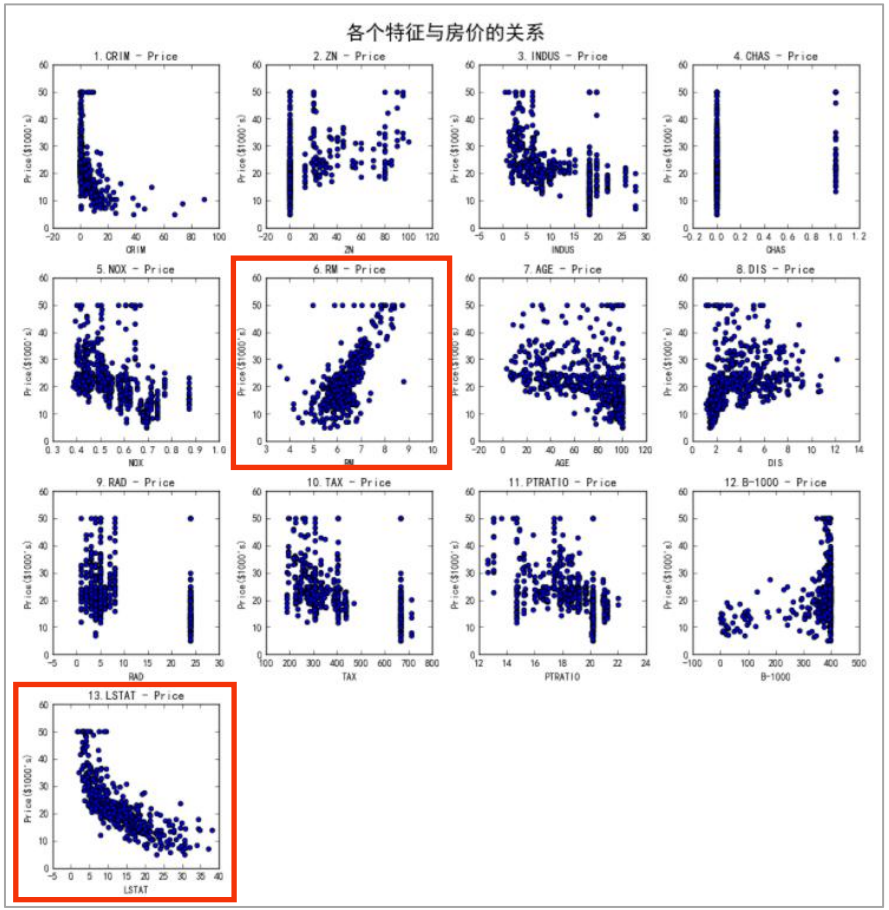
10.6 实例：波士顿房价预测

10.6.1 实例：波士顿房价预测(1)
一元线性回归

- 波士顿房价数据集


各个特征与房价的关系
1. CRIM - Price
2. ZN - Price
3. INDUS - Price
4. CHAS - Price
5. NOX - Price
6. RM - Price
7. AGE - Price
8. DIS - Price
9. RAD - Price
10. TAX - Price
11. PTRATIO - Price
12. B-1000 - Price
13. LSTAT - Price
Price($1000's)
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B-1000
LSTAT

加载数据集

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
boston_housing = tf.keras.datasets.boston_housing
(train_x, train_y), (test_x, test_y) = boston_housing.load_data()
```

In [3]:
```
train_x.shape, train_y.shape
```

Out[3]:
```
((404, 13), (404,))
```

In [4]:
```
test_x.shape, test_y.shape
```

Out[4]:
```
((102, 13), (102,))
```

数据处理

```
In [5]: x_train=train_x[:,5]
        y_train=train_y

In [6]: x_train.shape, y_train.shape
Out[6]: ((404,), (404,))

