

第5章 线性回归

卿来云

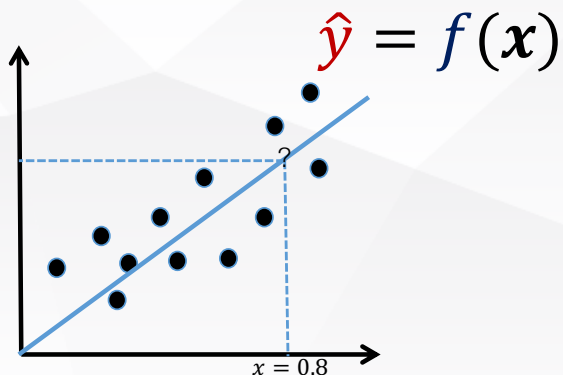
lyqing@ucas.ac.cn

Outline

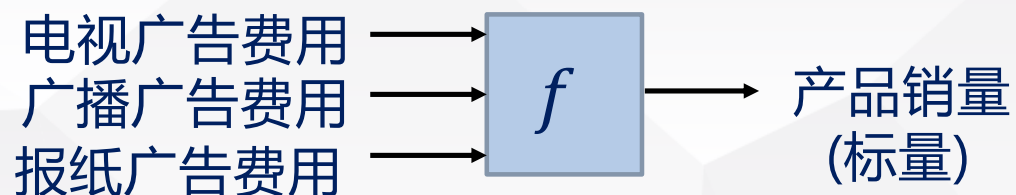
- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
- 回归任务的模型性能评价
- 线性回归案例分析

回归

- 监督学习中，给定训练数据 $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ ，其中 N 为训练样本数目， i 为样本索引， \mathbf{x}_i 为第 i 个样本的输入特征， y_i 为对应的输出/响应，
- 当 $y_i \in \mathbb{R}$ 时，该监督学习任务为一个回归任务。根据训练样本 \mathcal{D} ，学习一个从输入 \mathbf{x} 到输出 y 的映射 f 。
- 测试：对新的测试数据 \mathbf{x} ，用学习到的映射 f 对其进行预测： $\hat{y} = f(\mathbf{x})$



例：产品销量预测



训练数据:

输入

电视广告费用	广播广告费用	报纸广告费用	产品销量
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

输出

$$\mathbf{x} = (x_1, x_2, x_3)$$

$$\mathbf{y} = f(\mathbf{x}) = f(x_1, x_2, x_3)$$

➤ 线性回归

■ 机器学习的三要素

- 1. 函数集合 $\{f_1, f_2, \dots\}$
- 2. 目标函数 $J(f)$ ：函数的好坏
- 3. 优化算法：找到最佳函数

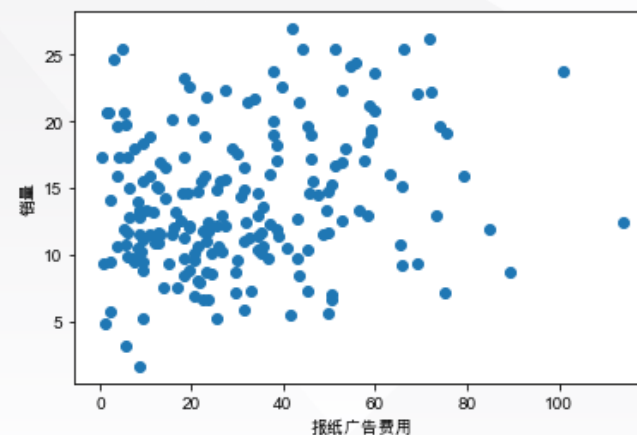
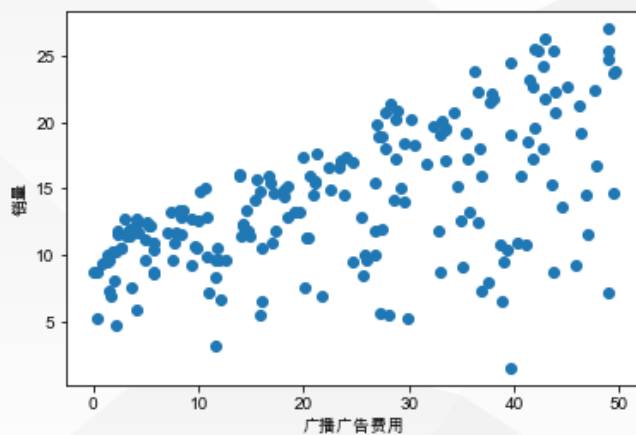
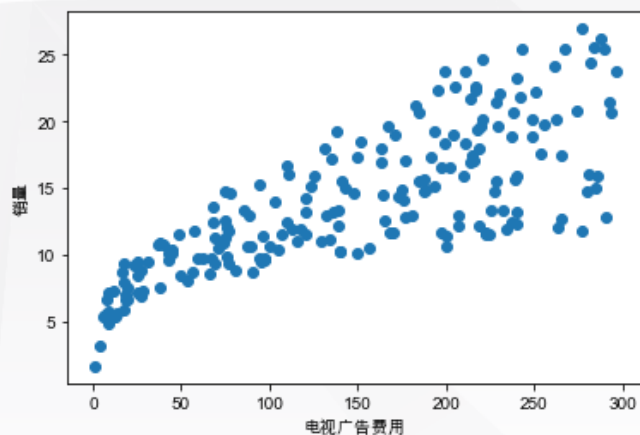
■ 线性回归：函数集合为输入特征的线性组合，即假设输出 y 与输入 x 之间的关系为线性关系

$$\hat{y} = f(x) = \underset{\text{截距项}}{w_0} + \sum_{j=1}^D \underset{\text{第}j\text{维特征的权重/回归系数}}{w_j} x_j = [w_0 \quad w_1 \quad \dots \quad w_D] \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_D \end{bmatrix} = \mathbf{w}^T \mathbf{x}$$

- D : 特征维数, j 为特征索引
- $\mathbf{x} = (1, x_1, \dots, x_D)^T$: 在 D 维特征的基础上, 增加一个常数项1 (用于表示截距项)

数据分析

■ 可通过散点图/相关系数初步检查目标与特征之间是否符合线性关系



产品销量 (sales) 与电视广告费用 (TV) 以及广播广告费用 (radio) 线性相关程度较高，相关系数大于0.5；
产品销量 (sales) 与报纸广告费用 (newspaper) 线性相关程度不高（相关系数小于0.3 ）。

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
- 回归任务的模型性能评价
- 线性回归案例分析

➤ 线性回归

■ 机器学习的三要素

- 1. 函数集合 $\{f_1, f_2, \dots\}$
- 2. 目标函数 $J(f)$: 函数的好坏
- 3. 优化算法 : 找到最佳函数

■ 回归任务目标函数 :

$$J(f, \lambda) = \sum_{i=1}^N L(\hat{y}_i, y_i) + \lambda R(f)$$

损失函数 正则项

➤ 损失函数

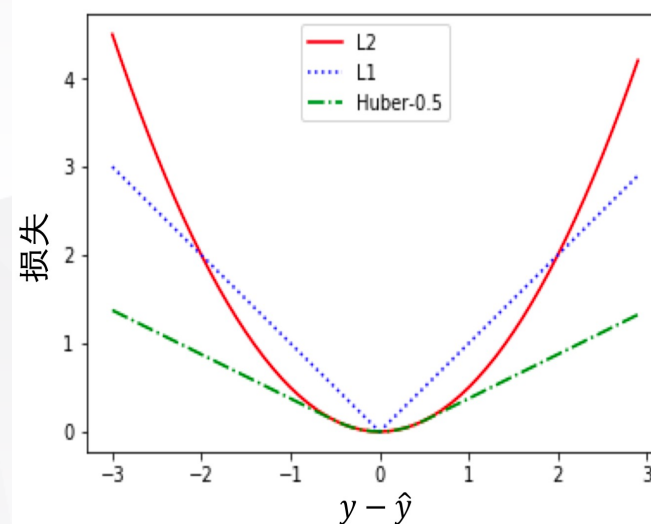
- 对回归问题，最常用的损失函数为**L2损失**，即预测残差的平方：

$$L(\hat{y}, y) = (\hat{y} - y)^2 = r^2$$

其中 预测残差 $r = y - \hat{y}$

- 训练集上所有训练数据的预测残差的平方和记为 (Residual Sum of Squares , RSS)

$$\text{RSS}(f) = \sum_{i=1}^N r_i^2 = \sum_{i=1}^N (\hat{y}_i - y_i)^2$$



➤ L2损失函数的概率解释

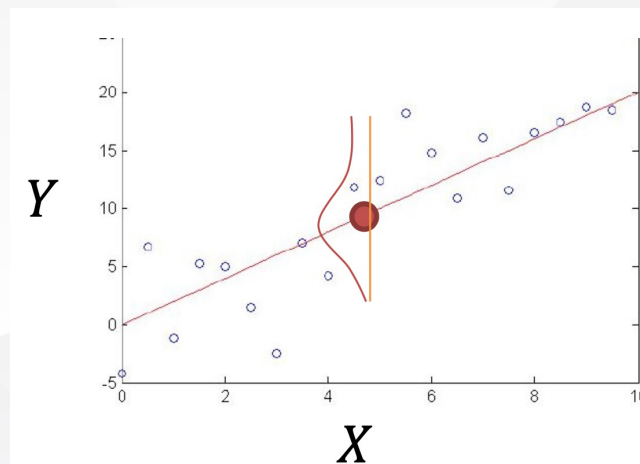
- 最小L2损失函数等价于高斯白噪声下的极大似然。
- 在回归任务中，令模型预测值和真实值之间的差异为噪声 ε 。假设噪声 ε_i 相互独立且与输入无关，且 ε_i 的分布为0均值的正态分布，即

$\varepsilon_i \sim N(0, \sigma^2)$ ，则

- $y_i = f(x_i) + \varepsilon_i$

- $y_i | x_i \sim N(f(x_i), \sigma^2)$

- 所以
$$p(y_i | x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - f(x_i))^2}{2\sigma^2}\right)$$



注意：在给定 x_i 的情况下， $f(x_i)$ 为常数

➤ L2损失函数的概率解释

■ 单个数据点的概率密度函数为：

$$p(y_i|\mathbf{x}_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}\right)$$

■ 则log似然函数为

$$\begin{aligned}\ell(f) = \ln p(\mathcal{D}) &= \sum_{i=1}^N \ln p(y_i|\mathbf{x}_i) \\ &= \sum_{i=1}^N \ln \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}\right) \right] \\ &= -\frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}\end{aligned}$$

➤ L2损失函数的概率解释

■ log似然

$$\ell(f) = -\frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(x_i))^2}{2\sigma^2}$$

■ 取最大值时， $-\sum_{i=1}^N \frac{(y_i - f(x_i))^2}{2\sigma^2}$ 取最大值，从而 $\sum_{i=1}^N \frac{(y_i - f(x_i))^2}{2\sigma^2}$ 取最小值。

■ $\sum_{i=1}^N (y_i - f(x_i))^2 = \text{RSS}(f)$ 刚好是残差平方和/训练集上的L2损失之和

所以：极大似然估计等价于最小二乘（最小L2损失函数）。

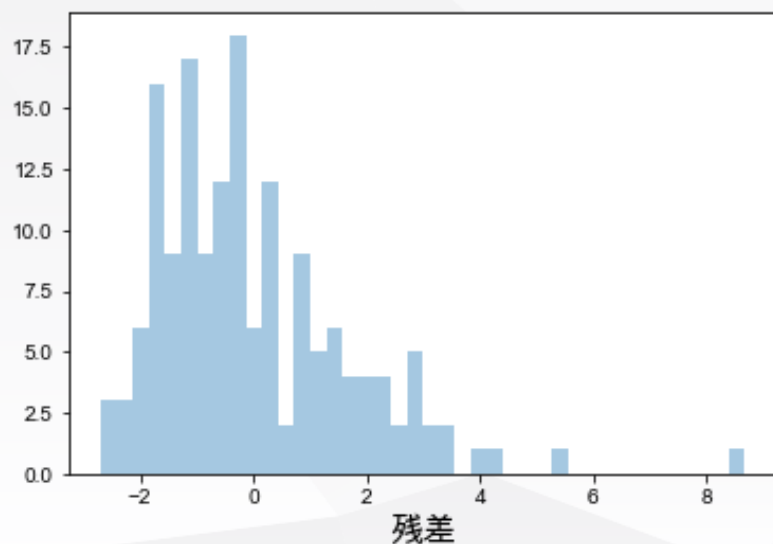
➤ 负log似然损失

- 极大似然估计等价于负log似然最小，因此负log似然也被称为一种损失函数：负log似然损失。
- L2损失是高斯白噪声假设下的负log似然损失。
- 分类任务中Logistic回归中采用的损失函数也是负log似然损失。

