

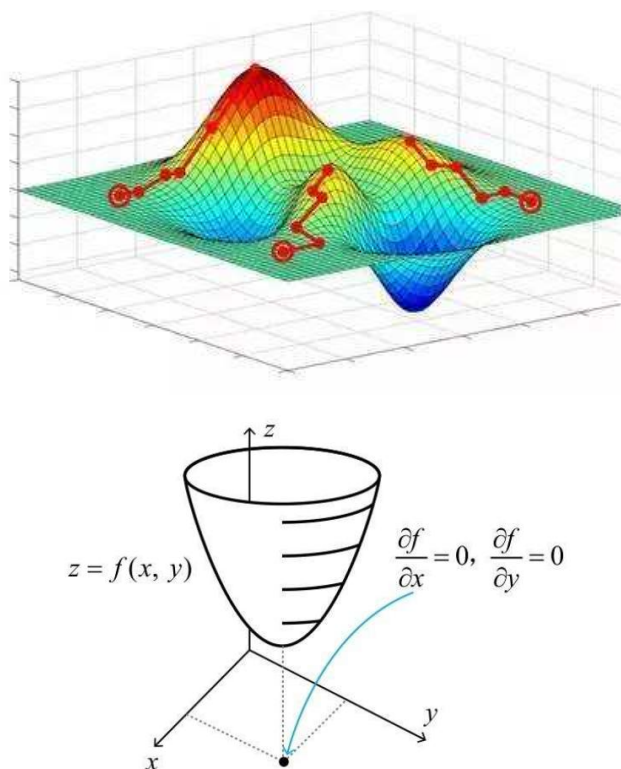
- 1、归一化 Normalization
 - 1.1、归一化目的
 - 1.2、归一化本质
 - 1.3、最大值最小值归一化
 - 1.4、Z-score标准化
- 2、正则化 Regularization
 - 2.1、过拟合欠拟合
 - 2.2、套索回归 (Lasso)
 - 2.3、岭回归 (Ridge)
- 3、线性回归衍生算法
 - 3.1、Lasso算法使用
 - 3.2、Ridge算法使用
 - 3.3、Elastic-Net算法使用
- 4、多项式回归
 - 4.1、多项式回归基本概念
 - 4.2、多项式回归实战1.0 (普通线性回归)
 - 4.3、多项式回归实战 (梯度下降)
- 5、代码实战天猫双十一销量预测
- 6、代码实战中国人寿保费预测
 - 6.1、数据加载与介绍
 - 6.2、EDA数据探索
 - 6.3、特征工程
 - 6.4、特征升维
 - 6.5、模型训练与评估

1、归一化 Normalization

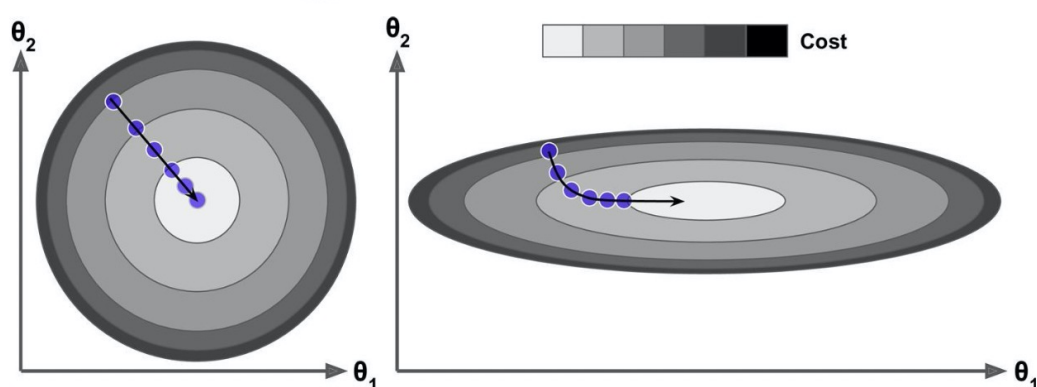
1.1、归一化目的

梯度下降的原理和应用，我们已经在前面课程中进行了学习。

不同方向的**陡峭度**是不一样的，即不同维度的数值大小是不同。也就是说梯度下降的快慢是不同的：



如果拿多元线性回归举例的话，因为多元线性回归的损失函数 MSE 是凸函数，所以我们可以把损失函数看成是一个碗。然后下面的图就是从碗上方去俯瞰！哪里是损失最小的地方呢？当然对应的就是碗底的地方！所以下图碗中心的地方颜色**较浅**的区域就是损失函数最小的地方。



上面两张图都是进行梯度下降，你有没有发现，略有不同啊？两张图形都是**鸟瞰图**，左边的图形做了归一化处理，右边是没有做归一化的俯瞰图。

啥是归一化呢？请带着疑问跟我走~

我们先来说一下为什么没做归一化是右侧图示，举个例子假如我们客户数据信息，有两个维度，一个是用户的年龄，一个是用户的月收入，目标变量是快乐程度。

| name | age | salary | happiness |
|-------|-------|--------|-----------|
| 代码简史 | 36 | 7000 | 100 |
| 智哥 | 38 | 200000 | 180 |
| 大王 | 23 | 300000 | 164 |
| | | | |

我们可以里面写出线性回归公式， $y = \theta_1 x_1 + \theta_2 x_2 + b$ ，那么这样每一条样本不同维度对应的数量级不同，原因是每个维度对应的物理含义不同嘛，但是计算机能理解 36 和 7000 分别是年龄和收入吗？计算机只是拿到一堆数字而已。

我们把 x_1 看成是年龄， x_2 看成是收入， y 对应着快乐程度。机器学习就是在知道 x, y 的情况下解方程组调整出最优解的过程。根据公式我们也可以发现 y 是两部分贡献之和，按常理来说，一开始并不知道两个部分谁更重要的情况下，可以想象为两部分对 y 的贡献是一样的即 $\theta_1 x_1 = \theta_2 x_2$ ，如果 $x_1 \ll x_2$ ，那么最终 $\theta_1 \gg \theta_2$ （远大于）。

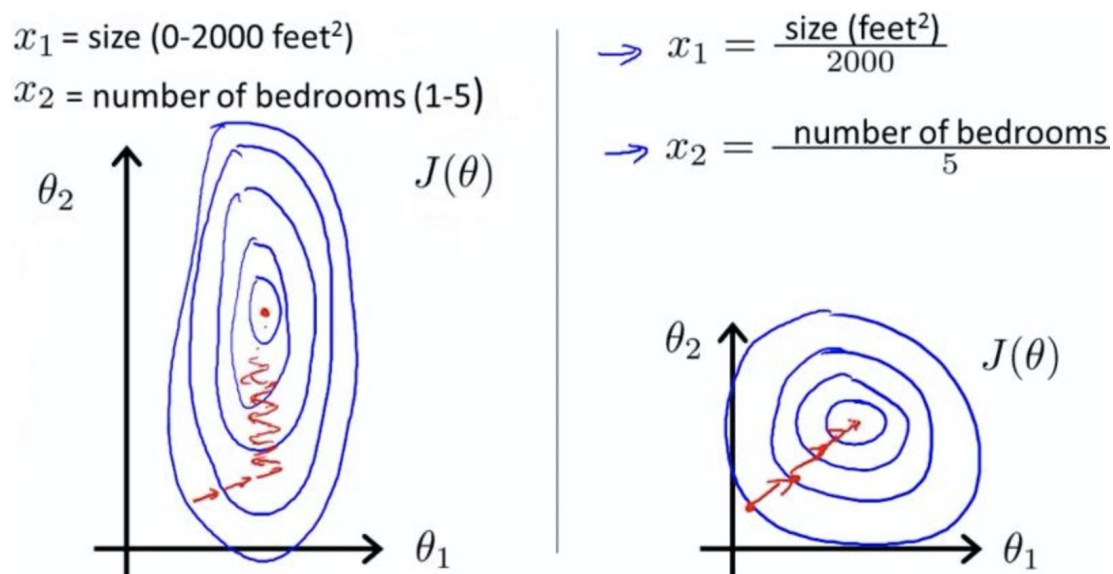
这样是不是就比较好理解为什么之前右侧示图里为什么 $\theta_1 > \theta_2$ ，看起来就是椭圆。再思考一下，梯度下降第 1 步的操作，是不是所有的维度 θ 都是根据在期望 μ 为 0 方差 σ 为 1 的正太分布随机生成的，说白了就是一开始的 θ_1 和 θ_2 数值是差不多的。所以可以发现 θ_1 从初始值到目标位置 θ_1^{target} 的距离要远大于 θ_2 从初始值到目标位置 θ_2^{target} 。

因为 $x_1 \ll x_2$ ，根据梯度公式 $g_j = (h_\theta(x) - y)x_j$ ，得出 $g_1 \ll g_2$ 。根据梯度下降公式： $\theta_j^{n+1} = \theta_j^n - \eta * g_j$ 可知，每次调整 θ_1 的幅度 \ll （远小于） θ_2 的调整幅度。

总结一下，根据上面得到的两个结论，它俩之间是互相矛盾的，意味着最后 θ_2 需要比 θ_1 更少的迭代次数就可以收敛，而我们要最终求得最优解，就必须每个维度 θ 都收敛才可以，所以会出现 θ_2 等待 θ_1 收敛的情况。讲到这里对应图大家应该可以理解为什么右图是先顺着 θ_2 的坐标轴往下走再往右走的原因了吧。

结论:

归一化的一个目的是，使得梯度下降在不同维度 θ 参数（不同数量级）上，可以步调一致协同的进行梯度下降。这就好比社会主义，一小部分人先富裕起来了，先富带后富，这需要一定的时间，先富的这批人等待其他的人富裕起来；但是，更好途经是实现共同富裕，最后每个人都不能落下，优化的步伐是一致的。



经过归一化处理，收敛的速度，明显快了！

1.2、归一化本质

做归一化的目的是要实现“**共同富裕**”，而之所以梯度下降优化时不能达到步调一致的根本原因其实还是 x_1 和 x_2 的数量级不同。所以什么是归一化？

答案自然就出来了，就是把 x_1 和 x_2 的数量级统一，扩展一点说，如果有更多特征维度，就要把各个特征维度 x_1 、 x_2 、.....、 x_n 的数量级统一，来做到无量纲化。

1.3、最大值最小值归一化

也称为离差标准化，是对原始数据的线性变换，使结果值映射到[0 - 1]之间。转换函数如下：

$$X^* = \frac{X - X_{min}}{X_{max} - X_{min}}$$

其实我们很容易发现使用最大值最小值归一化（min-max标准化）的时候，优点是一定可以把数值归一到 0 ~ 1 之间，缺点是如果有一个离群值（比如马云的财富），正如我们举的例子一样，会使得一个数值为 1，其它数值都几乎为 0，所以受离群值的影响比较大！

代码演示：

```
1 import numpy as np
2 x_1 = np.random.randint(1,10,size = 10)
3 x_2 = np.random.randint(100,300,size = 10)
4 x = np.c_[x_1,x_2]
5 print('归一化之前的数据：')
6 display(x)
7 x_ = (x - x.min(axis = 0)) / (x.max(axis = 0) -
   x.min(axis = 0))
8 print('归一化之后的数据：')
9 display(x_)
```

使用scikit-learn函数:

```
1 import numpy as np
2 from sklearn.preprocessing import MinMaxScaler
3 x_1 = np.random.randint(1,10,size = 10)
4 x_2 = np.random.randint(100,300,size = 10)
5 x = np.c_[x_1,x_2]
6 print('归一化之前的数据: ')
7 display(x)
8 min_max_scaler = MinMaxScaler()
9 x_ = min_max_scaler.fit_transform(x)
10 print('归一化之后的数据: ')
11 display(x_)
```

1.4、Z-score标准化

这种方法给予原始数据的均值（mean）和标准差（standard deviation）进行数据的标准化，叫做Z-score标准化。经过处理的数据符合标准正态分布，即均值为0，标准差为1，转化函数为：

$$X^* = \frac{X - \mu}{\sigma}$$

其中 μ 为所有样本数据的均值， σ 为所有样本数据的标准差。

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

相对于最大值最小值归一化来说，因为标准归一化除以了标准差，而标准差的计算会考虑到所有样本数据，所以受到离群值的影响会小一些，这就是除以方差的好处！但是，0-均值标准化不一定会把数据缩放到 0 ~ 1 之间了。既然是0均值，也就意味着，有正有负！

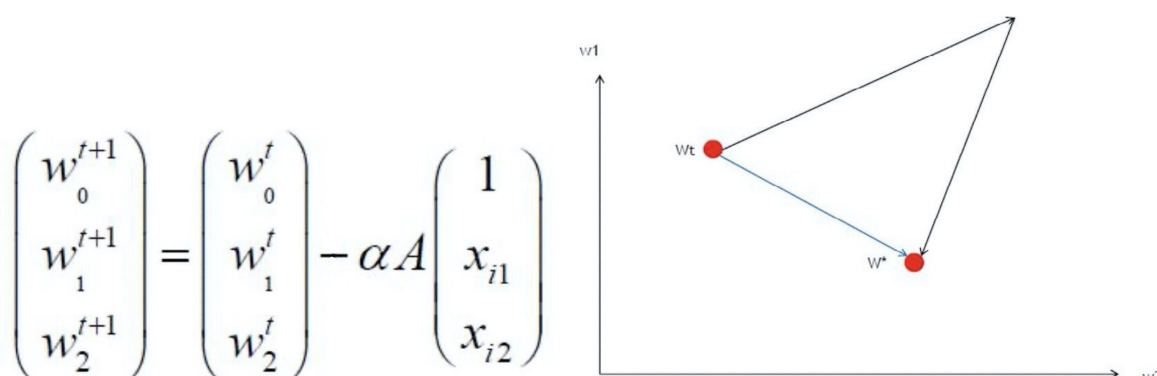
```
1 import numpy as np
2 x_1 = np.random.randint(1,10,size = 10)
3 x_2 = np.random.randint(100,300,size = 10)
4 x = np.c_[x_1,x_2]
5 print('归一化之前的数据：')
6 display(x)
7 x_ = (x - x.mean(axis = 0)) / x.std(axis = 0)
8 print('归一化之后的数据：')
9 display(x_)
```

使用scikit-learn函数：

```
1 import numpy as np
2 from sklearn.preprocessing import StandardScaler
3 x_1 = np.random.randint(1,10,size = 10)
4 x_2 = np.random.randint(100,300,size = 10)
5 x = np.c_[x_1,x_2]
6 print('归一化之前的数据：')
7 display(x)
8 standard_scaler = StandardScaler()
9 x_ = standard_scaler.fit_transform(x)
10 print('归一化之后的数据：')
11 display(x_)
```

那为什么要减去均值呢？其实做均值归一化还有一个特殊的好处（对比最大值最小值归一化，全部是正数0~1），我们来看一下梯度下降的式子，你就会发现 α 是**正数**，不管 A 也就是 梯度 g 是正还是负（A 就是 $\hat{y} - y = h_{\theta}(x) - y$ ），对于所有的维度 x ，比如这里的 x_1 和 x_2 来说， α 乘上 A 都是一样的符号，那么每次迭代的时候 w_1^{t+1} 和 w_2^{t+1} 的更新幅度**符号**也必然是一样的，这样就会像下图有右侧所示：要想从 w_t

更新到 w^* 就必然要么 w_1 和 w_2 同时变大再同时变小，或者就 w_1 和 w_2 同时变小再同时变大。不能如图上所示蓝色的最优解路径，即 w_1 变小的同时 w_2 变大！



那我们如何才能做到让 w_1 变小的时候 w_2 变大呢？归其根本还是数据集 X 矩阵（经过min-max归一化）中的数据均为正数。所以如果我们可以让 x_1 和 x_2 它们符号不同，比如有正有负，其实就可以在做梯度下降的时候有更多的可能性去让更新**尽可能**沿着最优解路径去走。

结论：**0-均值标准化**处理数据之后，属性有正有负，可以让梯度下降沿着最优路径进行~

注意：

我们在做**特征工程**的时候，很多时候如果对训练集的数据进行了预处理，比如这里讲的归一化，那么未来对测试集的时候，和模型上线来新的数据的时候，都要进行**相同**的数据预处理流程，而且所使用的均值和方差是来自当时训练集的均值和方差！

因为我们**人工智能**要干的事情就是从**训练集**数据中找**规律**，然后利用找到的规律去预测新产生的数据。这也就是说假设训练集和测试集以及未来新来的数据是属于**同分布**的！从代码上面来说如何去使用训练集的均值和方差呢？就需要把 scaler 对象持久化，回头模型上线的时候再加载进来去对新来的数据进行处理。

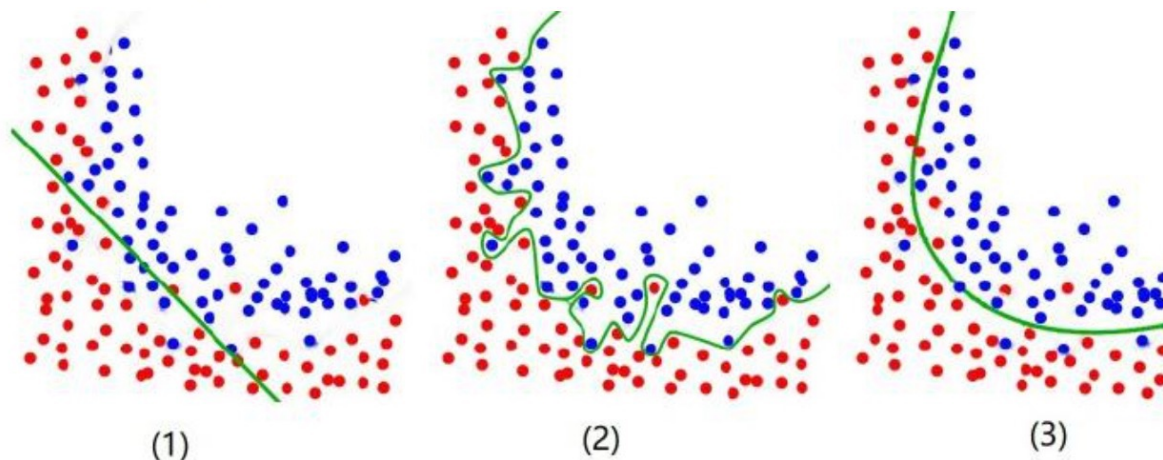

```
1 # 训练数据
2 import numpy as np
3 from sklearn.preprocessing import StandardScaler
4 x_1 = np.random.randint(1,10,size = 10)
5 x_2 = np.random.randint(100,300,size = 10)
6 x = np.c_[x_1,x_2]
7 print('归一化之前的数据: ')
8 display(x)
9 standard_scaler = StandardScaler() # 模型, 归一化
10 x_ = standard_scaler.fit_transform(x)
11 print('归一化之后的数据: ')
12 display(x_)
```