In [7]: x_test=test_x[:,5]
        y_test=test_y

In [8]: x_test.shape, y_test.shape
Out[8]: ((102,), (102,))
```

- 设置超参数

```
In [9]: learn_rate = 0.04
        iter = 2000
        display_step =200
```

- 设置模型参数初始值

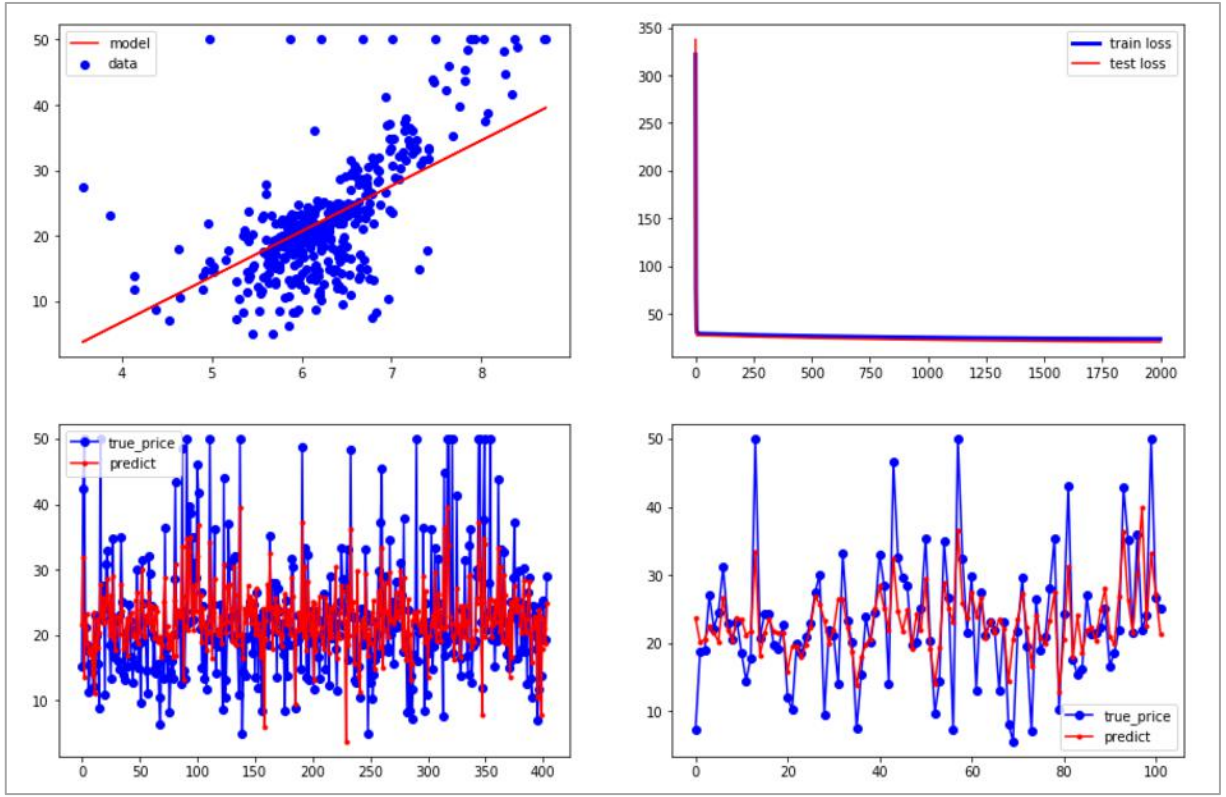
```
In [10]: np.random.seed(612)
         w = tf.Variable(np.random.randn())
         b = tf.Variable(np.random.randn())

