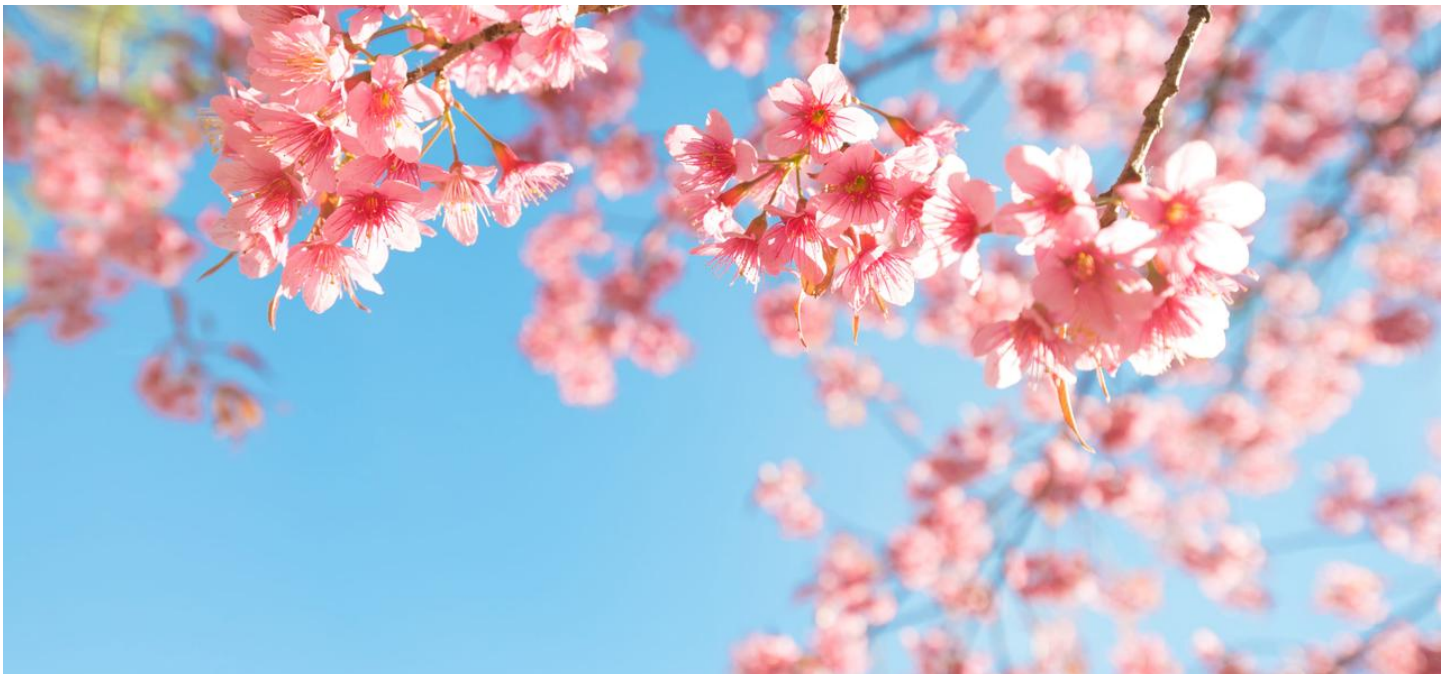
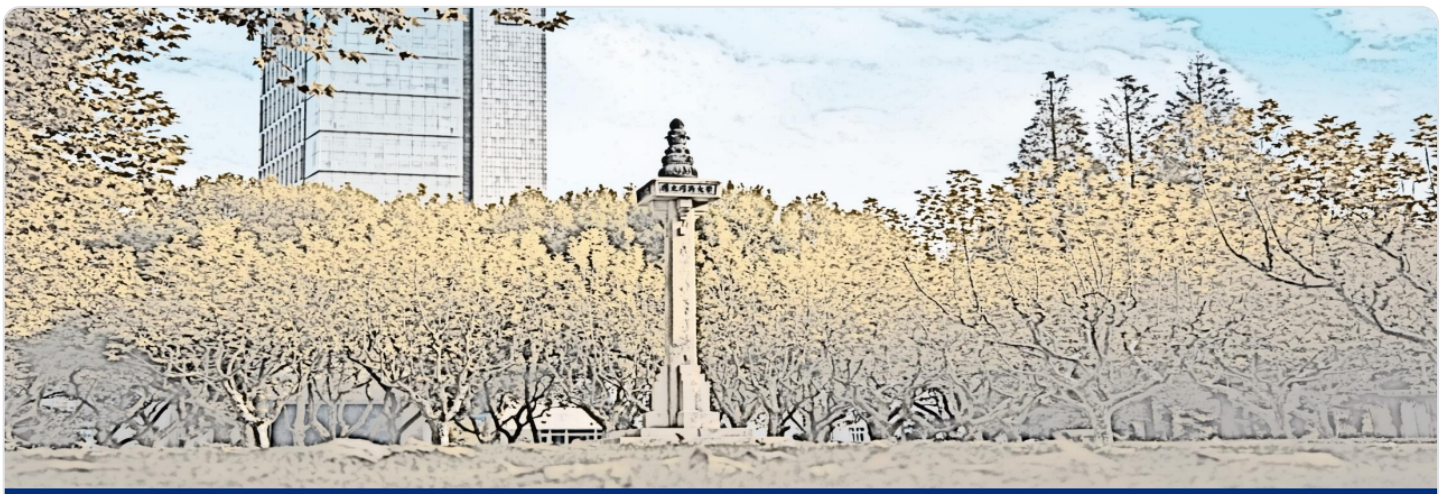


回归分析



第三章 监督学习 3.2.pdf

大纲

回归分析是一种用于预测数值型目标变量的监督学习方法。本章主要介绍线性回归、多项式回归以及正则化等概念，并通过实例演示如何使用Scikit-learn库进行模型训练和评估。

3.2.1 线性回归

- **任务**: 建立输入特征与输出之间的线性关系，预测目标值。
- **模型**: y = w1 * x1 + w2 * x2 + b + ε
- **损失函数**: 常用均方误差 (MSE) 或均方根误差 (RMSE)。
- **训练方法**: 正规方程法或梯度下降法。
- **评估指标**: 决定系数 (R2), MAE, MSE, RMSE 等。

3.2.2 梯度下降法

- **思想**: 沿梯度相反方向迭代调整参数，使损失函数最小化。
- **步骤**: 随机初始化参数，计算梯度，更新参数，重复迭代直至收敛。
- **关键问题**: 梯度计算、收敛性、学习率设置、迭代次数限制。
- **常用算法**: 批量梯度下降法、随机梯度下降法、小批量随机梯度下降法。

3.2.3 多项式回归

- **解决线性回归局限性**: 通过添加高次项特征来拟合非线性数据。
- **多项式特征**: 使用原特征的幂次方或组合生成新特征。
- **Scikit-learn 工具**: PolynomialFeatures 生成多项式特征，Pipeline 串联模型训练步骤。

3.2.4 正则化

- **目的**: 防止过拟合，提高模型泛化能力。
- **原理**: 在损失函数中添加惩罚项，限制模型复杂度。
- **常见方法**: 岭回归 (L2 正则化)、Lasso 回归 (L1 正则化)、弹性网络。
- **Scikit-learn 工具**: Ridge, Lasso, ElasticNet 类实现正则化回归。

实例演示

- **女大学生肺活量预测**: 使用 SGDRegressor 和特征缩放进行线性回归分析。
- **多项式回归应用**: 使用 Pipeline 和 PolynomialFeatures 进行多项式回归和正则化。
- **加州房价预测**: 使用 LinearRegression, Ridge, Lasso, ElasticNet 等模型进行房价预测，并比较性能。

线性回归

线性回归的数学原理和思想主要围绕着以下两点：

1. **线性模型**:

- 线性回归假设输入特征和输出之间存在线性关系，可以用线性方程表示：y = w1 * x1 + w2 * x2 + ⋯ + wn * xn + b
- 其中，x1, x2, ⋯, xn 是输入特征，w1, w2, ⋯, wn 是特征对应的权重，b 是偏置项。
- 线性模型简单易懂，易于建模，但无法处理非线性关系。

2. **最小二乘法**:

最小二乘法是一种求解线性回归模型参数的经典方法。其核心思想是找到一个参数组合，使得所有样本的预测值与真实值之间的误差平方和最小。下面是求解过程：

⚽ **建立模型**:

首先，我们需要建立一个线性回归模型，表示输入特征和输出之间的线性关系：

y = w1 * x1 + w2 * x2 + ⋯ + wn * xn + b

其中，x1, x2, ⋯, xn 是输入特征，w1, w2, ⋯, wn 是特征对应的权重，b 是偏置项。

🍞 **定义损失函数**:

接下来，我们定义损失函数，用于衡量模型预测值与真实值之间的误差。常用的损失函数是均方误差 (MSE):

L(w) = 1/m * Σ(y - f(x))^2

其中，y 是真实值，f(x) = w1 * x1 + w2 * x2 + ⋯ + wn * xn + b 是模型预测值，m 是样本数量。

🏝️ **求导**:

为了找到使损失函数最小的参数组合，我们需要对损失函数关于每个参数求偏导数：

∇L(w) = [∂L/∂w1, ∂L/∂w2, ⋯, ∂L/∂wn]

🦄 **设置偏导数为零**:

