LASSO回归

在线性回归损失函数的基础上，增加了对权重的限制，作为正则化项

将**有限**的权重，放到**更重要的**特征维度上

$$L_{\text{linear}}=\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$$

线性回归

$$L_{\text{ridge}}=\sum_{i=1}^{N}(y_i-\hat{y}_i)^2+\lambda\|\mathbf{w}\|_2 \qquad \|\mathbf{w}\|_2=\sum_{i=1}^{D}w_i^2=\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

岭回归

$$L_{\text{lasso}}=\sum_{i=1}^{N}(y_i-\hat{y}_i)^2+\lambda\|\mathbf{w}\|_1 \qquad \|\mathbf{w}\|_1=\sum_{i=1}^{D}|w_i|$$

LASSO回归

正则化项的作用：
使每个权重都不会过大
若 某个权重过大，则
自变量x一旦发生微小的改变
就会导致输出发生巨大改变
增大了过拟合的危险

牺牲了精度
提高了稳定性

比较岭回归，**LASSO回归**
可以使学习得到的权重
更加**稀疏（集中），不重要的系数可以为0**

$$L_{\text{lasso}} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \|\mathbf{w}\|_1$$

不满足处处可导，因此

$$\frac{\partial L_{\text{lasso}}}{\partial \mathbf{w}} = 0$$

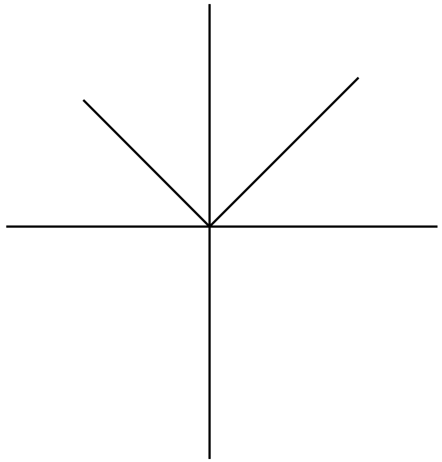
无法直接求导
没有闭式解

求解方法：坐标下降法

依次对w_i进行寻优，使梯度趋于0，
寻优的过程中只对w_i进行更改，其他的w_j不变

不断迭代，直到每个w_i都不产生显著变化为止

寻优方法**采用梯度下降搜索法**



$$\mathrm{L}_{\text{lasso}}=\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2+\lambda\|\mathbf{w}\|_1=\frac{1}{N}\sum_{i=1}^{N}\left(y_i-\sum_{j=1}^{D}x_{ij}w_j\right)^2+\lambda\sum_{j=1}^{D}|w_j|$$

$$=\frac{1}{N}\sum_{i=1}^{N}\left(y_i^2-2y_i\sum_{j=1}^{D}x_{ij}w_j+\left(\sum_{j=1}^{D}x_{ij}w_j\right)^2\right)+\lambda\sum_{j=1}^{D}|w_j|$$

梯度下降公式

$$w_j=w_j-lr*\frac{\partial \mathrm{L}_{\text{lasso}}}{\partial w_j}$$

$$=w_j-lr*\left(\frac{1}{N}\mathbf{x}_j^{T}*(\hat{\mathbf{y}}-\mathbf{y})+\lambda\,\mathrm{sign}(w_j)\right)$$

$$\frac{\partial \mathrm{L}_{\text{lasso}}}{\partial w_j}=\frac{1}{N}\sum_{i=1}^{N}\left(-2y_ix_{ij}+2x_{ij}\left(\sum_{j=1}^{D}x_{ij}w_j\right)\right)+\lambda\,\mathrm{sign}(w_j)$$

$$=\frac{1}{N}\sum_{i=1}^{N}\left(2x_{ij}\left(\sum_{j=1}^{D}x_{ij}w_j-y_i\right)\right)+\lambda\,\mathrm{sign}(w_j)=\frac{1}{N}\sum_{i=1}^{N}\left(2x_{ij}\left(\hat{y}_i-y_i\right)\right)+\lambda\,\mathrm{sign}(w_j)$$

$$=\frac{2}{N}\mathbf{x}_j^{T}*(\hat{\mathbf{y}}-\mathbf{y})+\lambda\,\mathrm{sign}(w_j)$$