In [11]: w.numpy().dtype, b.numpy().dtype
Out[11]: (dtype('float32'), dtype('float32'))
```

- 训练模型

In [12]:
```
mse_train=[]
mse_test=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:

        pred_train = w*x_train+b
        loss_train = 0.5*tf.reduce_mean(tf.square(y_train-pred_train))

        pred_test = w*x_test+b
        loss_test = 0.5*tf.reduce_mean(tf.square(y_test-pred_test))

    mse_train.append(loss_train)
    mse_test.append(loss_test)

    dL_dw,dL_db = tape.gradient(loss_train,[w,b])
    w.assign_sub(learn_rate*dL_dw)
    b.assign_sub(learn_rate*dL_db)

    if i % display_step == 0:
        print("i: %i, Train Loss: %f, Test Loss: %f" % (i, loss_train,loss_test))
```

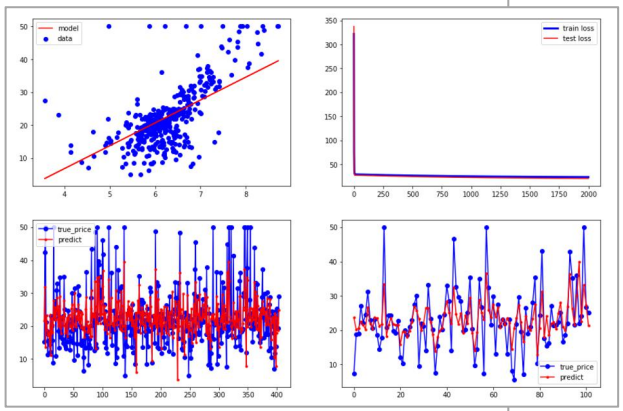
```
i: 0, Train Loss: 321.837585, Test Loss: 337.568665
i: 200, Train Loss: 28.122614, Test Loss: 26.237764
i: 400, Train Loss: 27.144741, Test Loss: 25.099329
i: 600, Train Loss: 26.341951, Test Loss: 24.141077
i: 800, Train Loss: 25.682898, Test Loss: 23.332981
i: 1000, Train Loss: 25.141848, Test Loss: 22.650158
i: 1200, Train Loss: 24.697674, Test Loss: 22.072004
i: 1400, Train Loss: 24.333027, Test Loss: 21.581432
i: 1600, Train Loss: 24.033665, Test Loss: 21.164263
i: 1800, Train Loss: 23.787907, Test Loss: 20.808695
i: 2000, Train Loss: 23.586145, Test Loss: 20.504940
```

- 可视化输出


model
data
train loss
test loss
true_price
predict
true_price
predict

10.6.1 波士顿房价预测：一元线性回归

In [13]:
```
plt.figure(figsize=(15, 10))

plt.subplot(221)
plt.scatter(x_train, y_train, color="blue", label="data")
plt.plot(x_train, pred_train, color="red", label="model")
plt.legend(loc="upper left")

plt.subplot(222)
plt.plot(mse_train, color="blue", linewidth=3, label="train loss")
plt.plot(mse_test, color="red", linewidth=1.5, label="test loss")
plt.legend(loc="upper right")

plt.subplot(223)
plt.plot(y_train, color="blue", marker="o", label="true_price")
plt.plot(pred_train, color="red", marker=".", label="predict")
plt.legend()

plt.subplot(224)
plt.plot(y_test, color="blue", marker="o", label="true_price")
plt.plot(pred_test, color="red", marker=".", label="predict")
plt.legend()

plt.show()
```



神经网络&深度学习 Google TENSORFLOW

10.6.2 实例：波士顿房价预测(2)
多元线性回归

中国大学MOOC

波士顿房价数据集

序号	变量名	说 明	示 例
1	CRIM	城镇人均犯罪率	0.00632
2	ZN	超过25000平方英尺的住宅用地所占比例	18.0
3	INDUS	城镇非零售业的商业用地所占比例	2.31
4	CHAS	是否被Charles河流穿过（取值1：是；取值0：否）	0
5	NOX	一氧化碳浓度	0.538
6	RM	每栋住宅的平均房间数	6.575
7	AGE	早于1940年建成的自住房屋比例	65.2
8	DIS	到波士顿5个中心区域的加权平均距离	4.0900
9	RAD	到达高速公路的便利指数	1
10	TAX	每10000美元的全值财产税率	296
11	PTRATIO	城镇中师生比例	15.3
12	B	反映城镇中的黑人比例的指标，越靠近0.63越小；$B=1000*(BK-0.63)^2$，其中BK是黑人的比例。	396.90
13	LSTAT	低收入人口的比例	7.68
14	MEDV	自住房屋房价的平均房价（单位为1000美元）	24.0

一维数组归一化

