欠拟合与过拟合

课前准备

- 下载Anaconda软件，请点击[这里](#)进行下载。

本节要点

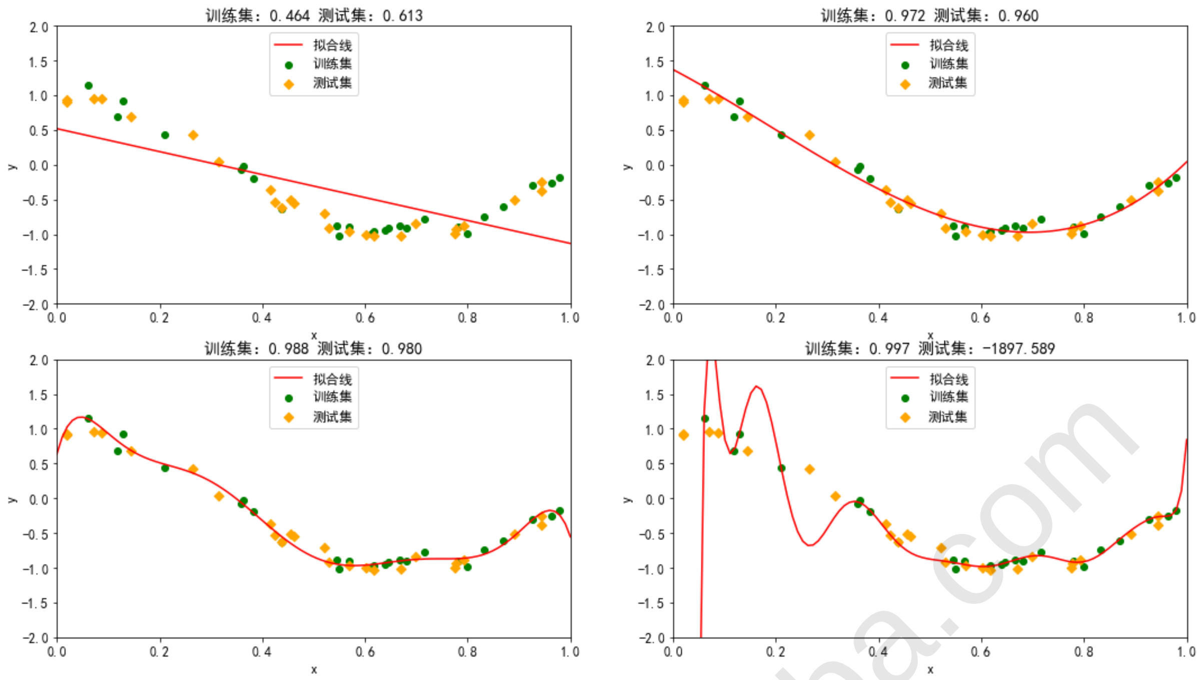
- 欠拟合与过拟合概念与产生原因。
- 多项式扩展原理。
- L1，L2与Elastic Net正则化。

再谈拟合

概念

之前，我们介绍过拟合的概念。拟合指构建一种算法，使得算法的预测值能够符合样本的真实值。与拟合相关的两个概念是**欠拟合与过拟合**。

- 欠拟合：模型过于简单，未能充分捕获样本数据的特征。表现为模型在训练集上的效果不好。
- 过拟合：模型过于复杂，过分捕获样本数据的特征，从而将样本数据中一些特殊特征当成了共性特征。表现为模型在训练集上的效果非常好，但是在未知数据上的表现效果不好。



解决方案

欠拟合解决方案：

- 增加模型复杂度。
  - 引入新的特征，例如多项式扩展。
  - 使用复杂的模型，例如由线性模型改成非线性模型。
- 增加迭代次数。

过拟合解决方案：

- 收集更多的数据。

- 降低模型的复杂度。
  - 使用简单的模型。
  - 减少特征数量。
  - 使用正则化。
- 减少迭代次数。
  - early stopping。

课堂练习

关于欠拟合与过拟合，说法正确的是（ ）。【不定项】
A 某模型在训练集上的效果明显大于测试集，这是欠拟合现象。
B 如果模型在训练集上的效果与在测试集上的效果差不多，则模型既没有欠拟合，也没有过拟合。
C 当产生过拟合时，首先想到的解决方案，是收集更多的数据。
D 模型既不是越简单越好，也不是越复杂越好。

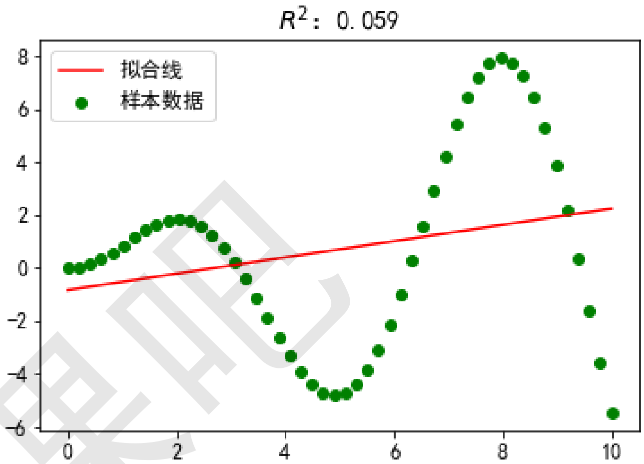
多项式扩展

欠拟合现象

我们可以使用线性回归模型来拟合数据，然而，在现实中，数据未必总是线性（或接近线性）的。当数据并非线性时，直接使用LinearRegression的效果可能会较差，产生欠拟合。

