第5章 线性回归

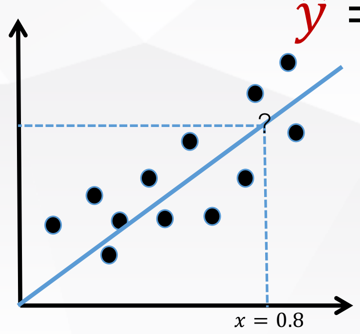
Outline

- 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
- 线性回归案例分析

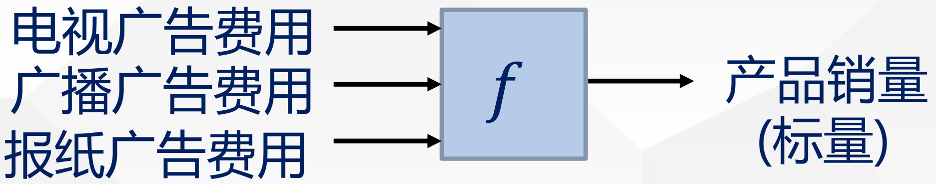
回归

- 监督学习中，给定训练数据$\mathcal{D}=\{\boldsymbol{x}_i,y_i\}_{i=1}^N$，其中$N$为训练样本数目，$i$为样本索引，$\boldsymbol{x}_i$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$\boldsymbol{x}$到输出y的映射f。
- 测试：对新的测试数据$\boldsymbol{x}$ ，用学习到的映射f对其进行预测：$\hat{y}=f(\boldsymbol{x})$

$$\hat{y}=f(\boldsymbol{x})$$



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

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

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

线性回归

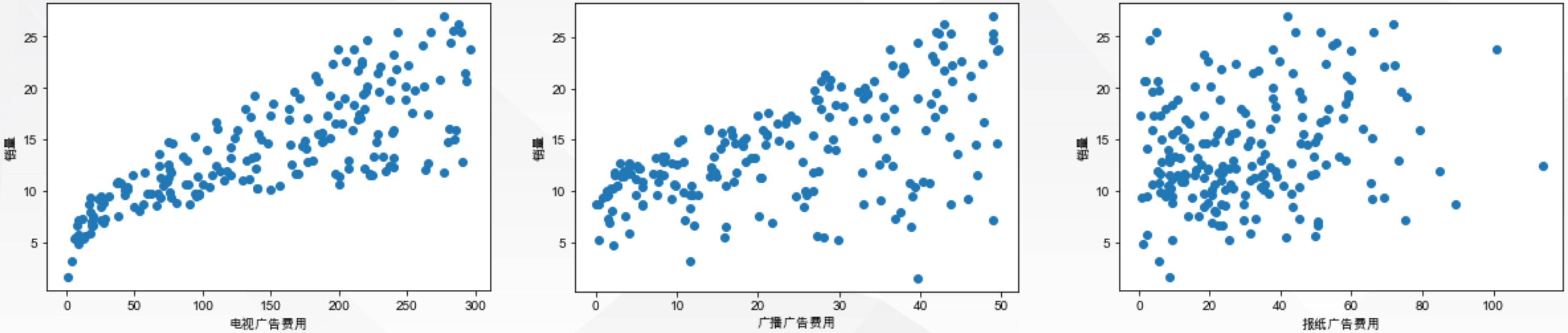
- 机器学习的三要素
  - 1. 函数集合$\{f_1, f_2, \dots\}$
  - 2. 目标函数$J(f)$：函数的好坏
  - 3. 优化算法：找到最佳函数
- 线性回归：函数集合为输入特征的线性组合，即假设输出y与输入$\boldsymbol{x}$之间的关系为线性关系

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

- D: 特征维数，j为特征索引
- $\boldsymbol{x} = (1, x_1, \dots, x_D)^{\mathrm{T}}$：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：

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

损失函数

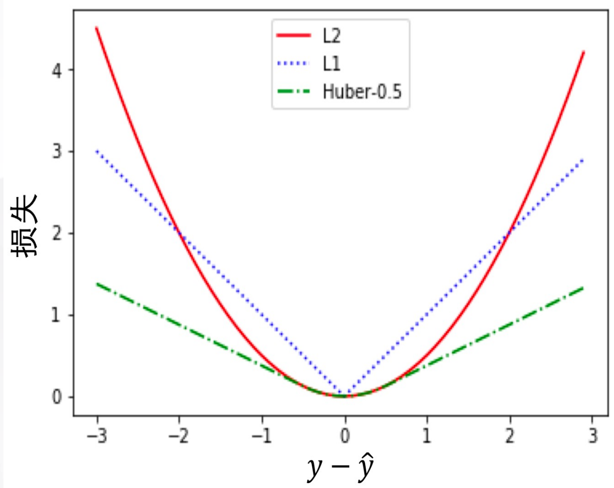
■对回归问题，最常用的损失函数为L2损失，即预测残差的平方：

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

其中 预测残差$r = y - \hat{y}$

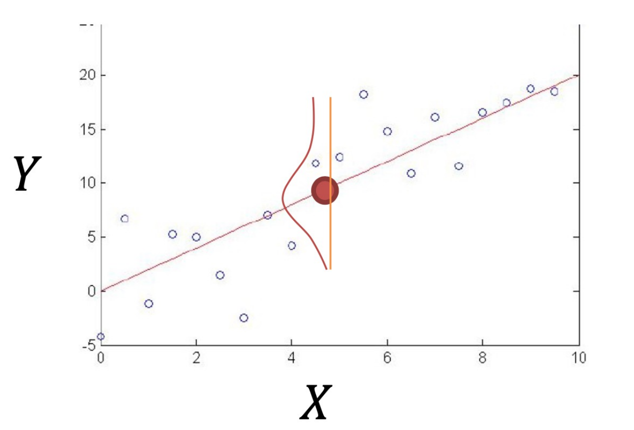
■ 训练集上所有训练数据的预测残差的平方和记为（ Residual Sum of Squares，RSS ）

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



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
- $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$
- $y_i | \boldsymbol{x}_i \sim N(f(\boldsymbol{x}_i), \sigma^2)$
- 所以 $p(y_i|\boldsymbol{x}_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y_i - f(\boldsymbol{x}_i)\right)^2}{2\sigma^2}\right)$



注意：在给定$\boldsymbol{x}_i$的情况下，$f(\boldsymbol{x}_i)$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：

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

- 则log似然函数为

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

L2损失函数的概率解释

- log似然

$$\ell(f) = -\frac{N}{2}\ln(2\pi) - N\ln\sigma - \sum_{i=1}^{N}\frac{\left(y_i - f(\boldsymbol{x}_i)\right)^2}{2\sigma^2}$$

- 取最大值时，$-\sum_{i=1}^{N}\frac{\left(y_i-f(\boldsymbol{x}_i)\right)^2}{2\sigma^2}$取最大值，从而$\sum_{i=1}^{N}\frac{\left(y_i-f(\boldsymbol{x}_i)\right)^2}{2\sigma^2}$取最小值。
- $\sum_{i=1}^{N}\left(y_i - f(\boldsymbol{x}_i)\right)^2 = \mathrm{RSS}(f)$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）。

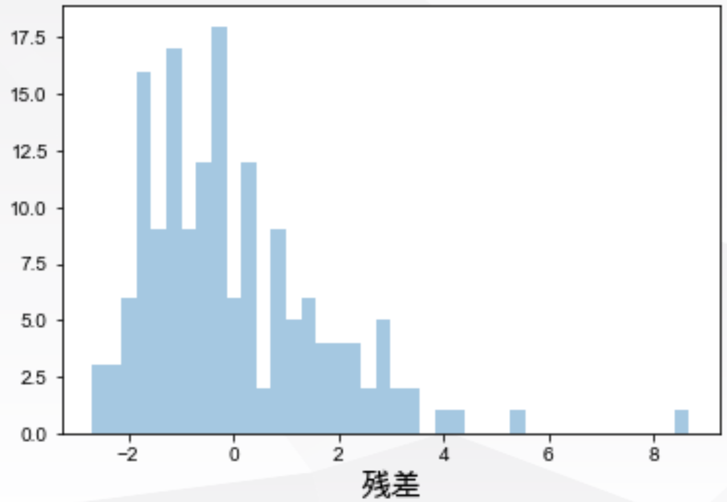
负log似然损失

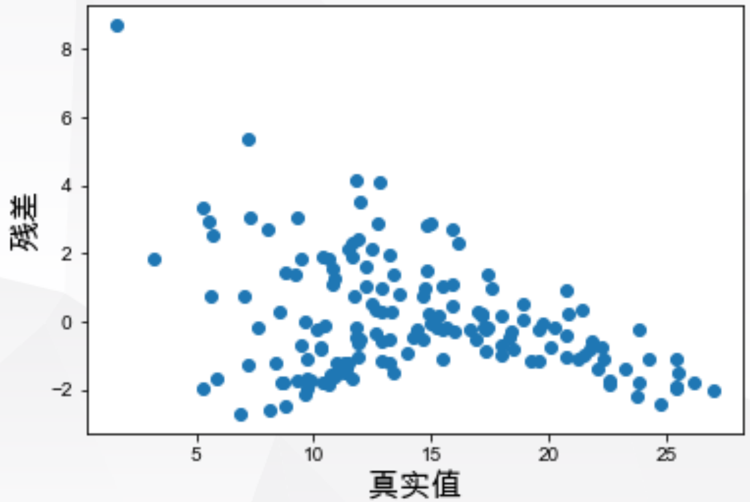
- 极大似然估计等价于负log似然最小，因此负log似然也被称为一种损失函数：负log似然损失。

- 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





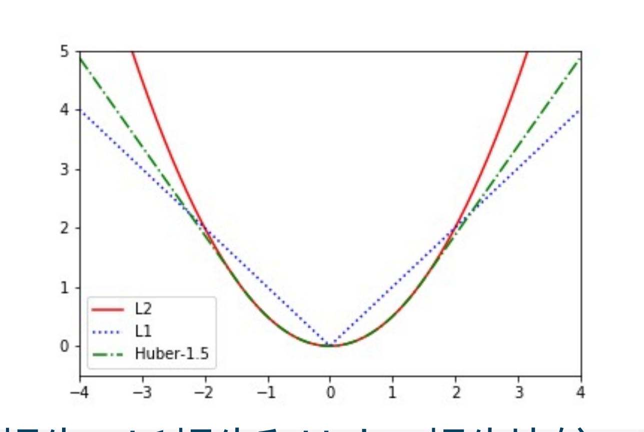
残差的分布并不符合0均值的正态分布
该模型（最小二乘回归模型）预测效果并不好

从真实值和残差的散点图来看，真实值较小和较大时，预测残差大多<0,其余情况残差大多>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