```
In [1]: import numpy as np

In [2]: area=np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
                       106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
        room=np.array([3, 2, 2, 3, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2])

In [3]: x1=(area -area.min())/(area.max()-area.min())
        x2=(room -room.min())/(room.max()-room.min())

In [4]: x1, x2

Out[4]: (array([0.99912223, 0.63188501, 0.58251042, 0.84935264, 0.3542901 ,
                0.57153829, 0.84584156, 0.73612025, 0.65591398, 1.        ,
                0.07504937, 0.        , 0.23140224, 0.17676103, 0.37689269,
                0.43120474]),
         array([1. , 0.5, 0.5, 1. , 0. , 0.5, 1. , 0.5, 0.5, 1. , 0. , 0. , 0. ,
                0. , 0.5, 0.5]))
```

二维数组归一化——循环实现

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
boston_housing = tf.keras.datasets.boston_housing
(train_x, train_y), (test_x, test_y) = boston_housing.load_data()
```

In [3]:
```
train_x.shape, train_y.shape
```

Out[3]:
```
((404, 13), (404,))
```

In [4]:
```
test_x.shape, test_y.shape
```

Out[4]:
```
((102, 13), (102,))
```

```
In [1]: import numpy as np

In [2]: x = np.array([[3.,  10,  500],
                      [2.,  20,  200],
                      [1.,  30,  300],
                      [5.,  50,  100]])

In [3]: x.dtype, x.shape
Out[3]: (dtype('float64'), (4, 3))

In [4]: len(x)
Out[4]: 4

In [5]: x.shape[0], x.shape[1]
Out[5]: (4, 3)
```

```
In [5]: x.shape[0],x.shape[1]

Out[5]: (4, 3)

In [6]: for i in range(x.shape[1]):
            x[:,i]=(x[:,i]-x[:,i].min())/(x[:,i].max()-x[:,i].min())

In [7]: x

Out[7]: array([[0.5 , 0.  , 1.  ],
               [0.25, 0.25, 0.25],
               [0.  , 0.5 , 0.5 ],
               [1.  , 1.  , 0.  ]])
```

二维数组归一化——广播运算

```
In [8]: x = np.array([[3.,  10.,  500.],
                      [2.,  20.,  200.],
                      [1.,  30.,  300.],
                      [5.,  50.,  100.]])

In [9]: x.min(axis=0)
Out[9]: array([  1.,  10., 100.])

In [10]: x.max(axis=0)
Out[10]: array([  5.,  50., 500.])

In [11]: x.max(axis=0)- x.min(axis=0)
Out[11]: array([  4.,  40., 400.])
```

```
In [11]: x.max(axis=0)- x.min(axis=0)
Out[11]: array([  4.,   40., 400.])

In [12]: x-x.min(axis=0)
Out[12]: array([[  2.,   0., 400.],
                [  1.,  10., 100.],
                [  0.,  20., 200.],
                [  4.,  40.,   0.]])

In [13]: (x-x.min(axis=0)) /(x.max(axis=0)- x.min(axis=0))
Out[13]: array([[0.5 , 0.  , 1.  ],
                [0.25, 0.25, 0.25],
                [0.  , 0.5 , 0.5 ],
                [1.  , 1.  , 0.  ]])
```