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False
plt.rcParams["font.size"] = 12

x = np.linspace(0, 10, 50)
# 构造非线性数据。
y = x * np.sin(x)
# 为x扩展一个维度，从一维变成二维。也可以通过reshape来实现同样的功能。
X = x[:, np.newaxis]
lr = LinearRegression()
lr.fit(X, y)
plt.scatter(x, y, c="g", label="样本数据")
plt.plot(X, lr.predict(X), "r-", label="拟合线")
plt.legend()
plt.title(f"$R^2$: {lr.score(X, y):.3f}")
```

```
Text(0.5, 1.0, '$R^2$: 0.059')
```



多项式扩展规则

此时，我们可以尝试使用多项式扩展来进行改进。**多项式扩展**，可以认为是对现有数据进行的一种转换，通过生成新的特征，从而将数据由低维映射到高维空间中，这样模型就可以拟合更广泛的数据。

我们将原始的特征称为**输入特征**，将多项式扩展之后的特征称为**输出特征**，多项式扩展的规则如下（假设为n阶扩展）：

- 对所有输入特征进行乘积组合。
- 每个输入特征具有一个指数，指数的取值范围为$[0, n]$。
- 遍历所有指数可能的取值，但是需要保证所有输入特征的指数之和不超过n（小于等于n）。
- 对每个符合上述条件的指数组合，作为扩展之后的输出特征。

假设，我们有如下的二元线性模型：

$$\hat{y} = w_0 + w_1x_1 + w_2x_2$$

如果该模型的拟合效果不佳，我们就可以对该模型进行多项式扩展。例如，我们进行2阶扩展（也可以进行更高阶的扩展），符合条件的指数组合为：

$$[x_1^0x_2^0,\ x_1^1x_2^0,\ x_1^0x_2^1,\ x_1^2x_2^0,\ x_1^1x_2^1,\ x_1^0x_2^2]$$

因此，经过多项式扩展之后，最终的输出特征为：

$$[1,\ x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2]$$

模型也最终变为：

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

当进行多项式扩展后，特征数量由之前的2个变成5个（特征维度从2维变成5维），从而可以更灵活的去拟合数据。

课堂练习

经过多项式扩展之后，最高次项已经不再是1，这样的数据，还能适用于线性回归这样的模型吗？
A 可以。
B 不能。
C 有时能，有时不能。
D 不好确定。

经过多项式扩展后，我们依然可以使用之前的线性回归模型去拟合数据。这是因为，我们可以假设：

$$\vec{z} = (z_1, z_2, z_3, z_4, z_5) = (x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

这样，之前的模型就会变成：

$$\hat{y} = w_0 + w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5$$

从而，我们依然可以认为，这还是一种线性模型。

PolynomialFeatures

我们可以使用sklearn中提供的PolynomialFeatures类来实现多项式扩展。通过powers_属性可以获取扩展之后每个输入特征的指数组合（矩阵）。关于指数矩阵，描述如下：

- 矩阵的每一列对应输入特征，即列数等于输入特征数。
- 矩阵的每一行对应输出特征，即行数等于输出特征数。
- 矩阵的形状为[输出特征数， 输入特征数]。
- 矩阵的值表示每个输入特征的指数值。
    - powers_[i, j]表示第i个输出特征中，第j个输入特征的指数值。

例如，如果输入样本的特征数为2，多项式扩展阶数为2，则指数矩阵为：

$$powers = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 1 & 1 \\ 0 & 2 \end{bmatrix}$$

对于两个输入特征x_1与x_2来讲，多项式转换之后的值为：

$$[x_1^0x_2^0,\ x_1^1x_2^0,\ x_1^0x_2^1,\ x_1^2x_2^0,\ x_1^1x_2^1,\ x_1^0x_2^2]$$

即：

$$[1,\ x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2]$$

```python
from sklearn.preprocessing import PolynomialFeatures

X = np.array([[1, 2], [3, 4]])
# degree：扩展的阶数。阶数越高，则输出特征越多。
# include_bias：是否包含偏置，默认为True。
poly = PolynomialFeatures(2, include_bias=True)
# 对输入数据进行转换。
# 相当于调用fit之后，再调用transform。
# poly.fit(X)
# r = poly.transform(X)
r = poly.fit_transform(X)
print("转换之后的结果：")
print(r)
print("指数矩阵：")
print(poly.powers_)
print("输入的特征数量：", poly.n_input_features_)
print("输出的特征数量：", poly.n_output_features_)

for x1, x2 in X:
    for e1, e2 in poly.powers_:
        print(x1 ** e1 * x2 ** e2, end="\t")
    print()
```