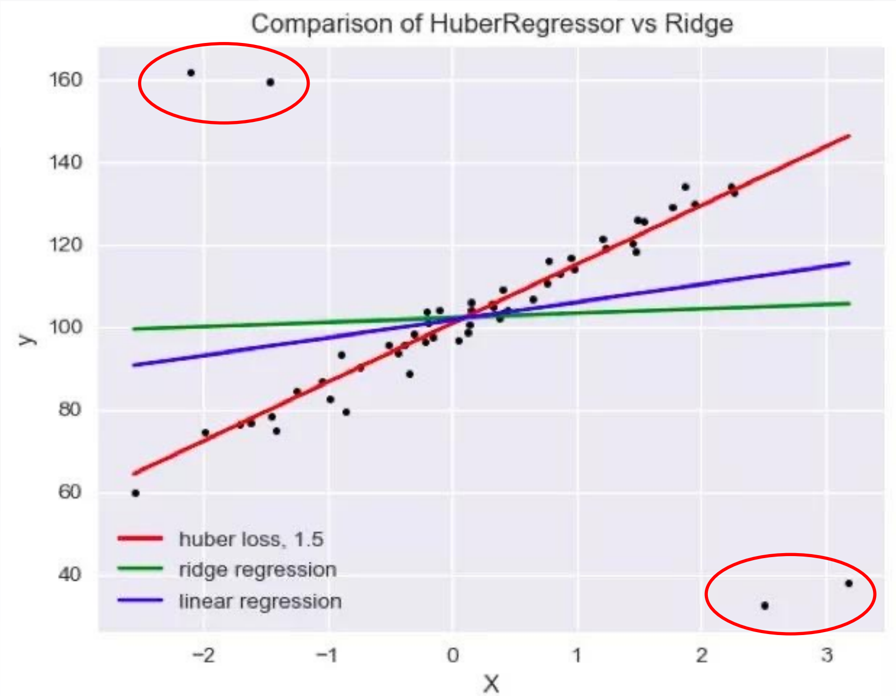
胡伯（Huber）损失函数

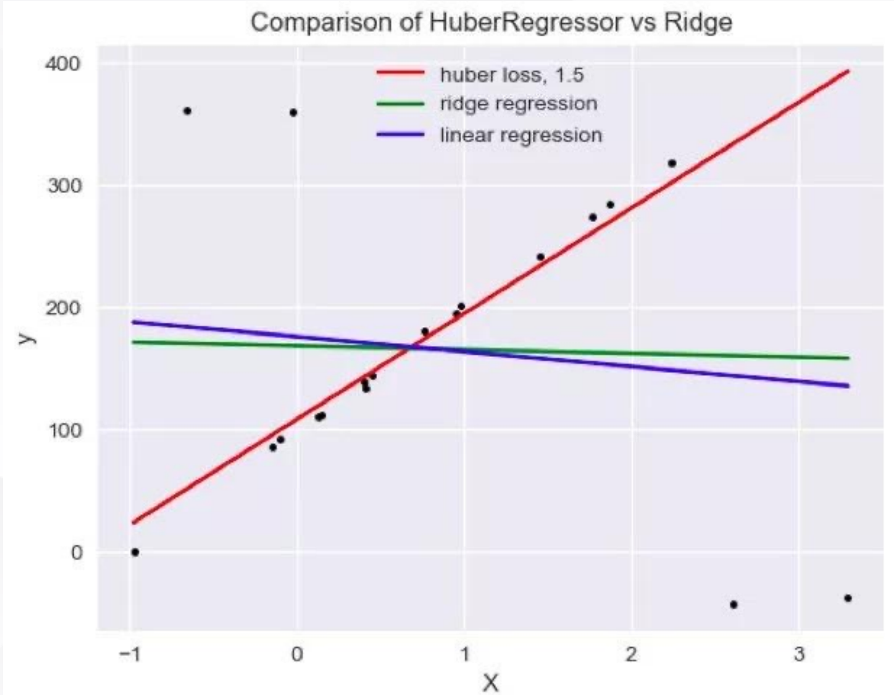
- L2损失的好处是处处连续，优化求解方便。
- 但L2损失对噪声点敏感（当预测残差r的绝对值较大时，r^2很大）。
- 考虑L1（$|r|$）损失函数，但$|r|$在$r=0$处不连续，优化求解麻烦。
- 因此综合L1和L2损失函数，得到Huber损失：

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



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

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

Outline

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：

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

- 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

- $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$，正则项$R(f) = R(\boldsymbol{w})$
- L2正则：$R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{D} w_j^2$
- L1正则：$R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_{j=1}^{D} |w_j|$
- 其中$\boldsymbol{w}$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：$\Delta \boldsymbol{x} \rightarrow \Delta y$）

$$\Delta y = f(\boldsymbol{x} + \Delta \boldsymbol{x}) - f(\boldsymbol{x}) = \left(\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{x} + \Delta \boldsymbol{x}) + b\right) - \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b\right) = \boldsymbol{w}^{\mathrm{T}}\Delta \boldsymbol{x}$$

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：

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

- 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*)

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

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*)

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

属性	说明	备注
coef_	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。
intercept_	截距项。	

方法	说明	备注
fit(X, y[, sample_weight])	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（如精度度不同），可设置样本权重。
predict(X)	使用训练好的模型进行预测，返回预测值	
score(X, y[, sample_weight])	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：$L(f(\boldsymbol{x}_i,\boldsymbol{w}),y_i)=(f(\boldsymbol{x}_i,\boldsymbol{w})-y_i)^2$
- 正则项取L2正则：$R(\mathbf{w})=\sum_{j=1}^{D}w_j^2$
- 线性回归模型：$f(\boldsymbol{x};\boldsymbol{w})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}=\sum_{j=0}^{D}w_jx_j \quad (x_0=1)$
- 得到岭回归的目标函数为：

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

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

岭回归

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

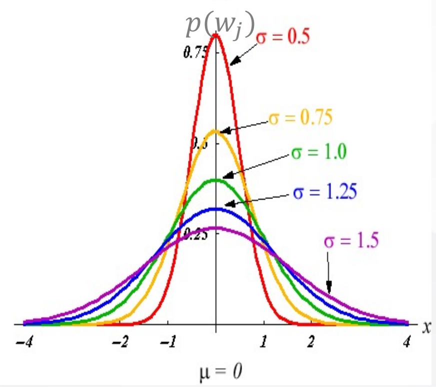
- 回忆：贝叶斯估计
  - 数据的似然：$\ell(f)=\ln p(\mathcal{D}|\boldsymbol{\theta})=\sum_{i=1}^{N}\ln p(y_i|\boldsymbol{x}_i)$
  - 参数的先验： $p(\boldsymbol{\theta})$
  - 参数的后验：$p(\boldsymbol{\theta}|\mathcal{D})\propto p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})$
  - 参数的最大后验估计： $\hat{\boldsymbol{\theta}}=\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\,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

岭回归

■岭回归的贝叶斯估计

- 数据的似然： $\ell(f)=\sum_{i=1}^{N}\ln p(y_i|\boldsymbol{x}_i)-\frac{N}{2}\log(2\pi)-N\ln\sigma-\sum_{i=1}^{N}\frac{\left(y_i-f(\boldsymbol{x}_i)\right)^2}{2\sigma^2}$
- 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：

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

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

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

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

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

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

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

■等价于岭回归的目标函数（去掉负号，取最大变成取最小）：

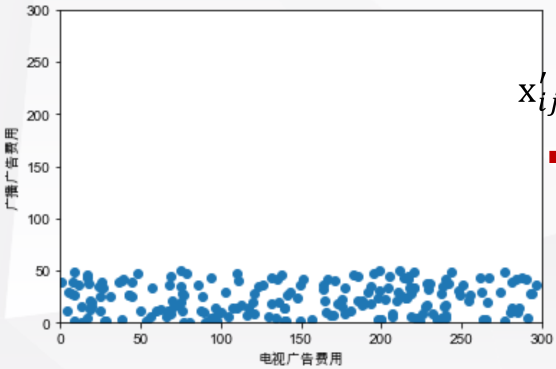
$$J(\boldsymbol{w},\lambda) = (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + \lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$$

特征缩放

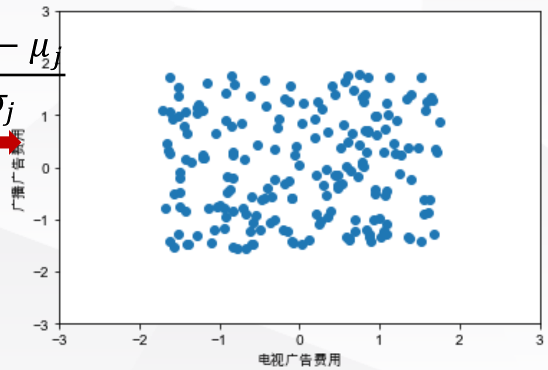
- 在正则项中，各特征的参数地位相同，这就要求各特征的单位相同，这可以对特征进行缩放（去量纲）实现。
- 最常用的特征缩放为特征标准化
  - sklearn中的实现：StandardScaler