2、正则化 Regularization

2.1、过拟合欠拟合

1. 欠拟合 (under fit) : 还没有拟合到位, 训练集和测试集的准确率都还没有到达最高, 学的还不到位。
2. 过拟合 (over fit) : 拟合过度, 训练集的准确率升高的同时, 测试集的准确率反而降低。学的过度了 (走火入魔), 做过的卷子都能再次答对 (死记硬背), 考试碰到新的没见过的题就考不好 (不会举一反三)。

3. 恰到好处 (just right) : 过拟合前, 训练集和测试集准确率都达到巅峰。好比, 学习并不需要花费很多时间, 理解的很好, 考试的时候可以很好的把知识举一反三。



正则化就是防止过拟合, 增加模型的**鲁棒性**, 鲁棒是 Robust 的音译, 也就是强壮的意思。就像计算机软件在面临攻击、网络过载等情况下能够不死机不崩溃, 这就是软件的鲁棒性。鲁棒性调优就是让模型拥有更好的鲁棒性, 也就是让模型的泛化能力和推广能力更加的强大。

举例子说明: 下面两个式子描述同一条直线哪个更好?

$$y = 0.3x_1 + 0.4x_2 + 0.5$$

$$y = 3x_1 + 4x_2 + 5$$

第一个更好, 因为下面的公式是上面的十倍, 当 w 越小公式的容错的能力就越好。因为把测试数据带入公式中如果测试集原来是 $[32, 128]$ 在带入的时候发生了一些偏差, 比如说变成 $[30, 120]$, 第二个模型结果就会比第一个模型结果的偏差大的多。公式中 $y = W^T x$, 当 x 有一点错误, 这个错误会通过 w 放大。但是 w 不能太小, 当 w 太小时 (比如都趋近 0), 模型就没有意义了, 无法应用。想要有一定的容错率又要保证正确率就要由正则项来发挥作用了!

所以正则化(鲁棒性调优)的本质就是牺牲模型在训练集上的正确率来提高推广、泛化能力, W 在数值上越小越好, 这样能抵抗数值的**扰动**。同时为了保证模型的正确率 W 又不能极小。故而人们将原来的损失函数加上一个**惩罚项**, 这里面损失函数就是原来固有的损失函数, 比如回

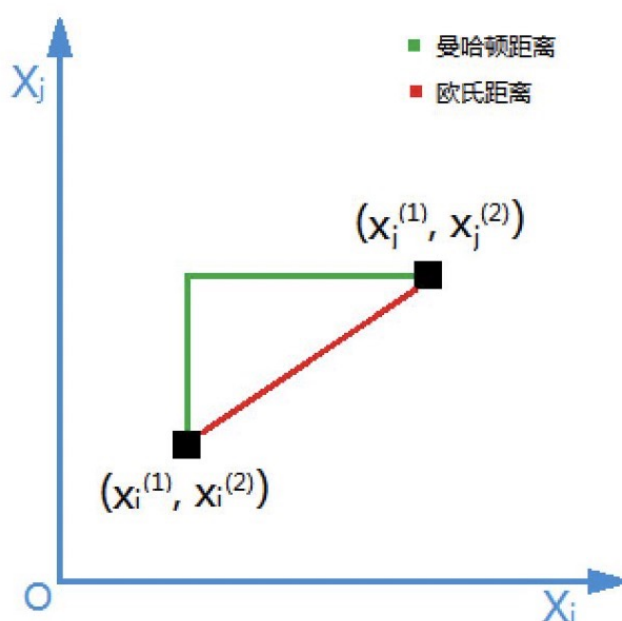
归的话通常是 MSE，分类的话通常是 cross entropy 交叉熵，然后在加上一部分惩罚项来使得计算出来的模型 W 相对小一些来带来泛化能力。

常用的惩罚项有 L1 正则项或者 L2 正则项：

- $L_1 = \|w\|_1 = \sum_{i=1}^n |w_i|$ 对应曼哈顿距离
- $L_2 = \|w\|_2 = \sqrt{\sum_{i=1}^n (w_i)^2}$ 对应欧氏距离

其实 L1 和 L2 正则的公式数学里面的意义就是范数，代表空间中向量到原点的距离：

$$L_p = \|X\|_p = \sqrt[p]{\sum_{i=1}^n x_i^p}, X = (x_1, x_2, \dots, x_n)$$



当我们把多元线性回归损失函数加上 L2 正则的时候，就诞生了 Ridge 岭回归。当我们把多元线性回归损失函数加上 L1 正则的时候，就孕育出来了 Lasso 回归。其实 L1 和 L2 正则项惩罚项可以加到任何算法的损失函数上面去提高计算出来模型的泛化能力的。

2.2、套索回归 (Lasso)

先从线性回归开始，其损失函数如下：

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

L1正则化的损失函数，令 $J_0 = J(\theta)$ ：

$$J = J_0 + \alpha * \sum_{i=1}^n |w_i|$$

$$\text{令 } L_1 = \alpha * \sum_{i=1}^n |w_i| :$$

$$J = J_0 + L_1$$

Lasso 回归的损失函数可以表示为：

$$J = J_0 + L_1 = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \alpha \sum_{i=1}^n |w_i|$$

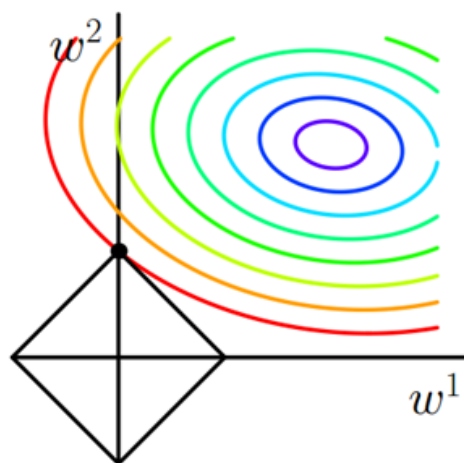
上面公式意味着：原始损失函数 J_0 后添加 L1 正则项相当于对 J_0 做了一个**约束**！！！！

假设我们有一个篮球队，我们想要选出五名最佳球员组成首发阵容。我们可以根据他们的**得分、篮板、助攻**等统计数据来评估他们的表现。但是，我们还需要考虑球员的**防守能力**，以确保球队在比赛中的整体表现。因此，我们在评估球员的得分、篮板和助攻等统计数据（即**原始损失函数 J_0** ）的同时，也需要考虑他们的防守能力（即 **L1 正则项**）。

在这个例子中，我们可以将选择五名首发球员的过程看作是一个**优化问题**。我们希望找到一组球员，使得他们的得分、篮板、助攻等统计数据**总和最高**，同时还要满足一定的**防守能力**要求。这就是在原始损失函数（即球员的进攻数据）后添加 **L1 正则项**（即球员的防守数据）的目的。这样，我们在求解这个优化问题时，就需要在保证**进攻数据最大化**的同时，满足**防守能力**的要求。因此，我们可以说，在原始损失函数 J_0 后添加 L1 正则项相当于对原始损失函数做了一个**约束**。

在 Lasso 回归中，原始损失函数 J_0 是衡量模型**预测能力**的指标，而 L1 正则项是衡量模型**复杂度**的指标。通过在原始损失函数后添加 L1 正则项，我们在求解优化问题时，需要在保证预测能力的同时，满足模型**复杂度**的要求。这就是为什么在原始损失函数后添加 L1 正则项相当于对原始损失函数做了一个约束， α 是正则化系数。

注意到 L1 正则化是权值的**绝对值**之和。 J 是带有绝对值符号的函数，因此 J 是不完全**可微**（求导）的。机器学习的任务就是要通过一些方法（比如梯度下降）求出损失函数的最小值。**考虑二维的情况**，即只有两个权值 w_1 、 w_2 ，此时 $L_1 = |w_1| + |w_2|$ 。对于梯度下降法，求解 J_0 过程可以画出等值线，同时 L1 正则化的函数 L_1 也可以在 w_1 、 w_2 所在的平面上画出来：



图中等值线是 J_0 的等值线，是椭圆形。黑色方框是 L_1 函数的图形， $L_1 = |w_1| + |w_2|$ 这个函数画出来，就是一个方框。

在图中，当 J_0 等值线与 L_1 图形首次相交的地方就是**最优解**。上图中 J_0 与 L_1 在 L_1 的一个顶点处相交，这个顶点就是最优解。注意到这个顶点的值是 $(w_1, w_2) = (0, w)$ 。可以直观想象，因为 L_1 函数有很多『突出的角』（二维情况下四个，多维情况下更多）， J_0 与这些角接触的机率会远大于与 L_1 其它部位接触的机率（这是很直觉的想象，突出的角比直线的边离等值线更近写），而在这些角上，会有很多权值等于0（因为角就在坐标轴上），这就是为什么L1正则化可以产生稀疏模型（很多权重等于0了），进而可以用于**特征选择**。

而正则化前面的系数 α ，可以控制 L_1 图形的大小。 α 越小， L_1 的图形越大（上图中的黑色方框）； α 越大， L_1 的图形就越小，可以小到黑色方框只超出原点范围一点点，这是最优解的值 $(w_1, w_2) = (0, w)$ 中的 w 可以取到很小的值的原因所在。