```
转换之后的结果：
[[ 1.  1.  2.  1.  2.  4.]
 [ 1.  3.  4.  9. 12. 16.]]
指数矩阵：
[[0 0]
 [1 0]
 [0 1]
 [2 0]
 [1 1]
 [0 2]]
输入的特征数量： 2
输出的特征数量： 6
1	1	2	1	2	4
1	3	4	9	12	16
```

课堂练习

如果有2个输入特征，经过多项式3阶扩展后，会有几个输出特征（含偏置项）？

A 7个

B 8个

C 9个

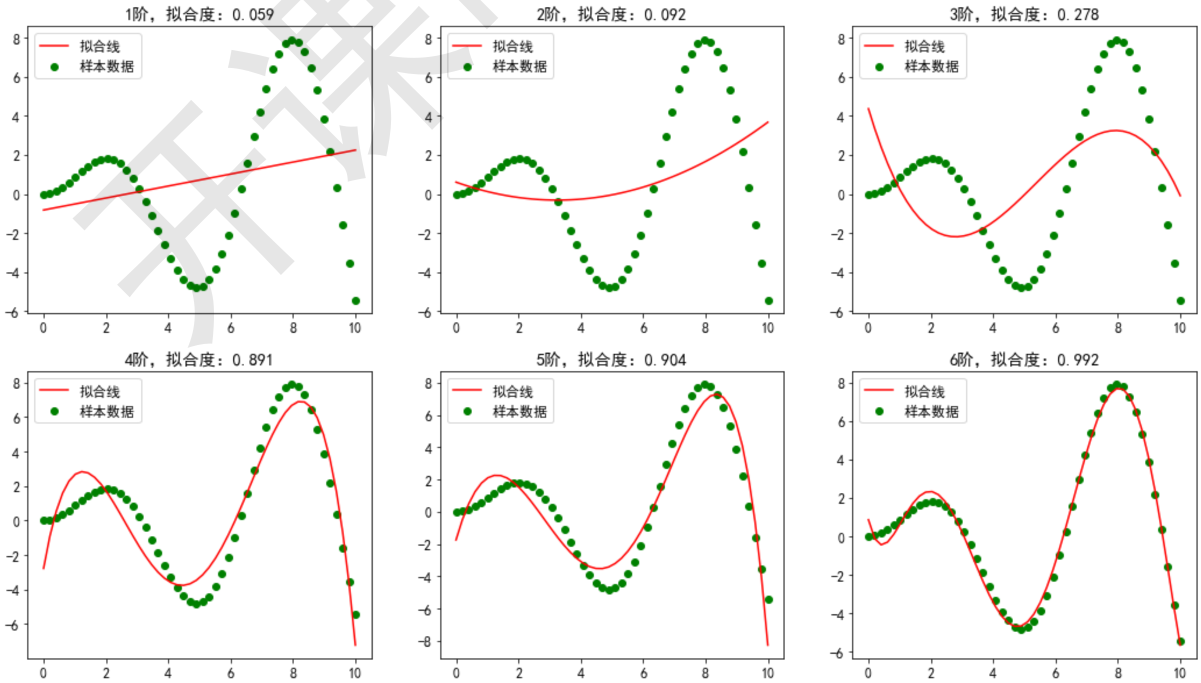
D 10个

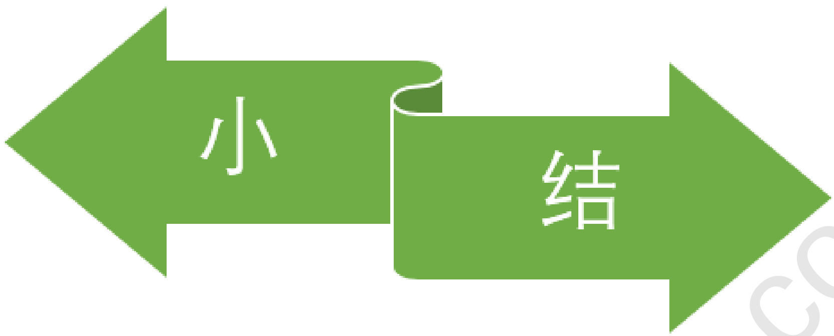
解决欠拟合

现在，就让我们对之前的程序来进行多项式扩展，尝试解决欠拟合问题。

```python
x = np.linspace(0, 10, 50)
y = x * np.sin(x)
X = x[:, np.newaxis]
figure, ax = plt.subplots(2, 3)
figure.set_size_inches(18, 10)
ax = ax.ravel()
```