其中：$\mathbf{x}_j$ 表示训练数据**X的第j列**

$$w_j = w_j - lr * \frac{\partial L_{\text{lasso}}}{\partial w_j}$$

$$= w_j - lr * \left(\frac{1}{N} \mathbf{x}_j^{\ T} * (\hat{\mathbf{y}} - \mathbf{y}) + \lambda \text{sign}(w_j) \right)$$

```python
# 坐标轴下降法
def CoordinateDescent(X,Y,epochs,lr,lam):
    N,D= X.shape
    XMat = np.mat(X)
    YMat = np.mat(Y)
    w = np.ones([D,1])

    # 进行 epoches 轮迭代
    for k in range(epochs):
        # 保存上一轮的w
        pre_w = copy.copy(w)
        # 逐维度进行参数寻优
        for i in range(D):
            # 在每个维度上找到最优的w_i
            for j in range(epochs):
                Y_hat = XMat*w
                g_i = XMat[:,i].T*(Y_hat-YMat)/N + lam *np.sign(w[i])
                # 进行梯度下降
                w[i] = w[i] - g_i*lr
                if np.abs(g_i)<1e-3:
                    break
        # 计算上一轮的w 和当前轮 w 的差值，如果每个维度的w都没有什么变化则退出
        diff_w  = np.array(list(map(lambda x:abs(x)<1e-3,pre_w-w)))
        if diff_w.all():
            break
    return w
```

权重初始化

注意与梯度下降法的区别

其他回归方法的比较

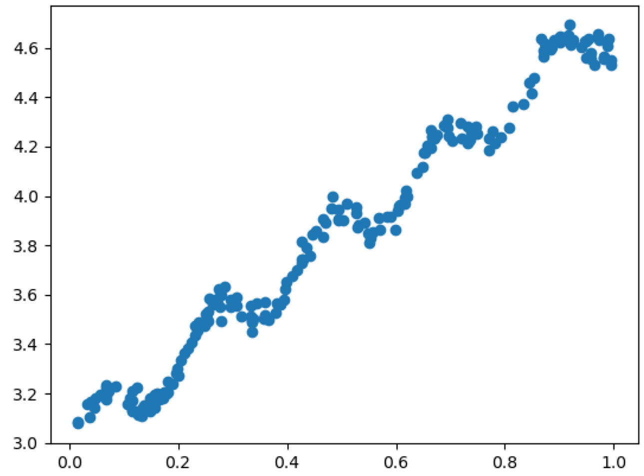
```python
import numpy as np
import copy
import matplotlib.pyplot as plt
from stand_regression import linear_Regress
from ridge_regression import ridge_regress

def load_DataSet(file_data,col_X=(0,1),col_Y=(2),add_bias=False):
    data_X = np.loadtxt(file_data,dtype=float,delimiter="\t",usecols=col_X)
    data_Y=  np.loadtxt(file_data,dtype=float,delimiter="\t",usecols=col_Y)

    # 如果需要偏置，则为data_X 增加一个数值是1的维度
    if add_bias:
        N,D = np.shape(data_X)
        add_dim = np.ones((N,1))
        data_X = np.concatenate((data_X,add_dim),axis=-1)

    # 对data_Y 进行维度修正 使其为 (N,1)
    if len(np.shape(data_Y))==1:
        data_Y = np.expand_dims(data_Y,axis=-1)


    return data_X,data_Y
```