将偏导数设置为 0，可以得到一个包含 m+1 个方程的线性方程组：

[∂L/∂w1] = 0 [∂L/∂w2] = 0 ⋯ [∂L/∂wn] = 0

🥛 **解线性方程组**:

解这个线性方程组，就可以得到使损失函数最小的参数组合 w1, w2, ⋯, wn, b。

🎹 **解析解**:

对于线性回归模型，可以通过正规方程法直接计算出解析解：

w = (X^T * X)^(-1) * X^T * y

其中，X 是输入特征矩阵，y 是目标向量。

总结:

最小二乘法通过求解线性方程组，找到了使预测误差最小的参数组合，从而建立了线性回归模型。

数学表达式:

- 损失函数 (MSE):
  - L(w) = 1/m * Σ(y - f(x))^2 = 1/m * Σ(y - (w1 * x1 + w2 * x2 + ⋯ + wn * xn + b))^2
- 正规方程解:
  - w = (X^T * X)^(-1) * X^T * y
- 梯度下降法:
  - w(t+1) = w(t) - η * ∇L(w(t))

总结:

线性回归的数学原理和思想是将输入特征和输出之间的线性关系转化为最小化预测误差的问题，并利用最小二乘法或梯度下降法求解最优参数，从而建立能够预测目标值的模型。

误差分析函数/决定系数

线性回归的常用损失函数主要包括以下几种：

1. **均方误差 (MSE)**:

MSE 是最常用的损失函数，它衡量了模型预测值与真实值之间的平均误差平方：

MSE = 1/m * Σ(y - f(x))^2

其中，y 是真实值，f(x) 是模型预测值，m 是样本数量。

2. **均方根误差 (RMSE)**:

RMSE 是 MSE 的平方根，它以相同的量纲表示误差，更直观地反映了预测值与真实值之间的差异：

RMSE = sqrt(MSE)

3. **平均绝对误差 (MAE)**:

MAE 是预测值与真实值之间绝对误差的平均值，它对异常值更敏感：

MAE = 1/m * Σ|y - f(x)|

4. **Huber 损失**:

Huber 损失结合了 MSE 和 MAE 的优点，对异常值更加鲁棒：

Huber 损失 = {0.5 * (y - f(x))^2, |y - f(x)| <= δ} {δ * (|y - f(x)| - 0.5 * δ), |y - f(x)| > δ}

其中，δ 是一个超参数，用于控制 MSE 和 MAE 的切换点。

5. **Log-cosh 损失**:

Log-cosh 损失是 Huber 损失的一个变体，它在计算上更加稳定：

Log-cosh 损失 = log(cosh(y - f(x)))

选择损失函数:

选择损失函数需要考虑数据的特点和应用场景。MSE 和 RMSE 是最常用的损失函数，适用于大多数回归问题。MAE 对异常值更敏感，适用于数据中存在较多异常值的情况。Huber 损失和 Log-cosh 损失可以有效地处理异常值，适用于数据质量较差的情况。

决定系数 (R^2) 是衡量回归模型拟合优度的重要指标，它表示模型解释的变异占总变异的比例。具体来说，R^2 值越接近 1，说明模型对数据的拟合越好，即模型能够解释更多的数据变异性。

计算公式:

R^2 = 1 - (Σ(y - f(x))^2) / (Σ(y - y_mean)^2)

其中：

- y 是真实值
- f(x) 是模型预测值
- y_mean 是真实值的平均值
- Σ 表示求和

解读:

- R^2 的取值范围是 [0, 1]。
- R^2 = 1 表示模型完全拟合数据，预测值与真实值完全一致。
- R^2 = 0 表示模型没有解释任何变异，预测值与真实值完全无关。
- R^2 越接近 1，说明模型对数据的拟合越好，解释的变异越多。

局限性:

- R^2 只能衡量模型解释的变异，不能衡量模型预测的准确性。
- R^2 容易受到数据集规模和特征数量等因素的影响。
- R^2 不能区分过拟合和欠拟合。

案例分析1

1. 导入必要的库。

```
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
```

2. 加载diabetes数据集，并将其转换为 pandas DataFrame。

```
data = load_diabetes()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target
```

3. 分离特征和目标变量。

```
X = df.drop('Target', axis=1)
y = df['Target']
```

4. 划分训练集和测试集。

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5. 训练线性回归模型。

```
model = LinearRegression()
model.fit(X_train, y_train)
```

6. 输出模型的系数和截距。

```
print("Coefficients:", model.coef_)
```

```
print("Intercept:", model.intercept_)
```

输出结果：

```
Coefficients: [  37.90402135 -241.96436231  542.42875852  347.70384391
-931.48884588
  518.06227698  163.41998299  275.31790158  736.1988589    48.67065743]
Intercept: 151.34560453985995
```

7. 生成测试集的预测值。

```
y_pred = model.predict(X_test)
```

8. 计算预测值与实际值之间的 Pearson 相关系数。

```
r = pearsonr(y_test, y_pred)[0]
print("Pearson Correlation Coefficient:", r)
```

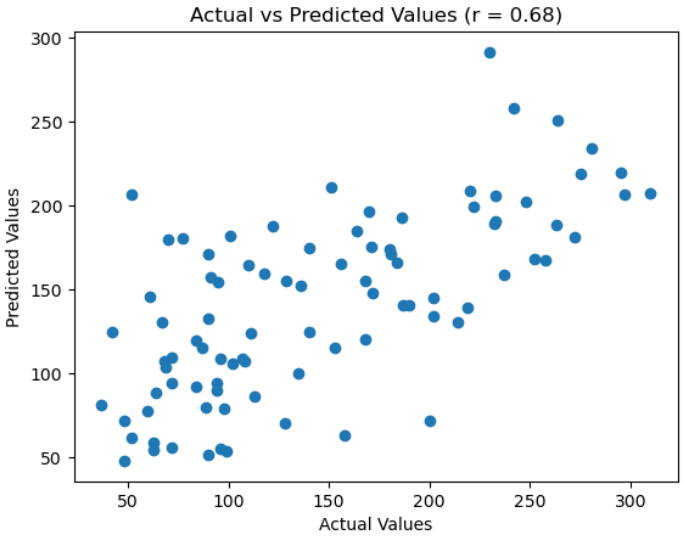
输出结果：

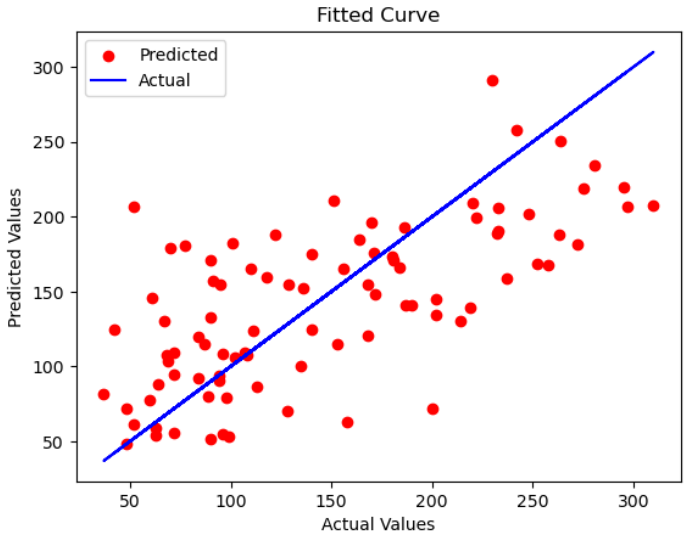
```
Pearson Correlation Coefficient: 0.6779729382980617
```

9. 绘制预测值与实际值的散点图,以及拟合曲线

```
# 绘制预测值与实际值的散点图
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (r = {:.2f})'.format(r))
plt.show()

# 绘制拟合曲线
plt.scatter(y_test, y_pred, color='red', label='Predicted')
plt.plot(y_test, y_test, color='blue', label='Actual')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Fitted Curve')
```

```
plt.legend()
plt.show()
```





10. 计算平均绝对误差 (MAE)、均方误差 (MSE)、均方根误差 (RMSE) 和决定系数 (R^2)，并且储存到一个DataFrame 中并打印。

```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
metrics_df = pd.DataFrame({
    'Metric': ['MAE', 'MSE', 'RMSE', 'R2'],
    'Value': [mae, mse, rmse, r2]
})
print(metrics_df)
```

输出结果：

```
  Metric        Value
0    MAE    42.794095
1    MSE  2900.193628
2   RMSE    53.853446
3     R2     0.452603
```

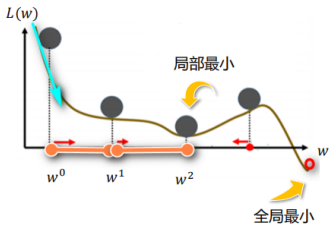
梯度下降法

原理与思考

梯度下降法是优化算法中的一种，其核心思想是沿着损失函数的梯度（即函数变化最快的方向）相反方向迭代地调整参数，从而使得损失函数最小化。

3.2.2 梯度下降法 | 1个参数的情形

假设损失函数 $L(\boldsymbol{w})$ 有1个参数向量 $\boldsymbol{w}$，求 $\boldsymbol{w}^* = arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$



随机选择一个初始 $\boldsymbol{w}^0$，

计算 $\frac{dL}{dw}|_{w=w^0}$，更新 $w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}$

计算 $\frac{dL}{dw}|_{w=w^1}$，更新 $w^2 \leftarrow w^1 - \eta \frac{dL}{dw}|_{w=w^1}$

…… 迭代多次

η 称为学习率，是超参数。

梯度下降法优化过程

已知X增广后的训练集数据 (X,y)，设置学习率η，迭代次数T

损失函数$L(\boldsymbol{w})$，求 $\boldsymbol{w}^* = arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$

梯度下降法优化过程：

随机初始化 $\boldsymbol{w}^0$

重复迭代更新参数，依次令 t =1,2,3,...T：

计算梯度 $\frac{\partial L}{\partial w}|_{w=w^{t-1}}$，

更新参数 $\boldsymbol{w}^t \leftarrow \boldsymbol{w}^{t-1} - \eta \frac{\partial L}{\partial w}|_{w=w^{t-1}}$

直到参数值收敛(或达到设定的迭代次数T)，得最优解$\mathbf{w}^*$。

思考：

1. 梯度怎么算?
2. 算法能否收敛出一个最小值?
3. 学习率怎么设置?
4. 迭代次数怎么限定?