```python

# 进行1阶到6阶的多项式扩展。（1阶相当于没有扩展）
for n in range(1, 7):
    poly = PolynomialFeatures(degree=n, include_bias=False)
    X_transform = poly.fit_transform(X)
    lr = LinearRegression()
    lr.fit(X_transform, y)
    ax[n - 1].set_title(f"{n}阶，拟合度：{lr.score(X_transform, y):.3f}")
    ax[n - 1].scatter(x, y, c="g", label="样本数据")
    ax[n - 1].plot(x, lr.predict(X_transform), "r-", label="拟合线")
    ax[n - 1].legend()
```





流水线

在上例中，我们使用多项式对训练数据进行了转换（扩展），然后使用线性回归类（LinearRegression）在转换后的数据上进行拟合。可以说，这是两个步骤。我们虽然可以分别去执行这两个步骤，然而，当数据预处理的工作较多时，可能会涉及更多的步骤（例如标准化，One-Hot编码，特征选择等操作），此时分别执行每个步骤会显得过于繁琐。

流水线（Pipeline类）可以将每个评估器视为一个步骤，然后将多个步骤作为一个整体而依次执行，这样，我们就无需分别执行每个步骤。例如，在上例中，我们就可以将多项式转换与训练模型两个步骤视为一个整体，一并执行。

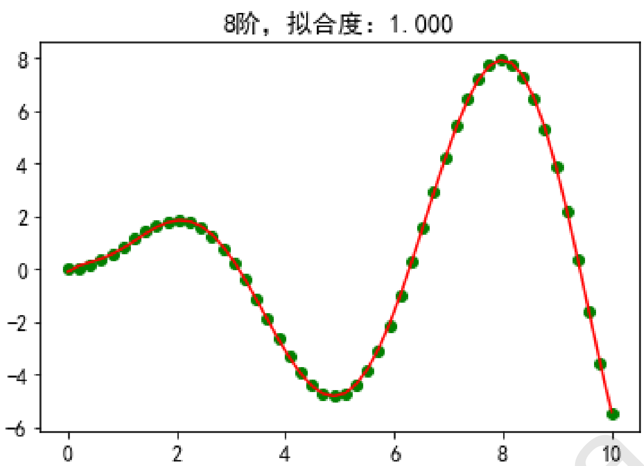
流水线具有最后一个评估器的所有方法。当通过流水线对象调用方法f时，会执行这样的过程（假设流水线具有n个评估器）：

- 如果f是fit或fit_transform方法，则会首先对前n - 1个评估器依次调用fit_transform方法，然后在最后一个评估器上调用f方法。
- 如果f是其他方法，则会首先对前n - 1个评估器依次调用transform方法，然后在最后一个评估器上调用f方法。

例如，当在流水线上调用fit方法时，将会依次在每个评估器上调用fit_transform方法，前一个评估器将转换之后的结果传递给下一个评估器，直到最后一个评估器调用fit方法为止（最后一个评估器不会调用transform方法）。而当在流水线上调用predict方法时，则会依次在每个评估器上调用transform方法，在最后一个评估器上调用predict方法。

```python
from sklearn.pipeline import Pipeline

x = np.linspace(0, 10, 50)
y = x * np.sin(x)
X = x[:, np.newaxis]
# 定义流水线的步骤。类型为一个列表，列表中的每个元素是元组类型，
# 格式为：[(步骤名1，评估器1)，(步骤名2， 评估器2)，……， (步骤名n, 评估器n)
steps = [("poly", PolynomialFeatures(include_bias=False)), ("lr", LinearRegression())]
pipe = Pipeline(steps)
# 设置流水线的参数。所有可用的参数，可以通过pipeline.get_params()获取。
pipe.set_params(poly__degree=8)
pipe.fit(X, y)
score = pipe.score(X, y)
plt.title(f"8阶，拟合度：{score:.3f}")
plt.scatter(X, y, c="g", label="样本数据")
plt.plot(X, pipe.predict(X), "r-", label="拟合线")
```