波士顿房价数据多元线性回归

- 加载数据集

```
In [1]: import tensorflow as tf
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: boston_housing = tf.keras.datasets.boston_housing
        (train_x, train_y), (test_x, test_y) = boston_housing.load_data()

In [3]: train_x.shape, train_y.shape
Out[3]: ((404, 13), (404,))

In [4]: test_x.shape, test_y.shape
Out[4]: ((102, 13), (102,))

In [5]: num_train=len(train_x)
        num_test=len(test_x)
```

数据处理

In [6]:
```
x_train=(train_x-train_x.min(axis=0))/(train_x.max(axis=0)- train_x.min(axis=0))
y_train=train_y

x_test=(test_x-test_x.min(axis=0))/(test_x.max(axis=0)- test_x.min(axis=0))
y_test=test_y
```

In [7]:
```
x0_train = np.ones(num_train).reshape(-1,1)
x0_test = np.ones(num_test).reshape(-1,1)
```

In [8]:
```
X_train=tf.cast(tf.concat([x0_train,x_train],axis=1),tf.float32)
X_test=tf.cast(tf.concat([x0_test,x_test],axis=1),tf.float32)
```

In [9]:
```
X_train.shape,X_test.shape
```

Out[9]: (TensorShape([404, 14]), TensorShape([102, 14]))

```
In [10]: Y_train=tf.constant(y_train.reshape(-1, 1),tf.float32)
         Y_test=tf.constant(y_test.reshape(-1, 1),tf.float32)

In [11]: Y_train.shape,Y_test.shape

Out[11]: (TensorShape([404, 1]), TensorShape([102, 1]))
```

```
In [10]: Y_train=tf.constant(y_train.reshape(-1, 1),tf.float32)
         Y_test=tf.constant(y_test.reshape(-1, 1),tf.float32)

In [11]: Y_train.shape,Y_test.shape

Out[11]: (TensorShape([404, 1]), TensorShape([102, 1]))
```

- 设置超参数

```
In [12]: learn_rate = 0.01
         iter= 2000
         display_step =200
```

- 设置模型变量初始值

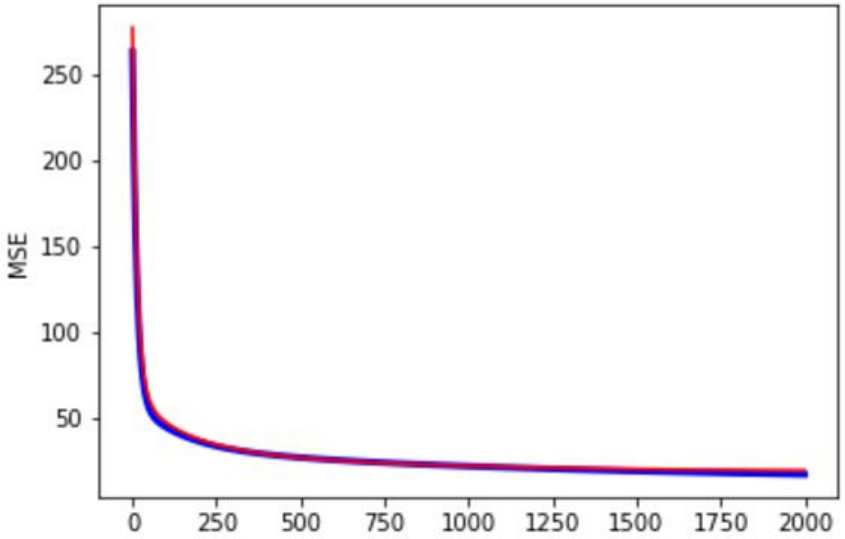
```
In [13]: np.random.seed(612)
         W = tf.Variable(np.random.randn(14,1),dtype=tf.float32)
```