思考1：梯度计算

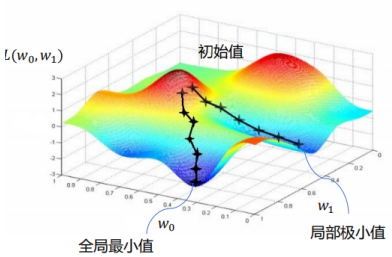
增广矩阵 $\boldsymbol{X} \leftarrow [\mathbf{1}, \boldsymbol{X}]$，1个参数 $\boldsymbol{w} \leftarrow \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}$

$$L(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)})^2 = \frac{1}{\boldsymbol{m}}(\boldsymbol{y} - \boldsymbol{X}\cdot\boldsymbol{w})^T(\boldsymbol{y} - \boldsymbol{X}\cdot\boldsymbol{w})$$

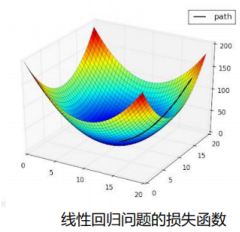
梯度 $\frac{\partial L}{\partial w}$ $\quad \frac{\partial L}{\partial \boldsymbol{w}} = \frac{2}{\boldsymbol{m}}\boldsymbol{X}^T\cdot(\boldsymbol{X}\cdot\boldsymbol{w} - \boldsymbol{y})$ 残差

```
gradients = 2/m * (X.T) @ (X @ w - y)
```

思考2：梯度下降法更新损失函数收敛吗?



若损失函数是非凸函数，存在局部极小值。
初始点不同，结果可能不同.....



线性回归问题的损失函数

线性回归问题，因损失函数L是凸函数，没有局部极小值。

思考3：学习率的设置

- **学习率（η）：是梯度下降法的一个重要超参数，它决定了每次下降的步长。**
  - 如果η太低，算法要经过大量迭代才能**收敛**，很耗时。
  - 如果η太高，可能越过山谷直达另一边，甚至可能比之前的起点还高。这导致算法**发散**!
- **学习率调度**：开始步长较大，然后越来越小，让算法停在全局最小值。确定每次迭代学习率的函数，这种方法称为**学习率调度**(learning schedule)。

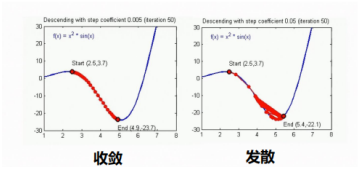


Image by Adam Harley
https://www.cs.ryerson.ca/~aharley/neural-networks/

学习率调度方法

- **一种流行且简单想法：**

  用某因子控制学习率η衰减，每隔几代（t表迭代次数）降低学习率。

  刚开始离目标很远，使用较大的学习率；

  迭代几次后接近目标，就降低学习率。

  如，1/t 衰减：$\eta^t = \frac{\eta}{\sqrt{t+1}}$

- **学习率不能一刀切**

  给不同的参数不同的学习率，如Adagrad方法

迭代次数设置的要点可以概括如下：

1. **迭代次数的影响**:
   - 如果迭代次数太低：算法可能无法达到最优解就停止计算，导致结果不理想。

- 如果迭代次数太高：即使模型已经达到了最优解，继续进行迭代而参数不变会导致时间浪费。

2. **简单做法**:
   - 先设定一个较大的迭代次数上限。
   - 当梯度向量（即目标函数的变化率）的值变得非常小时，也就是其范数低于某个预设的小阈值ε时，中断迭代过程。这个阈值ε通常被称为容差或容忍度。
   - 这样做的原因是当梯度下降到足够小的值时，意味着优化过程已经接近最小值点，进一步迭代不会带来显著改进。

通过这种方式，可以在保证找到近似最优解的同时避免不必要的额外迭代，从而节省计算资源。

三种梯度下降法

1. **Vanilla GD（批量梯度下降）**：
   - 原理：每次迭代时使用整个训练集来计算损失函数关于所有参数的梯度，然后根据这个梯度来更新模型的参数。
   - 特点：由于使用了全部数据，因此计算的梯度是全局最优解的方向，但这种方法需要大量的内存和处理时间，尤其是在处理大规模数据集时。
2. **随机梯度下降法（SGD）**：
   - 原理：每次迭代时从训练集中随机选取一个样本来近似地估计梯度，然后根据这个梯度来更新模型的参数。
   - 特点：这种方法速度快且具有很好的随机性，但由于只使用了一个样本，所以得到的梯度可能不是全局最优解的方向，而是局部或随机的方向。
3. **小批量随机梯度下降法（Mini-batch gradient descent）**：
   - 原理：每次迭代时从训练集中随机选取一小批样本（通常为32、64或128个）来计算梯度，然后根据这些样本的平均梯度来更新模型的参数。
   - 特点：这种方法介于批量梯度下降法和随机梯度下降法之间，它既利用了部分数据的统计特性来引导搜索过程，又保持了较好的收敛速度和稳定性。

SGD代码实现

```
def SGD(X,y,n_epochs=50,n_iter=1000):
  m = X.shape[0] 样本数
  w = np.random.randn(X.shape[1],1) 随机初始化权重向量
  etalist = [] 学习率列表
  losslist = [] 损失列表
  for epoch in range(n_epochs):
```

```
    for i in range(m):
      random_index = np.random.randint(m)
      x_i = X[random_index:random_index + 1] 随机选取1个样本
      y_i = y[random_index:random_index + 1]
      gradients = 2 * x_i.T.dot(x_i.dot(w) - y_i) 计算梯度
      eta = learning_schedule(epoch * m + i) 调节学习率
      w = w - eta * gradients 更新参数

      etalist.append(eta)
      loss = np.sum((X_b.dot(w) - y)**2)/m 计算损失
      losslist.append(loss)
```

1. **初始化参数**：首先定义了一个函数 `SGD(X, y, n_epochs=50, n_iter=1000)`，其中 `X` 是特征矩阵，`y` 是对应的目标变量向量，`n_epochs` 是最大迭代轮数，默认为50，`n_iter` 是每个epoch内的迭代次数，默认为1000。
2. **获取样本数量**：通过 `m = X.shape[0]` 获取数据集中的样本总数。
3. **权重初始化**：使用 `np.random.randn()` 方法对权重 `w` 进行随机初始化，形状与特征的数量一致。
4. **学习率列表和损失列表初始化**：分别创建两个空列表 `etalist` 和 `losslist` 用于存储每一步的学习率和对应的损失值。
5. **开始迭代过程**：
   - 外层循环遍历所有的epochs。
   - 内层循环在每个epoch内执行多次迭代，每次迭代从所有样本中随机选择一个样本更新一次权重。
6. **随机选取样本**：在每次迭代过程中，使用 `np.random.randint(m)` 随机选择一个样本索引 `random_index`。
7. **提取当前样本的特征和目标值**：利用选中的索引从 `X` 和 `y` 中提取出单个样本的特征 `x_i` 和目标值 `y_i`。
8. **计算梯度**：对于线性回归问题，梯度是通过预测值与真实值的差乘以输入特征的逆点积计算的，即 `gradients = 2 * x_i.T.dot(x_i.dot(w) - y_i)`。
9. **调整学习率**：通过调用 `learning_schedule(epoch * m + i)` 动态地调整学习率 `eta`，这通常是为了让学习率随着迭代的进行逐渐减小。
10. **更新权重**：使用当前的梯度和学习率更新权重 `w`，公式为 `w = w - eta * gradients`。
11. **记录学习率和损失值**：将本次迭代使用的学习率添加到 `etalist` 中，并将计算出的损失值添加到 `losslist` 中。
12. **计算总损失**：在每个epoch结束后，计算整个数据集上的平均平方误差作为损失值，并将其添加到 `losslist` 中。