```
[<matplotlib.lines.Line2D at 0x1b125c59b48>]
```



课堂练习

关于流水线（Pipeline），说法正确的有（ ）。【不定项】
A 使用流水线，可以省略频繁的fit与transform过程。
B 流水线可以加快程序的运行速度。
C 流水线可以将多个步骤作为一个整体而执行。
D 流水线内的所有步骤（评估器），都必须具有transform方法。

正则化

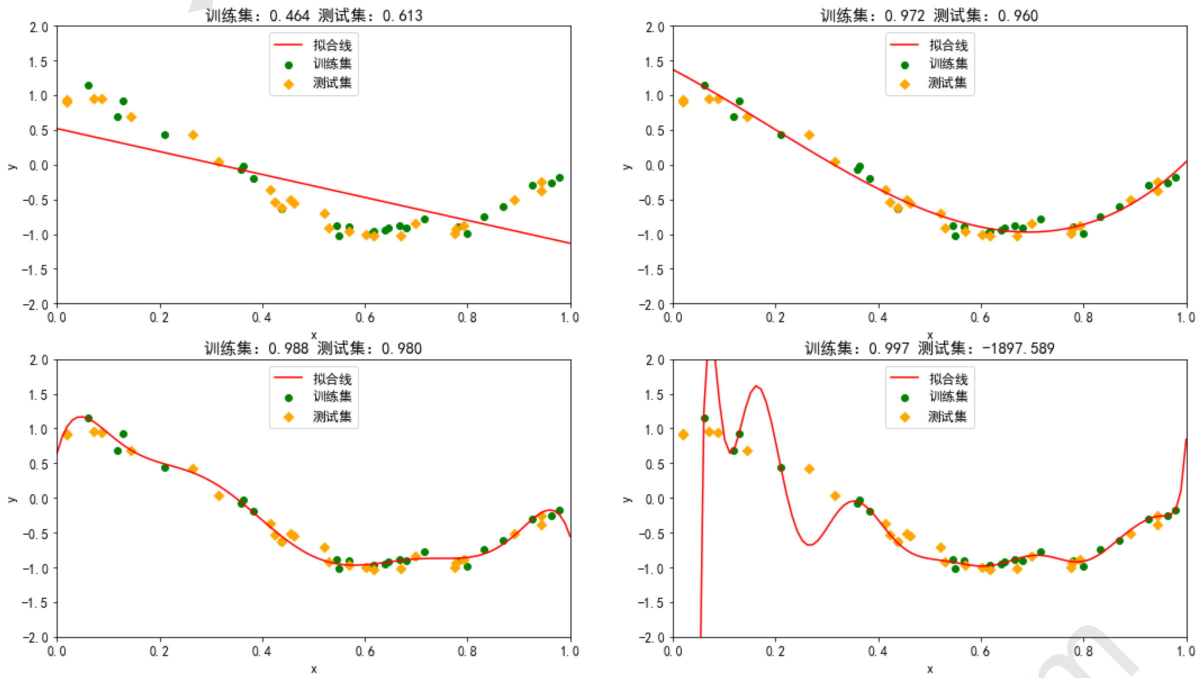
过拟合现象

通过之前的程序，我们发现，使用多项式扩展完美的解决了欠拟合问题。如果我们使用更多阶（8阶）的多项式扩展，甚至可以将拟合度提高为1。但是，问题来了，多项式扩展时，是否阶数越多越好呢?

```
from sklearn.model_selection import train_test_split

def true_fun(x):
    """数据分布的函数，用于生成由x -> y的映射。

    根据x数组中的每个元素，返回该元素的余弦计算值y。

    Parameters
    ----------
    x : array-like
        训练数据集。

    Returns
    -------
    y : array-like
        x对应的余弦映射结果。

    """

    return np.cos(1.5 * np.pi * x)


def fit_and_plot(model):
    """用来训练模型，并且绘制模型的拟合效果。

    Parameters
    ----------
    model : object
        模型对象。

    """

    np.random.seed(0)
    x = np.random.rand(50)
    # 在映射函数上，增加一定的误差（噪声）。这样更符合现实中数据的分布。
    # 误差服从正态分布。
    y = true_fun(x) + np.random.randn(len(x)) * 0.1
    X = x[:, np.newaxis]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=0)
    model.fit(X_train, y_train)
    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)
    plt.scatter(X_train, y_train, c="g", label="训练集")
    plt.scatter(X_test, y_test, c="orange", marker="D", label="测试集")
    s = np.linspace(0, 1, 100).reshape(-1, 1)
    plt.plot(s, model.predict(s), c="r", label="拟合线")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="upper center")
    plt.title(f"训练集：{train_score:.3f} 测试集：{test_score:.3f}")


```

```python
# 定义多项式扩展的阶数。
degrees = [1, 3, 8, 15]
plt.figure(figsize=(18, 10))
for i, n in enumerate(degrees):
    plt.subplot(2, 2, i + 1)
    pipe = Pipeline([("poly", PolynomialFeatures(degree=n, include_bias=False)),
            ("lr", LinearRegression())])
    fit_and_plot(pipe)
    # named_steps返回字典对象，提供流水线中每个步骤的名称（key）与对象（value）的映射。
    print(pipe.named_steps["lr"].coef_)
```