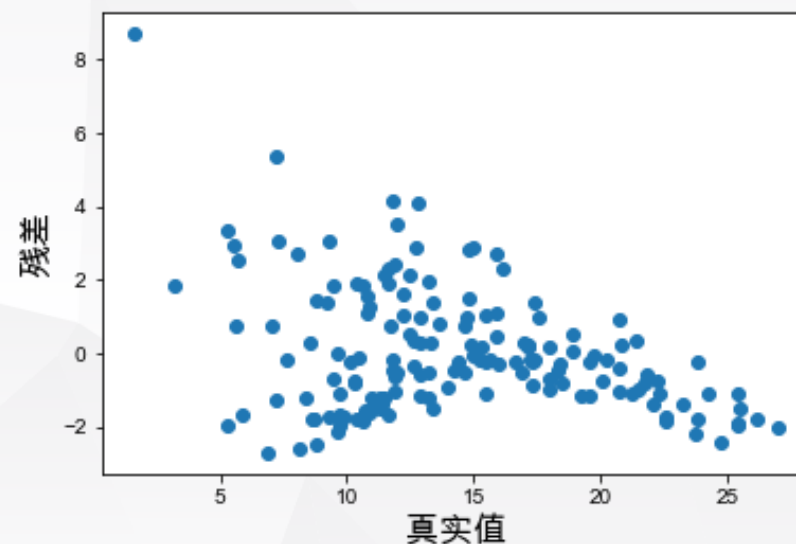
残差分布

■ 我们也可以通过残差的分布来检验回归模型是否足够正确

- 如果模型预测合理，残差应为0均值的正态分布



残差的分布并不符合0均值的正态分布
该模型（最小二乘回归模型）预测效果并不好

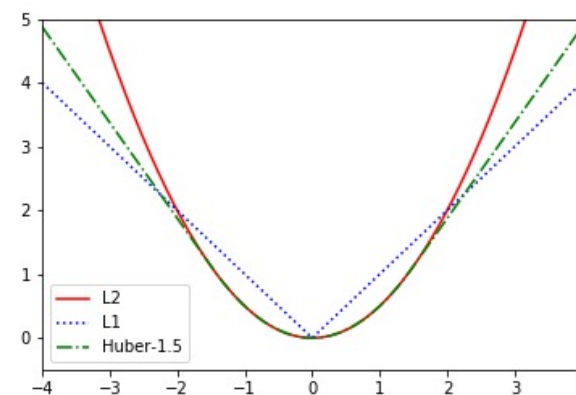


从真实值和残差的散点图来看，真实值较大时，预测残差大多 <0 。模型还没有完全建模 y 与 x 之间的关系，还有一部分关系残留在残差中

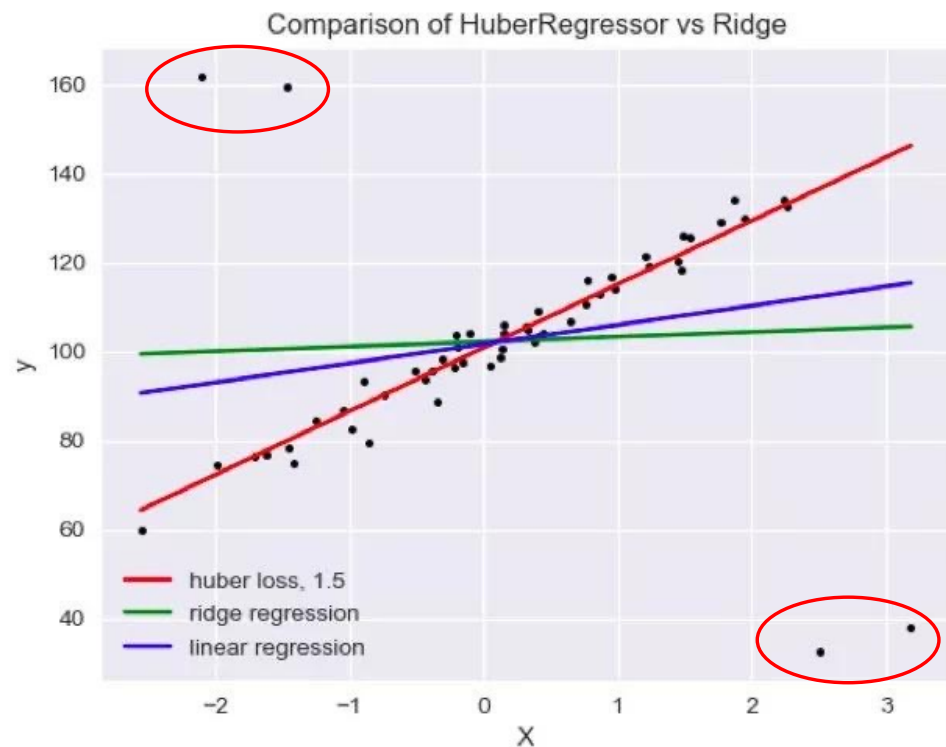
➤ 胡伯 (Huber) 损失函数

- L2损失的好处是处处连续，优化求解方便。
- 但L2损失对噪声点敏感（当预测残差 r 的绝对值较大时， r^2 很大）。
- 考虑L1（ $|r|$ ）损失函数，但 $|r|$ 在 $r = 0$ 处不连续，优化求解麻烦。
- 因此综合L1和L2损失函数，得到Huber损失：

$$L_{\delta}(\hat{y}, y) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2 & |r| \leq \delta \\ \delta|\hat{y} - y| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$



L2损失、L1损失和Huber损失比较
横轴：预测残差 $r = y - \hat{y}$
纵轴：损失函数的值



左上方和右下出现了一些异常点
OLS和Ridge regression (L2损失) 都不同程度上受到了异常点的影响, 而Huber损失受影响很小



正常点所占的比重更小, OLS和Ridge regression (L2损失) 所决定出的回归模型几乎不工作, Huber损失性能很好

➤ 采用Huber损失的回归模型：HuberRegressor

■ sklearn中实现Huber损失的回归模型：HuberRegressor

Huber损失：

```
from sklearn.linear_model import HuberRegressor  
huber = HuberRegressor()  
huber.fit(X_train, y_train)  
y_train_pred_huber = huber.predict(X_train)
```

L2损失：

```
from sklearn.linear_model import LinearRegression  
lr = LinearRegression()  
lr.fit(X_train, y_train)  
y_train_pred_lr = lr.predict(X_train)
```

➤ 采用Huber损失的回归模型：SGDRegressor

- 当训练样本很多时，可采用随机梯度下降来优化模型参数
- sklearn中的实现为SGDRegressor
 - 损失函数参数 `loss= 'huber'`
 - 和HuberRegressor中稍有不同，详见sklearn文档：https://scikit-learn.org/stable/modules/linear_model.html#huber-regression
- 当有数据噪声严重时，还可采用RANSAC (RANdom SAmple Consensus)识别噪声
- sklearn中对应的类为RANSACRegressor

思考题

■ 回归任务中L1损失对应的噪声模型是什么分布？L1损失最小也等价于极大似然估计吗？

■ 提示：Laplace分布为

$$x \sim \text{Laplace}(\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right)$$

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
- 回归任务的模型性能评价
- 线性回归案例分析

➤ 线性回归

■ 机器学习的三要素

- 1. 函数集合 $\{f_1, f_2, \dots\}$
- 2. 目标函数 $J(f)$: 函数的好坏
- 3. 优化算法 : 找到最佳函数

■ 回归任务目标函数 :

$$J(f, \lambda) = \sum_{i=1}^N \underbrace{L(\hat{y}_i, y_i)}_{\text{损失函数}} + \underbrace{\lambda R(f)}_{\text{正则项}}$$

- 如果目标函数只考虑模型对训练样本的拟合，容易产生过拟合。
- 抑制过拟合：在目标函数中加入正则项

正则项

- 正则项通常是与模型 f 的复杂度有关。

- 目标函数

$$J(f, \lambda) = \sum_{i=1}^N L(\hat{y}_i, y_i) + \lambda R(f)$$

- 称为结构风险。

- 训练集上的损失称为经验风险。

➤ 线性回归中常用正则函数

■ $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$, 正则项 $R(f) = R(\mathbf{w})$

■ L2正则 : $R(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{j=1}^D w_j^2$

■ L1正则 : $R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{j=1}^D |w_j|$

■ 其中 \mathbf{w} 为模型参数 , D 为参数的维数。

■ 注意 : 正则项不对截距项惩罚 , 因为截距项不影响模型的复杂度
(函数的平滑程度 : $\Delta \mathbf{x} \rightarrow \Delta y$)

$$\Delta y = f(\mathbf{x} + \Delta \mathbf{x}) - f(\mathbf{x}) = (\mathbf{w}^T (\mathbf{x} + \Delta \mathbf{x}) + b) - (\mathbf{w}^T \mathbf{x} + b) = \mathbf{w}^T \Delta \mathbf{x}$$

➤ 无正则：最小二乘

- 由于线性回归模型简单，在确定无需正则时，可以没有正则项
- 此时目标函数为：

$$J(\mathbf{w}) = \sum_{i=1}^N L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \sum_{i=1}^N (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

- Scikit-Learn中实现的最小二乘线性回归为：LinearRegression

>> LinearRegression

■ `class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)`

$$J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

参数	说明	备注
fit_intercept	模型中是否包含截距项。	如果数据已经中心化（训练样本集的y的均值为0），可设置为False。
normalize	是否对输入特征x做归一化（减去均值，除以L2模），使得每个样本的模长为1。	对数据进行归一化会使得超参数学习更鲁棒，且几乎和样本数目没有关系。回归中用的不多，更多的时候用标准化
copy_X	是否拷贝数据x。设置为False时，x会被重写覆盖	
n_jobs	并行计算时使用CPU的数目。	-1表示使用所有的CPU资源（建议与设置为CPU核的数目相同，现在CPU基本上都支持超线程，-1可能对应的是超线程后的值，最好设置为CPU核的数目）。

LinearRegression

■ `class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)`

$$J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

属性	说明	备注
<code>coef_</code>	回归系数/权重，与特征维数相同。	对多任务回归，标签 y 为二维数组，则回归系数也是二维数组。
<code>intercept_</code>	截距项。	
方法	说明	备注
<code>fit(X, y[, sample_weight])</code>	模型训练。X,y为训练数据，可设置每个样本的权重sample_weight	如果样本来自不同设备（如精度不同），可设置样本权重。
<code>predict(X)</code>	使用训练好的模型进行预测，返回预测值	
<code>score(X, y[, sample_weight])</code>	评估模型预测性能，返回模型预测的 R^2 分数（预测值和真实值 y 的差异）。	请见回归模型评价指标部分。

➤ 岭回归：L2正则的线性回归

■ 当损失函数取L2损失： $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$

■ 正则项取L2正则： $R(\mathbf{w}) = \sum_{j=1}^D w_j^2$

■ 线性回归模型： $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} = \sum_{j=0}^D w_j x_j \quad (x_0 = 1)$

■ 得到岭回归的目标函数为：

$$J(\mathbf{w}, \lambda) = \sum_{i=1}^N L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w}) = \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \sum_{j=1}^D w_j^2$$

$$= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

➤ 岭回归

- 岭回归的目标函数 $J(\mathbf{w}, \lambda) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$
- L2正则可视为参数先验分布为正态分布的贝叶斯估计。

■ 回忆：贝叶斯估计

- 数据的似然： $\ell(f) = \ln p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^N \ln p(y_i|\mathbf{x}_i)$
- 参数的先验： $p(\boldsymbol{\theta})$
- 参数的后验： $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})$
- 参数的最大后验估计： $\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D})$

岭回归

岭回归的贝叶斯估计

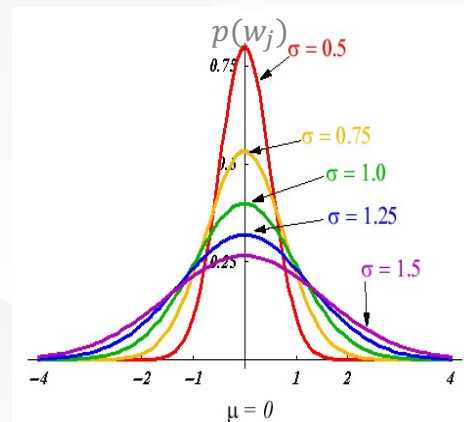
- 数据的似然： $\ell(f) = \sum_{i=1}^N \ln p(y_i | \mathbf{x}_i) - \frac{N}{2} \log(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}$
- 参数的先验： $w_j \sim N(0, \tau^2)$ ，且 w_j 相互独立，则

$$p(\mathbf{w}) = \prod_{j=1}^D p(w_j) = \prod_{j=1}^D \frac{1}{\sqrt{2\pi}\tau} \exp\left(-\frac{w_j^2}{2\tau^2}\right)$$

