特征值 `X[i]` 和其对应的一个标签值 `y[i]` 。
   - *训练集*（training set）
     - 用来训练模型的许多样本的集合。包括特征集 `train_X` 、标签集 `train_y` 。
     - 数学表示：$\mathbf{X}, \mathbf{y}$.其中，$\mathbf{X}$是n行d列矩阵，$\mathbf{y}$是n维列向量
   - *测试集*（test set）
     - 测试集是只使用一次的许多样本的集合，即在训练完成后评价最终的模型时使用。它既不参与学习参数过程，也不参数超参数选择过程，而仅仅使用于模型的评价。在打kaggle竞赛时，参赛者一般不能看到测试集。
   - *样本数*
     - 是什么：训练集/测试集中样本的个数
     - 表示：一般用 `n` 表示
   - *回归方程*
     - 是什么：假设自变量$\mathbf{x}$和因变量y之间的关系是线性的，即y可以表示为$\mathbf{x}$的各个分量的加权和。用$\mathbf{x}$来计算$\hat{y}$的方程。
     - 数学表示：
       - 对于单个：

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b.$$

       - 对于多个：

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b.$$

   - *参数*（parameters）
     - 是什么：模型通过学习数据而得到的并用于对新数据进行预测的变量。在线性回归中，通常是权重和偏置。在一个简单的线性回归模型中，有形如

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

       的方程。其中$\mathbf{w}$表示权重,b是一个标量(要用广播机制)。
     - pytorch中表示： `net.Parameters()`
     - 包括：
       a. *权重*（weights）
          - 是什么：参数之一，代表了自变量$\mathbf{x}$的各个分量对因变量$\hat{y}$的影响程度。在上述方程中，$\mathbf{w}$表示权重,是一个列向量。

- 数学表示：$\mathbf{w}$。是一个列向量
- pytorch中表示： `net[i].weight.data[j]` 表示神经网络 `net` 第 `i` 层中第 `j` 个神经单元对第 `i-1` 层的所有神经单元的权重。一般，是一个一维实数张量。

b. *偏置*（bias）
- 是什么：模型的参数之一，代表了输出在没有任何输入时的值（即在$\mathbf{x}=\mathbf{0}$时$\hat{y}$的值）。在上述方程中的b就是偏置，是一个标量。
- 数学表示：b。是一个标量。
- pytorch中表示： `net[i].bias.data[j]` 表示神经网络 `net` 第i层中第j个神经单元对第i-1层的所有神经单元的偏置。一般，是一个实数标量。

- *超参数*（Hyperparameters）：
  - 是什么：不能直接从数据中学习到的、需要在开始训练之前设置的参数。例如学习率、批量大小、迭代次数等
  - 包括:

    a. *学习率*（learning rate）
    - 是什么：在梯度下降中，控制在每次迭代中模型参数的更新幅度。较高的学习率可以导致学习过程快速收敛，但也可能跳过最佳值；较低的学习率确保了更稳定的收敛，但学习过程可能会很慢。
    - 数学表示：希腊字母 η

    b. *批量大小*（batch size）
    - 是什么：一轮迭代epoch中，每次所用来计算梯度的小批量样本数量。
    - 数学表示：希腊字母 β

    c. *迭代次数*（number of epochs）
    - 是什么：整个训练数据集循环通过学习算法的次数。太大了浪费算力；太小了损失函数可能无法收敛
- *损失函数*（Loss Function）（本题中用均方损失）
  - 是什么：衡量模型预测值与真实值之间的差异的函数。在线性回归中，通常使用均方误差（Mean Squared Error, MSE）作为损失函数。
  - 数学表示：
    - 单个样本的均方损失：

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2} \left(\hat{y}^{(i)} - y^{(i)}\right)^2.$$

    - 多个样本的损失函数：

$$L(\mathbf{w}, b) =\frac{1}{n}\sum_{i=1}^n l^{(i)}(\mathbf{w}, b) =\frac{1}{n} \sum_{i=1}^n \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2.$$