13. **返回结果**：最后，函数返回学习率列表、损失列表以及最终的权重 `w`。

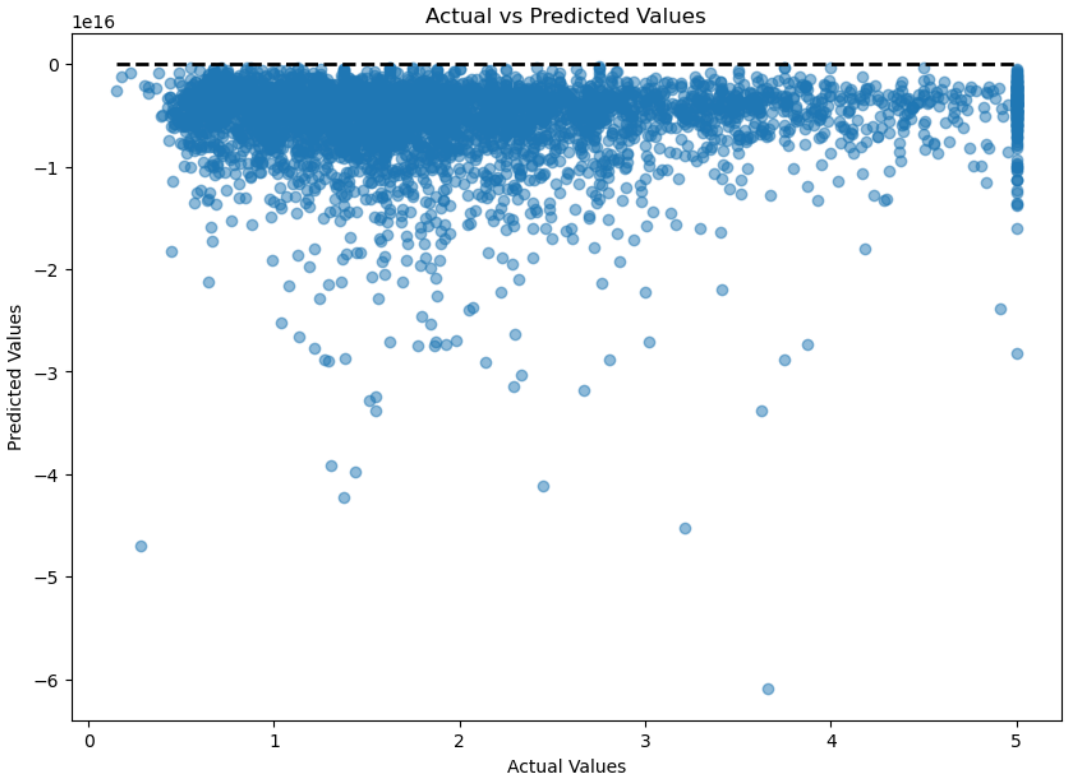
SGD案例

```python
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# 加载数据集
california_housing = fetch_california_housing()
X = california_housing.data
y = california_housing.target

# 分割数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 初始化SGD回归模型
sgd_regressor = SGDRegressor(max_iter=1000, tol=1e-3, eta0=0.1, random_state=42)

# 训练模型
sgd_regressor.fit(X_train, y_train)

# 进行预测
y_pred = sgd_regressor.predict(X_test)

# 计算均方误差和均方根误差
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")

# 绘制散点图
plt.figure(figsize=(10, 7))
plt.scatter(y_test, y_pred, alpha=0.5)  # alpha设置透明度

# 绘制参考线
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)

# 添加标签和标题
```

```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')

# 显示图表
plt.show()
```

输出结果：

```
Mean Squared Error: 4.917250740009454e+31
Root Mean Squared Error: 7012311131153162.0
```



特征缩放

标准化缩放（StandardScaler）：

- **原理**：标准化缩放是将数据转换为标准正态分布的方法。它通过减去均值然后除以标准差来实现数据的零均值化和单位方差化。公式如下所示：

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

其中 μ 是特征的平均值，σ 是特征的标准差。

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

归一化缩放（MinMaxScaler）：

- **原理**：归一化缩放也称为最小最大缩放，它是将原始数据按比例缩放到一个特定区间内的方法。通常情况下，这个区间是 [0,1]。其计算方式为：

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

其中 X_{min} 和 X_{max} 分别表示特征的最低值和最高值。

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)
```

- 标准化缩放适用于需要保持数据集原始分布形状的情况，如某些机器学习算法要求输入的特征具有相同的尺度或分布。
- 归一化缩放则常用于神经网络等模型，因为它们可能对输入值的范围有限制。

其他缩放：

- **Z-Score标准化**：这种方法与标准化缩放类似，但使用的是均值和标准差来进行转换。
- **RobustScaler**：这种缩放器使用的是数据的四分位数来缩放数据，而不是均值和标准差。这可以减少异常值的影响。
- **PowerTransformer**：这是一种更复杂的缩放方法，它可以尝试保留数据的原始分布的同时进行缩放。常见的实现包括Yeo-Johnson和Box-Cox变换。
- **QuantileTransformer**：这种方法可以将数据转换到均匀分布或高斯分布上，而不改变数据的原始分布形状。

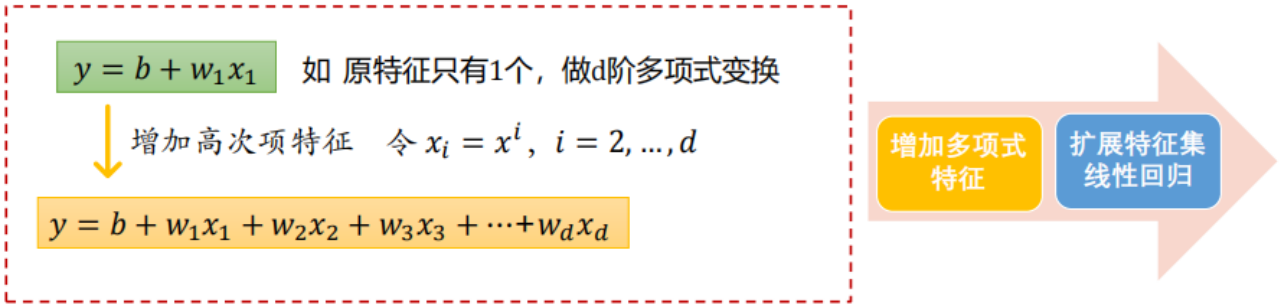
多项式回归

多项式回归原理

多项式回归是一种特殊的线性回归形式，其中自变量和因变量之间的关系被建模为自变量的多项式函数。在多项式回归中，自变量不仅仅是原始数据，而是通过多项式变换生成的特征。

- 可以使用线性模型 $f(x) = w^T x + b$，来拟合非线性数据。
- **一种简单的做法**：向特征集中添加新特征，新特征是每个原特征的幂次方或者原有特征的组合，然后在此扩展特征集上训练一个线性模型，这种技术称为**多项式回归。**

  新特征是由原有特征的多项式变换产生的，称为**多项式特征。**



多项式回归代码

我们可以使用sklearn中的PolynomialFeatures模块来进行多项式回归

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# 生成数据集
np.random.seed(42)  # 设置随机种子以确保结果可重复

# 生成X
X = np.random.uniform(-3, 3, (100, 1))

# 生成y
y = 0.5 * X**2 + 1.0 * X + 2.0 + np.random.normal(0, 1, (100, 1))

# 查看前5个数据点
X[:5], y[:5]

```

```python
# 使用PolynomialFeatures生成多项式特征
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# 创建线性回归模型
model = LinearRegression()
model.fit(X_poly, y)

# 进行预测
y_pred = model.predict(X_poly)

# 可视化结果
plt.scatter(X, y, color='blue', label='Actual data')
plt.scatter(X, y_pred, color='red', label='Predicted data')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Polynomial Regression Fit')
plt.legend()
plt.show()

# 误差评估
mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = mean_squared_error(y, y_pred, squared=False)
r2 = r2_score(y, y_pred)

# 将误差指标存储到DataFrame中
error_metrics = pd.DataFrame({
    'Mean Absolute Error': [mae],
    'Mean Squared Error': [mse],
    'Root Mean Squared Error': [rmse],
    'R^2 Score': [r2]
})

print(error_metrics)
```

多项式回归优化

`scikit-learn` 的 `pipeline` 模块提供了工具，用于构建复杂的数据处理流程，它允许用户将多个数据处理步骤（转换器）和最终的估计器（模型）串联成一个单一的序列。以下是 `pipeline` 模块的主要用途：

1. **序列化处理步骤**：将数据预处理（如特征选择、标准化、多项式特征生成等）和模型训练步骤串联起来，确保数据处理的一致性。
2. **简化代码**：通过管道，可以简化代码结构，避免重复编写数据处理和模型训练的代码。

3. **避免数据泄露**：在模型训练过程中，管道可以确保数据预处理步骤（如标准化）仅在训练数据上拟合，并在后续的预测步骤中使用相同的参数，从而避免数据泄露问题。
4. **易于维护和复用**：管道可以作为单个实体进行保存、加载和参数调优，使得模型维护和参数搜索更加方便。
5. **自动化工作流程**：在交叉验证或网格搜索等过程中，管道可以自动处理数据预处理和模型训练的各个步骤，无需手动干预。

```
# 创建一个管道，包含多项式特征生成和线性回归模型
pipeline = make_pipeline(
    PolynomialFeatures(degree=2, include_bias=False),
    StandardScaler(),  # 标准化，使模型训练更稳定
    LinearRegression()
)

# 使用管道进行训练和预测
pipeline.fit(X, y)
y_pred_pipeline = pipeline.predict(X)
```