- 参数的后验：

$$\ln p(\mathbf{w} | \mathcal{D}) \propto \ln p(\mathbf{w}) + \ln p(\mathcal{D} | \mathbf{w})$$

$$= -\frac{D}{2} \ln(2\pi) - D \ln \tau - \sum_{j=1}^D \frac{w_j^2}{2\tau^2} - \frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}$$





■最大后验估计(Maximum a posteriori estimation, MAP)

$$\begin{aligned}\hat{\mathbf{w}} &= \operatorname{argmax}_{\mathbf{w}} \ln p(\mathbf{w}|\mathcal{D}) \\&= \operatorname{argmax}_{\mathbf{w}} \left(-\frac{D}{2} \ln(2\pi) - D \ln \tau - \sum_{j=1}^D \frac{w_j^2}{2\tau^2} - \frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2} \right) \\&= \operatorname{argmax}_{\mathbf{w}} \left(-\sum_{j=1}^D \frac{w_j^2}{2\tau^2} - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2} \right) \quad (\text{去掉与}\mathbf{w}\text{无关的项}) \\&= \operatorname{argmin}_{\mathbf{w}} \left(\frac{\sigma^2}{\tau^2} \sum_{j=1}^D w_j^2 + \sum_{i=1}^N (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \right) \quad (\text{乘以}2\sigma^2)\end{aligned}$$

■等价于岭回归的目标函数（去掉负号，取最大变成取最小）：

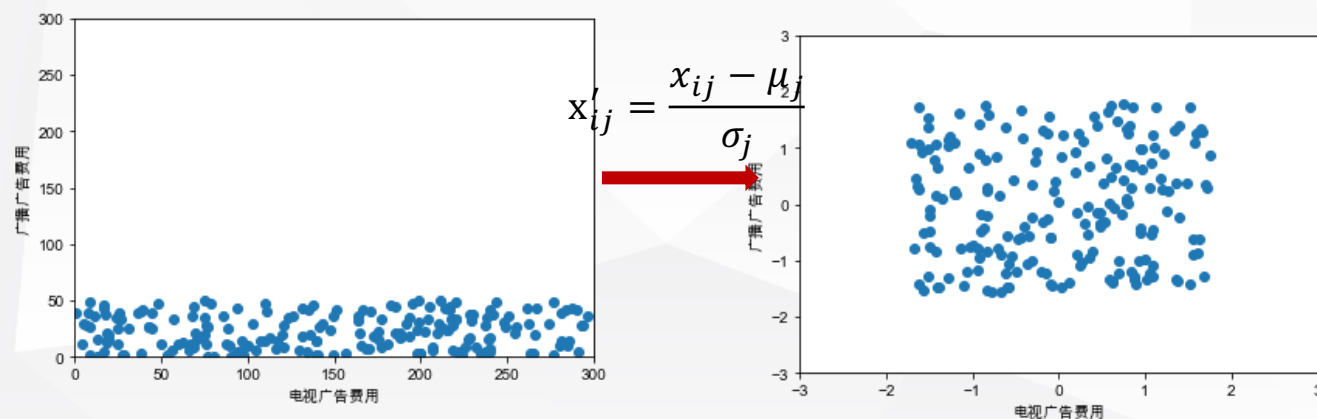
$$J(\mathbf{w}, \lambda) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

➤ 特征缩放

■ 在正则项中，各特征的参数地位相同，这就要求各特征的单位相同，这可以对特征进行缩放（去量纲）实现。

■ 最常用的特征缩放为特征标准化

- sklearn中的实现：StandardScaler



$$\mu_j = \frac{1}{N} \sum_{i=0}^N x_{ij}$$

$$\sigma_j = \sqrt{\frac{1}{N-1} \left(\sum_{i=0}^N x_{ij} - \mu_j \right)^2}$$

```
# 数据标准化
from sklearn.preprocessing import StandardScaler
# 构造输入特征的标准化器
ss_X = StandardScaler()

# 分别对训练和测试数据的特征进行标准化处理
X_train = ss_X.fit_transform(X_train)
X_test = ss_X.transform(X_test)
```

■ 亦可采用MinMaxScaler 将特征缩放到[0,1]。

➤ sklearn中的岭回归 : Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False,  
copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

参数说明 :

- *alpha*: 目标函数 $J(\mathbf{w}, \lambda) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$ 中的 λ
- *fit_intercept*、*normalize*、*copy_X*: 意义同LinearRegression
- 其余参数与优化计算有关 (请见优化求解部分) 。

属性同LinearRegression

常用方法同LinearRegression

➤ Lasso : L1正则的线性回归

- 当损失函数取L2损失 : $L(f(\mathbf{x}_i; \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$
- 正则项取**L1正则** : $R(\mathbf{w}) = \sum_{j=1}^D |w_j|$
- 线性回归模型 : $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} = \sum_{j=0}^D w_j x_j \quad (x_0 = 1)$
- 得到Lasso (Least Absolute Shrinkage and Selection Operator) 的目标函数为 :

$$\begin{aligned} J(\mathbf{w}, \lambda) &= \sum_{i=1}^N L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w}) = \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \sum_{j=1}^D |w_j| \\ &= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1 \end{aligned}$$

- Lasso的目标函数 $J(\mathbf{w}, \lambda) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$
- L1正则可视为参数先验分布为拉普拉斯分布的贝叶斯估计。

■ 贝叶斯估计

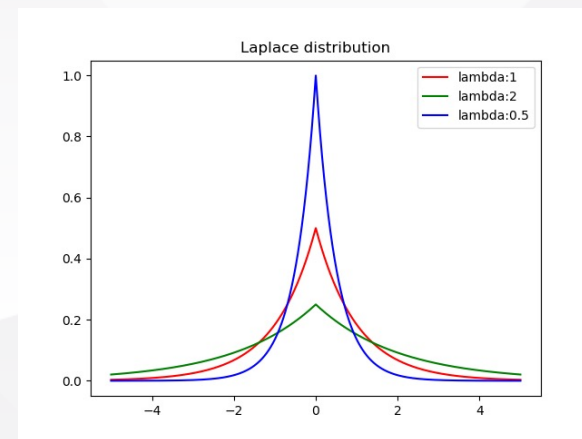
- 数据的似然： $\ell(f) = \sum_{i=1}^N \ln p(y_i | \mathbf{x}_i) - \frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}$
- 参数的先验： $w_j \sim \text{Laplace}(0, b)$ ，且 w_j 相互独立，则

$$p(\mathbf{w}) = \prod_{j=1}^D p(w_j) = \prod_{j=1}^D \frac{1}{2b} \exp\left(-\frac{|w_j|}{b}\right)$$

- 参数的后验：

$$\ln p(\mathbf{w} | \mathcal{D}) \propto \ln p(\mathbf{w}) + \ln p(\mathcal{D} | \mathbf{w})$$

$$= -D \ln(2b) - \sum_{j=1}^D \frac{|w_j|}{b} - \frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}$$





■最大后验估计(Maximum a posteriori estimation, MAP)

$$\hat{\mathbf{w}} = \operatorname{argmax}_{\mathbf{w}} \ln p(\mathbf{w}|\mathcal{D})$$

$$= \operatorname{argmax}_{\mathbf{w}} \left(= -D \ln(2b) - \sum_{j=1}^D \frac{|w_j|}{b} - \frac{N}{2} \ln(2\pi) - N \ln \sigma - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2} \right)$$

$$= \operatorname{argmax}_{\mathbf{w}} \left(- \sum_{j=1}^D \frac{|w_j|}{b} - \sum_{i=1}^N \frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2} \right) \quad (\text{去掉与}\mathbf{w}\text{无关的项})$$

$$= \operatorname{argmin}_{\mathbf{w}} \left(\frac{2\sigma^2}{b} \sum_{j=1}^D |w_j| + \sum_{i=1}^N (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \right) \quad (\text{乘以} 2\sigma^2)$$

■等价于Lasso回归的目标函数（去掉负号，取最大变成取最小）：

$$J(\mathbf{w}, \lambda) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \sum_{j=1}^D |w_j|$$

➤ sklearn中的Lasso : Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, fit_intercept=True, normalize=False,
precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False,
positive=False, random_state=None, selection='cyclic')
```

参数说明：

- alpha: 目标函数 $J(\mathbf{w}, \lambda) = \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1$ 中的 λ
- fit_intercept、normalize、copy_X: 意义同LinearRegression
- 其余参数与优化计算有关（请见优化求解部分）。

属性同LinearRegression

常用方法同LinearRegression

➤ 弹性网络：L1正则 + L2正则

- 正则项还可以为L1正则和L2正则的线性组合：

$$R(\mathbf{w}, \rho) = \sum_{j=1}^D \left(\rho |w_j| + \frac{(1-\rho)}{2} w_j^2 \right)$$

- 得到弹性网络的目标函数：

$$J(\mathbf{w}, \lambda, \rho) = \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \left(\rho \|\mathbf{w}\|_1 + \frac{(1-\rho)}{2} \|\mathbf{w}\|_2^2 \right), \quad 0 \leq \rho \leq 1.$$

- Sklearn中实现的弹性网络为：ElasticNet