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

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



$$x'_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$




```
# 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$J(\mathbf{w},\lambda) = (\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}) + \lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$中的$\lambda$
- 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：$L(f(\boldsymbol{x}_i;\boldsymbol{w}),y_i)=(f(\boldsymbol{x}_i,\boldsymbol{w})-y_i)^2$
- 正则项取L1正则：$R(\boldsymbol{w})=\sum_{j=1}^{D}|w_j|$
- 线性回归模型：$f(\boldsymbol{x},\boldsymbol{w})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}=\sum_{j=0}^{D}w_jx_j \quad (x_0=1)$
- 得到Lasso (Least Absolute Shrinkage and Selection Operator)的目标函数为：

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

$$=\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\|_2^2+\lambda\|\boldsymbol{w}\|_1$$

Lasso

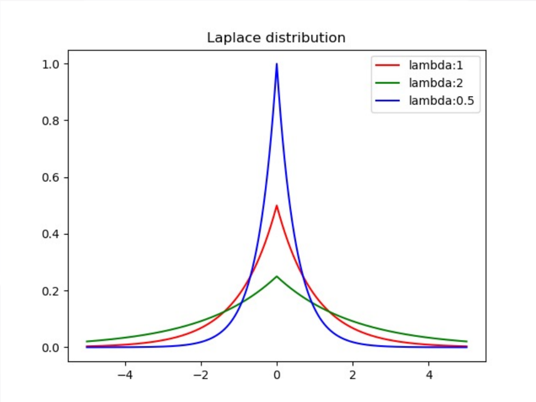
- Lasso的目标函数$J(\boldsymbol{w},\lambda) = \|\boldsymbol{Xw}-\boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$
- L1正则可视为参数先验分布为拉普拉斯分布的贝叶斯估计。

Lasso

■贝叶斯估计

- 数据的似然： $\ell(f)=\sum_{i=1}^{N}\ln p(y_i|\boldsymbol{x}_i)-\frac{N}{2}\ln(2\pi)-N\ln\sigma-\sum_{i=1}^{N}\frac{\left(y_i-f(\boldsymbol{x}_i)\right)^2}{2\sigma^2}$
- 参数的先验：$w_j\sim\mathrm{Laplace}(0,b)$ ，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：

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

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

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

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

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

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

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{2\sigma^2}{b} \sum_{j=1}^{D} |w_j| + \sum_{i=1}^{N} \left(y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right)^2 \right)$$ （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）：

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

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 $J(\mathbf{w},\lambda) = 1/2N \, \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$中的$\lambda$
- 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：

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

- 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： $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, J(\boldsymbol{w})$
- 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：$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \boldsymbol{0}$
- 若二阶海森（Hessian）矩阵

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

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：

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

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

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

- 梯度为：

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

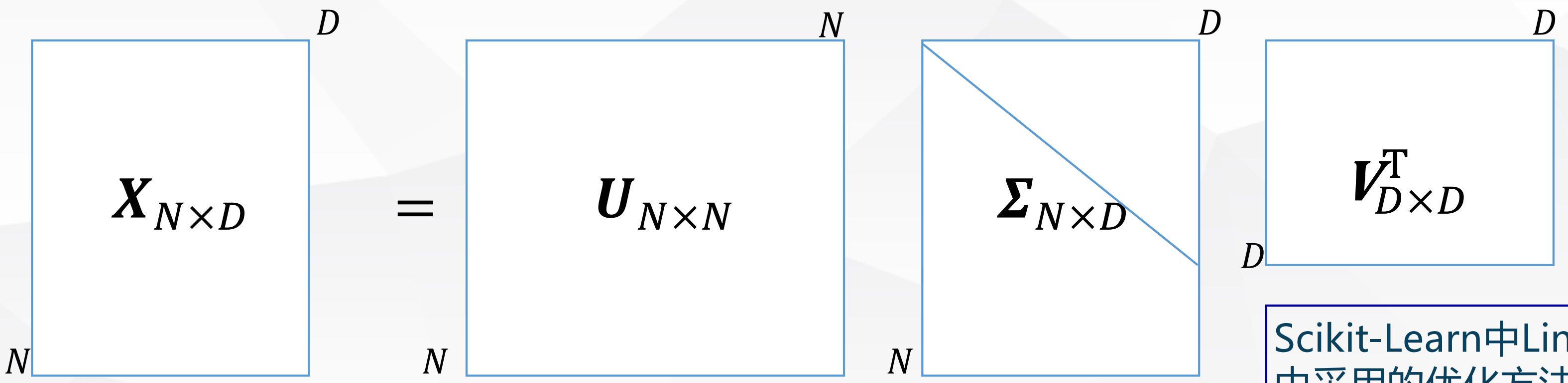
- 从而：

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \rightarrow \hat{\boldsymbol{w}}_{OLS} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

正规方程组(Normal Equations)

OLS的最优解： SVD

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



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

OLS的最优解： SVD