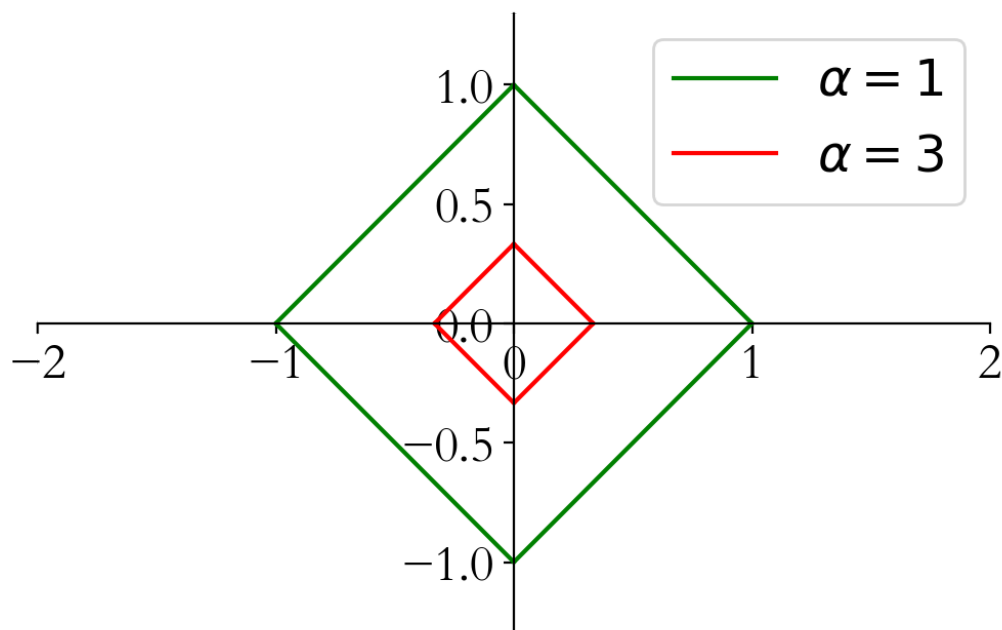
代码演示 α 取值大小对黑色方框的尺寸影响：

```
1 import matplotlib.pyplot as plt
2
3 #  $\alpha$  的值是: 1
4 #  $1 = x + y$ 
5 #  $y = 1 - x$ 
6 f = lambda x : 1- x
7 x = np.linspace(0,1,100)
8 plt.axis('equal')
9 plt.plot(x, f(x), color = 'green', label = '$\\alpha = 1$')
10 plt.plot(x, -f(x), color = 'green')
11 plt.plot(-x, f(x), color = 'green')
12 plt.plot(-x, -f(x), color = 'green')
13
14 #  $\alpha$  的值是: 3
```

```

15 # 1 = 3 * x + 3 * y
16 # y = 1/3 - x
17 f2 = lambda x : 1/3 - x
18 x2 = np.linspace(0,1/3,100)
19 plt.plot(x2, f2(x2),color = 'red',label = '$\\alpha = 3$')
20 plt.plot(x2, -f2(x2),color = 'red')
21 plt.plot(-x2,f2(x2),color = 'red')
22 plt.plot(-x2,-f2(x2),color = 'red')
23 # 一些列设置
24 plt.xlim(-2,2)
25 plt.ylim(-2,2)
26 ax = plt.gca()
27 ax.spines['right'].set_color('None') # 将图片的右框隐藏
28 ax.spines['top'].set_color('None') # 将图片的上边框隐藏
29 ax.spines['bottom'].set_position(('data', 0)) # x轴出现在y轴的-1 位置
30 ax.spines['left'].set_position(('data', 0))
31 plt.legend()

```



权重更新规则如下：

1. 损失函数:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$L_1 = \alpha * \sum_{i=1}^n |w_i|$$

$$J = J_0 + L_1$$

2. 更新规则:

$$\theta_j^{n+1} = \theta_j^n - \eta * \frac{\partial}{\partial \theta_j} J$$

$$\theta_j^{n+1} = \theta_j^n - \eta * \frac{\partial}{\partial \theta_j} (J_0 + L_1)$$

$$\frac{\partial}{\partial \theta_j} J_0 = \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\frac{\partial}{\partial \theta_j} L_1 = \alpha * \text{sgn}(w_i)$$

其中 J_0 即是线性回归的损失函数, L_1 是添加的正则项。 $\text{sgn}(w_i)$ 表示符号函数、指示函数, 值为: 1 或 -1。

$$\text{sgn}(w_i) = \begin{cases} 1, & w_i > 0 \\ -1, & w_i < 0 \end{cases}$$

注意当 $w_i = 0$ 时不可导。

综上所述，L1正则化权重更新如下：

$$\theta_j^{n+1} = \theta_j^n - \eta \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} - \eta * \alpha * \text{sgn}(w_i)$$

- Lasso回归和线性回归相比，多了一项： $-\eta * \alpha * \text{sgn}(w_i)$
- η 大于零，表示梯度下降学习率
- α 大于零，表示L1正则化系数
- 当 w_i 为正时候 $\text{sgn}(w_i) = 1$ ，直接减去 $\eta * \alpha$ （大于0），所以正的 w_i 变小了，向0靠近
- 当 w_i 为负时候 $\text{sgn}(w_i) = -1$ ，相当于直接加上 $\eta * \alpha$ （大于0），所以负的 w_i 变大了，绝对值变小，向0靠近

有的书本上公式会这样写，其中 λ 表示L1正则化系数：

$$\theta_j^{n+1} = \theta_j^n - \eta \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} - \eta * \lambda * \text{sgn}(w_i)$$

2.3、岭回归 (Ridge)

也是先从线性回归开始，其损失函数如下：

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

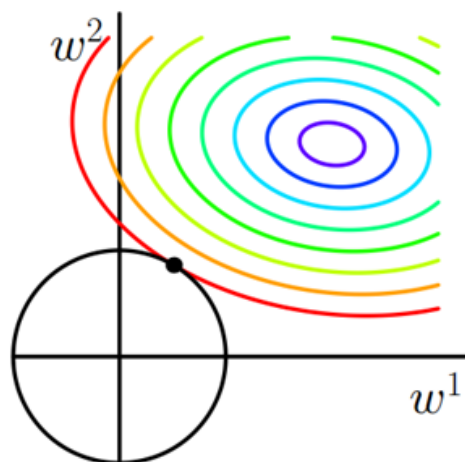
L2正则化的损失函数（对L2范数，进行了平方运算），令 $J_0 = J(\theta)$ ：

$$J = J_0 + \alpha * \sum_{i=1}^n (w_i)^2$$

$$\text{令 } L_2 = \alpha * \sum_{i=1}^n (w_i)^2 :$$

$$J = J_0 + L_2$$

同样可以画出他们在二维平面上的图形，如下：



二维平面下 L2 正则化的函数图形是个圆（绝对值的平方和，是个圆），与方形相比，被磨去了棱角。因此 J_0 与 L_2 相交时使得 w_1 、 w_2 等于零的机率小了许多（这个也是一个很直观的印象），这就是为什么 L2 正则化不具有稀疏性的原因，因为不太可能出现多数 w 都为 0 的情况（这种情况就叫稀疏性）！

权重更新规则如下：

1. 损失函数：

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$L_2 = \alpha * \sum_{i=1}^n (w_i)^2$$

$$J = J_0 + L_2$$

2. 更新规则：

$$\theta_j^{n+1} = \theta_j^n - \eta * \frac{\partial}{\partial \theta_j} J$$

$$\theta_j^{n+1} = \theta_j^n - \eta * \frac{\partial}{\partial \theta_j} (J_0 + L_2)$$

$$\frac{\partial}{\partial \theta_j} J_0 = \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\frac{\partial}{\partial \theta_j} L_2 = 2\alpha w_i$$

其中 J_0 即是线性回归的损失函数， L_2 是添加的正则项。

综上所述，L2正则化权重更新如下（ 2α 也是常数项，可以合并到一起用整体 α 替代）：

$$\theta_j^{n+1} = \theta_j^n (1 - \eta * \alpha) - \eta * \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

其中 α 就是正则化参数， η 表示学习率。从上式可以看到，与未添加L2正则化的迭代公式相比，每一次迭代， θ_j 都要先乘以一个小于1的因子（即 $(1 - \eta * \alpha)$ ），从而使得 θ_j 加速减小，因此总的来看， θ 相比不加L2正则项的线性回归可以获得更小的值。从而，实现了防止过拟合的效果，增加模型的鲁棒性~

有的书本上，公式写法可能**不同**：其中 λ 表示正则化参数。

$$\theta_j^{n+1} = \theta_j^n (1 - \eta * \lambda) - \eta * \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

3、线性回归衍生算法

接下来，我们一起学习一下scikit-learn中为我们提供的线性回归衍生算法，根据上面所学的原理，对比线性回归加深理解。

3.1、Lasso算法使用

这是scikit-learn官网给出的套索回归的，损失函数公式，注意，它用的矩阵表示，里面用到范数运算。

$$\min_w \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha ||w||_1$$

公式中多了一项： $\frac{1}{2n_{\text{samples}}}$ 这是一个常数项，去掉之后，也不会影响损失函数公式计算。在岭回归中，就没有这项。

L1正则化和普通线性回归系数对比：

```
1 import numpy as np
2 from sklearn.linear_model import Lasso
3 from sklearn.linear_model import SGDRegressor
4
5 # 1、创建数据集x, y
6 x = 2*np.random.rand(100, 20)
7 w = np.random.randn(20,1)
8 b = np.random.randint(1,10,size = 1)
9 y = x.dot(w) + b + np.random.randn(100, 1)
10
11 print('原始方程的斜率: ',w.ravel())
12 print('原始方程的截距: ',b)
```

```
13
14 lasso = Lasso(alpha= 0.5)
15 lasso.fit(X, y)
16 print('套索回归求解的斜率: ',lasso.coef_)
17 print('套索回归求解的截距: ',lasso.intercept_)
18
19 # 线性回归梯度下降方法
20 sgd = SGDRegressor(penalty='l2',alpha=0, l1_ratio=0)
21 sgd.fit(X, y.reshape(-1,))
22 print('随机梯度下降求解的斜率是: ',sgd.coef_)
23 print('随机梯度下降求解的截距是: ',sgd.intercept_)
```