```
class sklearn.linear_model.ElasticNet(alpha=1.0, l1_ratio=0.5, fit_intercept=True, normalize=False,
precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False,
random_state=None, selection='cyclic')
```

alpha参数为上述目标函数中的 λ ，l1_ratio为上述目标函数中的 ρ

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
 - 解析求解（正规方程组）
 - 梯度下降
 - 坐标轴下降
- 回归任务的模型性能评价
- 线性回归案例分析

➤ 目标函数最优值

- 给定超参数 λ 的情况下，目标函数最优解： $\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$
- 根据优化理论，函数极值点只能在边界点、不可导点、临界点（导数为0的点）
- 一阶偏导数组成的向量（梯度）为0向量： $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{0}$
- 若二阶海森（Hessian）矩阵

$$H = \frac{\partial^2 J(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^T} = \begin{bmatrix} \frac{\partial^2 J}{\partial w_1 \partial w_1} & \frac{\partial^2 J}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 J}{\partial w_1 \partial w_D} \\ \frac{\partial^2 J}{\partial w_2 \partial w_1} & \frac{\partial^2 J}{\partial w_2 \partial w_2} & \cdots & \frac{\partial^2 J}{\partial w_2 \partial w_D} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 J}{\partial w_D \partial w_1} & \frac{\partial^2 J}{\partial w_D \partial w_2} & \vdots & \frac{\partial^2 J}{\partial w_D \partial w_D} \end{bmatrix}$$

为正定矩阵，则在临界点目标函数取最小值。

➤ 最小二乘的解析求解：正规方程组

■ 最小二乘 (Ordinary Linear Square, OLS) 目标函数 (矩阵形式) 为：

$$J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\left[\frac{\partial (\mathbf{y}^T \mathbf{b})}{\partial \mathbf{y}} = \mathbf{b} \right]$$

■ 梯度为：

$$\left[\frac{\partial (\mathbf{y}^T \mathbf{A} \mathbf{y})}{\partial \mathbf{y}} = (\mathbf{A}^T + \mathbf{A}) \mathbf{y} \right]$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 2\mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = -2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{0}$$

■ 从而：

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y} \rightarrow \hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

正规方程组(Normal Equations)

➤ OLS的最优解：SVD

- $\hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{X}^\dagger \mathbf{y}$ 需要求矩阵 $\mathbf{A} = \mathbf{X}^T \mathbf{X}$ 的逆矩阵。
- 可采用数值更稳定的奇异值分解实现： $\mathbf{X} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$
- \mathbf{U} 为 $N \times N$ 的矩阵， \mathbf{U} 的列组成正交基，为 $\mathbf{X} \mathbf{X}^T$ 的特征向量
- $\mathbf{\Sigma}$ 是 $N \times D$ 的对角矩阵，对角线元素为矩阵 \mathbf{X} 的奇异值
- \mathbf{V} 是 $D \times D$ 的矩阵， \mathbf{V} 的列组成正交基，为 $\mathbf{X}^T \mathbf{X}$ 的特征向量

$$\begin{matrix} & D \\ & \boxed{\mathbf{X}_{N \times D}} \\ N & \end{matrix} = \begin{matrix} & N \\ & \boxed{\mathbf{U}_{N \times N}} \\ N & \end{matrix} \begin{matrix} & D \\ & \boxed{\mathbf{\Sigma}_{N \times D}} \\ N & \end{matrix} \begin{matrix} & D \\ & \boxed{\mathbf{V}_{D \times D}^T} \\ D & \end{matrix}$$

Scikit-Learn中LinearRegression中采用的优化方法。

➤ OLS的最优解：SVD

■ OLS目标函数： $J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ 。

■ \mathbf{X} 的SVD： $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$

■ 由于正交变换 \mathbf{U}^T 保范，

$$\begin{aligned}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 &= \|\mathbf{U}^T\mathbf{X}\mathbf{w} - \mathbf{U}^T\mathbf{y}\|_2^2 \\ &= \|\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{w} - \mathbf{U}^T\mathbf{y}\|_2^2 \\ &= \|\mathbf{\Sigma}\mathbf{V}^T\mathbf{w} - \mathbf{U}^T\mathbf{y}\|_2^2\end{aligned}$$

■ 令 $\mathbf{y}' = \mathbf{U}^T\mathbf{y}$ ， $\mathbf{w}' = \mathbf{V}^T\mathbf{w}$ ，则问题变成最小化 $\|\mathbf{\Sigma}\mathbf{w}' - \mathbf{y}'\|_2^2$ 。

■ 由于 $\mathbf{\Sigma}$ 为准三角矩阵，上述问题变得非常简单。

➤ OLS的最优解：SVD

■ 令 $y' = U^T y$, $w' = V^T w$, 则问题变成最小化 $\|\Sigma w' - y'\|_2^2$ 。

■ Σ 为准三角矩阵，所以： $w'_j = \frac{w'_j}{\sigma_j}$ 。

- 当奇异值 σ_j 很小（矩阵 X 接近不满秩）时， w'_j 数值不稳定。
- 矩阵 X 接近不满秩：特征之间存在共线性（有冗余）

■ 再根据 $w' = V^T w$, 得到 $w = V^{-T} w' = V w'$ 。

■ 或者：

$$X^\dagger = V \Sigma^\dagger U^T = V \begin{bmatrix} 1/\sigma_1 & 0 & \dots & \dots & 0 \\ 0 & \ddots & 0 & \dots & 0 \\ \vdots & 0 & 1/\sigma_r & 0 & 0 \\ \vdots & \vdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots \end{bmatrix} U^T \quad \hat{w}_{OLS} = X^\dagger y = V \Sigma^\dagger U^T y$$

➤ 岭回归的最优解：SVD

■ 岭回归的目标函数为： $J(\mathbf{w}, \lambda) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}^T\mathbf{w}$

■ 梯度：

$$\frac{\partial J(\mathbf{w}, \lambda)}{\partial \mathbf{w}} = -2\mathbf{X}^T\mathbf{y} + 2(\mathbf{X}^T\mathbf{X})\mathbf{w} + 2\lambda\mathbf{w} = \mathbf{0}$$

$$\hat{\mathbf{w}}_{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

■ 将 \mathbf{X} 的SVD分解代入： $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, $\mathbf{X}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T$

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{D}\mathbf{V}^T$$

$$\begin{aligned}\hat{\mathbf{w}}_{Ridge} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} = (\mathbf{V}\mathbf{D}\mathbf{V}^T + \lambda\mathbf{V}\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{y} \\ &= \mathbf{V}(\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{y} = \mathbf{V}(\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{y}\end{aligned}$$

➤ 岭回归的最优解：SVD

■ OLS : $\hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{V} \mathbf{\Sigma}^\dagger \mathbf{U}^T \mathbf{y}$

- 对矩阵 $\mathbf{X}^T \mathbf{X}$ 求逆
- 当矩阵 \mathbf{X} 接近不满秩时，结果不稳定

■ 岭回归 : $\hat{\mathbf{w}}_{Ridge} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$
 $= \mathbf{V} (\mathbf{D} + \lambda \mathbf{I})^{-1} \mathbf{\Sigma}^T \mathbf{U}^T \mathbf{y}$

- 对矩阵 $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}$ 求逆