  - 在pytorch中表示：一般写 `loss=nn.MSELoss()` 。其中， `MSELoss()` 是 `torch.nn` 的均方损失函数

2. *随机梯度下降*（stochastic gradient descent）：
- 是什么：梯度下降最简单的用法是计算损失函数（数据集中所有样本的损失均值） 关于模型参数的导数（在这里也可以称为梯度）。 但实际中的执行可能会非常慢：因为在每一次更新参数之前，我们必须遍历整个数据集。 因此，我们通常会在每次需要计算更新的时候随机抽取一小批样本， 这种变体叫做小批量随机梯度下降（minibatch stochastic gradient descent）。在每次迭代中，我们首先随机抽样一个小批量，它是由固定数量的训练样本组成的。 然后，我们计算小批量的平均损失关于模型参数的导数（也可以称为梯度）。 最后，我们将梯度乘以一个预先确定的正数，并从当前参数的值中减掉。
- 算法描述：

  在每次迭代中，我们首先随机抽样一个批量大小为$\mathcal{B}$的小批量，它是由固定数量的训练样本组成的。
  然后，我们计算小批量的平均损失关于模型参数的导数（也可称为梯度）。
  最后，我们将梯度乘以一个预先确定的学习率η，并从当前参数的值中减掉这一项，更新参数。
  我们用下面的数学公式来表示这一更新过程（∂表示偏导数）：

$$(\mathbf{w},b) \leftarrow (\mathbf{w},b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w},b)} l^{(i)}(\mathbf{w},b).$$

总结一下，算法的步骤如下：

（1）初始化模型参数的值，如随机初始化；

（2）从数据集中随机抽取小批量样本且在负梯度的方向上更新参数，并不断迭代这一步骤。

对于平方损失和仿射变换，我们可以明确地写成如下形式:

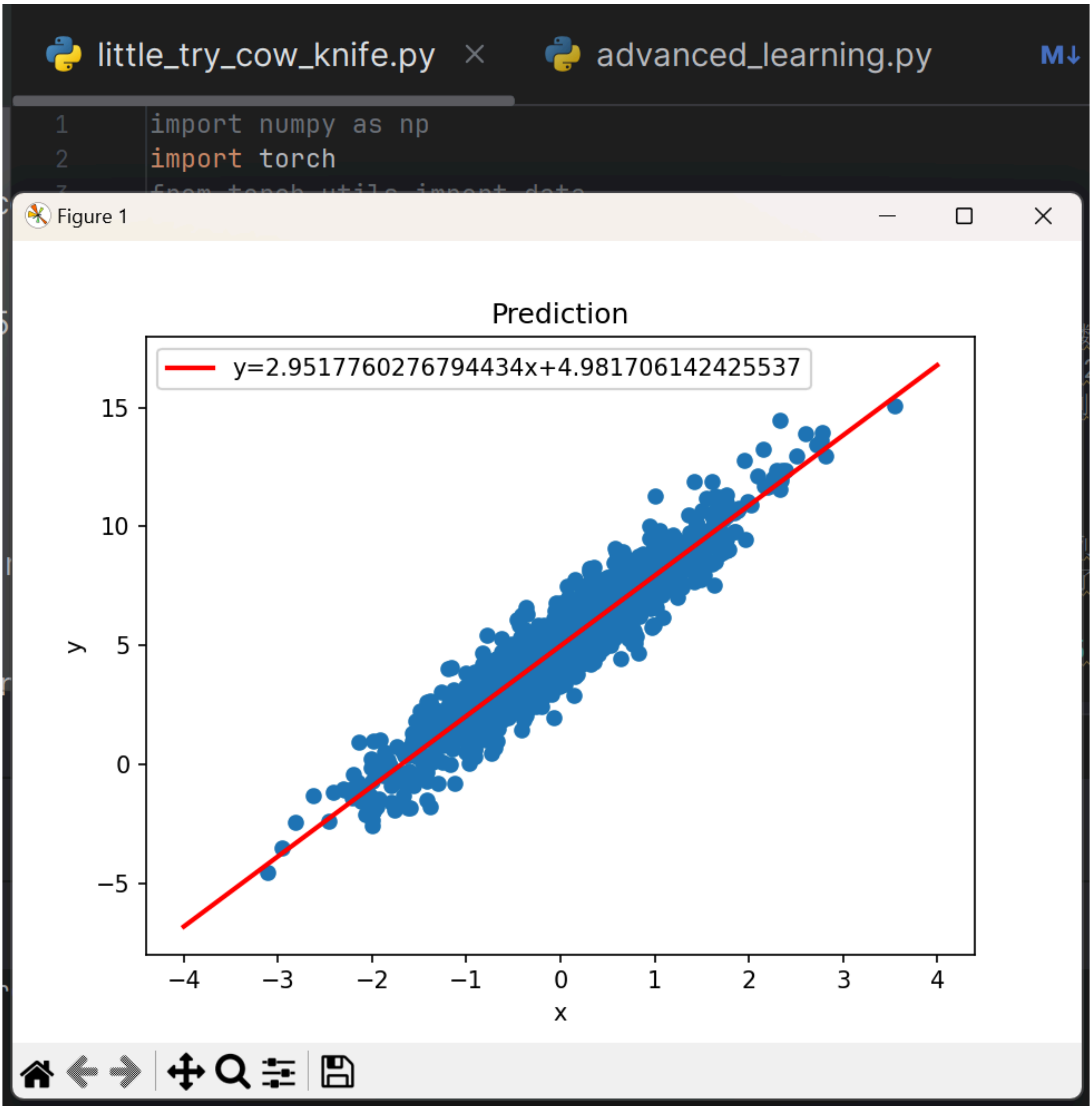
$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right),\\ b &\leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right). \end{aligned}$$