结论:

- 和没有正则项约束线性回归对比, 可知L1正则化, 将方程系数进行了缩减, 部分系数为0, 产生稀疏模型
- α 越大, 模型稀疏性越强, 越多的参数为0
- Lasso回归源码解析:
 - alpha: 正则项系数
 - fit_intercept: 是否计算 w_0 截距项
 - normalize: 是否做归一化
 - precompute: bool 类型, 默认值为False, 决定是否提前计算Gram矩阵来加速计算
 - max_iter: 最大迭代次数
 - tol: 结果的精确度
 - warm_start: bool类型, 默认值为False。如果为True, 那么使用用前一次训练结果继续训练。否则从头开始训练

3.2、Ridge算法使用

这是scikit-learn官网给出的岭回归的，损失函数公式，注意，它用的矩阵表示，里面用到范数运算。

$$\min_w ||Xw - y||_2^2 + \alpha ||w||_2^2$$

L2正则化和普通线性回归系数对比：

```
1 import numpy as np
2 from sklearn.linear_model import Ridge
3 from sklearn.linear_model import SGDRegressor
4
5 # 1、创建数据集x, y
6 x = 2*np.random.rand(100, 5)
7 w = np.random.randint(1,10,size = (5,1))
8 b = np.random.randint(1,10,size = 1)
9 y = x.dot(w) + b + np.random.randn(100, 1)
10
11 print('原始方程的斜率: ',w.ravel())
12 print('原始方程的截距: ',b)
13
14 ridge = Ridge(alpha= 1, solver='sag')
15 ridge.fit(x, y)
16 print('岭回归求解的斜率: ',ridge.coef_)
17 print('岭回归求解的截距: ',ridge.intercept_)
18
19 # 线性回归梯度下降方法
20 sgd = SGDRegressor(penalty='l2',alpha=0,l1_ratio=0)
21 sgd.fit(x, y.reshape(-1,))
22 print('随机梯度下降求解的斜率是: ',sgd.coef_)
23 print('随机梯度下降求解的截距是: ',sgd.intercept_)
```

结论:

- 和没有正则项约束线性回归对比, 可知L2正则化, 将方程系数进行了缩小
- α 增大求解出来的方程斜率变小
- Ridge回归源码解析:
 - alpha: 正则项系数
 - fit_intercept: 是否计算 w_0 截距项
 - normalize: 是否做归一化
 - max_iter: 最大迭代次数
 - tol: 结果的精确度
 - solver: 优化算法的选择

3.3、Elastic-Net算法使用

这是scikit-learn官网给出的弹性网络回归的, 损失函数公式, 注意, 它用的矩阵表示, 里面用到范数运算。

$$\min_w \frac{1}{2n_{\text{samples}}} \|Xw - y\|_2^2 + \alpha \rho \|w\|_1 + \frac{\alpha(1-\rho)}{2} \|w\|_2^2$$

Elastic-Net 回归, 即岭回归和Lasso技术的混合。弹性网络是一种使用 L1, L2 范数作为先验正则项训练的线性回归模型。这种组合允许学习到一个只有少量参数是非零稀疏的模型, 就像 Lasso 一样, 但是它仍然保持一些像 Ridge 的正则性质。我们可利用 l1_ratio 参数控制 L1 和 L2 的凸组合。

弹性网络在很多特征互相联系（相关性，比如**身高**和**体重**就很有关系）的情况下是非常有用的。Lasso 很可能只随机考虑这些特征中的一个，而弹性网络更倾向于选择两个。

在实践中，Lasso 和 Ridge 之间权衡的一个优势是它允许在迭代过程中继承 Ridge 的稳定性。

弹性网络回归和普通线性回归系数对比：

```
1 import numpy as np
2 from sklearn.linear_model import ElasticNet
3 from sklearn.linear_model import SGDRegressor
4
5 # 1、创建数据集x, y
6 x = 2*np.random.rand(100, 20)
7 w = np.random.randn(20,1)
8 b = np.random.randint(1,10,size = 1)
9 y = x.dot(w) + b + np.random.randn(100, 1)
10
11 print('原始方程的斜率: ',w.ravel())
12 print('原始方程的截距: ',b)
13
14 model = ElasticNet(alpha= 1, l1_ratio = 0.7)
15 model.fit(x, y)
16 print('弹性网络回归求解的斜率: ',model.coef_)
17 print('弹性网络回归求解的截距: ',model.intercept_)
18
19 # 线性回归梯度下降方法
20 sgd = SGDRegressor(penalty='l2',alpha=0, l1_ratio=0)
21 sgd.fit(x, y.reshape(-1,))
22 print('随机梯度下降求解的斜率是: ',sgd.coef_)
23 print('随机梯度下降求解的截距是: ',sgd.intercept_)
```

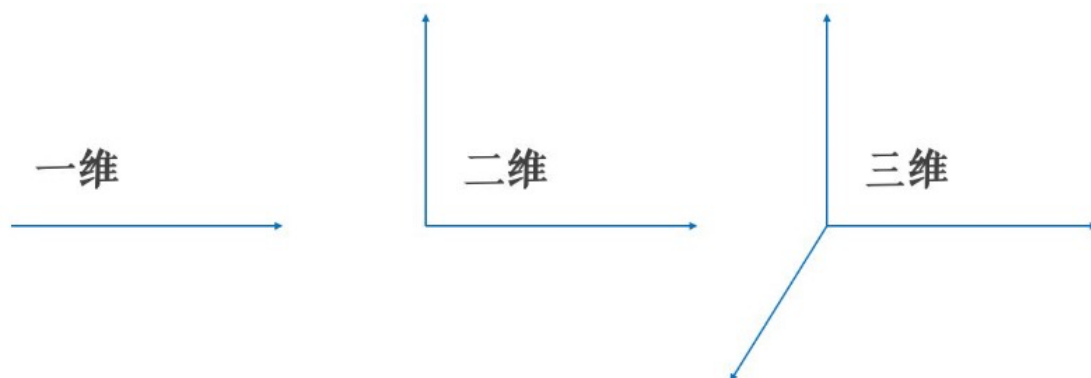
结论：

- 和没有正则项约束线性回归对比，可知Elastic-Net网络模型，融合了L1正则化L2正则化
- Elastic-Net 回归源码解析：
 - alpha: 混合惩罚项的常数
 - l1_ratio: 弹性网混合参数, $0 \leq l1_ratio \leq 1$, 对于 $l1_ratio = 0$, 惩罚项是L2正则惩罚。对于 $l1_ratio = 1$ 是L1正则惩罚。对于 0
 - fit_intercept: 是否计算 w_0 截距项
 - normalize: 是否做归一化
 - precompute: bool 类型，默认值为False，决定是否提前计算Gram矩阵来加速计算
 - max_iter: 最大迭代次数
 - tol: 结果的精确度
 - warm_start: bool类型，默认值为False。如果为True，那么使用用前一次训练结果继续训练。否则从头开始训练

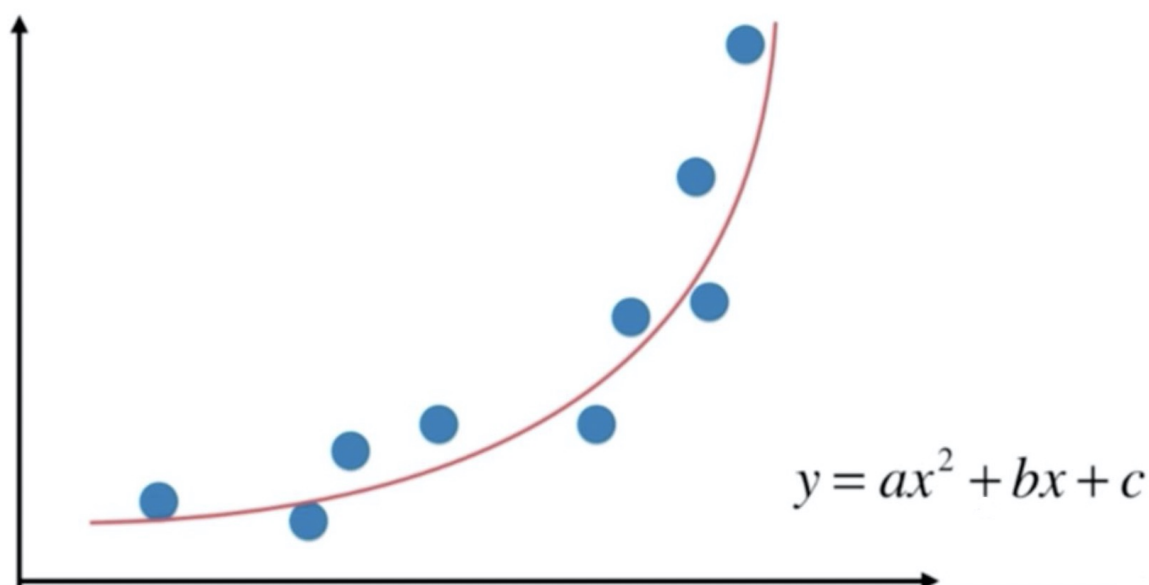
4、多项式回归

4.1、多项式回归基本概念

升维的目的是为了解决欠拟合的问题的，也就是为了提高模型的准确率为目的的，因为当维度不够时，说白了就是对于预测结果考虑的因素少的话，肯定不能准确的计算出模型。



在做升维的时候，最常见的手段就是将已知维度进行相乘（或者自乘）来构建新的维度，如下图所示。普通线性方程，无法拟合规律，必须是多项式，才可以完美拟合曲线规律，图中是二次多项式。