绘图函数

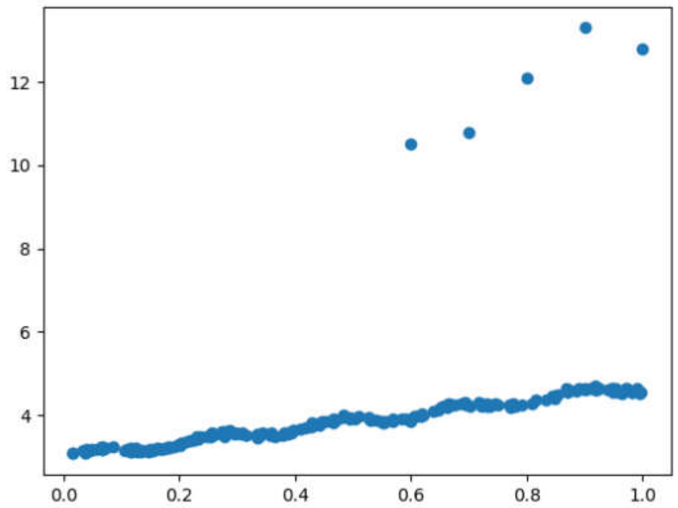
```python
def show_gress_with_W(X,Y,W=None,Colors=None):

    # 绘图
    fig = plt.figure() # 创建绘图对象
    ax = fig.add_subplot(1,1,1)
    ax.scatter(X[:,1],Y)

    # 回归预测
    if not W is None:
        for w,color in zip(W,Colors):
            Y_hat = np.dot(X,w)
            index=np.argsort(X[:,1])
            X_copy= X[index,:]
            Y_hat = Y_hat[index,:]

            ax.plot(X_copy[:,1],Y_hat,color=color)
    plt.show()
```

```
if __name__ == "__main__":
    # 数据加载
    X_train,Y_train = load_DataSet('ex0.txt')
    show_gress_with_W(X_train,Y_train)

    # 添加几个额外的数据

    x_add =np.zeros([5,2])
    x_add[:,1] = np.linspace(0.6,1,5)
    x_add[:,0] = 1

    y_add = np.zeros([5,1])
    y_add[:,0] = [10.5,10.8,12.1,13.3,12.8]

    X = np.concatenate([X_train,x_add],axis=0)
    Y = np.concatenate([Y_train,y_add],axis=0)
    show_gress_with_W(X,Y)
```