我们可以使用交叉验证的方式选出最优的阶数，方法如下：

1. 生成数据集。
2. 使用不同的多项式阶数（从1到20）分别进行拟合。
3. 使用交叉验证（例如，5折交叉验证）来评估每个模型的性能。
4. 选择具有最佳交叉验证分数的多项式阶数。
5. 使用所选阶数进行最终拟合，并进行可视化。

```
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# 设置随机种子以确保结果可重复
np.random.seed(42)

# 生成数据集
X = np.random.uniform(-3, 3, (100, 1))
```

```python
y = 0.5 * X**2 + 1.0 * X + 2.0 + np.random.normal(0, 1, (100, 1))

# 存储不同阶数模型的交叉验证分数
cv_scores = []

# 对1-20阶进行拟合和交叉验证
for degree in range(1, 21):
    # 生成多项式特征
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_poly = poly.fit_transform(X)

    # 创建线性回归模型
    model = LinearRegression()
    # 使用5折交叉验证
    scores = cross_val_score(model, X_poly, y,
scoring='neg_mean_squared_error', cv=5)
    # 记录平均分数
    cv_scores.append(-np.mean(scores))  # 取负值，因为我们希望最小化MSE

# 找到具有最佳交叉验证分数的阶数
optimal_degree = np.argmin(cv_scores) + 1  # 加1是因为索引从0开始

# 使用最佳阶数进行最终拟合
poly_optimal = PolynomialFeatures(degree=optimal_degree, include_bias=False)
X_poly_optimal = poly_optimal.fit_transform(X)
model_optimal = LinearRegression()
model_optimal.fit(X_poly_optimal, y)
y_pred_optimal = model_optimal.predict(X_poly_optimal)

# 可视化结果
plt.scatter(X, y, color='blue', label='Actual data')
plt.scatter(X, y_pred_optimal, color='red', label=f'Predicted data (Degree
{optimal_degree})')
plt.xlabel('X')
plt.ylabel('y')
plt.title(f'Polynomial Regression Fit (Optimal Degree: {optimal_degree})')
plt.legend()
plt.show()

# 返回最佳阶数和交叉验证分数
print(optimal_degree, cv_scores)
```

通过交叉验证，我们发现最佳的多项式阶数是2。以下是不同阶数模型的交叉验证分数（均方误差，MSE）：

- 阶数 1: MSE = 3.055

- 阶数 2: MSE = 0.850
- 阶数 3: MSE = 0.875
- 阶数 4: MSE = 0.904
- 阶数 5: MSE = 0.925
- 阶数 6: MSE = 0.948
- 阶数 7: MSE = 0.967
- 阶数 8: MSE = 0.987
- 阶数 9: MSE = 1.062
- 阶数 10: MSE = 1.072
- 阶数 11: MSE = 0.998
- 阶数 12: MSE = 1.015
- 阶数 13: MSE = 1.055
- 阶数 14: MSE = 1.134
- 阶数 15: MSE = 1.364
- 阶数 16: MSE = 1.401
- 阶数 17: MSE = 2.319
- 阶数 18: MSE = 7.978
- 阶数 19: MSE = 17.442
- 阶数 20: MSE = 25.024

随着阶数的增加，模型的复杂度增加，导致过拟合，这反映在较高的MSE分数上。因此，选择阶数为2是合理的。

过拟合与欠拟合

过拟合（Overfitting）

定义：过拟合是指模型对训练数据学习得太好，以至于它捕捉到了数据中的噪声和随机波动，而不是潜在的趋势或模式。这导致模型在训练数据上表现非常好，但在未见过的数据（如测试数据）上表现不佳。

特征：

- 训练误差低，测试误差高。
- 模型过于复杂，有大量的参数。

- 模型对训练数据中的每一个细节都学习得很好，包括噪声。

例子：假设你在训练一个用于识别猫的模型，如果模型不仅学习了猫的一般特征，还学习了训练集中特定猫的个体特征（如特定的斑点或颜色），那么它可能会在新的、不同的猫的图片上表现不佳。

欠拟合（Underfitting）

定义：欠拟合是指模型过于简单，未能捕捉到数据的基本结构或模式。这意味着模型既在训练数据上表现不佳，也在测试数据上表现不佳。

特征：

- 训练误差高，测试误差也高。
- 模型过于简单，参数太少。
- 模型未能捕捉到数据的主要趋势。

例子：继续上面的猫识别例子，如果模型只学习到了“猫是四足动物”这样的一般特征，而没有学习到猫的独特视觉特征，那么它可能无法准确地区分猫和其他四足动物。

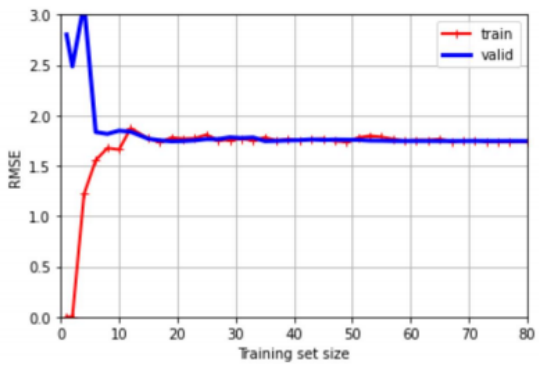
区别

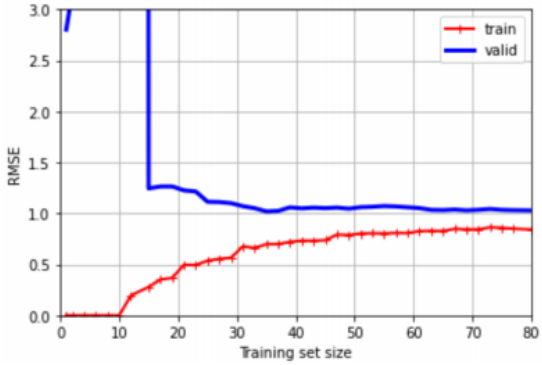
- **误差**：过拟合通常在训练集上误差很低，但在测试集上误差很高；而欠拟合在训练集和测试集上都有较高的误差。
- **模型复杂度**：过拟合通常发生在模型复杂度高的情形下，而欠拟合通常发生在模型复杂度低的情形下。
- **泛化能力**：过拟合的模型泛化能力差，无法很好地应用于新数据；欠拟合的模型同样泛化能力差，但原因在于模型过于简单，没有捕捉到足够的模式。

为了解决过拟合和欠拟合问题，可以采取以下措施：

- **过拟合**：简化模型、减少特征数量、增加训练数据、使用正则化方法等。
- **欠拟合**：增加模型复杂度、增加特征数量、减少正则化等

左图是线性回归的学习曲线，右图是10阶多项式回归的学习曲线。





- 两条曲线都到达了平稳状态，很接近并且误差很高。这是典型欠拟合现象。
- 若模型欠拟合训练数据，添加更多训练实例也无济于事。需要使用更复杂的模型或提供更好的特征。

- 曲线间存在间隙，而训练误差很低，说明训练数据上性能远好于验证数据上的性能，这是过拟合的典型标志。
- 改善过拟合模型的一种方法是向其提供更多的训练数据，直到验证误差达到训练误差为止。

误差与正则化

在统计学和机器学习中，偏差（Bias）、方差（Variance）、残差（Residual）和误差（Error）是评估模型性能的重要概念。下面是它们之间的区别：

偏差（Bias）

定义：偏差是指模型在多次训练过程中预测结果的期望与真实值之间的差距。它度量了模型在训练数据上的拟合程度，即模型对训练数据的准确度。

特点：

- 偏差高意味着模型可能过于简单，未能捕捉到数据中的关键关系（即欠拟合）。
- 偏差低意味着模型能够较好地捕捉到数据中的关系。

方差（Variance）

定义：方差是指模型在多次训练过程中，对于给定输入数据的预测结果的变化性。它度量了模型对于不同训练集的敏感程度。

特点：

- 方差高意味着模型对训练数据中的随机波动非常敏感，可能导致过拟合。
- 方差低意味着模型对于不同的训练集有稳定的预测结果。

残差（Residual）

定义：残差是指单个观测值与模型预测值之间的差异。它是实际数据点与回归线（或模型预测）之间的垂直距离。

特点：

- 残差用于衡量模型对单个数据点的拟合程度。
- 残差分析可以帮助识别数据中的异常值或模型未能解释的变异。

误差（Error）

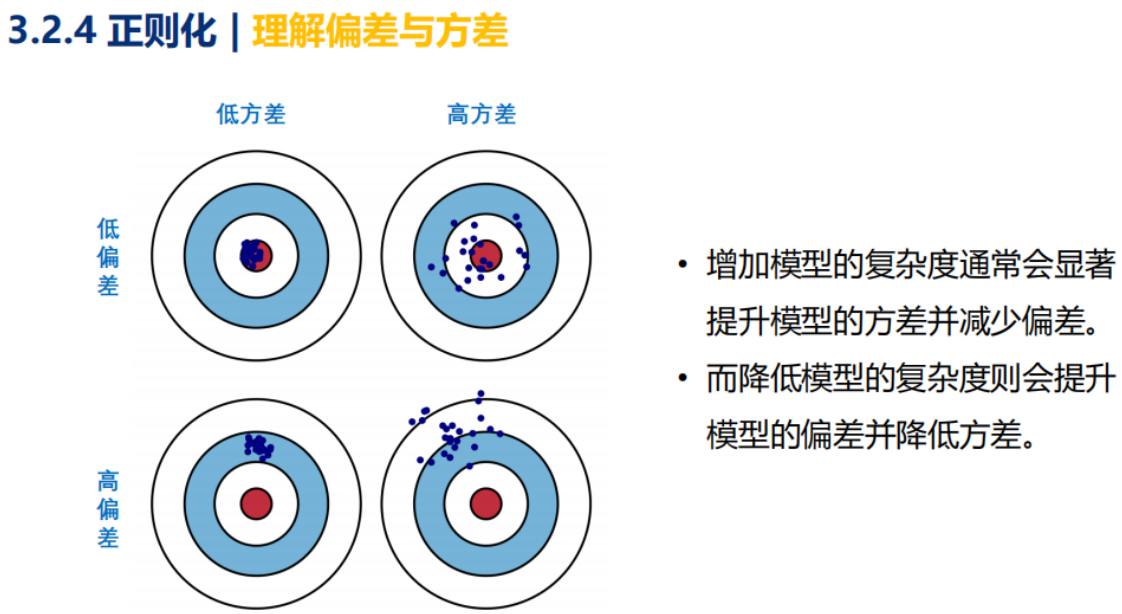
定义：误差是指模型预测值与真实值之间的差异。在统计和机器学习中，通常用误差来泛指任何类型的预测不准确。