批量大小和学习率的值通常是手动预先指定，而不是通过模型训练得到的。

一轮epoch最好满足：能够用完整个训练集，没有浪费。所以，当样本数不是批量大小的倍数时，剩下的一些数据也要纳入每一轮的梯度下降。

3. *调参*（hyperparameter tuning）：
   选择超参数的过程。超参数通常是我们根据训练迭代结果来调整的，而训练迭代结果是在独立的*验证数据集*（validation dataset）上评估得到的。在此题中，只根据training set的loss来调参。

流程记录

1. 小试牛刀
   拟合结果：
   

2. 基础学习
   i. 生成1000个样本。数据比较多，先写一个.py文件生成数据:

```python
import ...

"""用来随机生成training set"""
1 个用法
def generate(w, b, m,std1,std2):#w,b是给定的参数，m要生成的training set的个数
    X = torch.normal(0, std1, (m, len(w)))#生成正态分布的X（在这里是2*num矩阵），均值为0，标准差为1
    y = torch.matmul(X, w) + b#完全符合线性关系的y。可惜现实中碰不到
    y += torch.normal(0, std2, y.shape)#加噪声
    return X,y.reshape((-1, 1))#把y变成m维的列向量

#人造参数
true_w = torch.tensor([2, -3.4])
true_b = 4.2
m=1000
features, labels = generate(true_w, true_b, m, std1: 1, std2: 0.01)
X=torch.cat( tensors: (features,labels), dim: 1).tolist()
with open('training_set_for_basic_learning.csv', 'w', encoding='utf-8', newline='') as f:
    csv_write=csv.writer(f)
    for i in range(m):
        csv_write.writerow(X[i])
```