- OLS目标函数：$J(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2$。
- $\boldsymbol{X}$的SVD：$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$
- 由于正交变换$\boldsymbol{U}^{\mathrm{T}}$保范，

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

- 令$\boldsymbol{y}' = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}$，$\boldsymbol{w}' = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{w}$，则问题变成最小化$\|\boldsymbol{\Sigma}\boldsymbol{w}' - \boldsymbol{y}'\|_2^2$。
- 由于$\boldsymbol{\Sigma}$为准三角矩阵，上述问题变得非常简单。

OLS的最优解：SVD

- 令$\boldsymbol{y}' = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}$，$\boldsymbol{w}' = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{w}$，则问题变成最小化$\|\boldsymbol{\Sigma w}' - \boldsymbol{y}'\|_2^2$。
- $\boldsymbol{\Sigma}$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$\boldsymbol{X}$ 接近不满秩）时，w_j'数值不稳定。
  - 矩阵$\boldsymbol{X}$ 接近不满秩：特征之间存在共线性（有冗余）
- 再根据$\boldsymbol{w}' = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{w}$，得到$\boldsymbol{w} = \boldsymbol{V}^{-\mathrm{T}}\boldsymbol{w}' = \boldsymbol{V}\boldsymbol{w}'$。
- 或者：

$$\boldsymbol{X}^{\ddagger} = \boldsymbol{V\Sigma}^{\ddagger}\boldsymbol{U}^{\mathrm{T}} = \boldsymbol{V}\begin{bmatrix} 1/\sigma_1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & 0 \\ \vdots & 0 & 1/\sigma_r & 0 & 0 \\ \vdots & \vdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots \end{bmatrix}\boldsymbol{U}^{\mathrm{T}}$$

$$\widehat{\boldsymbol{w}}_{OLS} = \boldsymbol{X}^{\ddagger}\boldsymbol{y} = \boldsymbol{V\Sigma}^{\ddagger}\boldsymbol{U}^{\mathrm{T}}\ \boldsymbol{y}$$

岭回归的最优解： SVD

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

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

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

- 将$\boldsymbol{X}$的SVD分解代入：$\boldsymbol{X}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$，$\boldsymbol{X}^{\mathrm{T}}=\boldsymbol{V}\boldsymbol{\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}$

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

$$\widehat{\mathbf{w}}_{Ridge}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}=(\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}}+\lambda\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}})^{-1}\boldsymbol{V}\boldsymbol{\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}$$

$$=\boldsymbol{V}(\boldsymbol{D}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}=\boldsymbol{V}(\boldsymbol{D}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}$$

岭回归的最优解： SVD

- OLS：$\hat{\boldsymbol{w}}_{OLS} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{V}\boldsymbol{\Sigma}^{\ddagger}\boldsymbol{U}^{\mathrm{T}}\ \boldsymbol{y}$
  - 对矩阵$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$求逆
  - 当矩阵$\boldsymbol{X}$ 接近不满秩时，结果不稳定

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

$$= \boldsymbol{V}(\boldsymbol{D} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}$$

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

岭回归：权重衰减

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

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

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

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \, \nabla_{\boldsymbol{w}} J\Big|_{\boldsymbol{w}^{(t)}}$$