```
[-1.65201925]
[-3.89922088 -3.34968481  5.92093622]
[ 2.82447079e+01 -4.99295928e+02  3.53686879e+03 -1.31271460e+04
  2.71571014e+04 -3.15089674e+04  1.91826234e+04 -4.77061916e+03]
[ 2.07071311e+04 -4.97833841e+05  6.76423888e+06 -5.85834707e+07
  3.45651499e+08 -1.44778371e+09  4.41812369e+09 -9.97113468e+09
  1.67390055e+10 -2.08331288e+10  1.89560333e+10 -1.22494207e+10
  5.32316771e+09 -1.39466930e+09  1.66452216e+08]
```



常用的正则化

我们将之前的程序中，输出系数（权重与偏置）的注释取消，再次运行程序，我们会发现什么？

在线性回归中，模型过于复杂，通常表现为模型的参数过大（指绝对值过大），即如果模型的参数过大，就容易出现过拟合现象。我们可以通过正则化来降低过拟合的程度。**正则化**，就是通过在损失函数中加入关于权重的惩罚项，进而限制模型的参数过大，从而减低过拟合，增加的惩罚项，我们也称作正则项。

根据正则项的不同，我们可以将正则化分为如下几种：

- L2正则化
- L1正则化
- Elastic Net

L2正则化

L2正则化是最常使用的正则化，将所有权重的平方和作为正则项。使用L2正则的线性回归模型称为Ridge回归（岭回归）。加入L2正则化的损失函数为：

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 + \alpha\sum_{j=1}^{n}w_j^2$$

- m：样本数量。
- n：特征数量。
- α：惩罚系数（$\alpha > 0$）。

从包含正则项的损失函数中，我们可以发现，我们将损失函数分为两部分，一部分为原来的损失函数，另外一部分为正则项的惩罚，这样，如果当权重过大时，即使原来的损失函数值很小，但是整个项的损失值会很多，因此，整个损失值（二者的和）也不会很小，从而，权重过大的w就可能不会成为最佳解。

L1正则化

L1正则化使用所有权重的绝对值和作为正则项。使用L1正则的线性回归模型称为Lasso回归（Least Absolute Shrinkage and Selection Operator——最小绝对值收缩与选择因子）。

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 + \alpha\sum_{j=1}^{n}|w_j|$$

Elastic Net

Elastic Net（弹性网络），同时将绝对值和与平方和作为正则项，是L1正则化与L2正则化之间的一个折中。使用该正则项的线性回归模型称为Elastic Net算法。

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 + \alpha(p\sum_{j=1}^{n}|w_j| + (1-p)\sum_{j=1}^{n}w_j^2)$$

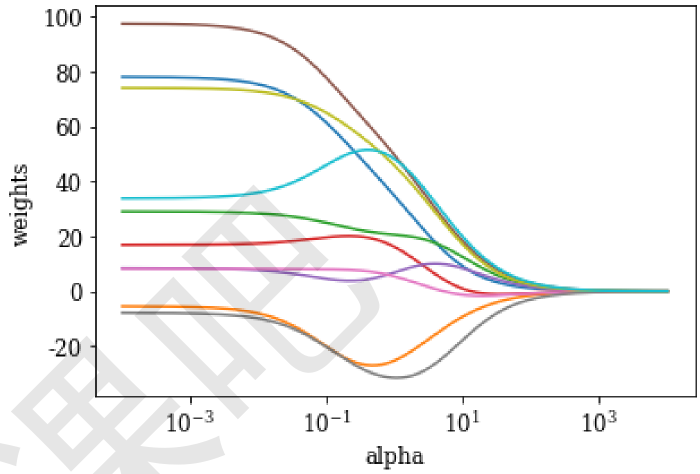
- p：L1正则化的比重（$0 <= p <= 1$）。

正则化对权重的影响

我们以L2正则化为例，来演示不同α取值下，正则化对权重的影响。

```
from sklearn.datasets import make_regression
from sklearn.linear_model import Ridge
# 注意：当坐标轴使用对数比例后，这里需要改成英文字体，否则无法正常显示。
plt.rcParams["font.family"] = "serif"

# 创建回归数据集。
# n_samples：样本数量。
# n_features：特征数量。
# coef：是否返回权重。默认为False。
# random_state：随机种子。
# bias：偏置。
# noise：增加的噪声干扰，值越大，干扰越大。
X, y, w = make_regression(n_samples=10, n_features=10, coef=True,
        random_state=1, bias=3.5, noise=0.0)

alphas = np.logspace(-4, 4, 200)
# 定义列表，用来保存在不同alpha取值下，模型最优的权重（w）值。
coefs = []
# 创建岭回归对象。
ridge = Ridge()
for a in alphas:
    # alpha：惩罚力度，值越大，惩罚力度越大。
    ridge.set_params(alpha=a)
    ridge.fit(X, y)
    # 将每个alpha取值下，Ridge回归拟合的最佳解（w）加入到列表中。
    coefs.append(ridge.coef_)

# gca  get current axes 获取当前的绘图对象。
ax = plt.gca()
# 当y是二维数组时，每一列会认为是一个单独的数据集。
ax.plot(alphas, coefs)
# 设置x轴的比例。（对数比例）
ax.set_xscale("log")
# 设置x轴的标签。
ax.set_xlabel("alpha")
# 设置y轴的标签。
ax.set_ylabel("weights")
```

```
Text(0, 0.5, 'weights')
```



课堂练习

关于正则化，以下说法错误的是（ ）。

A 常用的正则化有L1，L2与 Elastic Net三种。

B α值越大，正则化强度越大。

C 正则化，是通过在损失函数上，加入正则项，从而可以限制权重（w）过大。

D 如果模型比较复杂，就一定会产生过拟合。

小结

基本 扩展 延伸

正则化说明

- Lasso更容易产生稀疏解，这就减少了模型所依赖的特征数量。因此，可以使用Lasso进行特征选择。
- Ridge模型具有较高的稳定性。
- Elastic Net是Lasso与Ridge之间的一个折中，其可以像Lasso一样产生稀疏解，同时具有Ridge的部分稳定性。
- 当多个特征具有相关性时，Lasso可能只会随机选择其中的一个，而Elastic Net可能会选择多个。

扩展内容：关于Lasso为什么容易产生稀疏解，而Ridge不容易产生稀疏解，老梁提供辅助视频，供大家扩展学习。

通过Lasso实现特征选择

我们以模拟生成的数据为例，来演示通过Lasso结合SelectFromModel实现特征选择。