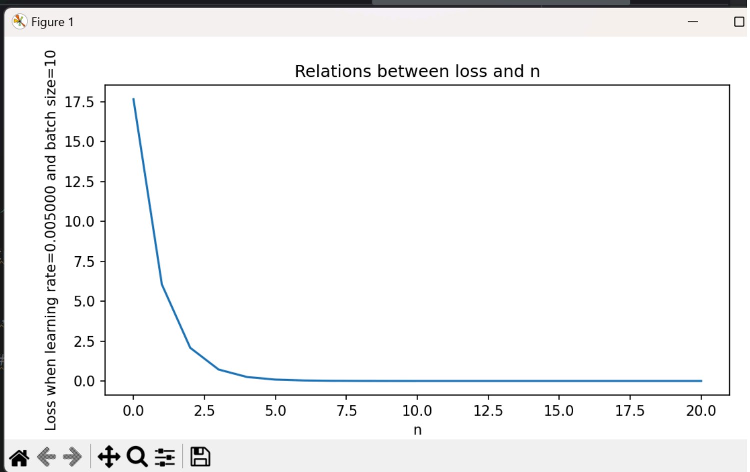
ii. 然后用一个.csv文件来存数据：

training_set_for_basic_learning.csv

```
976  1.4018323421478271,-1.6459463834762573,12.593379020690918
977  0.5140721797943115,0.10562876611948013,4.851898670196533
978  0.7909759283065796,0.6375572085380554,3.621629476547241
979  0.3864027261734009,-1.4380676746368408,9.860892295837402
980  0.2105979025363922,-0.225262612104415 9,5.389762878417969
981  0.3878855109214 7827,-1.3429667949676514,9.546656608581543
982  0.9227322340011597,-0.3120514750480652,7.101775646209717
983  3.1029746532440186,-0.6130111813545227,12.504938125610352
984  2.686011791229248,-0.13225388526916504,10.030187606811523
985  -1.3352587223052979,-0.5465253591537476,3.3697731494903564
986  0.02132505550980568,-1.0234640836715698,7.725276470184326
987  0.390977144241333,0.32069945335388184,3.8985435962677
```

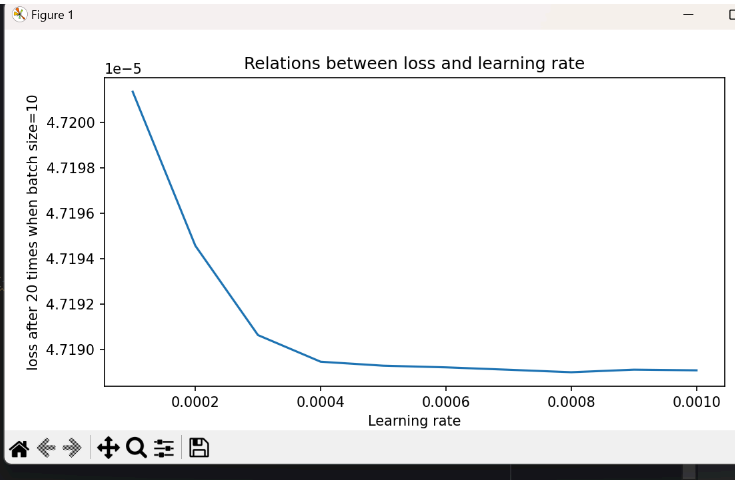
iii. 在一定学习率、批量大小下，损失函数与迭代次数的关系图：

```python
"""第一个关系：loss和迭代次数n"""
w=init_w
b=init_b
#对两个参数都要求导
l_n=[]#存储对应的损失函数值！
"""以下是把没有经过梯度下降损失的计入train_l[0]"""
with torch.no_grad():  # 不纳入梯度计算
    l_n.append((loss(net(features, w, b), labels)).mean())
"""核心代码2：一共进行times轮随机梯度下降，每一轮要用几乎（不排除有剩余的）整
for i in range(n):#进行n轮梯度下降
    for X, y in data_iter(batch_size, features, labels):#从整个t
        l = loss(net(X, w, b), y) #X和y的小批量损失
        l.sum().backward()#先求和，再反向传播！
        sgd( params: [w, b],yita, batch_size) #使用参数的梯度更新参数
    with torch.no_grad():#不纳入梯度计算
        l_n.append((loss(net(features, w, b), labels)).mean())#
        print(f'epoch={i + 1}, loss={float(l_n[i+1]):f}')

plt.figure(figsize=(8, 4),dpi=100)#设置图片的大小、像素
plt.plot( *args: list(range(0,n+1)),l_n)#显示一次数据点图
plt.xlabel('n')
plt.ylabel(f'Loss when learning rate={yita:f} and batch size={batch_size}')
plt.title('Relations between loss and n')
plt.show()
```