特点：

- 误差可以分为两类：随机误差和系统误差。随机误差是不可预测的，而系统误差是可预测的。
- 总误差可以分解为偏差、方差和不可减少的随机噪声。

区别

- **偏差**关注的是模型预测结果的系统性偏差，即模型是否能够捕捉到数据的基本趋势。
- **方差**关注的是模型预测结果的不稳定性，即模型是否对训练数据中的随机波动过于敏感。
- **残差**是针对单个数据点的预测误差，用于衡量模型对特定数据点的拟合程度。
- **误差**是一个更广泛的概念，它包括了所有导致预测值与真实值之间差异的因素，包括偏差、方差和随机噪声。



正则化

正则化是一种在机器学习中用于防止模型过拟合的技术。过拟合是指模型在训练数据上表现得非常好，但在未见过的新数据上表现不佳，因为它学习了训练数据中的噪声和细节，而没有捕捉到数据的真实分布。正则化通过在损失函数中添加一个惩罚项来实现，这个惩罚项会根据模型的复杂度对模型进行惩罚。

L1 正则化（Lasso）

L1 正则化在损失函数中添加了模型权重绝对值之和的惩罚。数学上可以表示为：

$$L1 = \lambda \sum_{i=1}^{n} |w_i|$$

其中 λ 是正则化参数，w_i 是模型参数。

L1 正则化的特点：

- 可以产生稀疏的权重矩阵，即某些权重会变为零，从而实现特征选择。

L2 正则化（Ridge）

L2 正则化在损失函数中添加了模型权重平方和的惩罚。数学上可以表示为：

$$L2 = \lambda \sum_{i=1}^{n} w_i^2$$

L2 正则化的特点：

- 不会产生稀疏权重矩阵，但会使得权重值趋向于分散，降低模型复杂度。

Elastic Net

Elastic Net 是 L1 和 L2 正则化的组合，它结合了两者的优点：

$$Elastic\ Net = \alpha\lambda_1 \sum_{i=1}^{n} |w_i| + \alpha\lambda_2 \sum_{i=1}^{n} w_i^2$$

其中 α 是混合参数，λ_1 和 λ_2 分别是 L1 和 L2 正则化的参数。

Dropout

Dropout 是一种针对神经网络特别有效的正则化技术。在训练过程中，dropout 随机地"丢弃"（即设置为零）网络中的一部分神经元，这样可以防止网络过度依赖某些特定的神经元。

早期停止（Early Stopping）

早期停止是一种简单的正则化方法，它通过在验证误差开始增加时停止训练来防止过拟合。这种方法不直接修改损失函数，而是通过控制训练过程来减少模型的复杂度。

正则化技术的选择取决于具体的应用场景和数据特性。通过适当选择正则化方法，可以有效地提高模型的泛化能力，使其在未见过的数据上也能做出更好的预测。

正则化回归

岭回归

岭回归，也称为Tikhonov正则化（L2正则化），是一种用于回归分析的线性模型，它通过在损失函数中引入L2正则化项来解决普通最小二乘法可能出现的过拟合问题。

岭回归（也称Tihkonov正则化）是线性回归的正则化版本，它将 $\alpha \sum_{i=1}^{m} w_i^2$ 作为**正则化项**（即惩罚项）添加到线性回归**损失函数**中：

$$L(\boldsymbol{w}) = \frac{1}{m}\sum_{j=1}^{m}(\boldsymbol{w}^T\boldsymbol{x}^{(j)} - y^{(j)})^2 + \alpha \sum_{i=1}^{m} w_i^2$$

m是样本数目

矩阵表示， $L(\boldsymbol{w}) = \frac{1}{m}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^{\ 2} + \alpha\|\boldsymbol{w}\|_2^{\ 2}$

其中$\|\boldsymbol{w}\|_2$表示权重向量的L2范数，这种正则化称为**岭正则化**，又称**L2正则化**。

- 超参数α控制对模型的正则化程度。α越大，正则强度越大。若α=0，则岭回归为线性回归；若α非常大，则所有权重最终都非常接近于零，结果是一条经过数据均值的平线。
- 加入正则化项的损失函数，迫使学习算法不仅拟合数据，还要使得模型权重尽可能小。

注意： 在执行岭回归之前要缩放数据。大多数正则化模型都需如此。

案例：

1.产生带白噪声一个线性数据集，含20个点。X有1个特征, 特征值是取自 [-3,3]的随机数， y=1+ 0.5 * X + np.random.randn(20, 1) / 1.5 。

2.用岭正则化的纯线性回归，来拟合这个数据集，正则化强度⍁分别取0, 10,100。从[-3,3]等间距取1000数据点构成 X_new，用上述模型预测 输出，画出拟合结果。

3.用岭正则多项式回归（degree=10)，来拟合这个数据集，⊠分别取 0, 10－3 , 1。 从[-3,3]等间距取1000数据点构成 X_new，用上述模型预测输出，画出拟合结果。

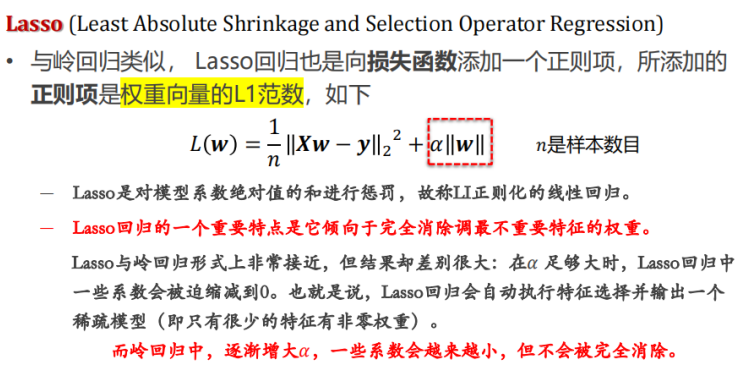
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures

# 生成带白噪声的线性数据集
np.random.seed(42)
X = np.random.uniform(-3, 3, (20, 1))
y = 1 + 0.5 * X + np.random.randn(20, 1) / 1.5

# 使用岭正则化的线性回归拟合数据集
alphas = [0, 10, 100]
X_new = np.linspace(-3, 3, 1000).reshape(-1, 1)

plt.figure(figsize=(12, 8))

# 绘制原始数据点
plt.scatter(X, y, color='blue', label='Data points')

# 对于每个alpha值，拟合模型并绘制结果
for alpha in alphas:
    model = Ridge(alpha=alpha)
    model.fit(X, y)
    y_new = model.predict(X_new)
    plt.plot(X_new, y_new, label=f'Ridge with alpha={alpha}')

plt.xlabel('X')
plt.ylabel('y')
plt.title('Ridge Regularized Linear Regression')
plt.legend()
plt.show()

# 使用岭正则多项式回归拟合数据集
alphas_poly = [0, 10**-3, 1]
poly = PolynomialFeatures(degree=10)

plt.figure(figsize=(12, 8))

# 绘制原始数据点
plt.scatter(X, y, color='blue', label='Data points')

# 对于每个alpha值，拟合模型并绘制结果
for alpha in alphas_poly:
```

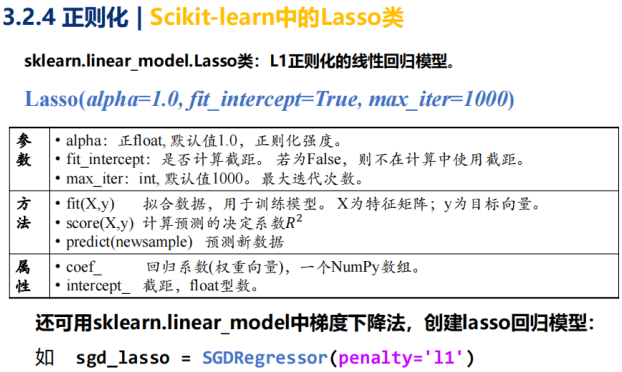
```
    model_poly_pipeline = make_pipeline(PolynomialFeatures(degree=10), Ridge(alpha=alpha))
    model_poly_pipeline.fit(X, y)
    y_new_poly_pipeline = model_poly_pipeline.predict(X_new)
    plt.plot(X_new, y_new_poly_pipeline, label=f'Polynomial Ridge with alpha={alpha}')