```
from sklearn.linear_model import Lasso

X, y = make_regression(n_samples=10, n_features=10, coef=False,
        random_state=1, bias=3.5, noise=1)
lasso = Lasso(alpha=1)
lasso.fit(X, y)
print(lasso.coef_)
```

```
[54.14243012 71.93458854  3.52415557 13.50646852 36.90416205  5.6709588
 -0.          -0.          73.44001013 71.26297921]
```

```
from sklearn.feature_selection import SelectFromModel

# estimator：评估器，即SelectFromModel类要进行特征选择的模型。
# threshold：阈值，当特征权重小于阈值时，丢弃该特征。
# prefit：传入的评估器（estimator参数）是否已经训练过了。默认为False。
sfm = SelectFromModel(estimator=lasso, threshold=1e-5, prefit=True)
X_transform = sfm.transform(X)
print(X_transform[:3])
# 返回布尔数组，用来表示是否选择对应的特征，True为选择，False为丢弃。
print(sfm.get_support())
```

```
[[ 1.13376944 -0.3224172  -1.09989127 -2.06014071 -0.87785842  0.04221375
  -0.38405435  1.46210794]
 [ 0.19829972  0.18656139  0.11900865 -0.20075807  0.37756379  0.12182127
   0.41005165 -0.22232814]
 [ 0.86540763 -0.52817175 -2.3015387  -0.61175641 -0.7612069   0.3190391
  -1.07296862  1.62434536]]
[ True  True False  True  True  True False  True  True  True]
```

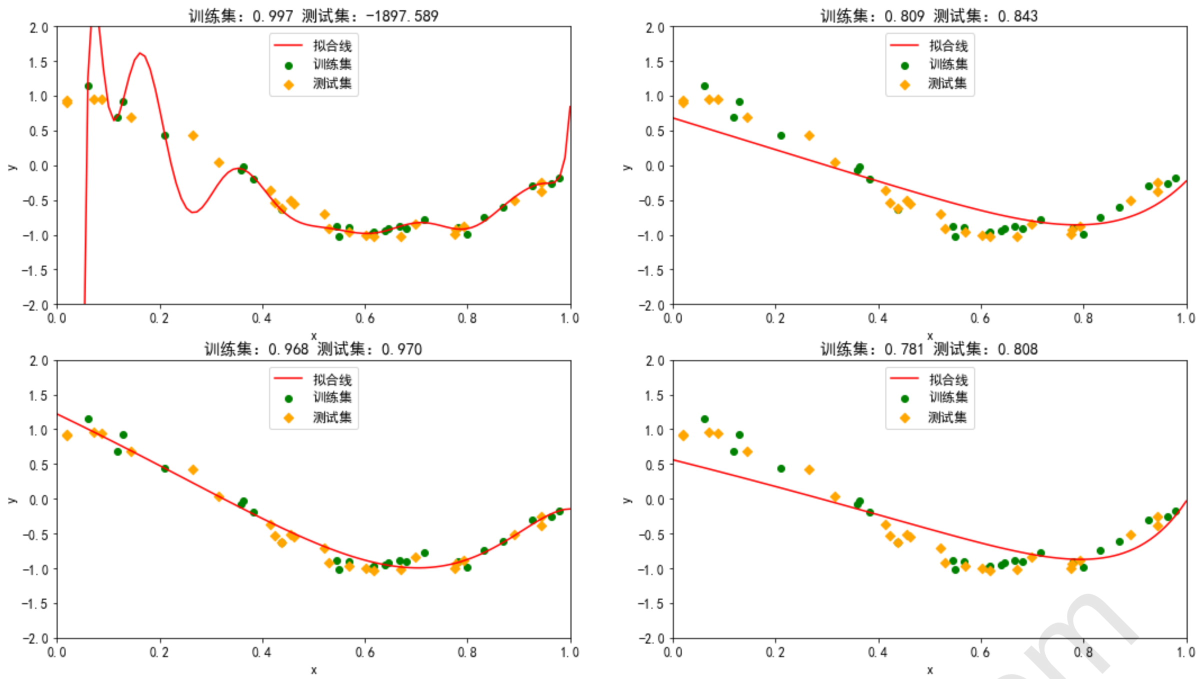
解决过拟合

现在，我们分别使用三种正则化方式，尝试解决过拟合问题。

```
from sklearn.linear_model import ElasticNet
# 将字体改回中文字体。
plt.rcParams["font.family"] = "SimHei"

np.random.seed(0)
x = np.random.rand(50)
y = true_fun(x) + np.random.randn(len(x)) * 0.1
X = x[:, np.newaxis]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
models = [("线性回归（无正则化）", LinearRegression()), ("L1正则化：", Lasso(alpha=0.02)),
          ("L2正则化", Ridge(alpha=0.02)), ("弹性网络", ElasticNet(alpha=0.02,
l1_ratio=0.5))]
plt.figure(figsize=(18, 10))
```