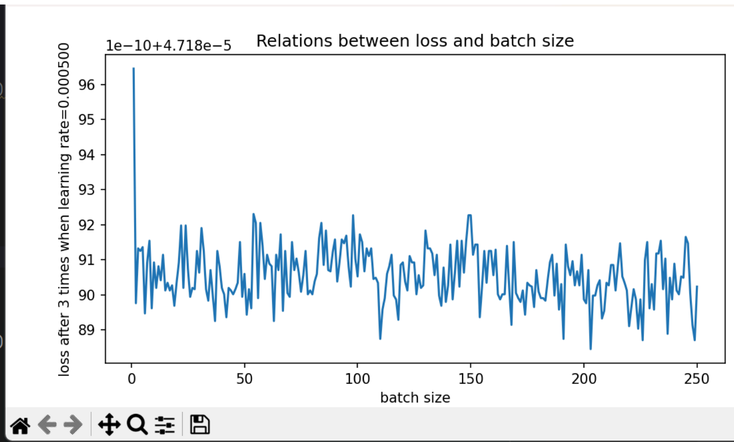
iv. 在一定迭代次数、批量大小下，损失函数与学习率的关系图：

```python
"""第二个关系：loss和学习率"""
lr=list(np.arange(0.0001,0.0011,0.0001))
l_lr=[]
for yita in lr:
    w=init_w
    b=init_b
    for i in range(n):
        for X,y in data_iter(batch_size,features,labels):
            l=loss(net(X,w,b),y)
            l.sum().backward()
            sgd( params: [w,b],yita,batch_size)
    with torch.no_grad():
        l_lr.append((loss(net(features, w, b), labels)).mean())#这
        print(f'learning rate={yita:f},loss={float(l_lr[-1]):f}')

plt.figure(figsize=(8,4),dpi=100)
plt.plot( *args: lr,l_lr)
plt.xlabel("Learning rate")
plt.ylabel(f"loss after {n} times when batch size={batch_size}")
plt.title('Relations between loss and learning rate')
plt.show()
```



v. 在一定迭代次数、学习率下，损失函数与批量大小的关系图：

```python
"""第三个关系：loss和批量大小"""
yita = 0.0005
n=3
bs=list(range(1,251,1))#+list(range(20,50,2))+list(range(50,255,5)
l_bs=[]
for batch_size in bs:
    w=init_w
    b=init_b
    for i in range(n):
        for X,y in data_iter(batch_size,features,labels):
            l=loss(net(X,w,b),y)
            l.sum().backward()
            sgd( params: [w,b],yita,batch_size)
    with torch.no_grad():
        l_bs.append((loss(net(features,w,b),labels)).mean())
        print(f'batch size={batch_size},loss={float(l_bs[-1]):f}')


plt.figure(figsize=(8,4),dpi=100)
plt.plot( *args: bs,l_bs)
plt.xlabel("batch size")
plt.ylabel(f"loss after {n} times when learning rate={yita:f}")
plt.title('Relations between loss and batch size')
plt.show()
```