plt.xlabel('X')
plt.ylabel('y')
plt.title('Ridge Regularized Polynomial Regression (Degree=10)')
plt.legend()
plt.show()
```

Lasso回归

Lasso回归（Least Absolute Shrinkage and Selection Operator）是一种用于估计线性回归模型参数的惩罚性学习方法。它的基本原理是在普通的最小二乘法回归的基础上加入L1正则化项（即系数的绝对值之和），以此来提高模型的预测准确性和解释性，尤其是在处理具有多重共线性特征的数据时。

多重共线性是指在统计学中，两个或多个自变量之间存在高度相关的情况。当模型中的自变量不是完全独立同分布时，就会出现多重共线性问题。具体来说，当一个或多个自变量的取值可以由其他自变量的线性组合得到时，就认为存在多重共线性。





ElasticNet

ElasticNet回归是线性回归的一种正则化方法，结合了岭回归（Ridge regression）和套索回归（Lasso regression）的特点，旨在解决这两种方法在某些情况下的不足。

原理：

- **正则化**：ElasticNet通过添加正则化术语到普通最小二乘法的成本函数中来防止过度拟合。这有助于保持模型参数较小，减少模型复杂度。

- **结合Ridge和Lasso**：ElasticNet是Ridge回归和Lasso回归的结合。Ridge回归使用L2正则化，而Lasso回归使用L1正则化。ElasticNet同时使用L1和L2正则化，可以通过一个参数调整两种正则化的相对影响力。
- **团体效应**：ElasticNet的一个优点是它可以捕获Ridge回归没有的团队效果(teamwork effect)。这意味着它倾向于将相关的特征捆绑在一起，而不是单独挑选它们。
- **数值稳定**：相对于Lasso，ElasticNet在复数共线性(feature correlation)的情况下更稳定，并且在特征数量多于观察数量的情况下仍然有效。

适用场景：

- 当特征数量非常大，并且相互之间有较强的相关性时，ElasticNet通常表现很好。
- 如果目标是同时执行特征选择和正则化，那么ElasticNet也是一个很好的选择，因为其L1正则化可以帮助剔除不重要的特征。
- 在数据量较大，且特征较多，但样本较少的情况下，ElasticNet能够更好地泛化，避免过拟合。

弹性网络介于岭回归和Lasso之间。其**正则项**是L1正则和L2正则的简单混合。

$$\textbf{损失函数：}\ L(\boldsymbol{w}) = \frac{1}{n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^{\,2} + r\alpha\|\boldsymbol{w}\| + \frac{1-r}{2}\alpha\|\boldsymbol{w}\|_2^{\,2}$$

其中，参数r控制混合比例。当r =0时，弹性网络等效于岭回归；
当r =1时，弹性网络等效于Lasso回归。

sklearn.linear_model.ElasticNet类

ElasticNet (*alpha=1.0, l1_ratio=0.5, fit_intercept=True, random_state=None*)

参数	• *alpha*：正则化强度，默认值1.0。增大alpha会使w更趋向于0，模型更简单。 • *fit_intercept*：是否计算截距。若为False，则计算中不使用截距(即数据应居中)。 • *l1_ratio*：混合比参数，取值[0,1]。取0时是L2正则，取1时是L1正则。
方法	• fit(X,y)　训练模型。X为特征矩阵；y为目标向量 • score(X,y) 计算预测的决定系数。 • predict(newsample)　预测新数据
属性	• coef_　回归系数(权重向量)，一个NumPy数组 • intercept_　截距，float型。

三种回归区别

Lasso回归（L1正则化）

适用场景：

- **特征选择**：当数据集的特征数量很多，且只有少数特征对目标变量有显著影响时，Lasso回归通过将一些系数压缩到零来实现特征选择。
- **稀疏数据**：适用于特征之间存在多重共线性，且希望模型解释性强的场景。

- **高维数据**：在特征数量远大于样本数量的情况下，Lasso回归能够有效地筛选出重要的特征。

Ridge回归（L2正则化）

适用场景：

- **多重共线性**：当数据特征之间存在多重共线性时，Ridge回归通过惩罚系数的平方和来减少共线性问题，使得模型更稳定。
- **特征数量多**：适用于特征数量多，但每个特征都对目标变量有一定影响的场景。
- **不需要特征选择**：Ridge回归不会将任何特征的系数压缩到零，因此当所有特征都重要时，Ridge回归是一个更好的选择。

Elastic Net回归（L1和L2正则化的组合）

适用场景：

- **特征选择与共线性**：当数据集同时具有大量特征和多重共线性问题时，Elastic Net结合了Lasso和Ridge的优点，既可以进行特征选择，又可以处理共线性问题。
- **混合类型的数据**：当数据中既有大量不重要的特征，又有一些相互关联的特征时，Elastic Net通过调整L1和L2正则化的比例来适应这种情况。
- **灵活性**：Elastic Net比单独使用Lasso或Ridge更加灵活，因为它可以通过调整混合参数（通常记为\rho$\rho$）来控制L1和L2正则化的相对重要性。

总结

- **Lasso**：适用于特征选择和稀疏数据。
- **Ridge**：适用于处理多重共线性问题，不需要特征选择。
- **Elastic Net**：适用于同时需要特征选择和处理多重共线性问题的场景，提供了比Lasso和Ridge更多的灵活性。

对比各种回归案例

要求：

(1）对加州房价数据集进行线性回归分析，输出训练得分和测试得分。

(2) 10阶多项式特征扩展，进行多项式回归分析，输出训练得分和测试得分。

计算pearson相关系数，绘制预测值与实际值的散点图。

(3) 在扩展后数据集上做岭回归正则化，⊠分别取10,1,0.1,0.001,0.00001，

输出训练得分和测试得分。画出各模型的回归系数散点图。

(4) 在扩展后数据集上做Lasso正则化，⊠分别取0.01,0.001,0.00001，最

大迭代次数100000，输出训练得分和测试得分。画出各模型的回归系

数散点图。

(5)在扩展后数据集上做ElasticNet正则化，⍁分别取0.01,0.001,0.00001，

最大迭代次数100000，l1_ratio=0.8，输出训练得分和测试得分。

导入数据：

```
from sklearn.datasets import fetch_california_housing
house = fetch_california_housing()
print(house.DESCR)
```

线性回归：

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X, y = house.data, house.target
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,
random_state=42)
lr = LinearRegression().fit(X_train,y_train)
print("训练集 R-score:{:.2f}".format(lr.score(X_train,y_train)))
print("测试集 R-score:{:.2f}".format(lr.score(X_test,y_test)))
```

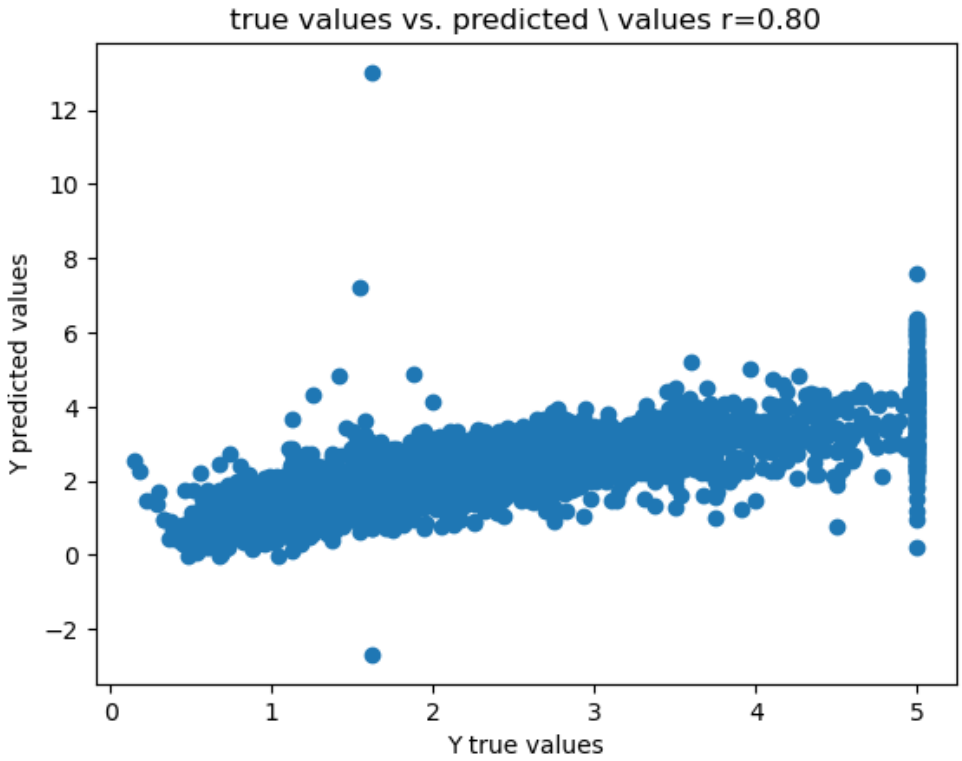
多项式回归:

```
from sklearn.preprocessing import MinMaxScaler,PolynomialFeatures
import numpy as np
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
poly = PolynomialFeatures(degree=2,include_bias=False)
X_trained = poly.fit_transform(X_train_scaled)
X_tested = poly.transform(X_test_scaled)
polylr = LinearRegression().fit(X_trained,y_train)
print("训练集得分:{:.2f}".format(polylr.score(X_trained,y_train)))
print("测试集得分:{:.2f}".format(polylr.score(X_tested,y_test)))
print("模型用到的特征数目:{}".format(np.sum(polylr.coef_!= 0)))
```