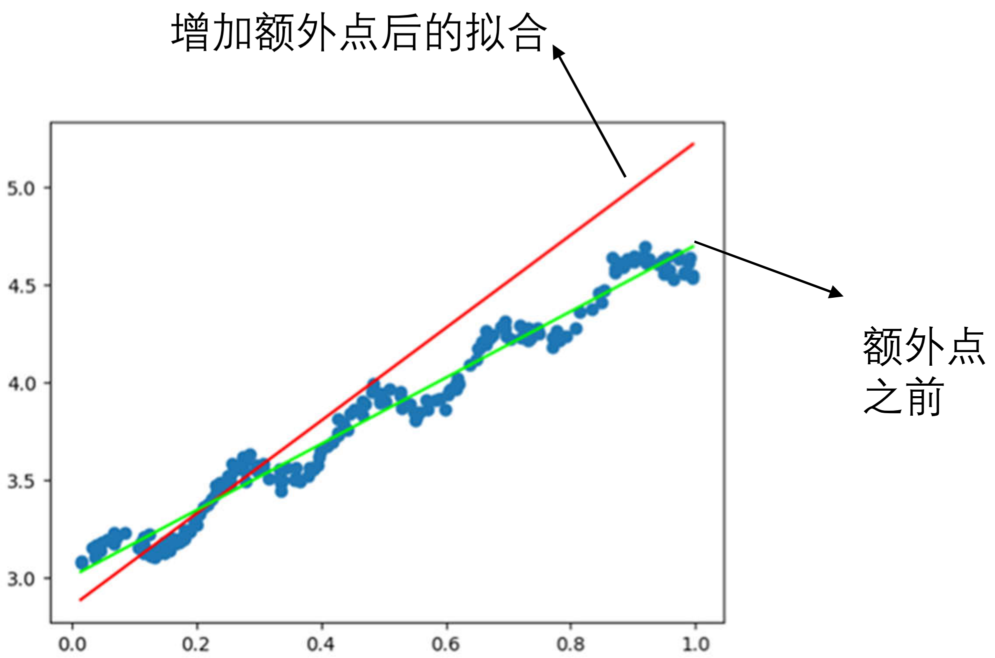
原始数据

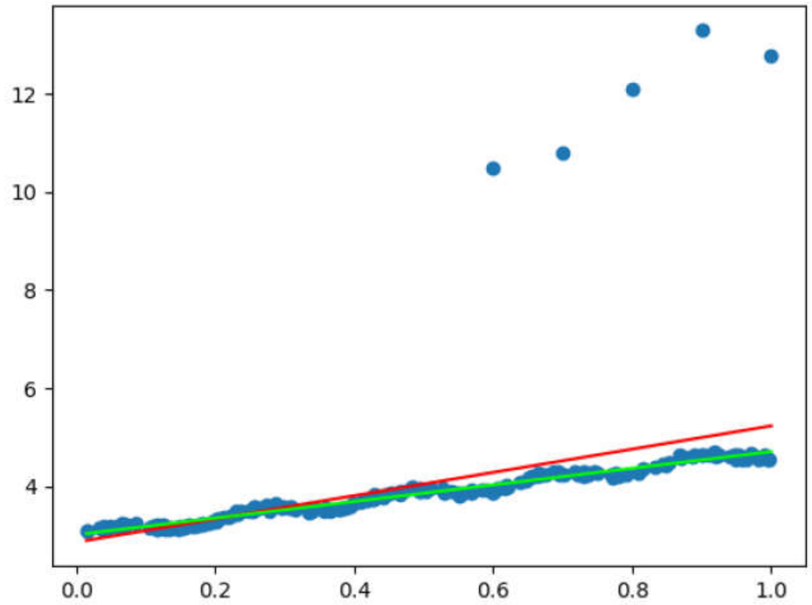


增加了额外的点

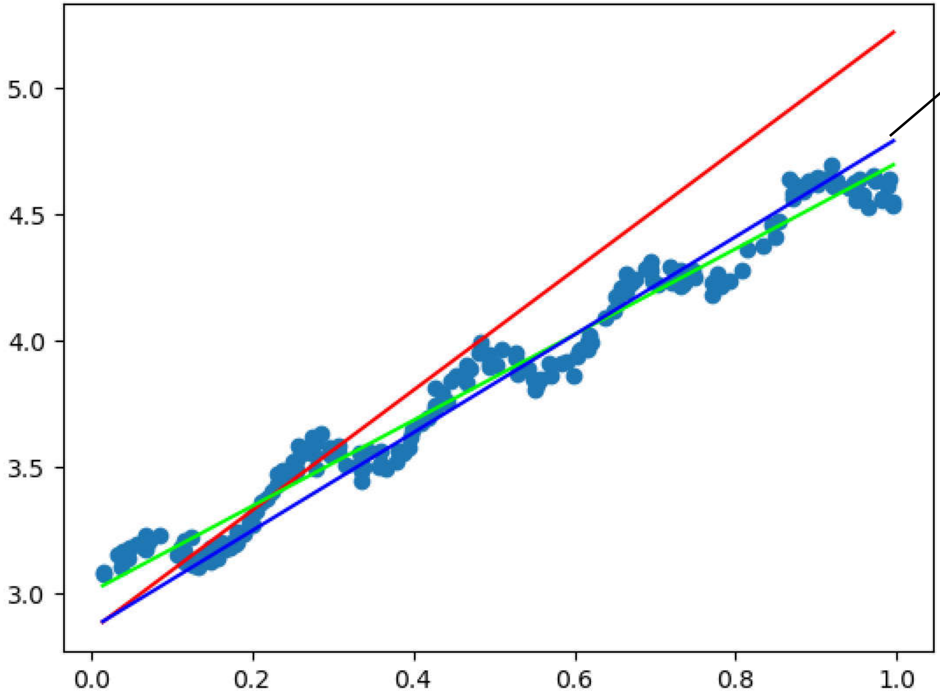


```
# 线性回归
W_linear = linear_Regress(X,Y)
W_linear_2 = linear_Regress(X_train,Y_train)
show_gress_with_W(X,Y,W=[W_linear,W_linear_2],Colors=[(1,0,0),(0,1,0)])
show_gress_with_W(X_train,Y_train,W=[W_linear,W_linear_2],Colors=[(1,0,0),(0,1,0)])
```