- 4. 若$J\left(\boldsymbol{w}^{(t)}\right) - J\left(\boldsymbol{w}^{(t+1)}\right) < \varepsilon$，返回$\boldsymbol{w}^{(t)}$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：$J(\boldsymbol{w}, \lambda) = (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$
- 目标函数的梯度：

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

$$= 2\boldsymbol{X}^{\mathrm{T}}\underbrace{\boxed{(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})}}_{\text{预测残差}r}$$

参数的更新量与输入与预测残差的相关性有关。

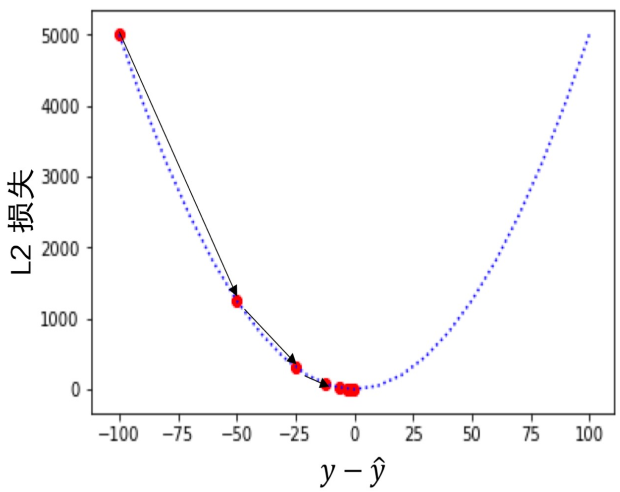
- 参数更新：

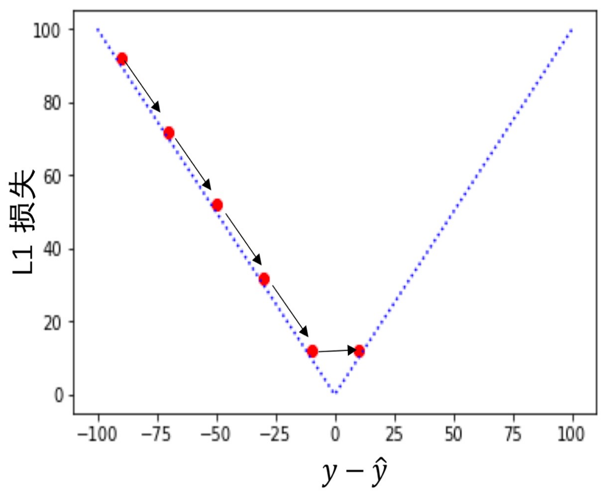
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta g\left(\boldsymbol{w}^{(t)}\right)$$

$$= \boldsymbol{w}^{(t)} - 2\eta \boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{w}^{(t)} - \boldsymbol{y}\right)$$

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：$J(\boldsymbol{w},\lambda)=(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})+\lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$
- 目标函数的梯度：

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

- 参数更新：

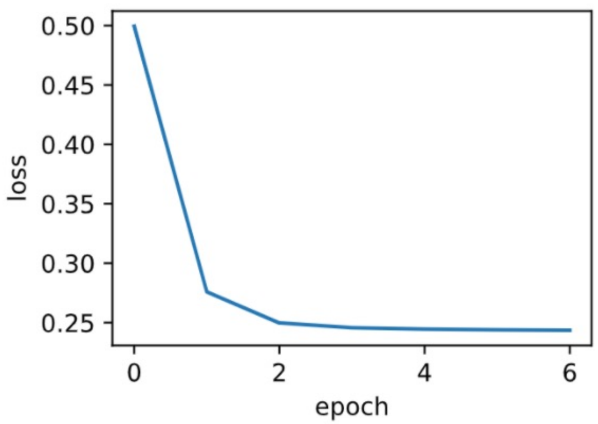
$$\begin{aligned}\boldsymbol{w}^{(t+1)}&=\boldsymbol{w}^{(t)}-\eta g\left(\boldsymbol{w}^{(t)}\right)\\&=\boldsymbol{w}^{(t)}-2\eta\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{w}^{(t)}-y\right)-2\eta\lambda\boldsymbol{w}^{(t)}\end{aligned}$$

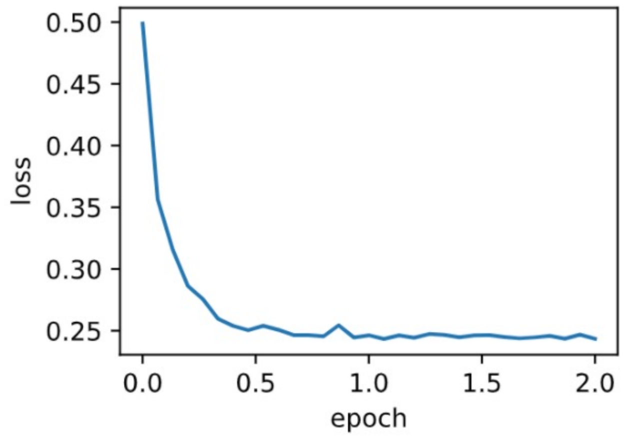
梯度下降的高阶话题

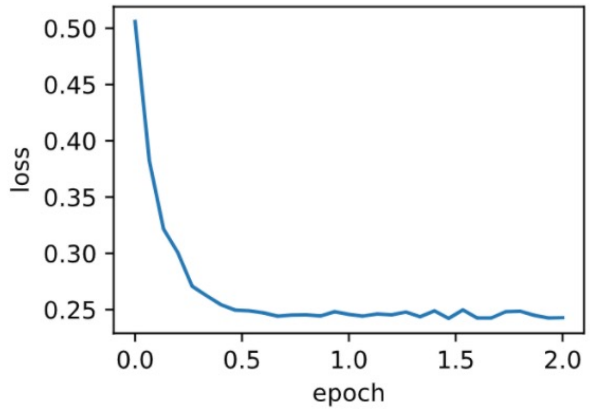
$$w^{(t+1)} = w^{(t)} - \eta g\left(w^{(t)}\right)$$

- 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

$(bathsize = 10)$

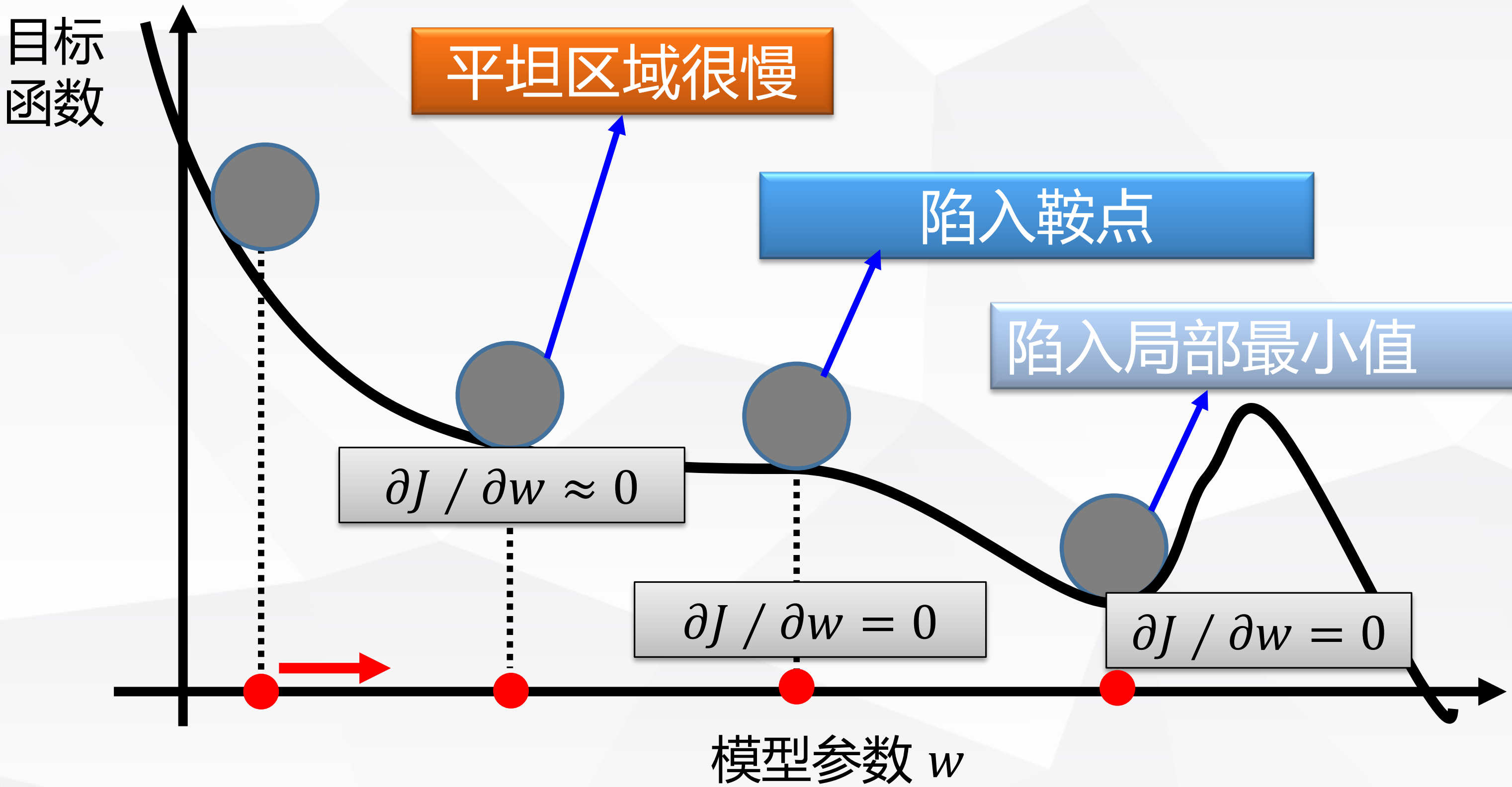
每处理一个batch更新一次模型参数

目标函数值的下降在1个迭代周期后就变得较为平缓

https://tangshusen.me/Dive-into-DL-PyTorch/#/chapter07_optimization/7.3_minibatch-sgd

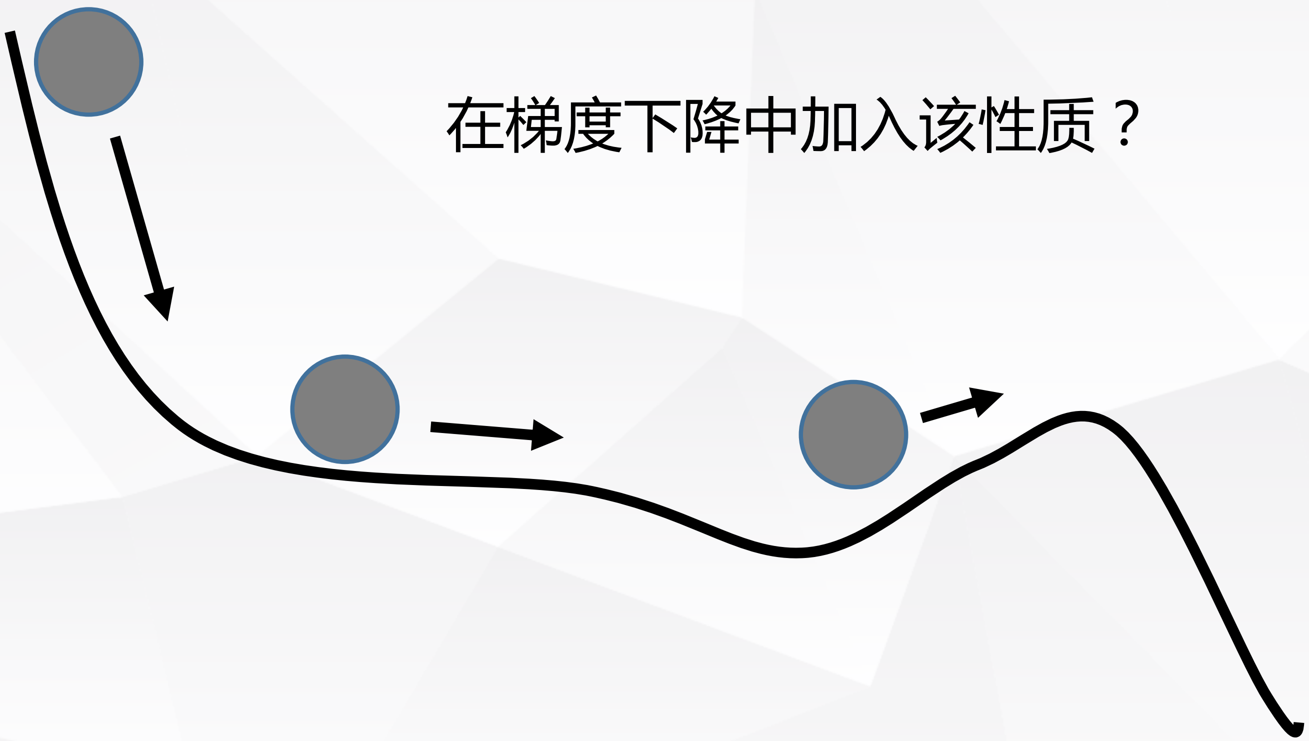
梯度下降

■梯度下降法在有些地方（Hessian矩阵病态）进展缓慢



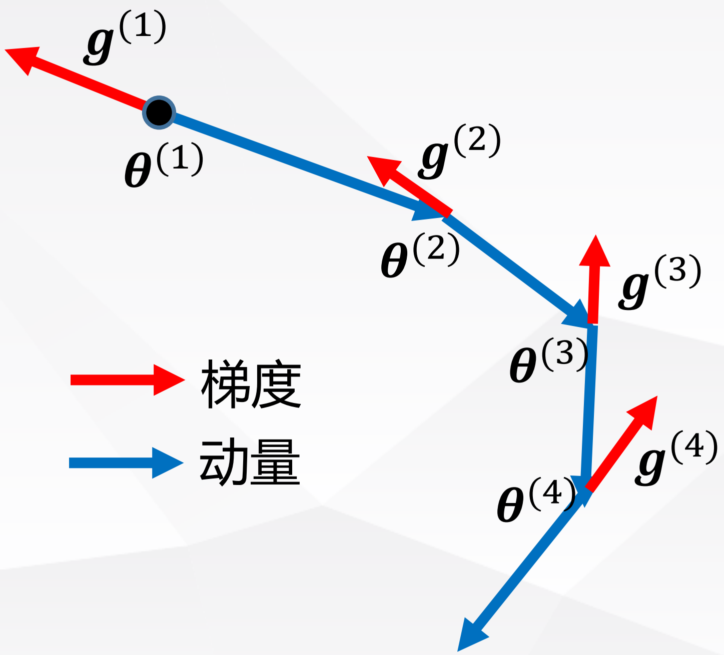
物理世界中……

■动量/惯性



在梯度下降中加入该性质？

回忆：朴素梯度下降



初始位置$\boldsymbol{\theta}^{(1)}$

计算$\boldsymbol{\theta}^{(1)}$处的梯度$\boldsymbol{g}^{(1)}$

移到$\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} - \eta \boldsymbol{g}^{(1)}$

计算$\boldsymbol{\theta}^{(2)}$处的梯度$\boldsymbol{g}^{(2)}$

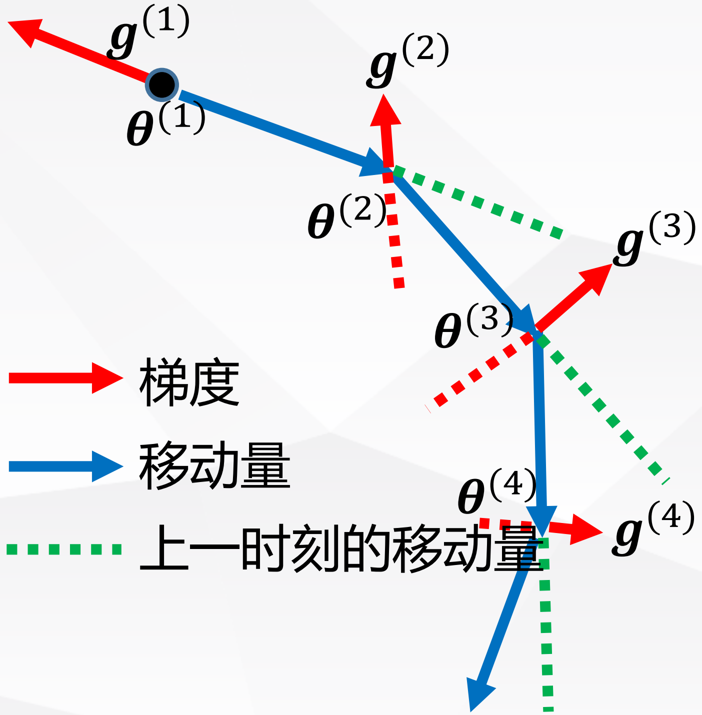
移到$\boldsymbol{\theta}^{(3)} = \boldsymbol{\theta}^{(2)} - \eta \boldsymbol{g}^{(2)}$

⋮

迭代，直到$\boldsymbol{g}^{(t)} \approx 0$

移动方向：动量法

动量：上一时刻的动量减去当前的梯度



初始动量$v^{(0)} = \mathbf{0}$

初始位置$\theta^{(1)}$

计算$\theta^{(1)}$处的梯度$g^{(1)}$

动量$v^{(1)} = \rho v^{(0)} - \eta g^{(1)}$

移到$\theta^{(2)} = \theta^{(1)} + v^{(1)}$

计算$\theta^{(2)}$处的梯度$g^{(2)}$

动量$v^{(2)} = \rho v^{(1)} - \eta g^{(2)}$

移到$\theta^{(3)} = \theta^{(2)} + v^{(2)}$

移动量不仅与梯度有关，还与前一时刻的移动量有关。

动量法

- 事实上，$\boldsymbol{v}^{(t)}$ 为之前所有梯度 $\boldsymbol{g}^{(0)}, \boldsymbol{g}^{(1)}, \ldots \boldsymbol{g}^{(t)}$ 的加权和

$$\boldsymbol{v}^{(0)} = 0$$

$$\boldsymbol{v}^{(1)} = \rho \boldsymbol{v}^{(0)} - \eta \boldsymbol{g}^{(1)} = -\eta \boldsymbol{g}^{(1)}$$

$$\boldsymbol{v}^{(2)} = \rho \boldsymbol{v}^{(1)} - \eta \boldsymbol{g}^{(2)} = -\rho\eta \boldsymbol{g}^{(1)} - \eta \boldsymbol{g}^{(2)}$$

$$\vdots$$

$$\begin{aligned} \boldsymbol{v}^{(t)} &= \rho \boldsymbol{v}^{(t-1)} - \eta \boldsymbol{g}^{(t)} \\ &= \rho\left(-\rho \boldsymbol{v}^{(t-2)} - \eta \boldsymbol{g}^{(t-1)}\right) - \eta \boldsymbol{g}^{(t)} \\ &= -\eta \sum_{j=1}^{t} \rho^{t-j} \boldsymbol{g}^{(j)} \end{aligned}$$