- 即使矩阵 \mathbf{X} 接近不满秩，结果仍然稳定
- 要得到不同 λ 对应的解，只需对 \mathbf{X} 一次 SVD 分解

➤ 岭回归：权重衰减

- $$\begin{aligned}\hat{\mathbf{w}}_{\text{Ridge}} &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \\ &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{X}) (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \\ &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{X}) \hat{\mathbf{w}}_{\text{OLS}}\end{aligned}$$
- $$\begin{aligned}\|\hat{\mathbf{w}}_{\text{Ridge}}\|_2 &= \|(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{X}) \hat{\mathbf{w}}_{\text{OLS}}\|_2 \\ &= \|(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I} - \lambda \mathbf{I}) \hat{\mathbf{w}}_{\text{OLS}}\|_2 \\ &= \|\hat{\mathbf{w}}_{\text{OLS}} - \lambda \mathbf{I} (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \hat{\mathbf{w}}_{\text{OLS}}\|_2 \\ &= \|(\mathbf{I} - \lambda (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1}) \hat{\mathbf{w}}_{\text{OLS}}\|_2 \\ &< \|\hat{\mathbf{w}}_{\text{OLS}}\|_2\end{aligned}$$

权重衰减：权重幅值变小

Lasso的无法求得解析解

■ Lasso的无法求得解析解，可用迭代求解。

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
 - 解析求解（正规方程组）
 - 梯度下降
 - 坐标轴下降
- 回归任务的模型性能评价
- 线性回归案例分析

➤ 梯度下降

- OLS和岭回归虽然可以用解析法求解，但当输入矩阵 X 很大时，SVD的计算量大，甚至不能载入内存。此时可用迭代方式求解。
- 梯度下降法是求解无约束优化问题最常采用的方法之一。
- 最小二乘回归和岭回归可采用梯度下降法求解，Lasso由于目标函数中有L1正则函数不可导，不能采用梯度下降法求解。

➤ Recall : 梯度下降法

- 1. 初始化 $w^{(0)}$ (上标括号中的数字表示迭代次数) ;
- 2. 计算函数 $J(w)$ 在当前位置 $w^{(t)}$ 处的梯度 $\nabla_w J|_{w^{(t)}}$
- 3. 根据当前学习率 η , 更新参数

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w J|_{w^{(t)}}$$

- 4. 若 $J(w^{(t)}) - J(w^{(t+1)}) < \varepsilon$, 返回 $w^{(t)}$ 为最佳参数 ;
- 否则 $t = t + 1$, 转第2步。

➤ 最小二乘

■ 最小二乘的目标函数： $J(\mathbf{w}, \lambda) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$

■ 目标函数的梯度：

$$\begin{aligned} g(\mathbf{w}) &= \frac{\partial J(\mathbf{w}, \lambda)}{\partial \mathbf{w}} = 2(\mathbf{X}^T \mathbf{X})\mathbf{w} - 2\mathbf{X}^T \mathbf{y} \\ &= 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \end{aligned}$$

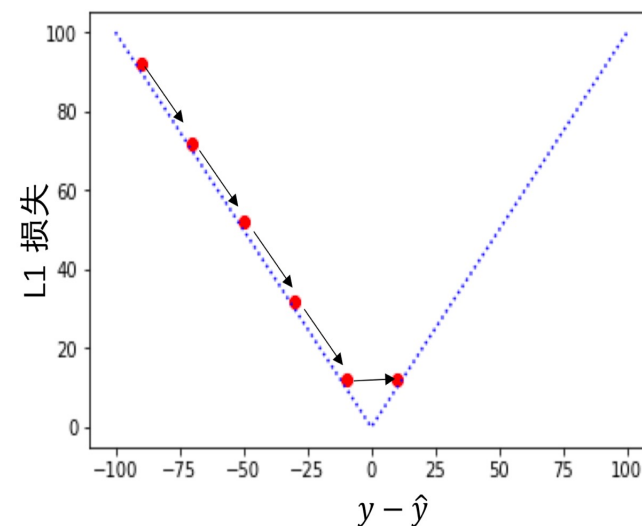
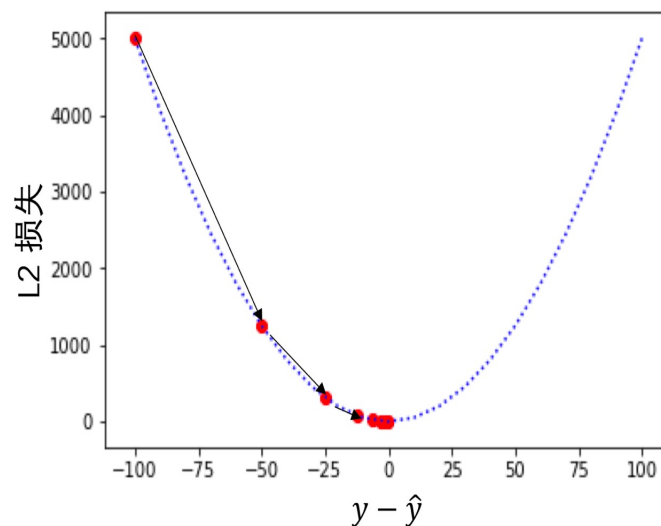
预测残差 r

参数的更新量与输入与预测残差的相关性有关。

■ 参数更新： $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta g(\mathbf{w}^{(t)})$
 $= \mathbf{w}^{(t)} - 2\eta \mathbf{X}^T(\mathbf{X}\mathbf{w}^{(t)} - \mathbf{y})$

➤ 损失函数对梯度的影响

- L2损失：梯度的绝对值为 $|y - \hat{y}|$ 。随损失增大而增大，损失趋于0时梯度也减小。这使得在训练结束时，使用采用L2损失的模型的结果会更精确。即使固定学习率，函数也能较快取得最小值。
- L1损失：梯度的绝对值始终为1。即使对于小的损失值，梯度也大。这不利于函数的收敛和模型的学习，可能导致在快要结束时错过了最小点 → 使用变化的学习率，在损失接近最小值时降低学习率



➤ 岭回归

■ 岭回归的目标函数： $J(\mathbf{w}, \lambda) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}^T\mathbf{w}$

■ 目标函数的梯度：

$$g(\mathbf{w}) = \frac{\partial J(\mathbf{w}, \lambda)}{\partial \mathbf{w}} = 2(\mathbf{X}^T\mathbf{X})\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w}$$

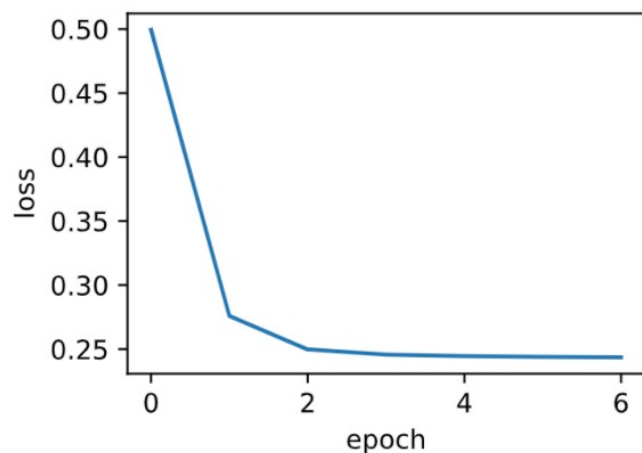
■ 参数更新：
$$\begin{aligned}\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} - \eta g(\mathbf{w}^{(t)}) \\ &= \mathbf{w}^{(t)} - 2\eta\mathbf{X}^T(\mathbf{X}\mathbf{w}^{(t)} - \mathbf{y}) - 2\eta\lambda\mathbf{w}^{(t)}\end{aligned}$$

➤ 梯度下降的高阶话题

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta g(\mathbf{w}^{(t)})$$

- 梯度的计算：计算梯度时需用到每个训练样本，当样本数目很多时，计算费用高：随机梯度下降、小批量梯度下降
- 动量：比梯度更好的移动方向
- 学习率：
 - 太小，收敛慢
 - 学习率太大，不收敛
 - 各参数公用一个学习率：特征缩放
 - 自适应学习率

例：小批处理梯度下降

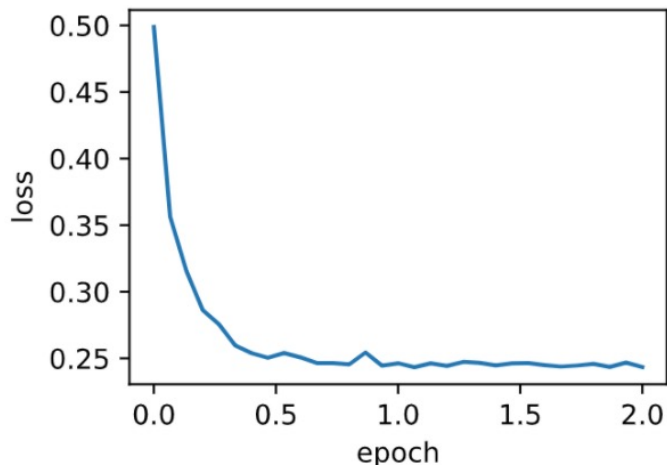


批处理梯度下降

($bathsize = N = 1500$)

1个迭代周期对模型参数只迭代1次。

6次迭代后目标函数值（训练损失）的下降趋向平稳

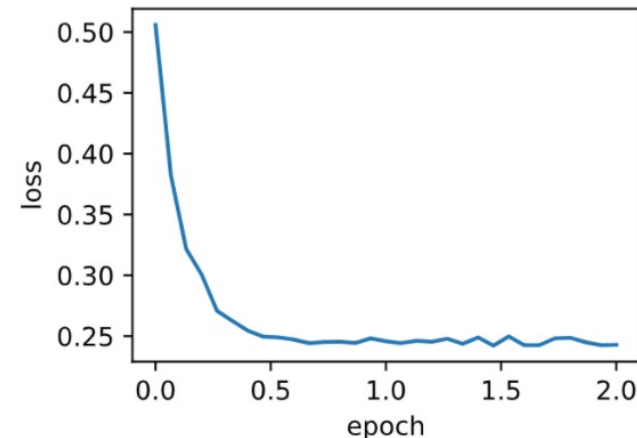


随机梯度下降

($bathsize = 1$)

每处理一个样本会更新一次模型参数

目标函数值的下降在1个迭代周期后就变得较为平缓
一个迭代周期耗时更多



小批量梯度下降

($bathsize = 10$)

每处理一个batch更新一次模型参数

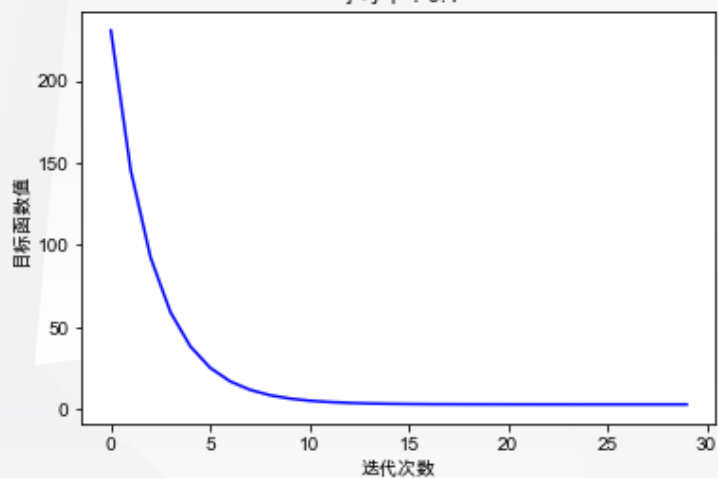
目标函数值的下降在1个迭代周期后就变得较为平缓

学习率

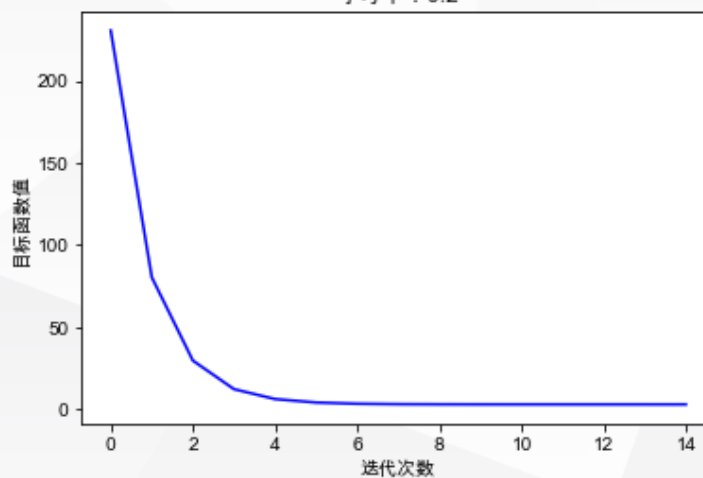
■ 学习率对训练的影响

- 目标函数变化太慢：学习率太低
- 目标函数出现NaN：通常意味着学习率太高
- 建议： $[1e-3 \dots 1e-5]$

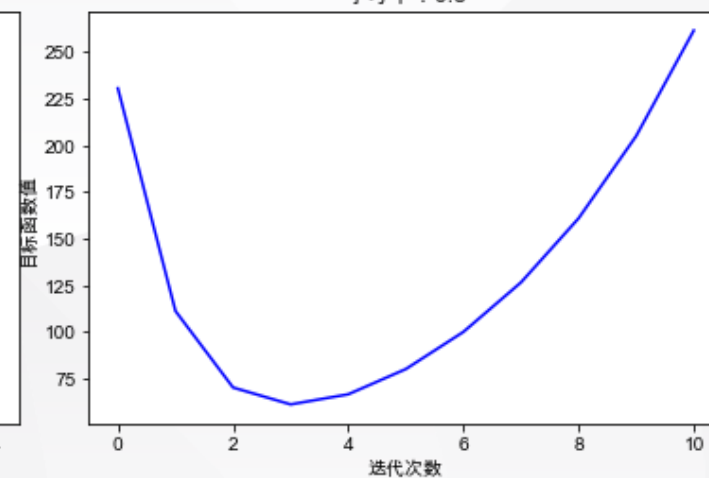
学习率：0.1



学习率：0.2

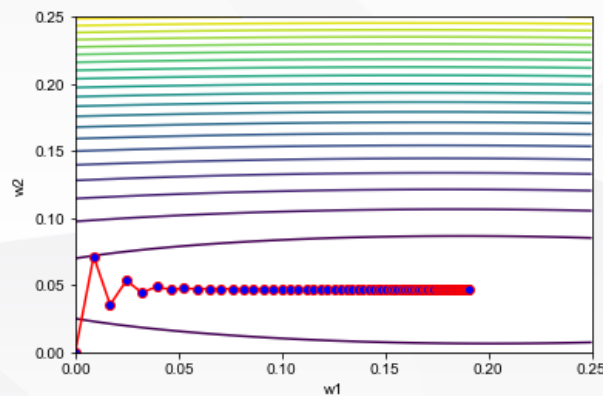
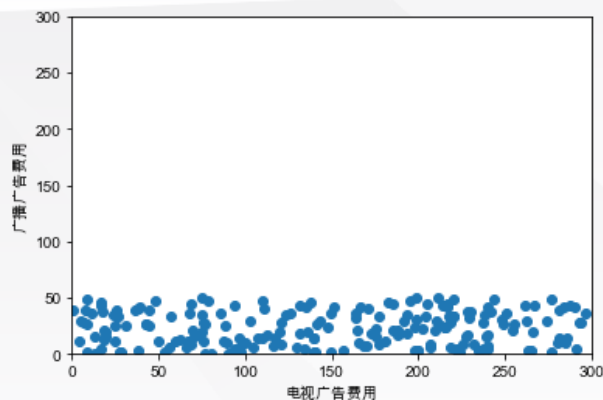


学习率：0.8

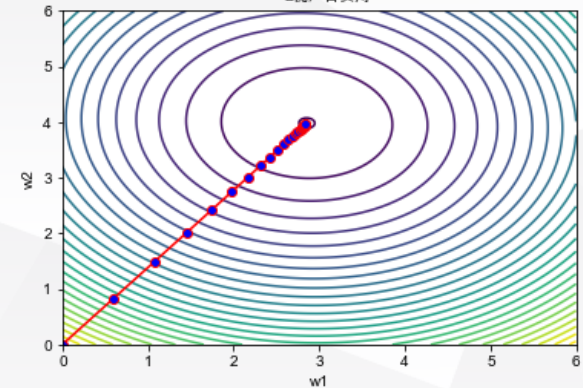
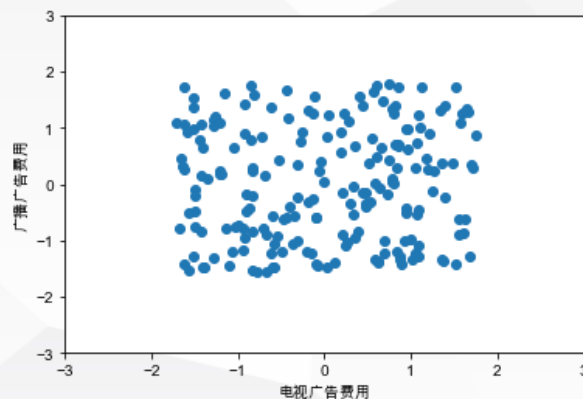


➤ 特征去量纲

- OLS的梯度下降更新换代： $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta g(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} - 2\eta \mathbf{X}^T(\mathbf{X}\mathbf{w}^{(t)} - \mathbf{y})$
- 各参数公用一个学习率：特征缩放/去量纲



特征没有标准化



特征标准化

特征标准化：

$$x'_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j}$$

➤ 回忆 : Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False,
copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

优化计算有关参数

参数	说明	备注
<i>max_iter</i>	共轭梯度求解器的最大迭代次数。	对于优化算法 <i>solver</i> 为'sparse_cg'和'lsqr'，默认值由scipy.sparse.linalg确定。 对于'sag'求解器，默认值为1000。
<i>tol</i>	解的精度，判断迭代收敛与否的阈值。 当(loss > previous_loss - tol)时迭代终止。	
<i>solver</i>	求解最优化问题的算法 可取：'auto'，'svd'，'cholesky'，'lsqr'， 'sparse_cg'，'sag'，'saga'	请见下页
<i>random_state</i>	数据洗牌时的随机种子。	仅用于'sag'求解器。

➤ 回忆 : Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False,
copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

*solver*参数取值：

取值	说明	备注
'auto'	根据数据类型自动选择求解器	
'svd'	使用X的奇异值分解来计算Ridge系数	对于奇异矩阵，比'cholesky'更稳定。
'cholesky'	使用标准的scipy.linalg.solve函数获得解析解	
'sparse_cg'	使用scipy.sparse.linalg.cg中的共轭梯度求解器	对大规模数据，比“cholesky”更合适
'lsqr'	使用专用的正则化最小二乘常数 scipy.sparse.linalg.lsqr	最快
'sag'	用随机平均梯度下降，当样本数n_samples和特征维数n_feature都很大时，比其他求解器更快	当fit_intercept为True时，'sag' 和 'saga'只支持稀疏输入。“sag” 和 'saga' 快速收敛仅在具有近似相同尺度的特征上被保证，数据需要标准化
'saga'	sag的改进版本	

➤ 随机梯度下降回归 : SGDRegressor

■ SGDRegressor适合大数据量训练集（样本数目>10000）。