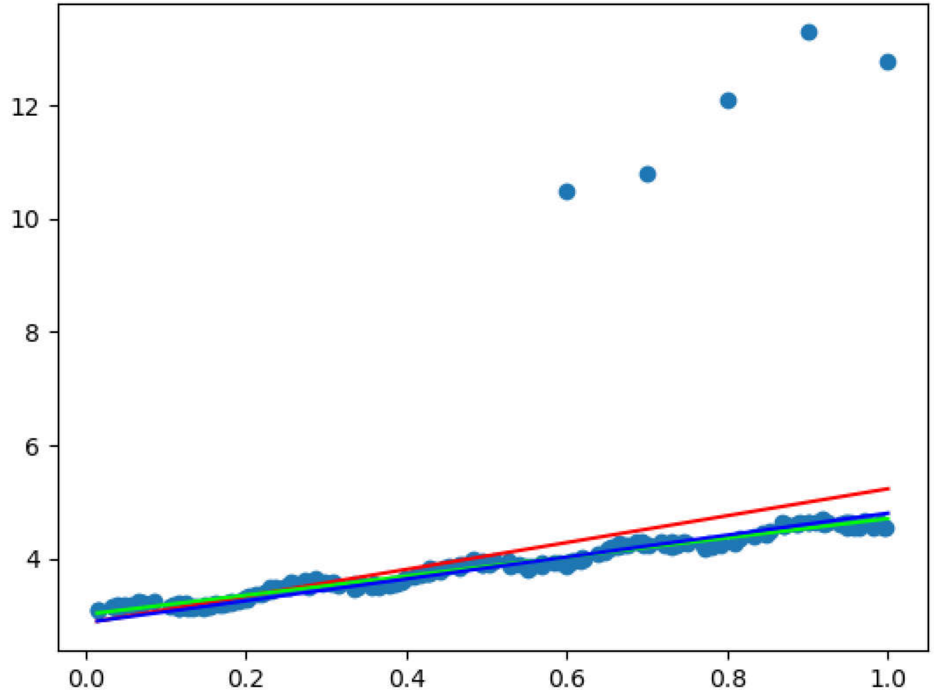




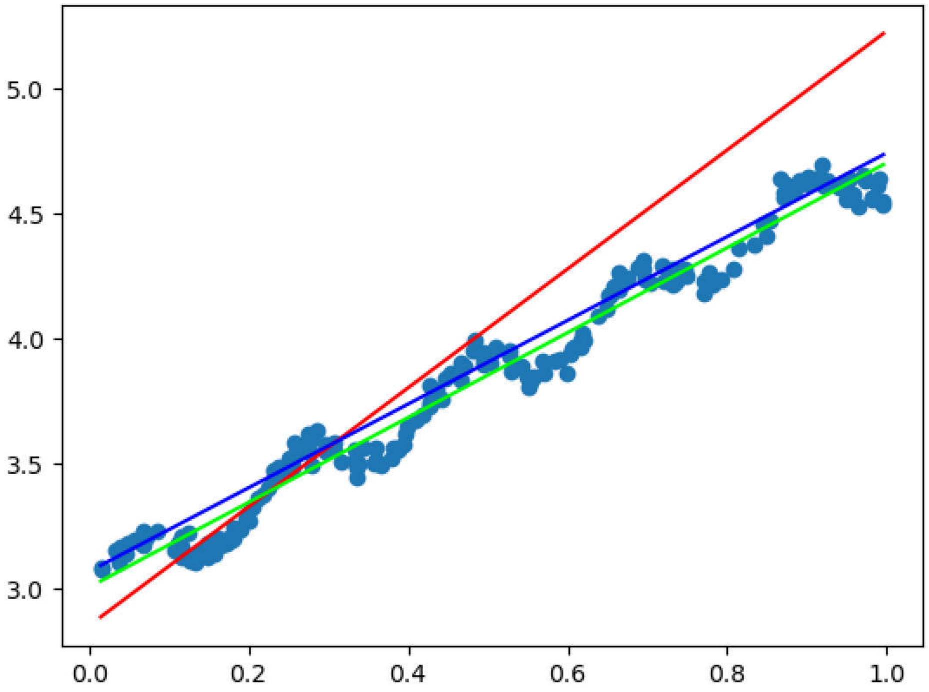
```
# 岭回归
W_ridge = ridge_regress(X,Y,lam= 15)
show_gress_with_W(X,Y,W=[W_linear,W_linear_2,W_ridge],Colors=[(1,0,0),(0,1,0),(0,0,1)])
show_gress_with_W(X_train,Y_train,W=[W_linear,W_linear_2,W_ridge],Colors=[(1,0,0),(0,1,0),(0,0,1)])
```