负梯度的指数衰减平均

初始动量 $\boldsymbol{v}^{(0)} = \mathbf{0}$

初始位置 $\boldsymbol{\theta}^{(1)}$

计算 $\boldsymbol{\theta}^{(1)}$ 处的梯度 $\boldsymbol{g}^{(1)}$

动量 $\boldsymbol{v}^{(1)} = \rho \boldsymbol{v}^{(0)} - \eta \boldsymbol{g}^{(1)}$

移到 $\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} + \boldsymbol{v}^{(1)}$

计算 $\boldsymbol{\theta}^{(2)}$ 处的梯度 $\boldsymbol{g}^{(2)}$

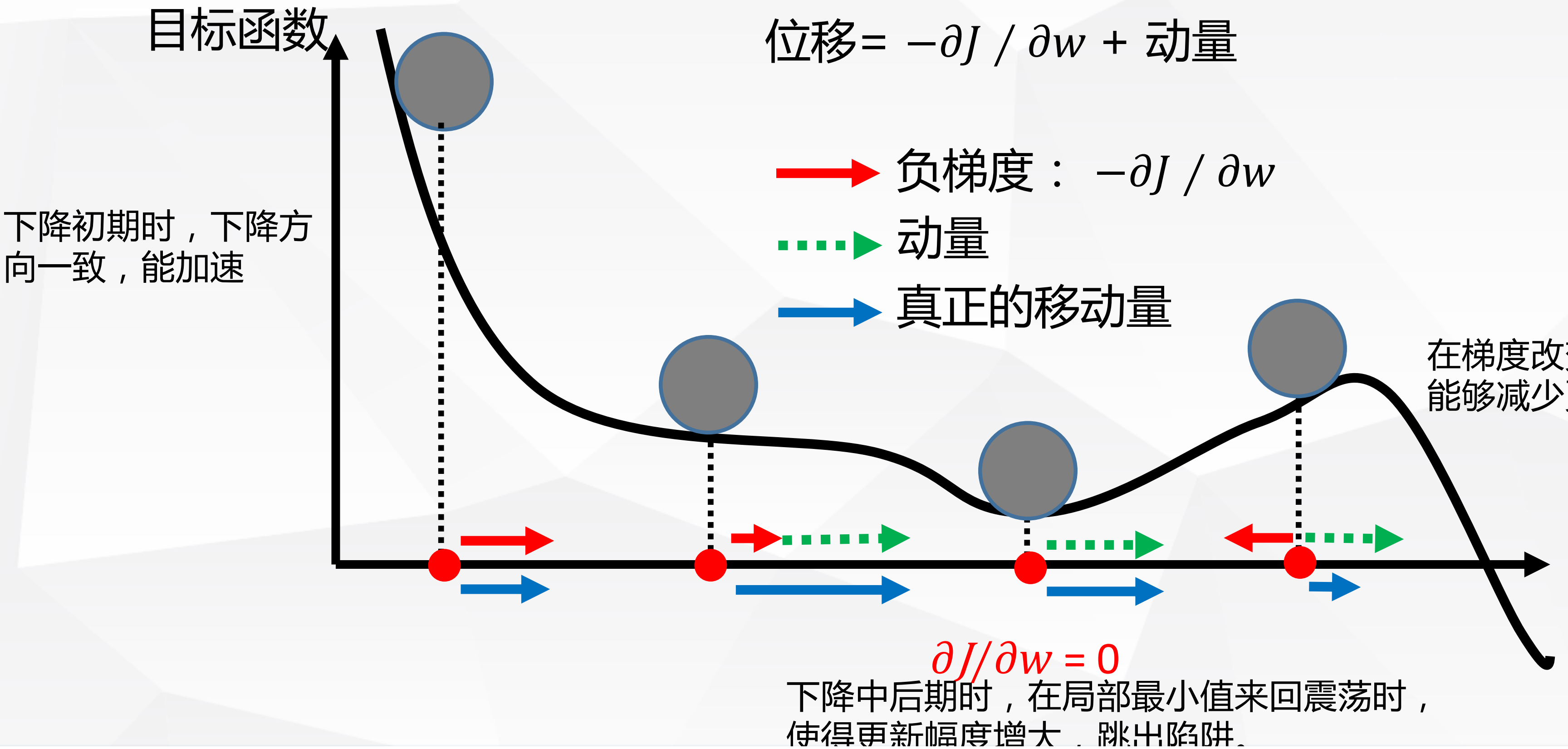
动量 $\boldsymbol{v}^{(2)} = \rho \boldsymbol{v}^{(1)} - \eta \boldsymbol{g}^{(2)}$

移到 $\boldsymbol{\theta}^{(3)} = \boldsymbol{\theta}^{(2)} + \boldsymbol{v}^{(2)}$

移动量不仅与梯度有关，还与前一时刻的移动量有关。

动量法

仍然不能保证到达全局最小值，
but give some hope ……



Nesterov动量法

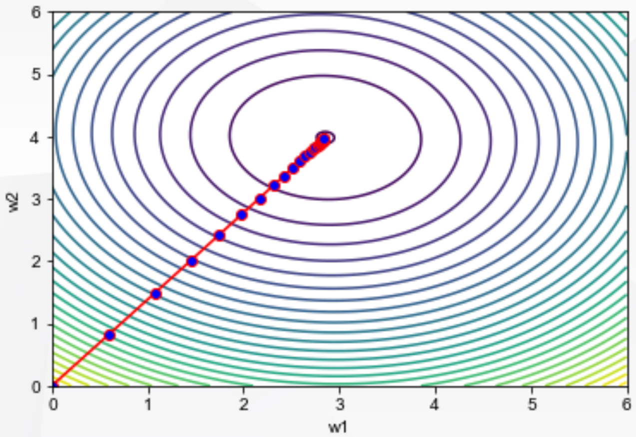
- **SGD with Nesterov Momentum（涅斯捷罗夫动量法）**
  - 不计算当前位置的梯度，而是计算如果按照速度方向走了一步，那个时候的梯度，再与速度一起计算更新方向

梯度下降：$\boldsymbol{v}^{(t)} = -\eta \nabla J(\boldsymbol{\theta}) \mid_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$

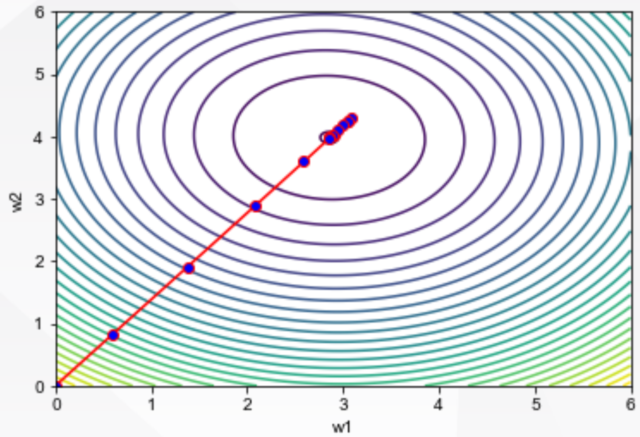
动量法：$\boldsymbol{v}^{(t)} = \rho \boldsymbol{v}^{(t-1)} - \eta \nabla J(\boldsymbol{\theta}) \mid_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$

NAG：$\boldsymbol{v}^{(t)} = \rho \boldsymbol{v}^{(t-1)} - \eta \nabla J |_{\boldsymbol{\theta}^{(t)}+\rho \boldsymbol{v}^{(t-1)}}$

动量法比较



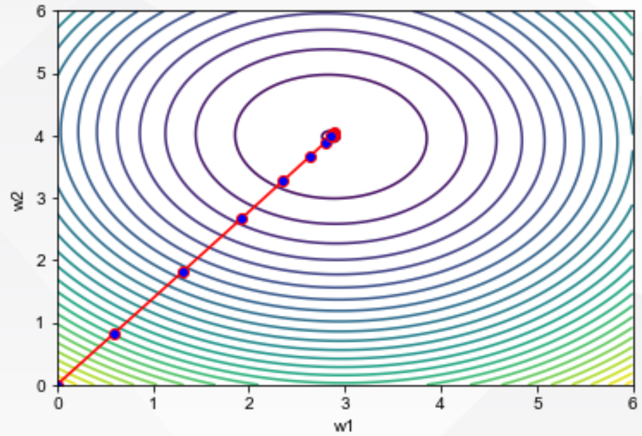
梯度下降



动量法

收敛快
迭代次数比梯度下降法少一半
尤其前几次迭代参数更新量大

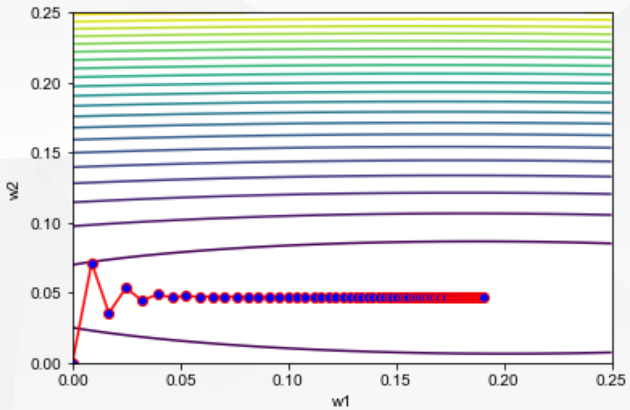
最后阶段搜索范围越过了最佳位置（学习率较大），这时两次更新方向相反，动量法会使得更新幅度减小，再慢慢回到最佳位置



NAG

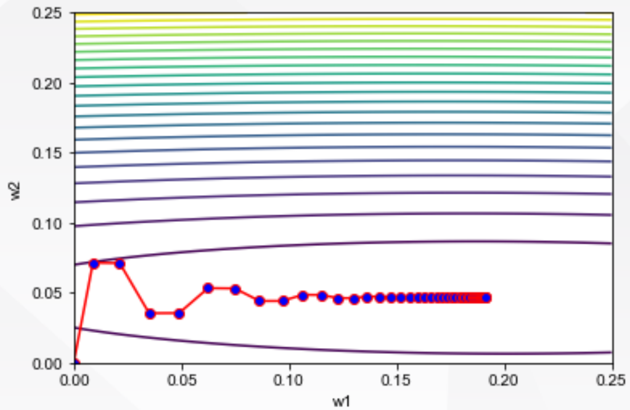
提前预知目标函数的信息，相当于多考虑了目标函数的二阶导数信息，类似牛顿法的思想，因此搜索路径更合理，收敛速度更快

动量法比较



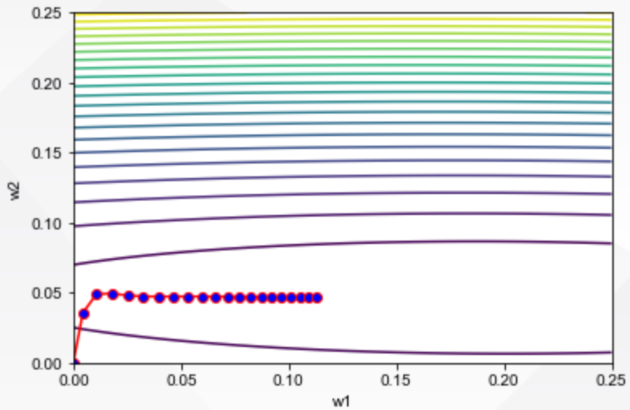
梯度下降

目标函数在竖直方向比在水平方向的斜率的绝对值更大，梯度下降法中参数在竖直方向比在水平方向移动幅度更大，在长轴上呈“之”字形反复跳跃，缓慢向最小值逼近。
（不同参数的梯度范围差异大，通常是因为特征没有去量纲）



动量法

竖直方向上的移动更加平滑，且在水平方向上更快逼近最优解，
因为此时竖直方向的当前梯度与之前的梯度方向相反相互抵消，移动的幅度小

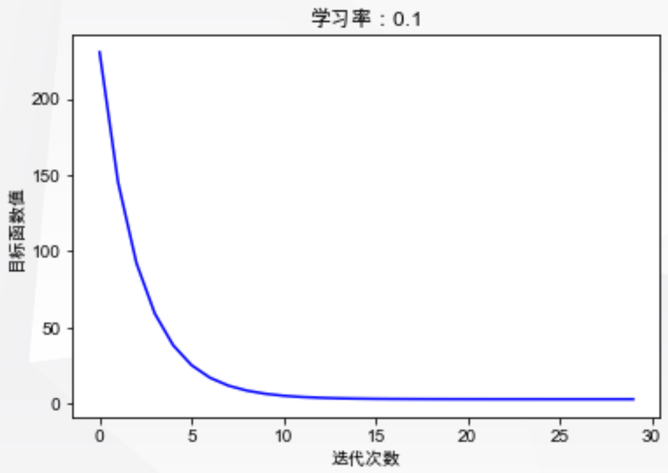


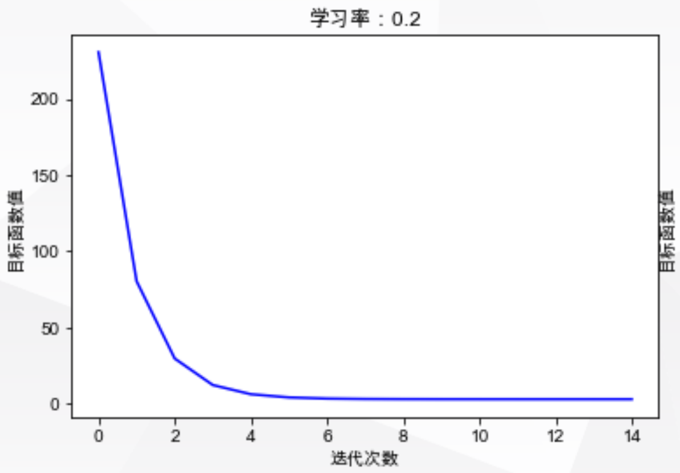
NAG

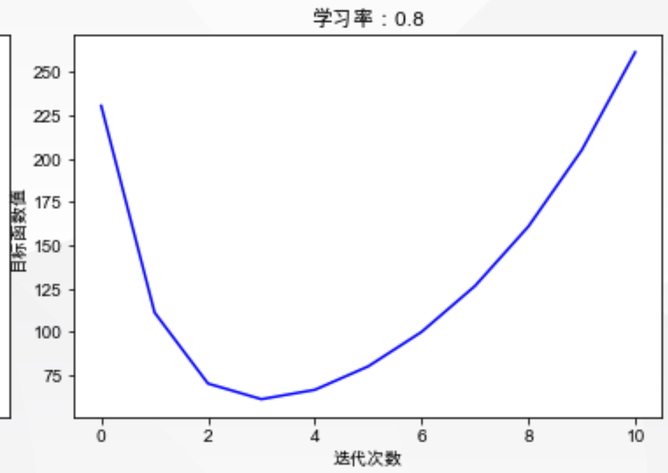
提前预知目标函数的信息，相当于多考虑了目标函数的二阶导数信息，类似牛顿法的思想，因此搜索路径更合理，收敛速度更快

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 ... 1e-5]

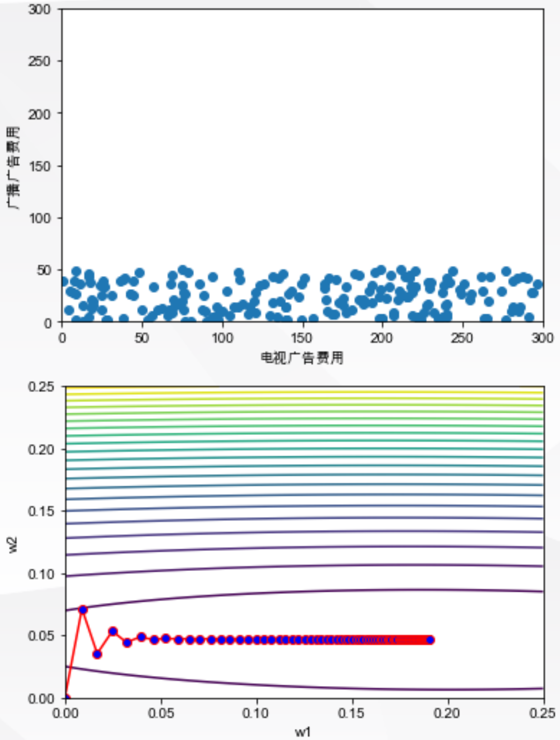




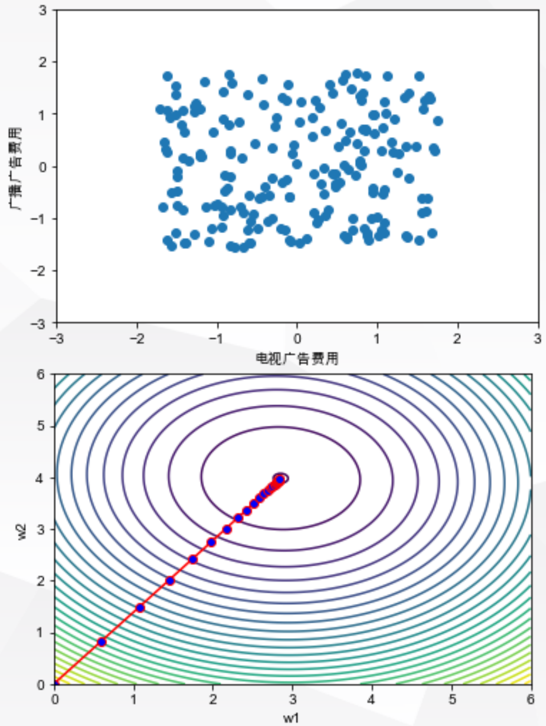


特征去量纲

- OLS的梯度下降更新换代：$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta g\left(\boldsymbol{w}^{(t)}\right) = \boldsymbol{w}^{(t)} - 2\eta \boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{w}^{(t)} - \boldsymbol{y}\right)$
- 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：

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

自适应学习率

- **AdaGrad**
  - 经常更新的参数学习率较小，尽量不被单个样本影响较大
  - 偶尔更新的参数学习率大一些，希望能从偶然出现的样本上多学一些
  - 使用二阶动量（迄今为止所有梯度值的平方和）来度量历史更新频率

$$\boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \varepsilon}} \odot \boldsymbol{g}^{(t)}$$

$\odot$：向量按元素乘
$\sqrt{()}$：对向量按元素求平方根
初始学速率η：一般设置为0.01
ϵ通常取很小的数：如10^{-6}

  - 存在问题：
    - 梯度会累加得越来越大，学习率衰减：学习速率衰减过快
    - 减缓陡峭区域的下降过程、加速平坦区域的过程

RMSProp

- 为了缓解Adagrad学习率衰减过快，RMSprop改变梯度累积为指数衰减的移动平均以丢弃遥远的历史。

AdaGrad：

$$\boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \varepsilon}} \odot \boldsymbol{g}^{(t)}$$

RMSprop ：

$$\boldsymbol{s}^{(t)} = \rho \boldsymbol{s}^{(t-1)} + (1-\rho) \boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \varepsilon}} \odot \boldsymbol{g}^{(t)}$$

Adam

■ **Adam**：Adaptive + Momentum，同时利用一阶动量和二阶动量