计算pearson相关系数，绘制预测值与实际值的散点图:

```
import matplotlib.pyplot as plt
```

```
from scipy.stats import pearsonr
y_pred = polylr.predict(X_tested)
plt.scatter(y_test,y_pred)
plt.xlabel('Y true values')
plt.ylabel('Y predicted values')
pearson = pearsonr(y_test,y_pred)[0]
plt.title('true values vs. predicted \ values r={:.2f}'.format(pearson))
```



岭回归代码：

```
from sklearn.linear_model import Ridge
ridge_coef=[]
for alpha in [10,1,0.1,0.001,0.00001]:
  ridge=Ridge(alpha).fit(X_trained,y_train)
  print('alpha={:}'.format(alpha))
  print("训练集得分:{:.2f}".format(ridge.score(X_trained,y_train)))
  print("测试集得分:{:.2f}".format(ridge.score(X_tested,y_test)))
  print("模型用到的特征数目:{}".format(np.sum(ridge.coef_!= 0)))
  ridge_coef.append(ridge.coef_)
```

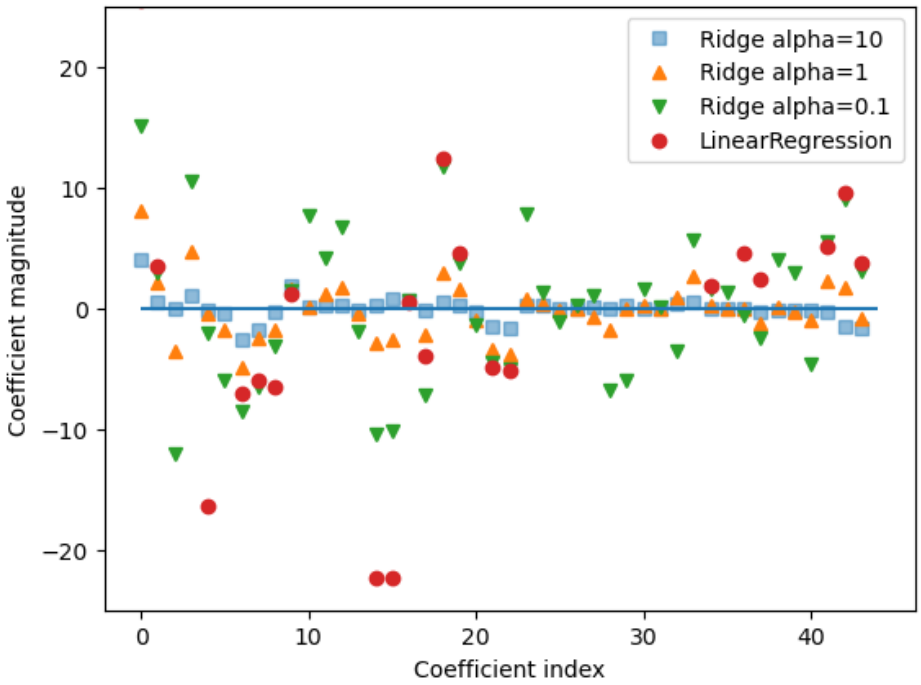
输出：

```
alpha=10
训练集得分:0.61
测试集得分:0.59
```

```
模型用到的特征数目:44
alpha=1
训练集得分:0.63
测试集得分:0.60
模型用到的特征数目:44
alpha=0.1
训练集得分:0.65
测试集得分:0.62
模型用到的特征数目:44
alpha=0.001
训练集得分:0.67
测试集得分:0.62
模型用到的特征数目:44
alpha=1e-05
训练集得分:0.68
测试集得分:0.64
模型用到的特征数目:44
```

绘制回归系数:

```
import matplotlib.pyplot as plt
plt.plot(ridge_coef[0],'s',label="Ridge alpha=10",alpha=0.5)
plt.plot(ridge_coef[1],'^',label="Ridge alpha=1")
plt.plot(ridge_coef[2],'v',label="Ridge alpha=0.1")
plt.plot(polylr.coef_,'o',label="LinearRegression")
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.hlines(0,0len(polylr.coef_))
plt.ylim(-25,25)
plt.legend(loc='best')
```



· Ridge正则化，增大alpha，会使得回归系数更加趋向于0，从而降低训练集的性能。

· 减少alpha，可让系数受到的限制更小。对于非常小的alpha值，系数几乎没有受到 限制，得到一个与多项式回归类似的模型。

Lasso回归代码：

```
from sklearn.linear_model import Lasso
lasso_coef=[]
for alpha in [0.01,0.001,0.00001]:
  lasso=Lasso(alpha=alpha,max_iter=100000).fit(X_trained,y_train)
  print('alpha={:}'.format(alpha))
  print("训练集得分:{:.2f}".format(lasso.score(X_trained,y_train)))
  print("测试集得分:{:.2f}".format(lasso.score(X_tested,y_test)))
  print("模型用到的特征数目:{}".format(np.sum(lasso.coef_!= 0)))
  lasso_coef.append(lasso.coef_)
```

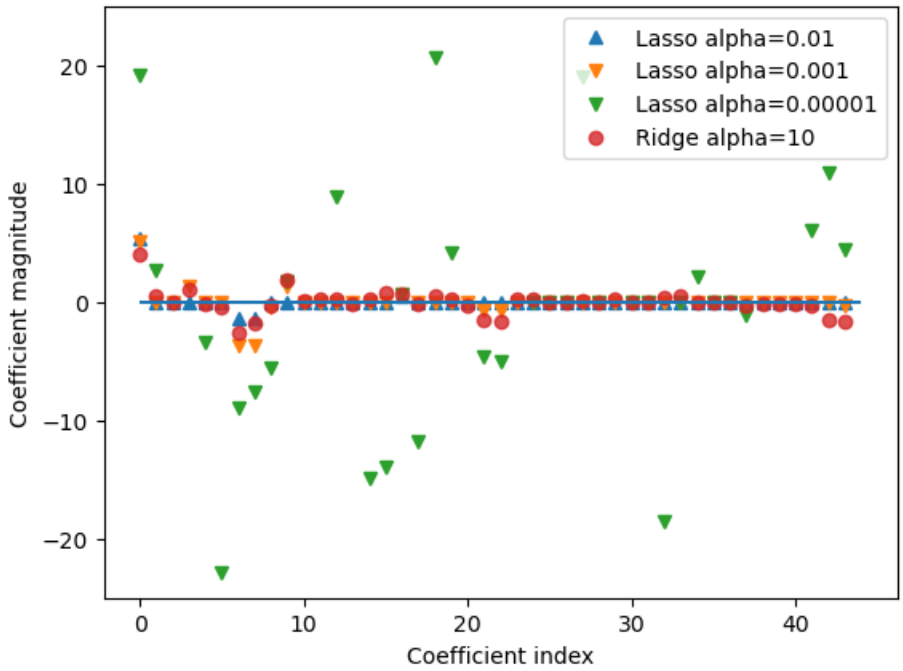
输出：

```
alpha=0.01
训练集得分:0.55
测试集得分:0.54
模型用到的特征数目:4
alpha=0.001
训练集得分:0.61
测试集得分:0.59
模型用到的特征数目:10
```

```
alpha=1e-05
训练集得分:0.66
测试集得分:0.62
模型用到的特征数目:35
```

绘制回归系数：

```
plt.plot(lasso_coef[0],'^',label="Lasso alpha=0.01")
plt.plot(lasso_coef[1],'v',label="Lasso alpha=0.001")
plt.plot(lasso_coef[2],'v',label="Lasso alpha=0.00001")
plt.plot(ridge_coef[0],'o',label="Ridge alpha=10",alpha=0.8)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.hlines(0,0,len(polylr.coef_))
plt.ylim(-25,25)
plt.legend(loc='best')
```



使用Elasticnet进行回归：

```
from sklearn.linear_model import ElasticNet
for alpha in [0.01,0.001,0.00001]:
  elastic_net = ElasticNet(alpha=alpha,l1_ratio=0.8,random_state=3,
  max_iter=100000).fit(X_trained,y_train)
  print('alpha={:}'.format(alpha))
  print("训练集得分:{:.2f}".format(elastic_net.score(X_trained,y_train)))
  print("测试集得分:{:.2f}".format(elastic_net.score(X_tested,y_test)))
  print("模型用到的特征数目:{}".format(np.sum(elastic_net.coef_!= 0)))
```

输出：

```
alpha=0.01
训练集得分:0.55
测试集得分:0.53
模型用到的特征数目:8
alpha=0.001
训练集得分:0.61
测试集得分:0.59
模型用到的特征数目:11
alpha=1e-05
训练集得分:0.65
测试集得分:0.63
模型用到的特征数目:38
```

🌟 alpha 参数控制着正则化的强度，它影响着模型复杂度和泛化能力。

l1_ratio 参数控制着 L1 正则化和 L2 正则化的比例。

· 通常，有正则化总比没有好。大多数情况下，应避免使用纯线性回归。

· 岭回归是不错的默认选择。

· 若实际用到的特征很可能只有少数几个，就应倾向于Lasso或弹性网络，因为它们会将无用特征的权重降为零。

· 一般而言，弹性网络优于Lasso回归，因为当特征数量超过训练样本实例数量，或者几个特征强相关时，Lasso回归表现可能很不稳定