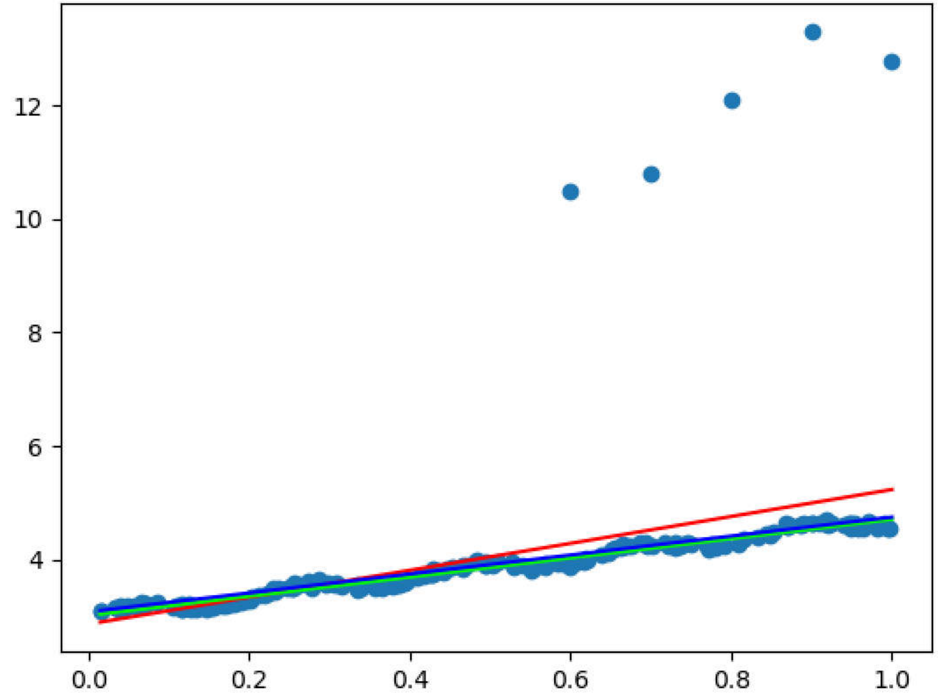


岭回归



```
# lasso 回归
W_lasso = CoordinateDescent(X,Y,lam= 0.13,lr = 0.001,epochs=250)
show_gress_with_W(X,Y,W=[W_linear,W_linear_2,W_lasso],Colors=[(1,0,0),(0,1,0),(0,0,1)])
show_gress_with_W(X_train,Y_train,W=[W_linear,W_linear_2,W_lasso],Colors=[(1,0,0),(0,1,0),(0,0,1)])
```





```
# 测试鲍鱼数据
 # 真实数据测试:
X,Y= load_DataSet('鲍鱼.txt',col_X=(0,1,2,3,4,5,6,7),col_Y=(8))
mean_X= np.mean(X,axis=0,keepdims=True)
std_X = np.std(X,axis=0,keepdims=True)
X = (X-mean_X)/std_X

mean_Y = np.mean(Y,axis=0,keepdims=True)
Y= Y-mean_Y
W_ridge = ridge_regress(X,Y,lam= 15)
print(W_ridge)
print(compute_error(X,Y,W_ridge))

W_lasso = CoordinateDescent(X,Y,lam= 0.1,lr = 0.001,epochs=250)
print(W_lasso)
print(compute_error(X,Y,W_lasso))
```

岭回归

```
[ 1.19264683]
 [ 0.50915326]
 [ 2.93751219]
 [-3.71652544]
 [-0.70517541]
 [ 1.70579472]]
4.932249213062436
```

Lasso回归

```
[[-7.77975064e-05]
 [ 1.22188905e-04]
 [ 5.59559838e-01]
 [ 4.54046963e-01]
 [-1.40430352e-05]
 [-1.41106428e+00]
 [-7.04837212e-05]
 [ 2.28874865e+00]]
```