```
class sklearn.linear_model.SGDRegressor(loss='squared_loss', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, random_state=None, learning_rate='invscaling', eta0=0.01, power_t=0.25, warm_start=False, average=False, n_iter=None)
```

- 支持的损失函数`loss`包括：

- `'squared_loss'` : L2损失
- `'huber'` : Huber损失
- `'epsilon_insensitive'` : ϵ 不敏感损失 (SVM)
- `'squared_epsilon_insensitive'`

参数`epsilon`是某些损失函数（`huber`、`epsilon_insensitive`、`squared_epsilon_insensitive`）需要的额外参数。

- 支持的正则函数`penalty`包括：

- `'none'` : 无正则
- `'l2'` : L2正则
- `'l1'` : L1正则
- `'elasticnet'` : L1正则+L2正则（参数`l1_ratio`为L1正则比例）

参数`alpha`是正则惩罚系数，也用于学习率计算（请见下页）。

目标函数为：

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N L(f(\mathbf{x}_i), y_i) + \alpha R(\mathbf{w})$$

➤ 随机梯度下降回归 : SGDRegressor

```
class sklearn.linear_model.SGDRegressor(loss='squared_loss', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, random_state=None, learning_rate='invscaling', eta0=0.01, power_t=0.25, warm_start=False, average=False, n_iter=None)
```

- 优化相关的参数包括 :

- `max_iter`: 最大迭代次数 (访问训练数据的次数 , `epoches` 的次数) 。SGD在接近 10^6 的训练样本时收敛。因此可将迭代数设置成 $\text{np.ceil}(10^6 / N)$, 其中 N 是训练集的样本数目。默认值是5。参数`n_iter`意义相同 , 已被抛弃。
 - `tol` : 停止条件。如果非None , 当 $(\text{loss} > \text{previous_loss} - \text{tol})$ 时迭代终止。
 - `shuffle` : 每轮SGD之前是否重新对数据进行洗牌。
 - `random_state` : 随机种子 , Scikit-Learn中如随机有关的算法均有此参数 , 含义相同。当参数`shuffle==TRUE`时用到。如果随机种子相同 , 每次洗牌得到的结果一样。可设置为某个整数。
 - `learning_rate` : 支持三种方式
 - `constant` : $\text{eta} = \text{eta0}$
 - `'optimal'` : $\text{eta} = 1.0 / (\text{alpha} * (\text{t} + \text{t0}))$
 - `'invscaling'` : $\text{eta} = \text{eta0} / \text{pow}(\text{t}, \text{power_t})$
- 随机梯度下降实现参考文献 :
- "[Stochastic Gradient Descent](#)" L. Bottou - Website, 2010
 - "[The Tradeoffs of Large Scale Machine Learning](#)" L. Bottou - Website, 2011.

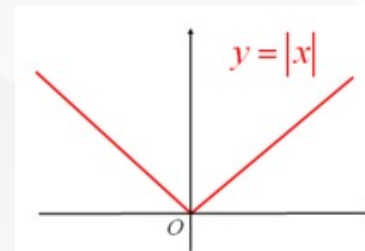
- `warm_start`: 是否从之前的结果继续。随机梯度下降中初始值可以是之前的训练结果 , 支持在线学习。初始值可在`fit`函数中作为参数传递。
- `average` : 是否采用平均随机梯度下降法 (ASGD) 。

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的的优化算法
 - 解析求解（正规方程组）
 - 梯度下降
 - 坐标轴下降
- 回归任务的模型性能评价
- 线性回归案例分析

次梯度法

- 当目标函数可导时，梯度下降法是非常有效的优化算法。
- Lasso的目标函数为： $J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1$ ，
- 其中绝对值函数 $\|\mathbf{w}\|_1$ 在 $w_j = 0$ 时不可导，无法计算梯度，无法用梯度下降法求解。
- **次梯度法**将梯度扩展为次梯度，可以解决这个难题。

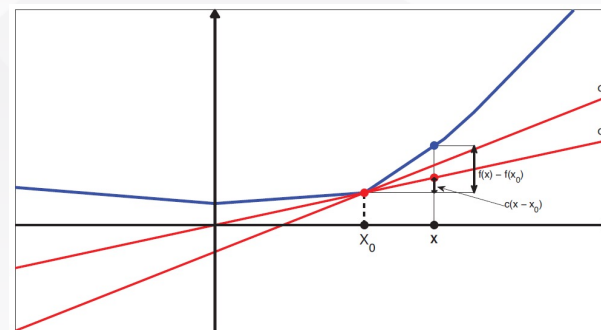


次梯度法

- 为了处理不平滑函数，扩展导数的表示，定义一个凸函数 f 在点 x_0 处的次导数/次梯度为一个标量 g ，使得

$$f(x) - f(x_0) \geq g(x - x_0), \forall x \in I$$

- 其中 I 为包含 x_0 的某个区间。



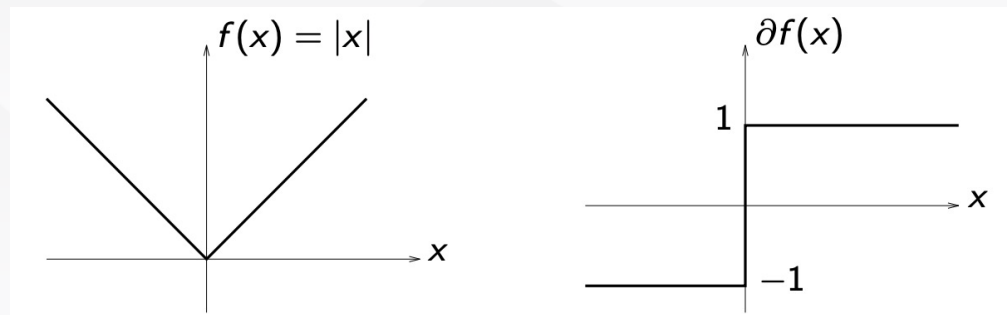
- 如图所示，对于定义域中的任何 x_0 ，我们总可以作出一条直线通过点 $(x_0, f(x_0))$ ，且直线要么与 f 相切，要么在它的下方。直线的斜率称为函数的次导数。
- 次梯度的集合为函数 f 在 x_0 处的次微分，记为 $\partial f(x_0)$ 。
- 定义次梯度集合为区间 $[a, b]$ ：

$$a = \lim_{x \rightarrow x_0^-} \frac{f(x) - f(x_0)}{x - x_0}, \quad b = \lim_{x \rightarrow x_0^+} \frac{f(x) - f(x_0)}{x - x_0}$$

次梯度法

■例：对凸函数 $f(x) = |x|$

$$\frac{\partial}{\partial w_j} |w_j| = \begin{cases} \{-1\} & \text{如果 } w_j < 0 \\ [-1, 1] & \text{如果 } w_j = 0 \\ \{+1\} & \text{如果 } w_j > 0. \end{cases}$$



当函数在 x_0 处可导时，次微分只有一个点组成，这个点就是函数在 x_0 处的导数。

■最优解条件： $0 \in \partial f(x^*) \iff f(x^*) = \min_x f(x)$ ，即当且仅当0属于函数 f 在点 x^* 处次梯度集合时， x^* 为极值点。

次梯度法

■将梯度下降法中的梯度换成次梯度，就得到次梯度法。

- 梯度下降法：
 1. 从 $t = 0$ 开始，初始化 $\mathbf{w}^{(0)}$ ；
 2. 计算目标函数 $J(\mathbf{w})$ 在当前值的梯度： $\nabla J(\mathbf{w}^{(t)})$ ；
 3. 根据学习率 η ，更新参数： $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla J(\mathbf{w}^{(t)})$ ；
 4. 判断是否满足迭代终止条件。如果满足，循环结束，返回最佳参数 $\mathbf{w}^{(t+1)}$ 和目标函数极小值 $J(\mathbf{w}^{(t+1)})$ ；否则转到第2步。

- 次梯度法：
 1. 从 $t = 0$ 开始，初始化 $\mathbf{w}^{(0)}$ ；
 2. 计算目标函数 $J(\mathbf{w})$ 在当前值的次微分： $\partial J(\mathbf{w}^{(t)})$ ；
 3. 根据学习率 η ，更新参数： $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \partial J(\mathbf{w}^{(t+1)})$ ；
 4. 判断是否满足迭代终止条件。如果满足，循环结束，返回最佳参数 $\mathbf{w}^{(t+1)}$ 和目标函数极小值 $J(\mathbf{w}^{(t+1)})$ ；否则转到第2步。

次梯度法

- 将梯度下降法中的梯度换成次梯度，就得到次梯度法。
- 与梯度下降算法不同，次梯度算法并不是下降算法（每次对参数的更新，并不能保证目标函数单调递减）。
- 因此一般情况下我们选择： $J(\mathbf{w}^*) = \min_{1, \dots, t} J(\mathbf{w}^{(t)})$ 。
- 虽然不能保证次梯度法中目标函数单调下降，可以证明对满足一定条件的凸函数，次梯度法是收敛的，只是收敛速度比梯度下降法慢。

坐标轴下降法

- 次梯度法收敛速度慢，Lasso求解推荐使用坐标轴下降法。
- **坐标轴下降法**：沿坐标轴方向搜索
 - 在每次迭代中，在当前点处沿一个坐标轴方向进行一维搜索。
 - 循环使用不同的坐标轴。一个周期的一维搜索迭代过程相当于一个梯度迭代。
 - 利用当前坐标系统进行搜索，无需计算目标函数的导数，只按照某一坐标方向进行搜索最小值。
- 梯度下降法：沿目标函数负梯度的方向搜索，梯度方向通常不与任何坐标轴平行。
- 坐标轴下降法在稀疏矩阵上的计算速度非常快。

➤ Lasso的坐标轴下降法求解

- Lasso的目标函数为： $J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$
- 坐标轴下降法：每次搜索一个维度
- 对第 j 维，目标函数第一部分为残差平方和 $RSS(\mathbf{w})$

$$\frac{\partial}{\partial w_j} RSS(\mathbf{w}) = \frac{\partial}{\partial w_j} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$

$$= \frac{\partial}{\partial w_j} \sum_{i=1}^N ((\mathbf{w}_{-j}^T \mathbf{x}_{i,-j} + w_j x_{i,j}) - y_i)^2$$

$$= -2 \sum_{i=1}^N (\mathbf{w}_{-j}^T \mathbf{x}_{i,-j} + w_j x_{i,j} - y_i) x_{i,j}$$

$$= \underbrace{2 \sum_{i=1}^N x_{i,j}^2 w_j}_{a_j} + \underbrace{2 \sum_{i=1}^N (\mathbf{w}_{-j}^T \mathbf{x}_{i,-j} - y_i) x_{i,j}}_{c_j} = a_j w_j + c_j$$

$$a_j = 2 \sum_{i=1}^N x_{i,j}^2$$

$$c_j = 2 \sum_{i=1}^N (\mathbf{w}_{-j}^T \mathbf{x}_{i,-j} - y_i) x_{i,j}$$

第 j 维特征与残差的相关性

利用去掉第 j 维后其他特征得到的预测的残差

注意： \mathbf{w}_{-j} 表示向量 \mathbf{w} 中去掉第 j 维的后向量； \mathbf{x}_{-j} 类似

➤ Lasso的坐标轴下降法求解

■ 次第 j 维对应目标函数次微分为

$$\partial_{J_{w_j}} \partial J(\mathbf{w}, \lambda) = (a_j w_j + c_j) + \lambda \frac{\partial \|\mathbf{w}\|_1}{\partial w_j} = \begin{cases} \{a_j w_j + c_j - \lambda\} & \text{如果 } w_j < 0 \\ [c_j - \lambda, c_j + \lambda] & \text{如果 } w_j = 0 \\ \{a_j w_j + c_j + \lambda\} & \text{如果 } w_j > 0. \end{cases}$$

$$\hat{w}_j(c_j) = \begin{cases} (-c_j + \lambda)/a_j & \text{如果 } c_j < -\lambda \\ 0 & \text{如果 } c_j \in [-\lambda, \lambda] \\ (-c_j - \lambda)/a_j & \text{如果 } c_j > \lambda \end{cases}$$

c_j 为预测残差与特性 x_j 的相关性
当 c_j 绝对值比较小 ($\leq \lambda$) 时, $w_j = 0$

稀疏：特征选择

$$a_j = 2 \sum_{i=1}^N x_{i,j}^2$$

$$c_j = 2 \sum_{i=1}^N (\mathbf{w}_{-j}^T \mathbf{x}_{i,-j} - y_i) x_{i,j}$$

➤ Lasso的坐标轴下降法求解

- 1. 计算 $a_j = 2 \sum_{i=1}^N x_{i,j}^2$;
- 2. 初始化参数 w (全0或随机) ;
- 3. 循环直到收敛 : 选择变化幅度最大的维度或者轮流更新 w_j :
 - 3.1 计算 $c_j = 2 \sum_{i=1}^N (w_{-j}^T x_{i,-j} - y_i) x_{i,j}$;
 - 3.2 计算 $\hat{w}_j = \begin{cases} (-c_j + \lambda)/a_j & \text{如果 } c_j < -\lambda \\ 0 & \text{如果 } c_j \in [-\lambda, \lambda] \\ (-c_j - \lambda)/a_j & \text{如果 } c_j > \lambda \end{cases}$;

注意 : 当特征与预测残差弱相关 , 即 $c_j \in [-\lambda, \lambda]$ 时 , $w_j = 0$ 。

当 $\lambda \geq \max_j (x_{:,j}^T y)$ 时 , 所有 $w_j = 0$ 。

其中 $x_{:,j}$ 表示所有样本第 j 维的特征值 , y 表示所有样本的标签 。

>> 回忆Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')
```