对于多项式回归来说主要是为了扩展线性回归算法来适应更广泛的数据集，比如我们数据集有两个维度 x_1 、 x_2 ，那么用多元线性回归公式就是： $\hat{y} = w_0 + w_1x_1 + w_2x_2$ ，当我们使用二阶多项式升维的时候，数据集就从原来的 x_1 、 x_2 扩展成了 x_1 、 x_2 、 x_1^2 、 x_2^2 、 x_1x_2 。因此多元线性回归就得去多计算三个维度所对应的w值：

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2。$$

此时拟合出来的方程就是曲线，可以解决一些线性回归的欠拟合问题！

4.2、多项式回归实战1.0（普通线性回归）

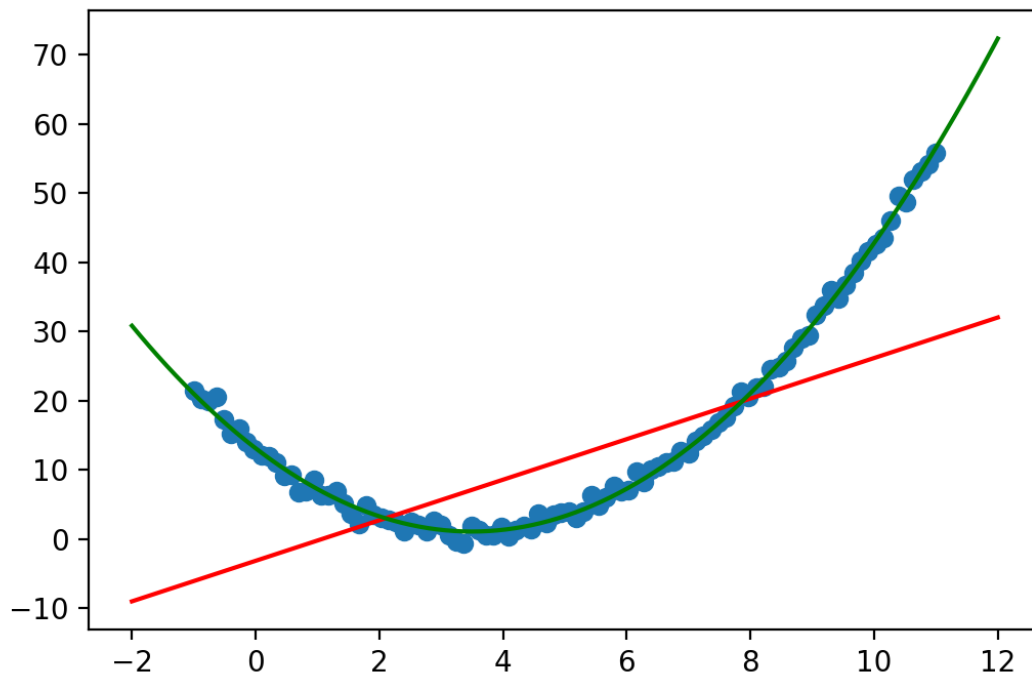
```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.linear_model import LinearRegression
4
5 # 1、创建数据，并进行可视化
6 X = np.linspace(-1,11,num = 100)
7 y = (X - 5)**2 + 3*X -12 + np.random.randn(100)
8 X = X.reshape(-1,1)
9 plt.scatter(X,y)
10
11 # 2、创建预测数据
12 X_test = np.linspace(-2,12,num = 200).reshape(-1,1)
13
14 # 3、不进行升维 + 普通线性回归
15 model_1 = LinearRegression()
16 model_1.fit(X,y)
17 y_test_1 = model_1.predict(X_test)
18 plt.plot(X_test,y_test,color = 'red')
19
20 # 4、多项式升维 + 普通线性回归
21 X = np.concatenate([X,X**2],axis = 1)
22 model_2 = LinearRegression()
23 model_2.fit(X,y)
24 # 5、测试数据处理，并预测
25 X_test = np.concatenate([X_test,X_test**2],axis = 1)
26 y_test_2 = model_2.predict(X_test)
27
28 # 6、数据可视化，切片操作
29 plt.plot(X_test[:,0],y_test_2,color = 'green')

```

结论:

- 不进行多项式升维，拟合出来的曲线，是线性的直线，和目标曲线无法匹配
- 使用np.concatenate()进行简单的，幂次合并，注意数据合并的方向 axis = 1
- 数据可视化时，注意切片，因为数据升维后，多了平方这一维



4.3、多项式回归实战（梯度下降）

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.preprocessing import
  PolynomialFeatures,StandardScaler
4 from sklearn.linear_model import SGDRegressor
5
6 # 1、创建数据，并进行可视化
7 x = np.linspace(-1,11,num = 100)
8 y = (x - 5)**2 + 3*x -12 + np.random.randn(100)
9 x = x.reshape(-1,1)
10 plt.scatter(x,y)
11
12 # 2、创建预测数据
13 x_test = np.linspace(-2,12,num = 200).reshape(-1,1)
14
15 # 3、使用PolynomialFeatures进行特征升维
16 poly = PolynomialFeatures()
17 poly.fit(x,y)
```



```

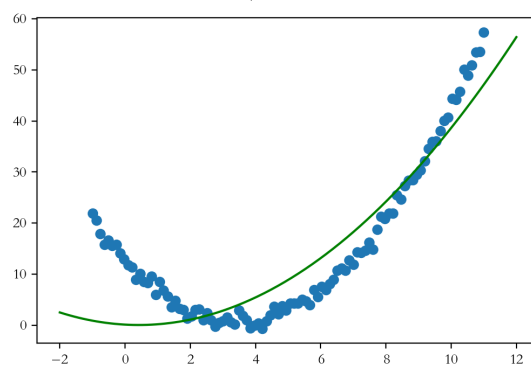
18 x = poly.transform(X)
19 s = StandardScaler()
20 x = s.fit_transform(X)
21 # model = SGDRegressor(penalty='l2',eta0 =
    0.0001,max_iter = 10000)
22 model = SGDRegressor(penalty='l2',eta0 = 0.01)
23 model.fit(X,y)
24
25 # 4、预测数据
26 X_test = poly.transform(X_test)
27 X_test_norm = s.transform(X_test)
28 y_test = model.predict(X_test_norm)
29 plt.plot(X_test[:,1],y_test,color = 'green')

```

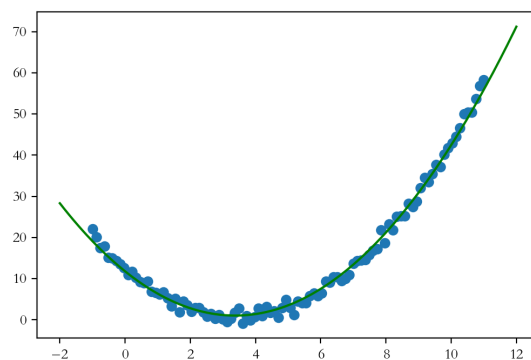
结论:

- eta0表示学习率，设置合适的学习率，才能拟合成功
- 多项式升维，需要对数据进行Z-score归一化处理，效果更佳出色
- SGD随机梯度下降需要调整参数，以使模型适应数据

未处理

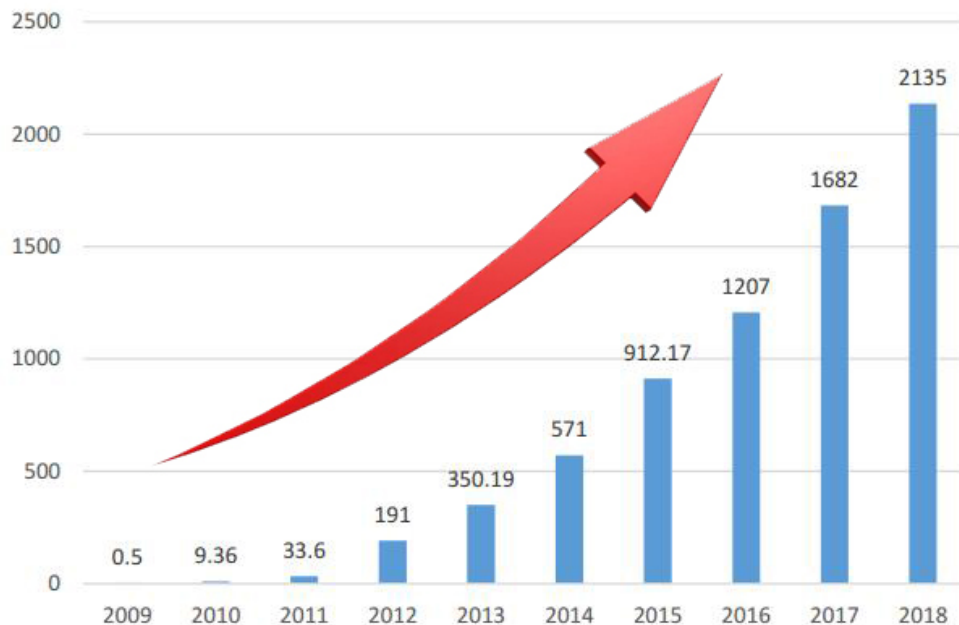


归一化



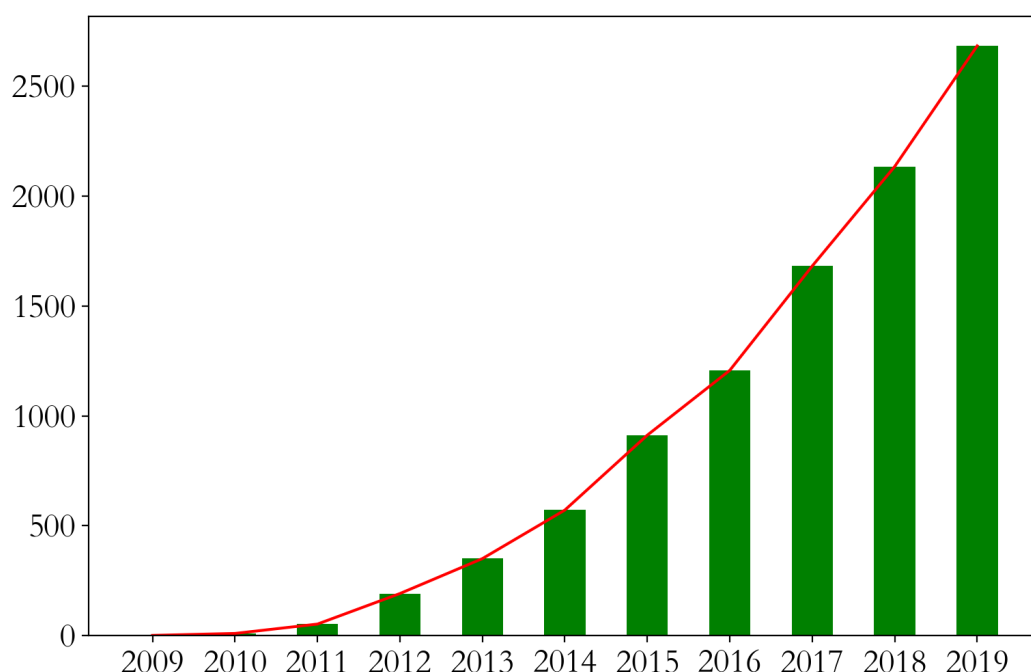
5、代码实战天猫双十一销量预测

天猫双十一，从2009年开始举办，第一届成交额仅仅0.5亿，后面呈现了爆发式的增长，那么这些增长是否有规律呢？是怎么样的规律，该如何分析呢？我们使用多项式回归一探究竟！



数据可视化，历年天猫双十一销量数据：