训练模型

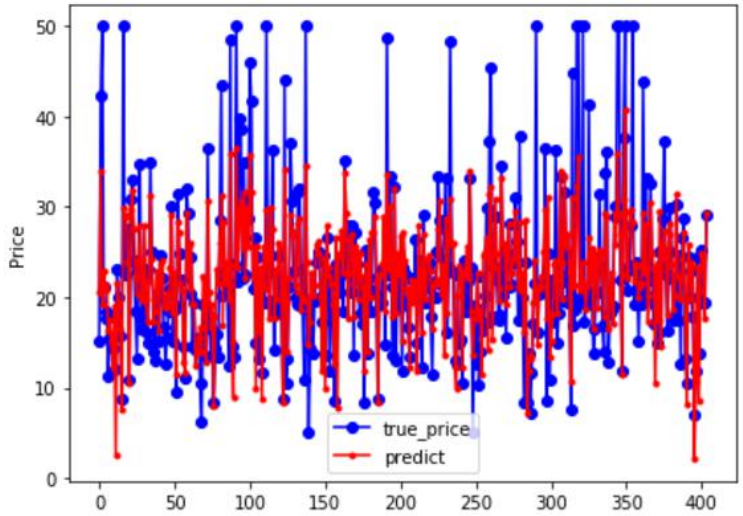
```
mse_train=[]
mse_test=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:

        PRED_train=tf.matmul(X_train,W)
        Loss_train=0.5* tf.reduce_mean(tf.square(Y_train-PRED_train))

        PRED_test=tf.matmul(X_test,W)
        Loss_test=0.5* tf.reduce_mean(tf.square(Y_test-PRED_test))

    mse_train.append(Loss_train)
    mse_test.append(Loss_test)

    dL_dW = tape.gradient(Loss_train,W)
    W.assign_sub(learn_rate*dL_dW)

    if i % display_step == 0:
        print("i: %i, Train Loss: %f, Test Loss: %f " % (i, Loss_train,Loss_test))
```

```
i: 0, Train Loss: 263.193451, Test Loss: 276.994110
i: 200, Train Loss: 36.176552, Test Loss: 37.562954
i: 400, Train Loss: 28.789461, Test Loss: 28.952513
i: 600, Train Loss: 25.520697, Test Loss: 25.333916
i: 800, Train Loss: 23.460522, Test Loss: 23.340532
i: 1000, Train Loss: 21.887278, Test Loss: 22.039747
i: 1200, Train Loss: 20.596283, Test Loss: 21.124847
i: 1400, Train Loss: 19.510204, Test Loss: 20.467239
i: 1600, Train Loss: 18.587009, Test Loss: 19.997717
i: 1800, Train Loss: 17.797461, Test Loss: 19.671591
i: 2000, Train Loss: 17.118927, Test Loss: 19.456863
```

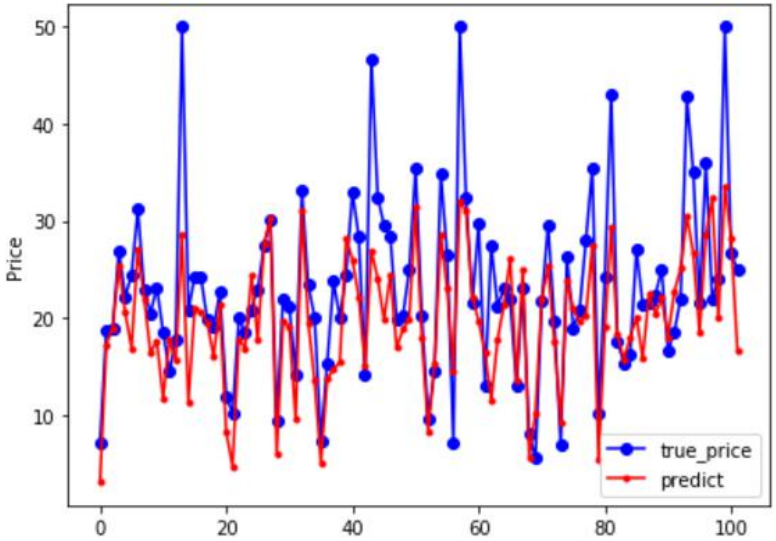
- 可视化输出




```
plt.plot(mse_train,color="blue",linewidth=3)
plt.plot(mse_test,color="red",linewidth=1.5)
```