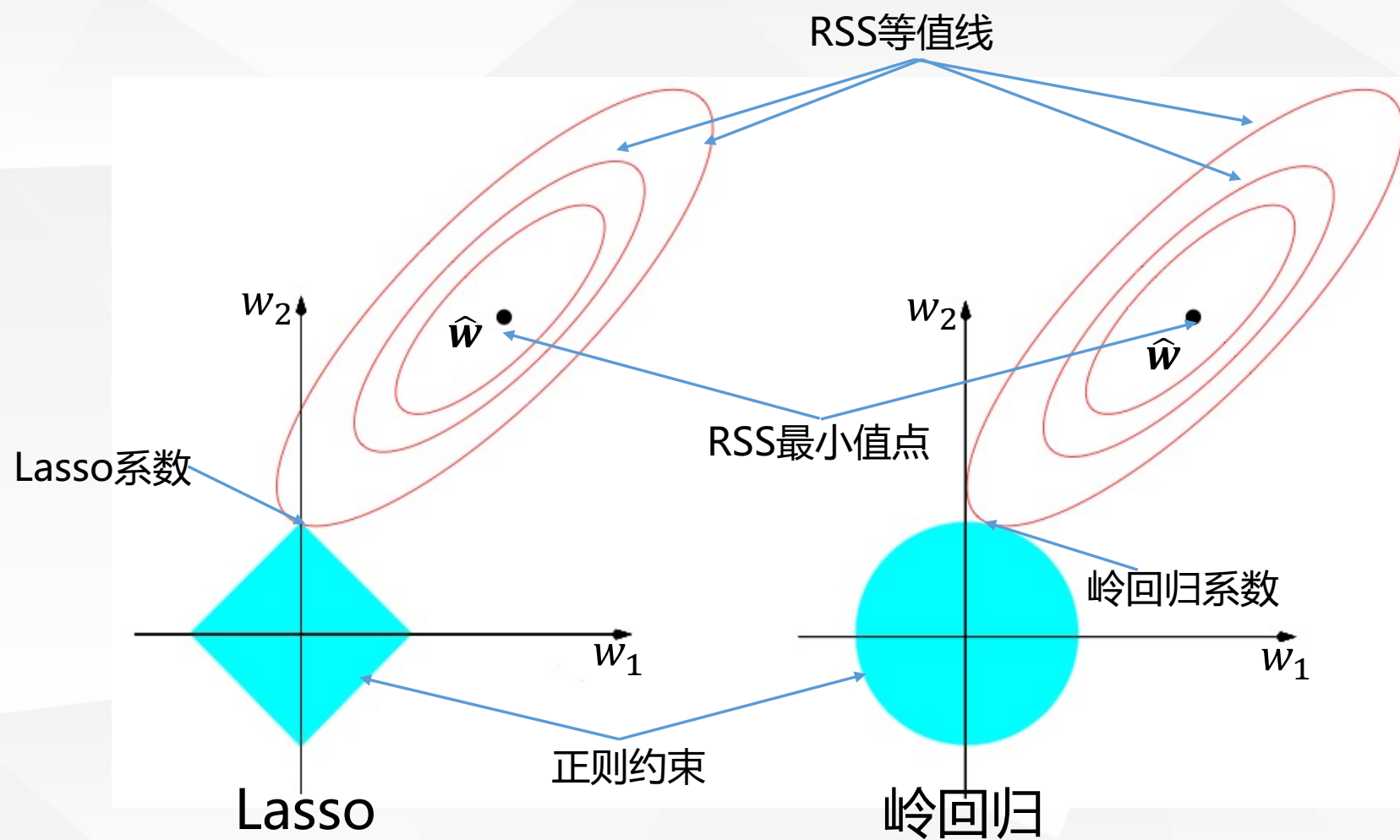
优化计算有关参数

参数	说明
<i>precompute</i>	是否使用预计算的 Gram 矩阵来加速计算。可取值：True, False, 'auto' 或数组 (array-like)。如果设置为 'auto' 则机器决定。
<i>max_iter</i>	最大迭代次数。
<i>tol</i>	解的精度，判断迭代收敛与否的阈值。 当更新量小于tol时，优化代码检查优化的 dual gap 并继续直到小于 tol 为止。
<i>warm_start</i>	是否从之前的结果继续。初始值可以是之前的训练结果，支持在线学习。初始值可在fit函数中作为参数传递。
<i>positive</i>	是否强制使系数为正。
<i>random_state</i>	随机选择特征的权重进行更新的随机种子。 当参数selection == 'random' 有效。
<i>selection</i>	选择特征权重更新的方式，可选项有：'cyclic'（循环更新），'random'（随机选择特征进行更新，通常收敛更快，尤其当参数 $tol > 10^{-4}$ 时）。

➤ 正则小结

- L2正则使得线性回归系数收缩，模型稳定。
- 当输入特征之间存在共线性时使用L2正则。
- L1正则也会收缩回归系数。当正则参数取合适值时，L1正则使得有些线性回归系数为0，得到稀疏模型。
- 当输入特征多，有些特征与目标变量之间相关性很弱时，L1正则可能只选择强相关的特征，模型解释性好。
- 注意：由于正则项中对不同维度的 w_j 同等对待，对输入特征 x 最好做去量纲（scaling）处理，使得不同维度的特征取值范围大致相同。

➤ L2正则 vs. L1正则



➤ 补充：拉格朗日乘子法与卡罗需-库恩-塔克（Karush–Kuhn–Tucker, KKT）条件



Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的的优化算法
- 回归任务的模型性能评价
 - 回归任务的模型性能评价指标
 - 线性回归模型超参数调优
- 线性回归案例分析

➤ 回归模型的评价指标

- 开方均方误差 (Rooted Mean Squared Error , RMSE)

$$\text{RMSE}(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2}$$

- 平均绝对误差 (Mean Absolute Error , MAE)

$$\text{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^N |\hat{y}_i - y_i|$$

- 绝对误差中值 (Median Absolute Error , MedAE)

$$\text{MedAE}(\hat{\mathbf{y}}, \mathbf{y}) = \text{median}(|\hat{y}_1 - y_1|, \dots, |\hat{y}_N - y_N|)$$

- 平均平方log误差 (Mean Squared Logarithmic Error, MSLE)

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

当 y 呈指数增长时可以使用 (如计数、一年的平均销量...)

➤ 回归模型的评价指标

- R^2 分数：既考虑了预测值与真值之间的差异，也考虑了问题本身真值之间的差异（ Scikit-Learn 回归模型的默认评价准则 ）

模型预测的MSE

$$SS_{res}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$
$$R^2(\hat{\mathbf{y}}, \mathbf{y}) = 1 - \frac{SS_{res}(\mathbf{y}, \hat{\mathbf{y}})}{SS_{tot}(\mathbf{y})}$$

$$SS_{tot}(\mathbf{y}) = \frac{1}{N} \sum_{i=1}^N (\bar{y} - y_i)^2$$

误差减少比例

用样本均值 \bar{y} 来做预测的MSE

（ 不考虑输入特征的影响，最简单粗暴的预测 ）

- 已解释的方差分数（ Explained variance score ）：最佳分数为1

$$\text{explained_variance}(\hat{\mathbf{y}}, \mathbf{y}) = 1 - \frac{\text{Var}(\hat{\mathbf{y}} - \mathbf{y})}{\text{Var}\{\mathbf{y}\}}$$

当残差的均值为0时，二者相同

➤ sklearn中的评价模型性能的方式

- estimator的score方法：每个学习器都有score方法，提供一个缺省的评估方法（回归为 R^2 分数）。
- Metric：metrics模块实现了一些函数，用来评估预测误差。
- Scoring参数：使用交叉验证评估模型的工具具有Scoring 参数。

<code>metrics.explained_variance_score(y_true, y_pred)</code>	Explained variance regression score function
<code>metrics.mean_absolute_error(y_true, y_pred)</code>	Mean absolute error regression loss
<code>metrics.mean_squared_error(y_true, y_pred[, ...])</code>	Mean squared error regression loss
<code>metrics.mean_squared_log_error(y_true, y_pred)</code>	Mean squared logarithmic error regression loss
<code>metrics.median_absolute_error(y_true, y_pred)</code>	Median absolute error regression loss
<code>metrics.r2_score(y_true, y_pred[, ...])</code>	R ² (coefficient of determination) regression score function.

➤ Scikit-Learn中的Scoring参数

Regression

'explained_variance'	<u>metrics.explained_variance_score</u>
'neg_mean_absolute_error'	<u>metrics.mean_absolute_error</u>
'neg_mean_squared_error'	<u>metrics.mean_squared_error</u>
'neg_mean_squared_log_error'	<u>metrics.mean_squared_log_error</u>
'neg_median_absolute_error'	<u>metrics.median_absolute_error</u>
'r2'	<u>metrics.r2_score</u>

同metrics——对应

➤ 线性回归模型的超参数调优

■ 线性回归模型的超参数为正则系数 λ

- 岭回归：sklearn中的实现为RidgeCV
- Lasso：sklearn中的实现为LassoCV
- 弹性网络：sklearn中的实现为ElasticNetCV

■ 超参数调优方法：

- 交叉验证
 - 岭回归（RidgeCV）：广义留一交叉验证
- 信息准则：BIC/AIC
 - 计算快，无需留出验证集（或重复 K 次交叉验证）

➤ 特殊的交叉验证：留一交叉验证

■ 当交叉验证的折数 $K = N$ 时，因为每次留出一个样本做验证集，我们称之为留一交叉验证。

- 折数更多，计算代价越高，通常当样本数非常少使用。

■ 对线性模型，可采用广义交叉验证（Generalized Cross Validation, GCV）近似留一交叉验证，极大降低交叉验证的计算量。

- Scikit-Learn 中岭回归（RidgeCV）采用 GCV 确定正则超参数。

$$GCV(\lambda) = \frac{1}{N} \sum_{i=1}^N \left(\frac{f(\mathbf{x}_i) - y_i}{1 - \frac{df(\lambda)}{N}} \right)^2, df(\lambda) = \sum_{j=1}^D \frac{\sigma_j^2}{\sigma_j^2 + \lambda}$$

其中 σ_j^2 为矩阵 $\mathbf{X}^T \mathbf{X}$ 的特征值

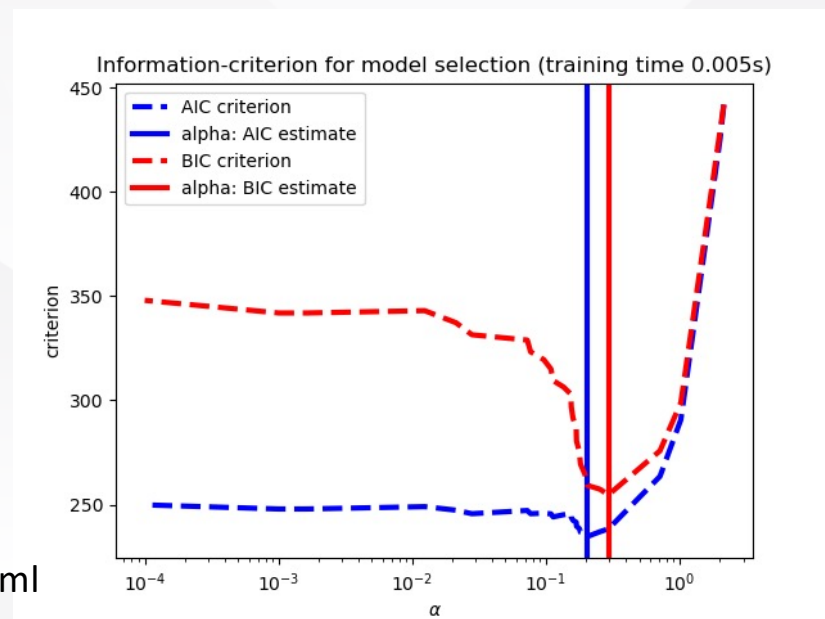
>> BIC/AIC

■ AIC : $AIC(\lambda) = N \ln(RSS(\mathbf{w}, \lambda)) + 2df(\lambda)$

■ BIC : $BIC(\lambda) = N \ln(RSS(\mathbf{w}, \lambda)) + \ln(N)df(\lambda)$

■ AIC/BIC : 依赖于对自由度的正确估计

- 假设模型必需是正确, 而且是对大样本 (渐近结果) 进行推导。
- 当问题是欠定时 (特征数大于样本数) 会崩溃



https://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html
#sphx-glr-auto-examples-linear-model-plot-lasso-model-selection-py

■ RidgeCV : 在一组正则参数 *alphas* 中找最佳的 *alpha*

`class sklearn.linear_model.RidgeCV(alphas=(0.1, 1.0, 10.0), fit_intercept=True, normalize=False, scoring=None, cv=None, gcv_mode=None, store_cv_values=False)`

- 与CV有关的参数：
 - *scoring*: 评价指标
 - *cv*: 交叉校验划分策略。默认是None，采用高效的留一交叉验证。
 - *gcv_mode*: 广义留一交叉验证的模式，可选：None, 'auto', 'svd', 'eigen'。

'auto' /None	当n_features > n_samples或X 为稀疏矩阵时，用 " eigen" ； 否则用 'svd' 。自动选择更经济的计算方式。
'svd'	用X的SVD分解 (对稀疏矩阵不适用)
'eigen'	$X^T X$ 的特征值分解

- *store_cv_values*: 是否将每个alpha对应的交叉验证的值存储在属性cv_values_ 中。

```
class sklearn.linear_model.LassoCV(eps=0.001, n_alphas=100, alphas=None, fit_intercept=True,
normalize=False, precompute='auto', max_iter=1000, tol=0.0001, copy_X=True, cv=None, verbose
=False, n_jobs=1, positive=False, random_state=None, selection='cyclic')
```

- 与CV有关的参数：
- Lasso的alphas可以通过两种方式设置：
 - 1. 设置参数eps和n_alphas，参数alphas为None：alpha超过一定值后，所有回归系数为0。所以算法可以自动确定alpha的最大值 α_{\max} ，再根据参数eps的值和 $\alpha_{\min}/\alpha_{\max} = \text{eps}$ ，确定alpha的最小值 α_{\min} ；最后在 α_{\min} 到 α_{\max} 区间的 \log_{10} 均匀取n_alphas个alpha：
 $\text{logspace}(\text{np.log10}(\alpha_{\max} * \text{eps}), \text{np.log10}(\alpha_{\max}))$ 。
 - 2. 同RidgeCV一样，人为设定alphas
 - cv: 交叉校验划分策略。默认是None，表示采用3折交叉验证

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
- 回归任务的模型性能评价
- 线性回归案例分析
 - 案例：广告费用与产品销量
 - 案例：共享单车骑行量预测