```
## Adam
first_moment = 0
second_moment = 0

while True:
    dx = compute_gradient(x)
    first_moment = rho_1 * first_moment + (1 - rho_1)*dx
    second_moment = rho_2 * second_moment + (1 - rho_2)*dx*dx
    x -= learning_rate first_moment /np.sqrt(second_moment + 1e-7)
```

AdaGrad / RMSProp

$$\boldsymbol{v}^{(t)} = \rho_1 \boldsymbol{v}^{(t-1)} - (1-\rho_1)\boldsymbol{g}^{(t)}$$

$$\boldsymbol{s}^{(t)} = \rho_2 \boldsymbol{s}^{(t-1)} + (1-\rho_2)\boldsymbol{g}^{(i)} \odot \boldsymbol{g}^{(i)}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)}+\epsilon}} \odot \boldsymbol{v}^{(t)}$$

•超参数建议： $\rho_1 = 0.9$，$\rho_2 = 0.999$，$\eta =$ 1e−3 或5e−4

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
max_iter	共轭梯度求解器的最大迭代次数。	对于优化算法*solver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。
tol	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。	
solver	求解最优化问题的算法 可取：'auto'，'svd'，'cholesky'，'lsqr'，'sparse_cg'，'sag'，'saga'	请见下页
random_state	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'

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

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(loss > previous_loss - 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'：eta = eta0
    - 'optimal'：eta = 1.0 / (alpha * (t + t0))
    - 'invscaling'：eta = eta0 / pow(t, power_t)

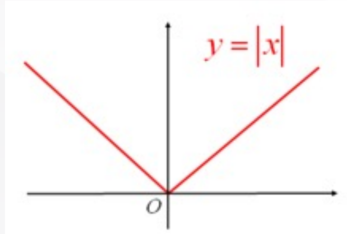
    - 随机梯度下降实现参考文献：
      - "Stochastic Gradient Descent" L. Bottou - Website, 2010
      - "The Tradeoffs of Large Scale Machine Learning" L. Bottou - Website, 2011.

- 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

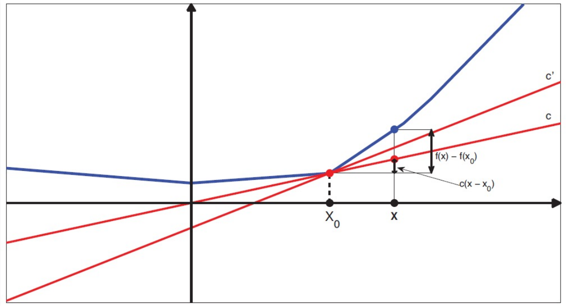


- 当目标函数可导时，梯度下降法是非常有效的优化算法。
- Lasso的目标函数为：$J(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$，
- 其中绝对值函数$\|\boldsymbol{w}\|_1$在$w_j = 0$时不可导，无法计算梯度，无法用梯度下降法求解。
- 次梯度法将梯度扩展为次梯度，可以解决这个难题。

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

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



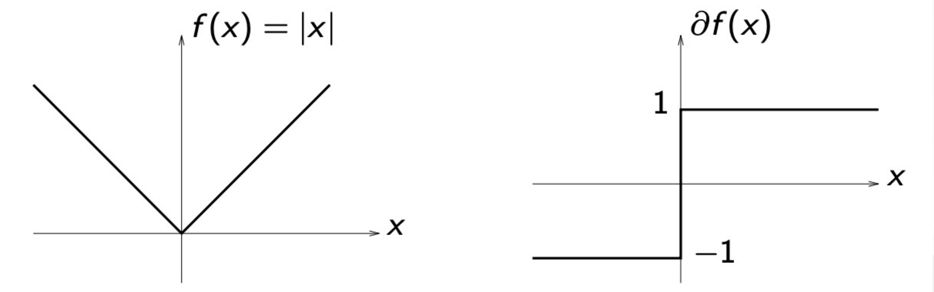
- 其中I为包含x_0的某个区间。
- 如图所示，对于定义域中的任何x_0，我们总可以作出一条直线通过点$(x_0, f(x_0))$，且直线要么与f相切，要么在它的下方。直线的斜率称为函数的次导数。
- 次梯度的集合为函数f在x_0处的次微分，记为$\partial f(x_0)$。
- 定义次梯度集合为区间$[a,b]$：

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

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：$0 \in \partial f(x^*)$ ↔ $f(x^*) = \min_x f(x)$，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$\boldsymbol{w}^{(0)}$；
- 2. 计算目标函数$J(\boldsymbol{w})$在当前值的梯度：$\nabla J\left(\boldsymbol{w}^{(t)}\right)$；
- 3. 根据学习率η，更新参数：$\boldsymbol{w}^{(t+1)}=\boldsymbol{w}^{(t)}-\eta\nabla J\left(\boldsymbol{w}^{(t)}\right)$；
- 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$\boldsymbol{w}^{(t+1)}$和目标函数极小值$J\left(\boldsymbol{w}^{(t+1)}\right)$；否则转到第2步。

- 次梯度法：
- 1. 从$t=0$开始，初始化$\boldsymbol{w}^{(0)}$；
- 2. 计算目标函数$J(\boldsymbol{w})$在当前值的次微分：$\partial J\left(\boldsymbol{w}^{(t)}\right)$；
- 3. 根据学习率η，更新参数：$\boldsymbol{w}^{(t+1)}=\mathbf{w}^{t}-\eta\partial J\left(\boldsymbol{w}^{(t+1)}\right)$；
- 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$\boldsymbol{w}^{(t+1)}$和目标函数极小值$J\left(\boldsymbol{w}^{(t+1)}\right)$；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：$J(\boldsymbol{w}^*) = \min_{1,\ldots,t} J\left(\boldsymbol{w}^{(t)}\right)$。
- 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：$J(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$
- 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$RSS(\boldsymbol{w})$

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

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

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

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

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

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

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

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

Lasso的坐标轴下降法求解

■次第j维对应目标函数次微分为

■最优解条件：$0 \in \partial J_{w_j}(\mathbf{w},\lambda)$：

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

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

稀疏：特征选择

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

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$\boldsymbol{w}$（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 $c_j = 2\sum_{i=1}^{N}\left(\boldsymbol{w}_{-j}^{\mathrm{T}}\boldsymbol{x}_{i,-j} - y_i\right)x_{i,j}$；