```
1 import numpy as np
2 from sklearn.linear_model import SGDRegressor
3 import matplotlib.pyplot as plt
4 plt.rcParams['font.size'] = 18
5 plt.figure(figsize=(9,6))
6
7 # 创建数据，年份数据2009 ~ 2019
8 x = np.arange(2009,2020)
9 y =
    np.array([0.5,9.36,52,191,350,571,912,1207,1682,2135,
    2684])
10 plt.bar(x,y,width = 0.5,color = 'green')
11 plt.plot(x,y,color = 'red')
12 _ = plt.xticks(ticks = x)
```

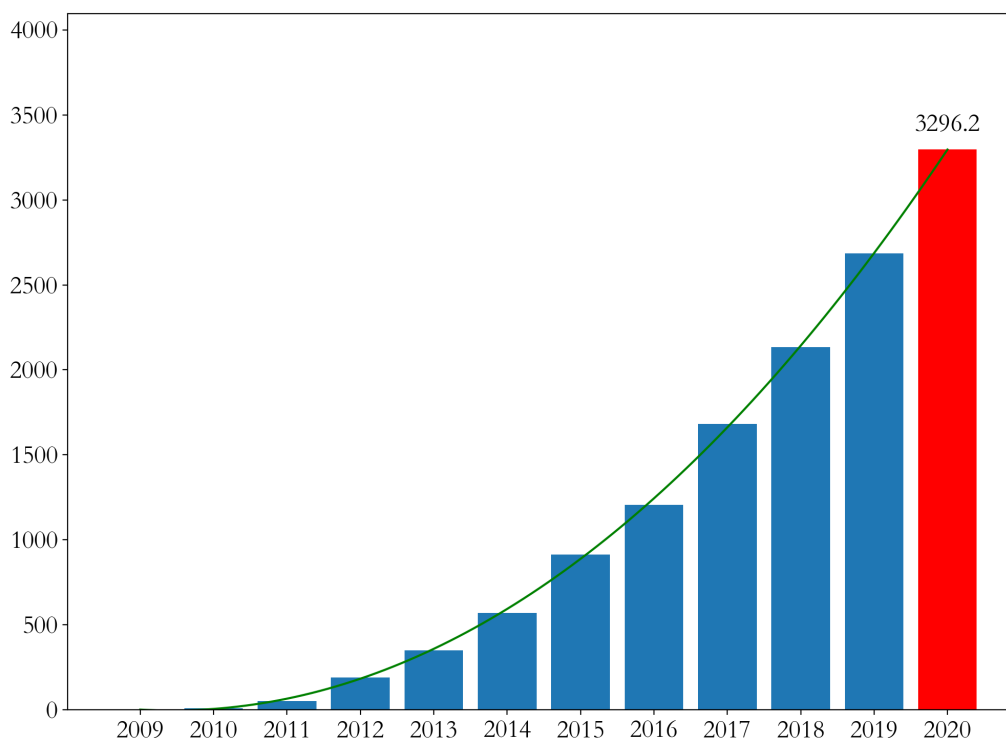


有图可知，在一定时间内，随着经济的发展，天猫双十一销量与年份的关系是多项式关系！假定，销量和年份之间关系是三次幂关系：

$$f(x) = w_1x + w_2x^2 + w_3x^3 + b$$

```
1 import numpy as np
2 from sklearn.linear_model import SGDRegressor
3 import matplotlib.pyplot as plt
4 from sklearn.preprocessing import PolynomialFeatures
5 from sklearn.preprocessing import StandardScaler
6 plt.figure(figsize=(12,9))
7
8 # 1、创建数据，年份数据2009 ~ 2019
9 x = np.arange(2009,2020)
10 y =
    np.array([0.5,9.36,52,191,350,571,912,1207,1682,2135,
    2684])
11
12 # 2、年份数据，均值移除，防止某一个特征列数据天然的数值太大而影响结果
```

```
13 x = x - x.mean()
14 x = x.reshape(-1,1)
15
16 # 3、构建多项式特征，3次幂
17 poly = PolynomialFeatures(degree=3)
18 x = poly.fit_transform(X)
19 s = StandardScaler()
20 x_norm = s.fit_transform(X)
21
22 # 4、创建模型
23 model = SGDRegressor(penalty='l2',eta0 = 0.5,max_iter
    = 5000)
24 model.fit(x_norm,y)
25
26 # 5、数据预测
27 x_test = np.linspace(-5,6,100).reshape(-1,1)
28 x_test = poly.transform(x_test)
29 x_test_norm = s.transform(x_test)
30 y_test = model.predict(x_test_norm)
31
32 # 6、数据可视化
33 plt.plot(x_test[:,1],y_test,color = 'green')
34 plt.bar(x[:,1],y)
35 plt.bar(6,y_test[-1],color = 'red')
36 plt.ylim(0,4096)
37 plt.text(6,y_test[-1] + 100,round(y_test[-1],1),ha =
    'center')
38 _ = plt.xticks(np.arange(-5,7),np.arange(2009,2021))
```



结论：

- 数据预处理，均值移除。如果特征**基准值和分散度**不同在某些算法（例如回归算法，KNN等）上可能会大大影响了模型的预测能力。通过均值移除，大大增强数据的**离散化**程度。
- 多项式升维，需要对数据进行Z-score归一化处理，效果更佳出色
- SGD随机梯度下降需要调整参数，以使模型适应多项式数据
- 从2020年开始，天猫双十一统计的成交额改变了规则为11.1日~11.11日的成交数据（之前的数据为双十一**当天**的数据），2020年成交额为**4980**亿元
- 可以，经济发展有其客观规律，前11年高速发展（曲线基本可以反应销售规律），到2020年是一个转折点

6、代码实战中国人寿保费预测

6.1、数据加载与介绍