```python
for i, (name, model) in enumerate(models):
    plt.subplot(2, 2, i + 1)
    pipe = Pipeline([("poly", PolynomialFeatures(degree=15, include_bias=False)), ("model", model)])
    fit_and_plot(pipe)
    print(model.coef_)
```

```
[ 2.07071311e+04 -4.97833841e+05  6.76423888e+06 -5.85834707e+07
  3.45651499e+08 -1.44778371e+09  4.41812369e+09 -9.97113468e+09
  1.67390055e+10 -2.08331288e+10  1.89560333e+10 -1.22494207e+10
  5.32316771e+09 -1.39466930e+09  1.66452216e+08]
[-2.28050995 -0.          0.          0.          0.          0.
  1.37135264  0.          0.          0.          0.          0.
  0.          0.          0.        ]
[-3.54177497 -0.98692712  0.47117231  0.99378303  1.03351056  0.86683848
  0.63550359  0.40472689  0.20025764  0.02892467 -0.11067832 -0.22293056
 -0.31276194 -0.38465655 -0.4423375 ]
[-1.86165016 -0.2996501  -0.          0.          0.01961693  0.20690235
  0.27988858  0.28810391  0.26055212  0.21407932  0.15841395  0.09914914
  0.0394854   0.          0.        ]
```



课堂练习

以下说法错误的是（ ）。
A α值越小，Lasso越倾向产生更多值为0的权重。
B Lasso可以实现特征选择。
C Ridge具有更好的稳定性。
D Elastic Net与Lasso都可能产生稀疏解。

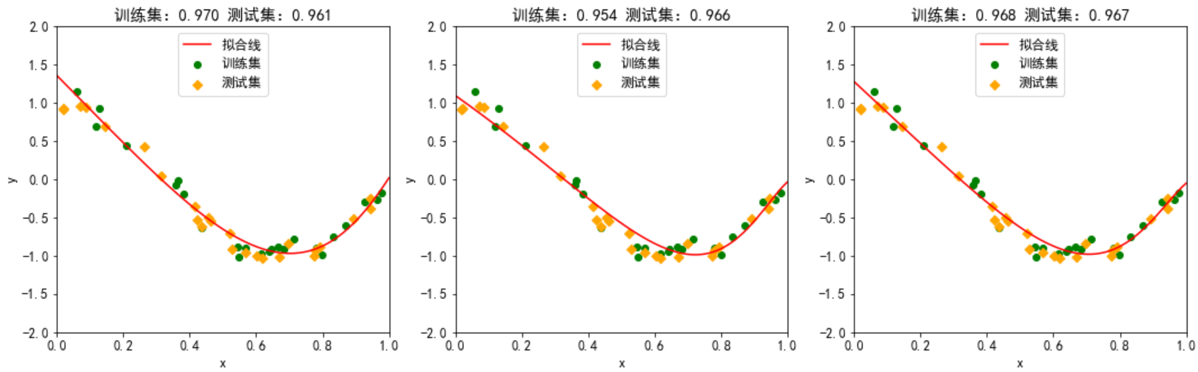
交叉验证调整超参数

通过刚才的示例，我们可知，通过正则化，已经有效的缓解了过拟合，不过，刚才示例中，超参数α的取值是比较随意的，如果使用合适的 α值，或许还能够提高模型的效果。这里，我们可以使用交叉验证来调整参数α的取值。

在sklearn中，提供了LassoCV, RidgeCV与ElasticNetCV类，可以用来交叉验证。

```python
from sklearn.linear_model import LassoCV, RidgeCV, ElasticNetCV

alphas = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5]
models = [("L1正则化: ", LassoCV(max_iter=5000)), ("L2正则化", RidgeCV()),
        ("弹性网络", ElasticNetCV(l1_ratio=0.5))]
plt.figure(figsize=(18, 5))
for i, (name, model) in enumerate(models):
    plt.subplot(1, 3, i + 1)
    pipe = Pipeline([("poly", PolynomialFeatures(degree=15)), ("model", model)])
    # 将模型设置为10折交叉验证，其实在models中，各个模型的构造器中，可以指定cv=10,但是需要三个都指定。
    # 这里在循环中，只需要使用一行代码就可以了。
    pipe.set_params(model__cv=10)
    pipe.set_params(model__alphas=alphas)
    fit_and_plot(pipe)
    # 输出最佳的超参数alpha。
    print(name, model.alpha_)
```

```
L1正则化:  0.001
L2正则化 0.05
弹性网络 0.001
```



拓展点

- Lasso回归容易产生稀疏解的原因。

总结

- 欠拟合与过拟合的现象。
- 欠拟合与过拟合的解决方案。
- 多项式扩展。
- 正则化。

作业

1. 对波士顿房价采用多项式扩展，是否可以提高训练集的回归效果。
2. 训练之后，是否也会提高测试集的效果呢?