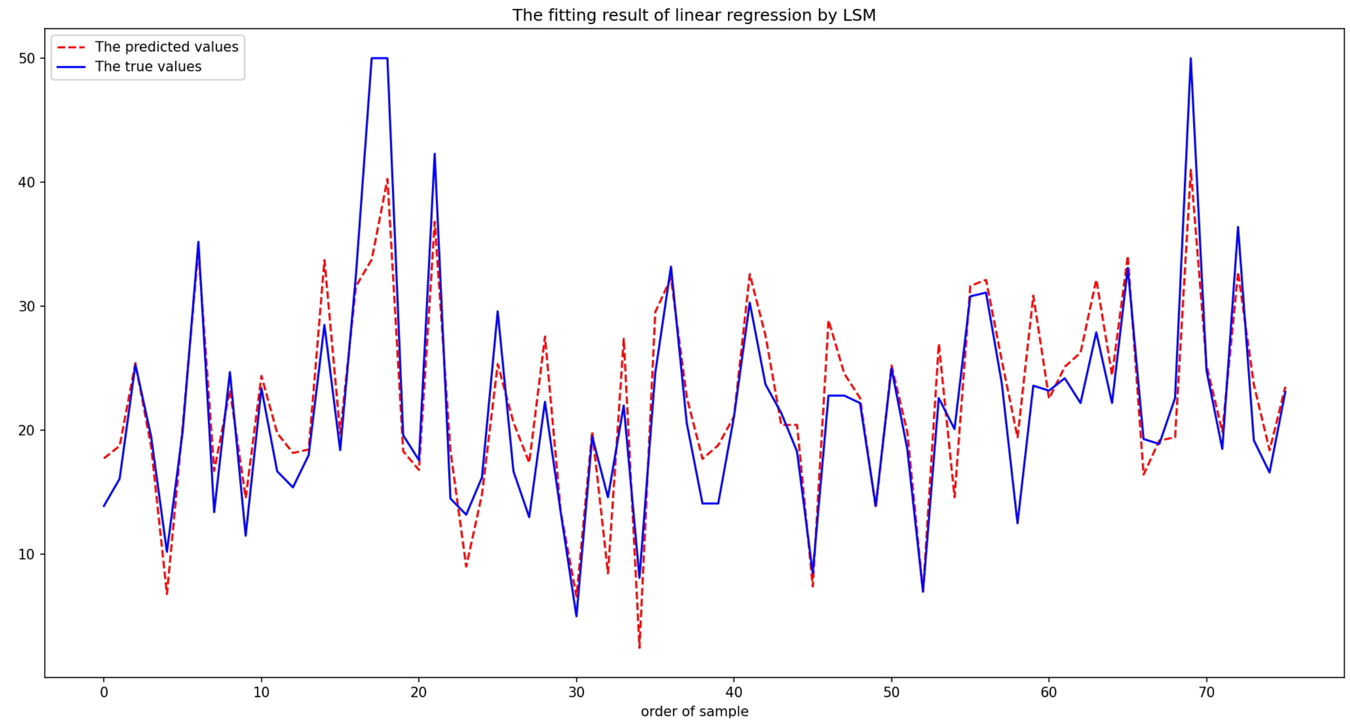


3. 进阶学习：

i. 从网上下载波士顿房价数据集，存在.csv文件中：

```
py    advanced_learning.py    linear_regression.md    boston.csv
CRIM,ZN,INDUS,CHAS,NOX,RM,AGE,DIS,RAD,TAX,PIRATIO,B,LSTAT,MEDV
0.00632,18,2.31,0,0.538,6.575,65.2,4.09,1,296,15.3,396.9,4.98,24
0.02731,0,7.07,0,0.469,6.421,78.9,4.9671,2,242,17.8,396.9,9.14,21.6
0.02729,0,7.07,0,0.469,7.185,61.1,4.9671,2,242,17.8,392.83,4.03,34.7
0.03237,0,2.18,0,0.458,6.998,45.8,6.0622,3,222,18.7,394.63,2.94,33.4
0.06905,0,2.18,0,0.458,7.147,54.2,6.0622,3,222,18.7,396.9,5.33,36.2
0.02985,0,2.18,0,0.458,6.43,58.7,6.0622,3,222,18.7,394.12,5.21,28.7
0.08829,12.5,7.87,0,0.524,6.012,66.6,5.5605,5,311,15.2,395.6,12.43,22.9
0.14455,12.5,7.87,0,0.524,6.172,96.1,5.9505,5,311,15.2,396.9,19.15,27.1
0.21124,12.5,7.87,0,0.524,5.631,100,6.0821,5,311,15.2,386.63,29.93,16.5
0.17004,12.5,7.87,0,0.524,6.004,85.9,6.5921,5,311,15.2,386.71,17.1,18.9
```

ii. 取430个样本作为训练集，用最小二乘法线性回归，取76个样本作为测试集，拟合结果图：


The fitting result of linear regression by LSM
The predicted values
The true values
50
40
30
20
10
0
10
20
30
40
50
60
70
order of sample