  - 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$\lambda \geq \max_j\left(\boldsymbol{x}_{:j}^{T}\boldsymbol{y}\right)$时，所有$w_j = 0$。
其中$\boldsymbol{x}_{:j}$表示所有样本第j维的特征值，$\boldsymbol{y}$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
precompute	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。
max_iter	最大迭代次数。
tol	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。
warm_start	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。
positive	是否强制使系数为正。
random_state	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。
selection	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

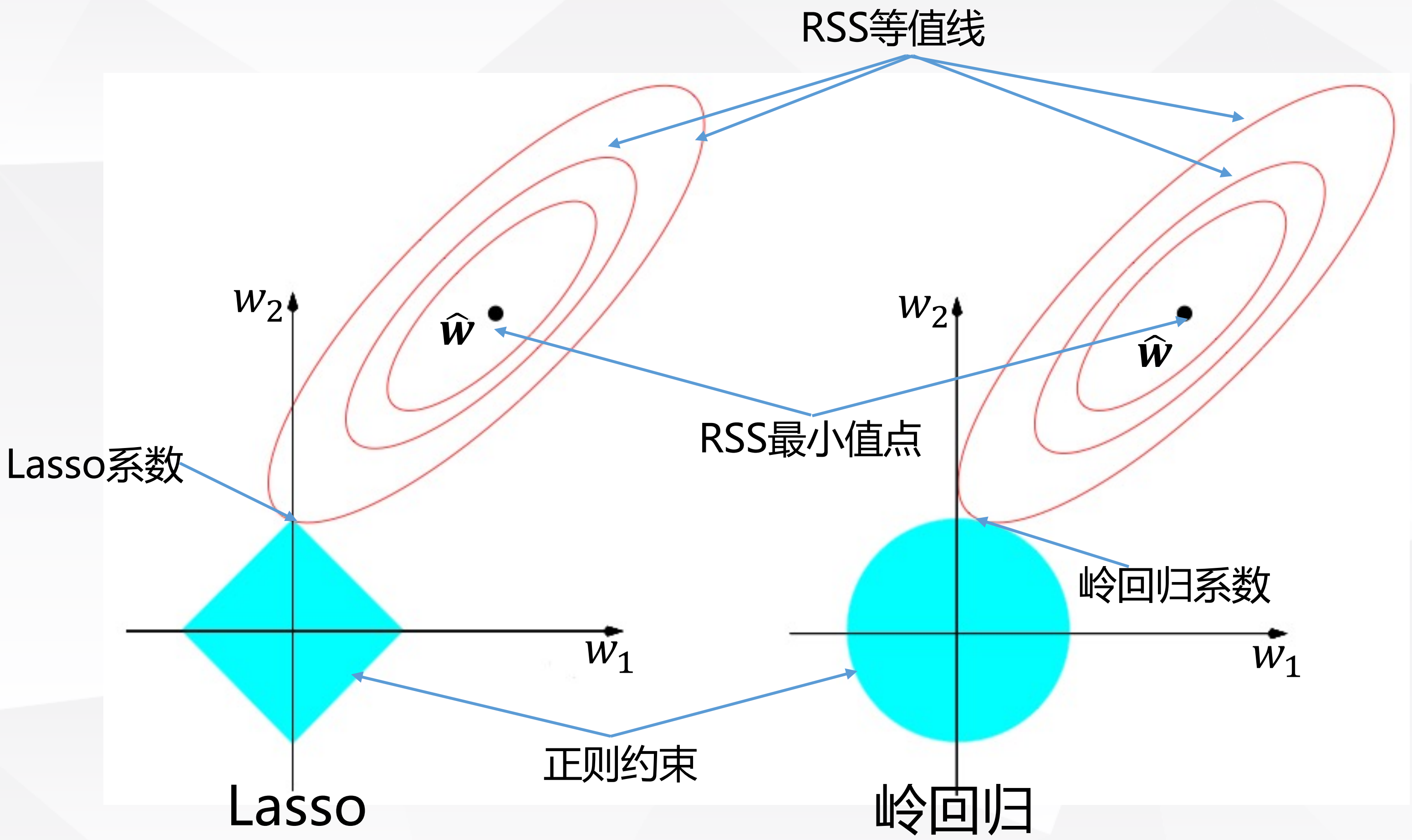
- L2正则使得线性回归系数收缩，模型稳定。
- 当输入特征之间存在共线性时使用L2正则。

- 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$\boldsymbol{X}$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



RSS等值线
w_2
$\hat{w}$
Lasso系数
RSS最小值点
w_2
$\hat{w}$
岭回归系数
w_1
w_1
正则约束
Lasso
岭回归

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）

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

- 平均绝对误差（Mean Absolute Error，MAE）

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

- 绝对误差中值（Median Absolute Error，MedAE）

$$\mathrm{MedAE}(\hat{\boldsymbol{y}},\boldsymbol{y})=\mathrm{median}(|\hat{y}_1-y_1|,\ldots,|\hat{y}_N-y_N|)$$

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

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

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

$$SS_{res}\left(\hat{\boldsymbol{y}}, \boldsymbol{y}\right) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$R^2(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \frac{SS_{res}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right)}{SS_{tot}(\boldsymbol{y})}$$

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

误差减少比例

用样本均值$\bar{y}$来做预测的MSE

（没有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

$$\text{explained\_variance}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \frac{\text{Var}\left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)}{Var\{\boldsymbol{y}\}}$$

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有Scoring 参数。

sklearn.metrics

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

Scikit-Learn中的Scoring参数

Regression	
‘explained_variance’	metrics.explained_variance_score
‘neg_mean_absolute_error’	metrics.mean_absolute_error
‘neg_mean_squared_error’	metrics.mean_squared_error
‘neg_mean_squared_log_error’	metrics.mean_squared_log_error
‘neg_median_absolute_error’	metrics.median_absolute_error
‘r2’	metrics.r2_score

同metrics一一对应

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

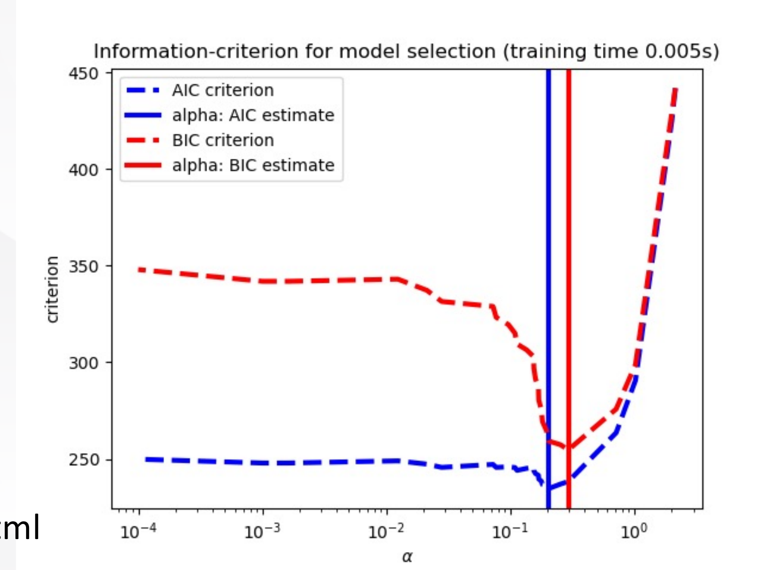
- 当交叉验证的折数$K = N$时，因为每次留出一个样本做验证集，我们称之为留一交叉验证。
  - 折数更多，计算代价越高，通常当样本数非常少使用。
- 对线性模型，可采用广义交叉验证（Generalized Cross Validation, GCV）近似留一交叉验证，极大降低交叉验证的计算量。
  - Scikit-Learns中岭回归（RidgeCV）采用GCV确定正则超参数。

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

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

BIC/AIC

- AIC： $AIC(\lambda) = N\ln\big(\mathrm{RSS}(\boldsymbol{w},\lambda)\big) + 2df(\lambda)$
- BIC： $BIC(\lambda) = N\ln\big(\mathrm{RSS}(\boldsymbol{w},\lambda)\big) + \ln(N)df(\lambda)$
- 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

■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'	$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$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。

LassoCV

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$\alpha_{\max}$，再根据参数eps的值和$\alpha_{\min}/\alpha_{\max} = \text{eps}$，确定alpha的最小值$\alpha_{\min}$；最后在$\alpha_{\min}$到$\alpha_{\max}$区间的$\log_{10}$均匀取n_alphas个alpha：logspace(np.log10($\alpha_{\max}$ * eps), 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

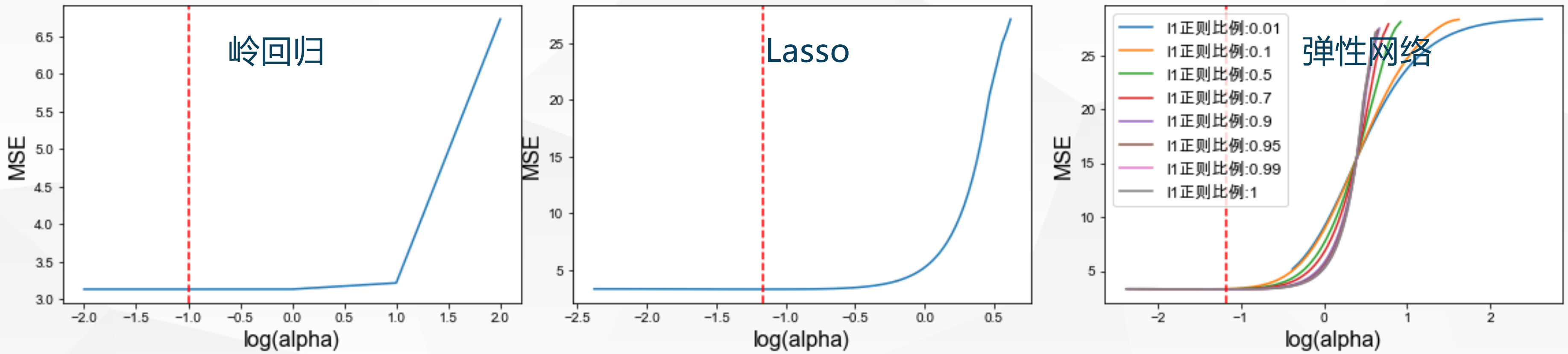
模型

- 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

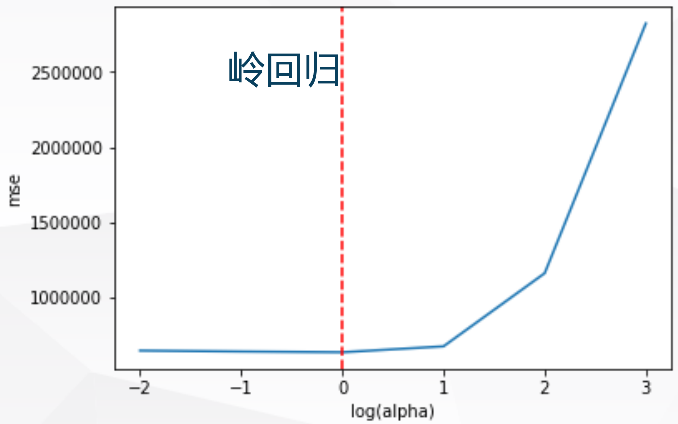
■特征工程： chp6_2_FE_bikesharing.ipynb

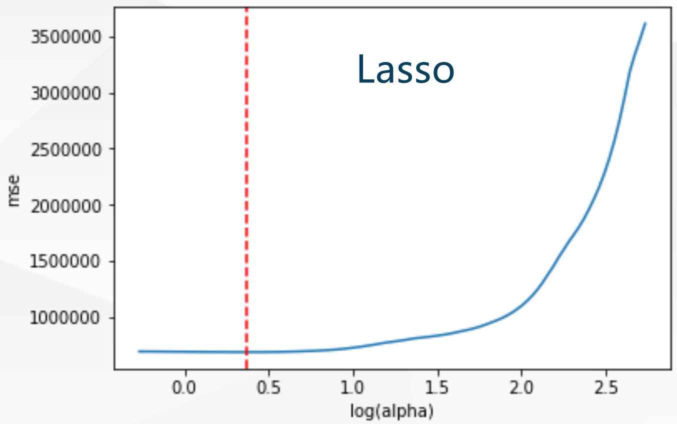
- 数值型特征做标准化处理：湿度、温度、体感温度、风速特征
- 类别型特征取值均不多，独热编码：季节、月份、星期、天气概况
- 二值特征编码为0/1：年份、是否为工作日、是否为节假日
- 每个日期只有2个样本，所以去掉特征“日期”

共享单车骑行量预测

3_Train_Test_Advertising.ipynb

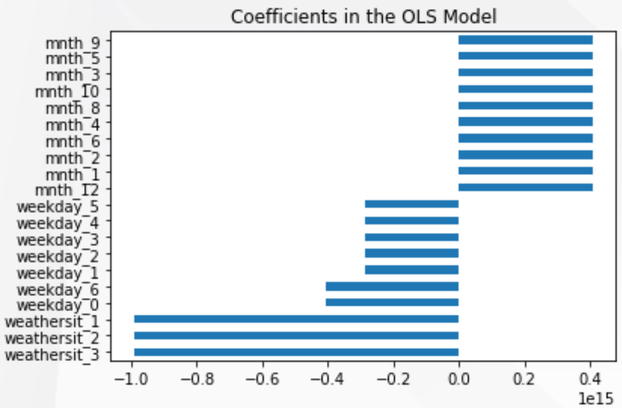
- 模型
  - 最小二乘线性回归（OLS）
  - 岭回归：正则参数λ
  - Lasso：正则参数λ
  - 弹性网络：正则参数λ和L1正则比例
- 数据集中随机80%样本为训练数据
- 超参数调优评价指标：MSE（红色竖线为最佳超参数）

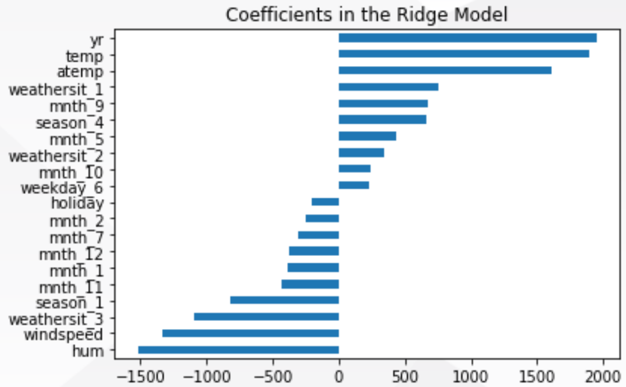


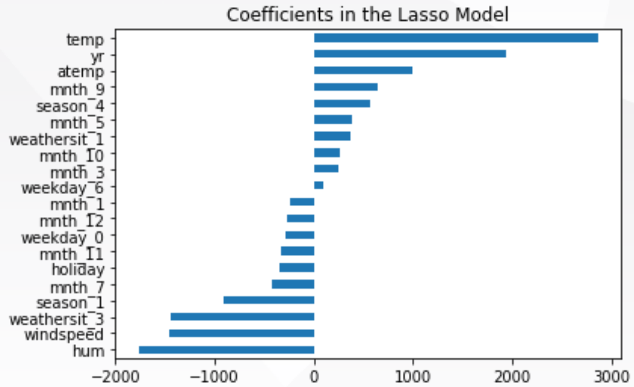


共享单车骑行量预测

■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







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