➤ 广告费用与产品销量

■ 数据分析：

- 200个样本，没有缺失值，看不出来有噪声数据
- 每个样本有3个数值型特征：TV, radio, newspaper
- 标签：产品销量
- 特征与特征之间的相关性不太高

■ 特征工程：

- 特征均为数值型特征

➤ 广告费用与产品销量

■ 数据分析： chp5_1_EDA_advertising.ipynb

- 200个样本，没有缺失值，看不出来有噪声数据
- 每个样本有3个数值型特征： TV, radio, newspaper
- 标签：产品销量
- 特征与特征之间的相关性不太高
- 特征TV, radio与标签相关性较高，特征newspaper相关性不太高

■ 特征工程： chp5__FE_advertising.ipynb

- 特征均为数值型特征，暂时不做过多处理，只是对特征做标准化处理，使得各特征处理后均为0均值、1标准差（并不要求是正态分布）

广告费用与产品销量

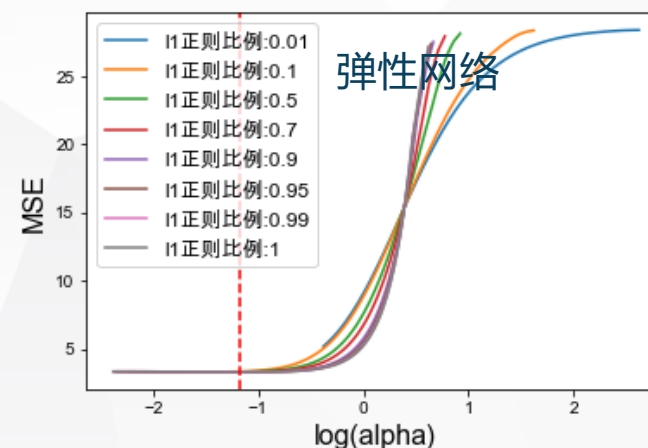
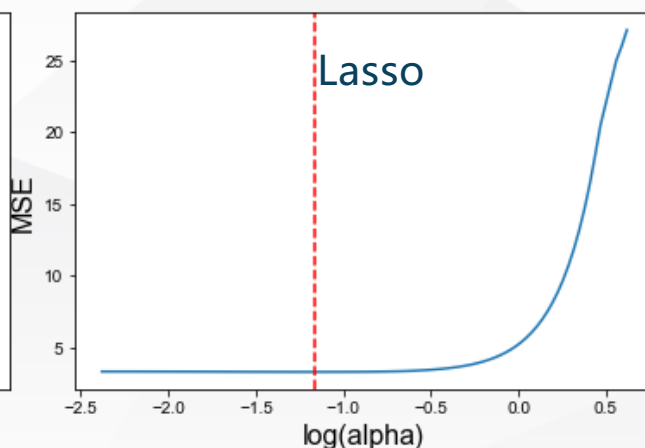
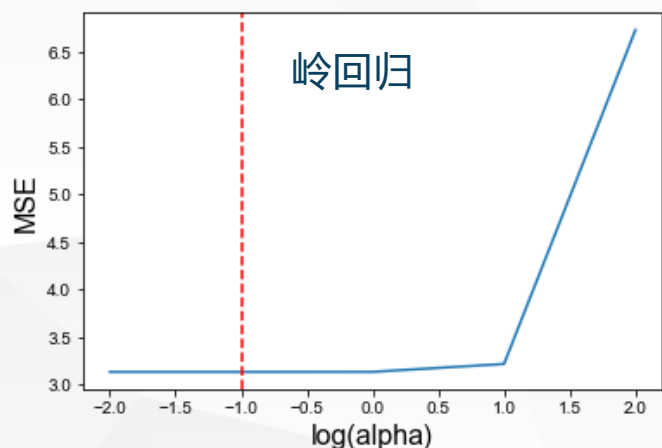
模型

- 最小二乘线性回归 (OLS)
- 岭回归：正则参数 λ
- Lasso：正则参数 λ
- 弹性网络：正则参数 λ 和L1正则比例

chp5_3_train_test_advertising.ipynb

数据集中随机80%样本为训练数据

超参数调优评价指标：MSE（红色竖线为最佳超参数）



➤ 广告费用与产品销量

■ 数据集中随机选择20%样本作为测试数据，其他80%样本为训练数据

广告数据集上不同线性回归模型的系数

特征	最小二乘线性回归系数	岭回归系数	Lasso系数	弹性网络系数
TV	3.983944	3.981524	3.921642	3.921642
radio	2.860230	2.858304	2.806374	2.806374
newspaper	0.038194	0.038925	0.000000	0.000000
截距项	13.969091	13.969282	13.972528	13.972528

权值收缩：岭回归、Lasso、弹性网络的回归系数的绝对值比OLS的小
Lasso的稀疏性：特征newspappwer的系数为0

➤ 广告费用与产品销量

广告数据集上不同线性回归模型的性能
(R方分数)

数据集	最小二乘 线性回归	岭回归	Lasso	弹性网络
训练集上性能	0.896285	0.896285	0.895925	0.895925
测试集上性能	0.893729	0.893865	0.899197	0.899197

最小二乘线性回归模型在训练集上性能最好，但在测试集上性能最差
(过拟合)

Lasso模型/弹性网络在测试集上性能最好

Outline

- 回归任务简介
- 线性回归模型
- 回归任务的损失函数
- 线性回归模型的正则项
- 线性回归的优化算法
- 回归任务的模型性能评价
- 线性回归案例分析
 - 数据探索分析
 - 数据编码和预处理
 - 案例：广告费用与产品销量
 - 案例：共享单车骑行量预测

➤ 共享单车骑行量预测

■ 数据分析：chp5_1_EDA_bikesharing.ipynb

- 两年每天的骑行量以及当天的天气情况
- 731个样本，没有缺失值
- 每个样本有12个特征：
 - 时间信息：年、季节、月份、日期、星期、是否为工作日、是否为节假日
 - 天气情况：天气概况（晴、阴、小雨雪、大雨雪）、湿度、温度、体感温度、风速
- 标签：骑行量
- 有些特征相关性很高（季节与月份、温度与体感温度），冗余大

➤ 共享单车骑行量预测

■ 特征工程：chp6_2_FE_bikesharing.ipynb

- 数值型特征做标准化处理：湿度、温度、体感温度、风速特征
- 类别型特征取值均不多，独热编码：季节、月份、星期、天气概况
- 二值特征编码为0/1：年份、是否为工作日、是否为节假日
- 每个日期只有2个样本，所以去掉特征“日期”

共享单车骑行量预测

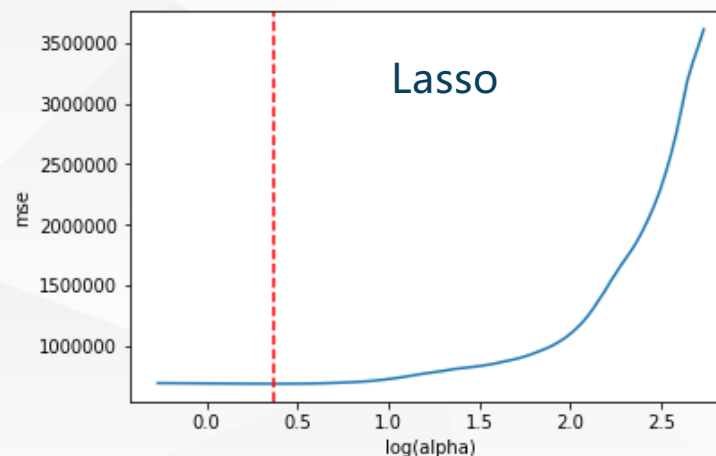
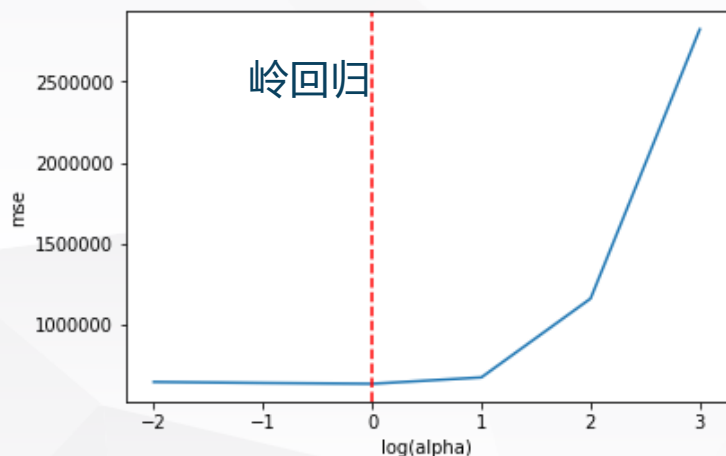
■ 模型

- 最小二乘线性回归 (OLS)
- 岭回归：正则参数 λ
- Lasso：正则参数 λ
- 弹性网络：正则参数 λ 和L1正则比例

3_Train_Test_Advertising.ipynb

■ 数据集中随机80%样本为训练数据

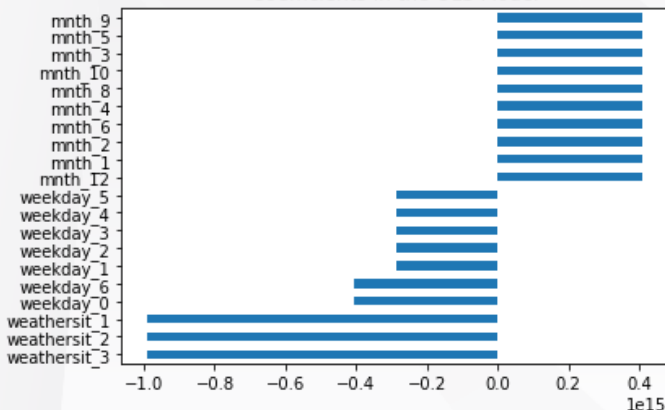
■ 超参数调优评价指标：MSE（红色竖线为最佳超参数）



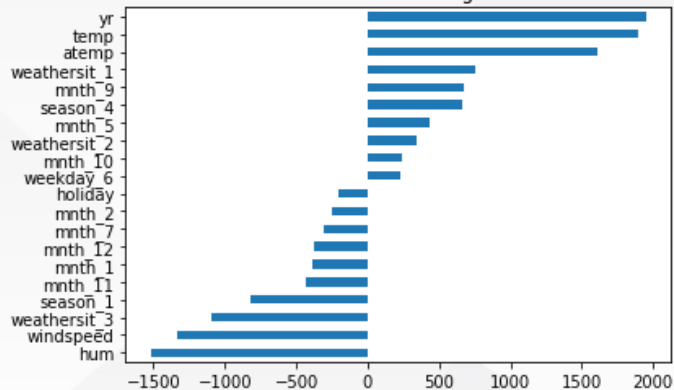
共享单车骑行量预测

■ 数据集中随机选择20%样本作为测试数据，其他80%样本为训练数据

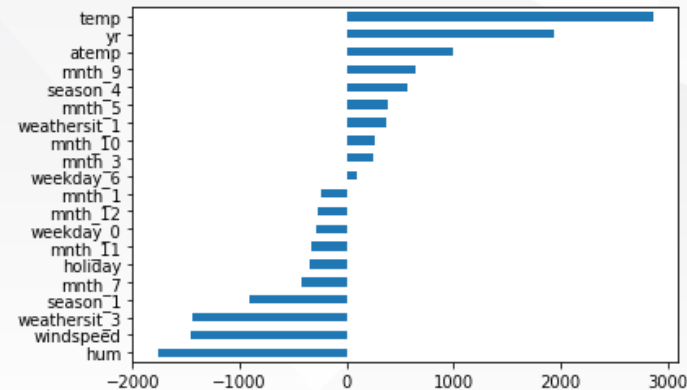
Coefficients in the OLS Model



Coefficients in the Ridge Model



Coefficients in the Lasso Model



权值收缩：岭回归、Lasso、弹性网络的回归系数的绝对值比OLS的**小得多**
Lasso的稀疏性：6个特征的系数为0

共享单车骑行量预测

共享单车数据集上不同线性回归模型的性能
(RMSE)

数据集	最小二乘线性回归	岭回归	Lasso
训练集上性能	752.257390	754.036662	752.643468
测试集上性能	785.595792	776.975361	784.878890

最小二乘线性回归模型在训练集上性能最好，但在测试集上性能最差
(过拟合)

岭回归在测试集上性能最好

➤ 本章总结

■ 线性回归

- 函数集合：输入特征的线性组合
- 目标函数
 损失函数：L2损失、L1损失、Huber损失
 正则项：L2正则、L1正则
- 优化方法：解析法、**梯度下降**、坐标下降

■ Sklearn中的线性回归实现

- LinearRegression
- Ridge/RidgeCV
- Lasso/LassoCV
- HuberRegression
- SGDRegression