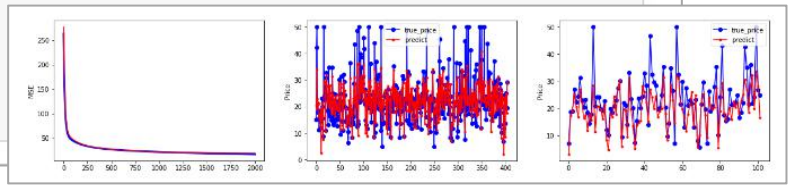

```
plt.plot(y_train,color="blue",marker="o",label="true_price")
plt.plot(PRED_train,color="red",marker=".",label="predict")
```




```
plt.plot(y_test,color="blue",marker="o",label="true_price")
plt.plot(PRED_test,color="red",marker=".",label="predict")
```

- 可视化输出

```
plt.figure(figsize=(20,4))

plt.subplot(131)
plt.ylabel("MSE")
plt.plot(mse_train,color="blue",linewidth=3)
plt.plot(mse_test,color="red",linewidth=1.5)

plt.subplot(132)
plt.plot(y_train,color="blue",marker="o",label="true_price")
plt.plot(PRED_train,color="red",marker=".",label="predict")
plt.legend()
plt.ylabel("Price")

plt.subplot(133)
plt.plot(y_test,color="blue",marker="o",label="true_price")
plt.plot(PRED_test,color="red",marker=".",label="predict")
plt.legend()
plt.ylabel("Price")

plt.show()
```



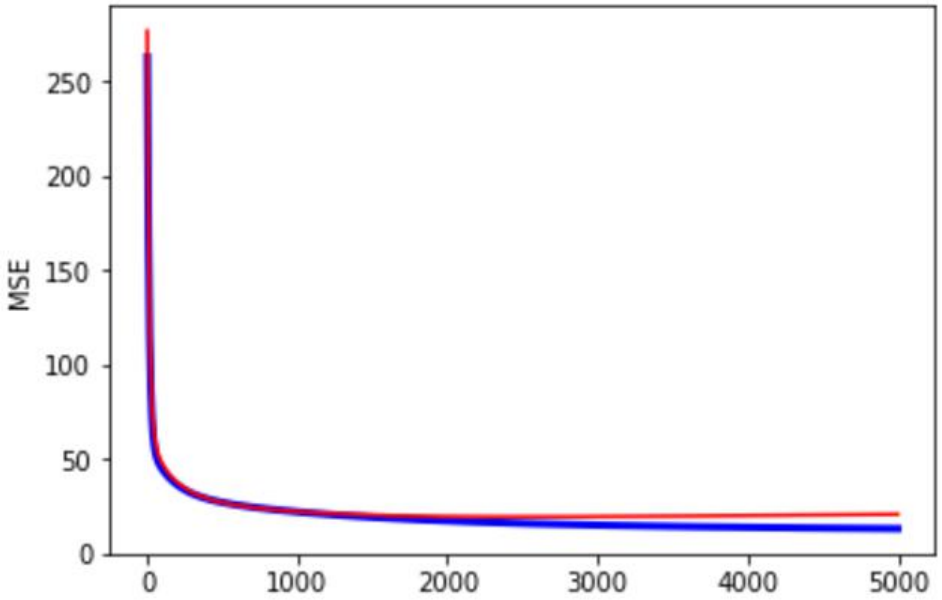
```
learn_rate = 0.01
iter= 8000
display_step =500
```