```
1 import numpy as np
2 import pandas as pd
3 data = pd.read_excel('./中国人寿.xlsx')
4 print(data.shape)
5 data.head()
```

数据介绍：

- 共计1338条保险数据，每条数据7个属性
- 最后一列charges是保费
- 前面6列是特征，分别为：年龄、性别、体重指数、小孩数量、是否抽烟、所在地区

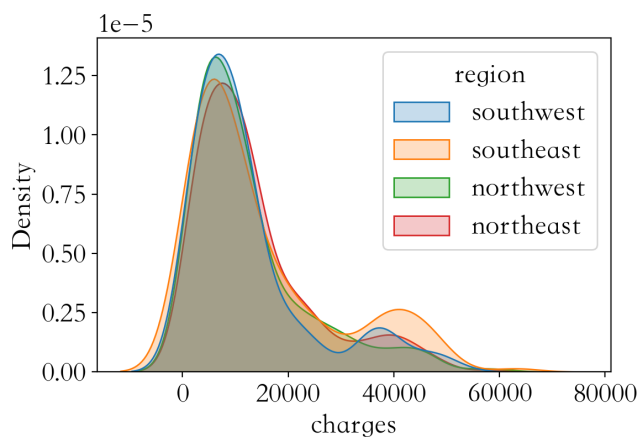
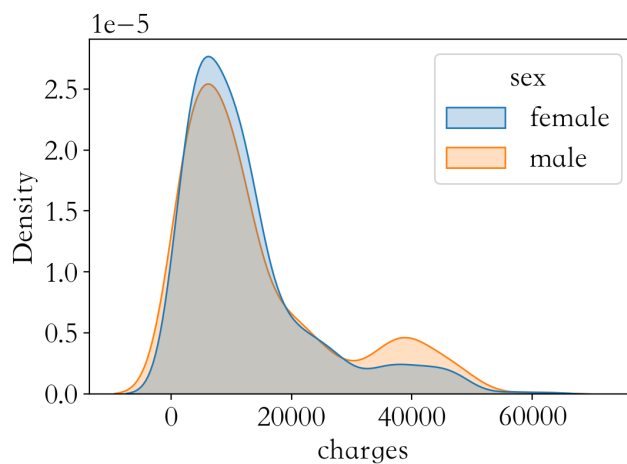
6.2、EDA数据探索

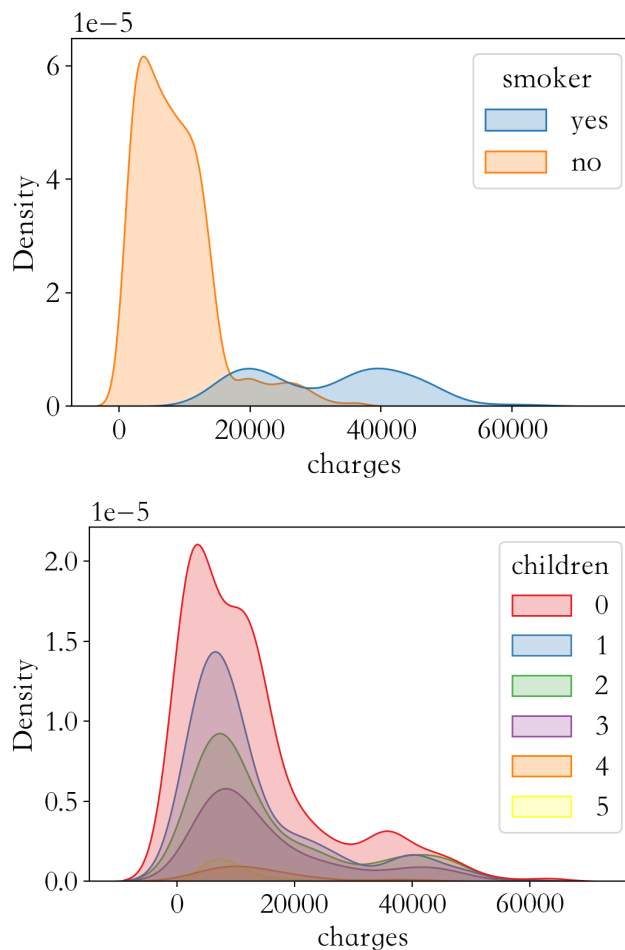
EDA（Exploratory Data Analysis，数据探索分析）是数据分析的第一步，它旨在了解数据的特征、结构和潜在规律，为进一步分析和建模提供基础。

```

1 import seaborn as sns
2 # 性别对保费影响
3 sns.kdeplot(data['charges'],shade = True,hue =
  data['sex'])
4
5 # 地区对保费影响
6 sns.kdeplot(data['charges'],shade = True,hue =
  data['region'])
7
8 # 吸烟对保费影响
9 sns.kdeplot(data['charges'],shade = True,hue =
  data['smoker'])
10
11 # 孩子数量对保费影响
12 sns.kdeplot(data['charges'],shade = True,hue =
  data['children'],palette='Set1')

```





总结：

- 不同性别对保费影响不大，不同性别的保费的概率分布曲线基本重合，因此这个特征无足轻重，可以删除
- 地区同理
- 吸烟与否对保费的概率分布曲线差别很大，整体来说不吸烟更加健康，那么保费就低，这个特征很重要
- 家庭孩子数量对保费有一定影响

6.3、特征工程

```
1 data = data.drop(['region', 'sex'], axis=1)
2 data.head() # 删除不重要特征
3
4 # 体重指数，离散化转换，体重两种情况：标准、肥胖
5 def convert(df, bmi):
6     df['bmi'] = 'fat' if df['bmi'] >= bmi else
    'standard'
```

```
7     return df
8 data = data.apply(convert, axis = 1, args=(30,))
9 data.head()
10
11 # 特征提取，离散型数据转换为数值型数据
12 data = pd.get_dummies(data)
13 data.head()
14
15 # 特征和目标值抽取
16 x = data.drop('charges', axis=1) # 训练数据
17 y = data['charges'] # 目标值
18 x.head()
```

6.4、特征升维

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.linear_model import ElasticNet
3 from sklearn.metrics import
  mean_squared_error, mean_squared_log_error
4
5 # 数据拆分
6 from sklearn.model_selection import train_test_split
7 from sklearn.preprocessing import PolynomialFeatures
8 X_train, X_test, y_train, y_test =
  train_test_split(X, y, test_size=0.2)
9
10 # 特征升维
11 poly = PolynomialFeatures(degree= 2, include_bias =
  False)
12 X_train_poly = poly.fit_transform(X_train)
13 X_test_poly = poly.fit_transform(X_test)
```

6.5、模型训练与评估

$$MSLE = \frac{1}{n} \sum_{i=1}^n (\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

普通线性回归：

```

1 # 升维数据
2 model = LinearRegression()
3 model.fit(X_train_poly,y_train)
4
5 print('训练数据表现: ',model.score(X_train_poly,y_train))
6 print('测试数据表现: ',model.score(X_test_poly,y_test))
7
8 print('训练数据MSE: ',mean_squared_error(y_train,model.predict(X_train_poly)))
9 print('测试数据MSE: ',mean_squared_error(y_test,model.predict(X_test_poly)))
10
11 print('训练数据MSLE: ',mean_squared_log_error(y_train,model.predict(X_train_poly)))
12 print('测试数据MSLE: ',mean_squared_log_error(y_test,model.predict(X_test_poly)))

```

未升维数据对比（表现差强人意，说明升维更棒）

```

1 # 未进行特征升维建模预测表现
2 model = LinearRegression()
3 model.fit(X_train,y_train)
4
5 print('训练数据表现: ',model.score(X_train,y_train))
6 print('测试数据表现: ',model.score(X_test,y_test))
7
8 print('训练数据
MSE: ',mean_squared_error(y_train,model.predict(X_train)))
9 print('测试数据MSE: ',mean_squared_error(y_test,
model.predict(X_test)))
10
11 print('训练数据
MSLE: ',mean_squared_log_error(y_train,model.predict(
X_train)))
12 print('测试数据MSLE: ',mean_squared_log_error(y_test,
model.predict(X_test)))

```

弹性网络回归:

```

1 # 升维后的数据, 特征变多了
2 import warnings
3 warnings.filterwarnings('ignore')
4 from sklearn.preprocessing import StandardScaler
5 # X_train_poly_norm =
scaler.fit_transform(X_train_poly)
6 # X_test_poly_norm = scaler.transform(X_test_poly)
7 model =
ElasticNet(alpha=0.1,l1_ratio=0.3,max_iter=1000)
8 model.fit(X_train_poly,y_train)
9
10 print('训练数据表
现: ',model.score(X_train_poly,y_train))
11 print('测试数据表现: ',model.score(X_test_poly,y_test))
12

```

```

13 print('训练数据
    MSE: ',mean_squared_error(y_train,model.predict(X_train_poly)))
14 print('测试数据MSE: ',mean_squared_error(y_test,
    model.predict(X_test_poly)))
15
16 print('训练数据
    MSLE: ',mean_squared_log_error(y_train,model.predict(
    X_train_poly)))
17 print('测试数据MSLE: ',mean_squared_log_error(y_test,
    model.predict(X_test_poly)))

```

未升维数据对比（表现差强人意，说明升维更棒）

```

1 # 未进行特征升维建模预测表现
2 model =
    ElasticNet(alpha=0.1,l1_ratio=0.3,max_iter=1000)
3 model.fit(X_train,y_train)
4
5 print('训练数据表现: ',model.score(X_train,y_train))
6 print('测试数据表现: ',model.score(X_test,y_test))
7
8 print('训练数据
    MSE: ',mean_squared_error(y_train,model.predict(X_train)))
9 print('测试数据MSE: ',mean_squared_error(y_test,
    model.predict(X_test)))
10
11 print('训练数据
    MSLE: ',mean_squared_log_error(y_train,model.predict(
    X_train)))
12 print('测试数据MSLE: ',mean_squared_log_error(y_test,
    model.predict(X_test)))

```

结论:

- 进行EDA数据探索，可以查看无关紧要特征

- 进行特征工程：删除无用特征、特征离散化、特征提取。这对机器学习都至关重要
- 对于简单的数据（特征比较少）进行线性回归，一般需要进行特征升维
- 选择不同的算法，进行训练和评估，从中筛选优秀算法