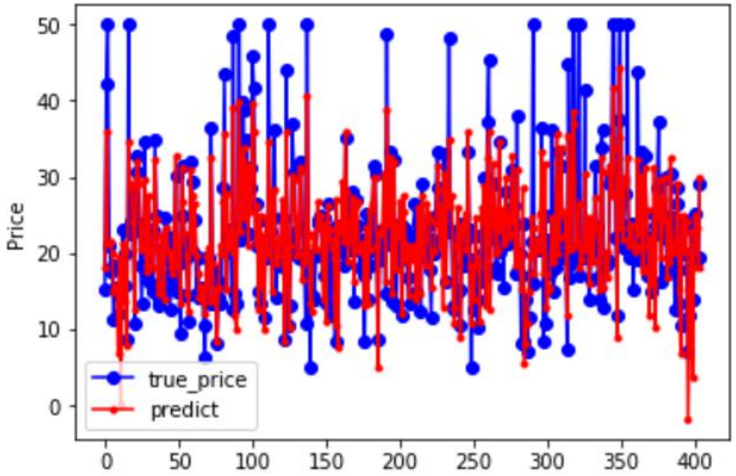
iter=**3000**
display=**100**

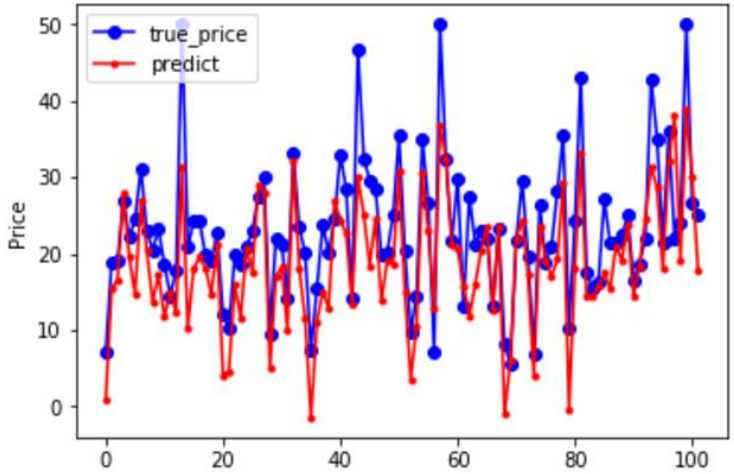
```
i: 0, Train Loss: 263.193451, Test Loss: 276.994110
i: 500, Train Loss: 26.911528, Test Loss: 26.827421
i: 1000, Train Loss: 21.887278, Test Loss: 22.039747
i: 1500, Train Loss: 19.030268, Test Loss: 20.212141
i: 2000, Train Loss: 17.118927, Test Loss: 19.456863
i: 2500, Train Loss: 15.797002, Test Loss: 19.260986
i: 3000, Train Loss: 14.858858, Test Loss: 19.365532
i: 3500, Train Loss: 14.177205, Test Loss: 19.623526
i: 4000, Train Loss: 13.671042, Test Loss: 19.949772
i: 4500, Train Loss: 13.287543, Test Loss: 20.295109
i: 5000, Train Loss: 12.991438, Test Loss: 20.631866
i: 5500, Train Loss: 12.758677, Test Loss: 20.945160
i: 6000, Train Loss: 12.572536, Test Loss: 21.227777
i: 6500, Train Loss: 12.421189, Test Loss: 21.477072
i: 7000, Train Loss: 12.296155, Test Loss: 21.693033
i: 7500, Train Loss: 12.191256, Test Loss: 21.877157
i: 8000, Train Loss: 12.101961, Test Loss: 22.031693
```

10.6.2 波士顿房价预测：多元线性回归

```
learn_rate = 0.01
iter= 8000
display_step =500
```




Price
true_price
predict
0
10
20
30
40
50
0
50
100
150
200
250
300
350
400


Price
true_price
predict
0
10
20
30
40
50
0
20
40
60
80
100