# 线性回归模型



# 线性回归

机器学习是根据训练数据对变量之间的关系进行建模。当输出变量（响应变量）$\boldsymbol{y} \in \mathbb{R}$ 是连续值时，我们称之为**回归分析**，即用函数描述一个或多个预测变量与响应变量 $\boldsymbol{y}$之间的关系，并根据该模型预测新观测值对应的响应。

- 给定训练数据 $\mathcal{D}=\{\mathbf{x}_i|y_i\}_{i=1}^N$，其中 $y\in\mathbb{R}$ 是连续值，一共有$N$个样本，回归分析的目标是学习一个从输入$\mathcal{X}$ 到输出 $\mathcal{Y}$ 的映射 $f$
- 对新的测试数据 $\mathbf{x}$，用学习到的映射 $f$ 对其进行预测：$\hat{y}=f(\mathbf{x})$
- 若假设映射 $f$ 是一个**线性函数**，即

$$y=f(\mathbf{x}|\mathbf{w})=\mathbf{w}^T\mathbf{x}$$

- 我们称之为**线性回归模型**。

例：Boston房价预测

波士顿房价数据集为大波士顿地区房价数据，是UCI ML（欧文加利福尼亚大学机器学习库）房价数据集的副本，链接为 http://archive.ics.uci.edu/ml/datasets/Housing。

波士顿房价数据已被用于许多涉及回归问题的机器学习论文中。该数据集有506个样本，每个样本包含波士顿某地区的房屋的13个属性和该地区的房价中位数。回归分析的任务是根据某地区房屋的13个属性$\mathbf{x}$预测该地区的房价中位数$y$。

若采用线性回归模型来做房价预测，我们假设房屋的13个属性$\mathbf{x}$与该地区的房价中位数$y$之间的关系为：

$$y=\mathbf{w}^T\mathbf{x}=\sum_{j=1}^{13}w_jx_j+w_0$$

其中$w_0$为截距项，向量$\mathbf{x}=(1,x_1,\ldots,x_{13})^T$，即13个房屋属性加上常数组成的向量（增加元素1是为了将截距项与属性统一处理，下面如无特别说明，通常假设向量$\mathbf{x}$包含常量1）。

**参考资料**


> - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
> - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.


数据集字段信息（按顺序）：

| CRIM | 城镇人均犯罪率 |
| --- | --- |
| ZN | 占地面积超过2.5万平方英尺的住宅用地比例 |
| INDUS | 城镇非零售业务地区的比例 |
| CHAS | 是否靠近查尔斯河 ( 1表示在河边；否则为0) |
| NOX | 一氧化氮浓度（每1000万份） |
| RM | 平均每居民房数 |
| AGE | 在1940年之前建成的所有者占用单位的比例 |
| DIS | 与五个波士顿就业中心的加权距离 |
| RAD | 辐射状公路的可达性指数 |
| TAX | 每10,000美元的全额物业税率 |
| PTRATIO | 城镇师生比例 |
| B | 1000(Bk - 0.63)^2 其中 Bk 是城镇的黑人比例 |
| LSTAT | 人口中地位较低人群的百分数 |
| MEDV | 以1000美元计算的自有住房的房价中位数 |

我们通常用二维表（类似关系数据库中的数据表）存储样本数据，在Python语言中可用pandas库来处理数据读取Boston房价数据集，并显示前5个样本的代码如下

```
import pandas as pd # 数据处理

data = pd.read_csv("boston_housing.csv")

#通过观察前5行，了解数据每列（特征）的概况
data.head()
```

上述代码运行结果如下表所示：

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.00632 | 18 | 2.31 | 0 | 0.538 | 6.575 | 65.2 | 4.09 | 1 | 296 | 15 | 396.9 | 4.98 | 24 |
| 1 | 0.02731 | 0 | 7.07 | 0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | 17 | 396.9 | 9.14 | 21.6 |
| 2 | 0.02729 | 0 | 7.07 | 0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | 17 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0 | 2.18 | 0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | 18 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0 | 2.18 | 0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | 18 | 396.9 | 5.33 | 36.2 |

# 目标函数

- 机器学习模型的目标函数通常包含两项：**损失函数**$L$和**正则项**$R$，分别代表度量模型与训练数据的匹配程度（损失函数越小越匹配）和对模型复杂度的“惩罚”以避免过拟合。

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) + R(\boldsymbol{\theta})$$

  因此目标函数最小要求模型和训练数据拟合得好，同时模型尽可能简单。这也体现了机器学习的基本准则：奥卡姆**剃刀**定律（**Occam's** Razor），即简单有效原则。

- 对回归问题，损失函数可以采用L2损失（也可以根据实际情况选择其他有意义的损失函数），得到

$$\mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) = (y_i - f(\mathbf{x}_i; \boldsymbol{\theta}))^2 ,$$

即残差的平方。对线性回归，所有样本的残差平方和为**残差平方和**(residual sum of squares，RSS)：

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x_i})^2 。$$

# 线性回归的正则项

## 1. 无正则：最小二乘线性回归（Ordinary Least Square，OLS）

- 由于线性模型比较简单，实际应用中有时正则项为空，得到最小二乘线性回归（Ordinary Least Square，OLS）（此时目标函数中只有残差平方和，“平方”的在古时侯的称谓为“二乘”），即

$$J(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x_i})^2 = RSS(\mathbf{w}) 。$$

- 还记得中学物理实验中求重力加速度的实验、用欧姆定律求电阻的实验吗？对了，当时我们就是用最小二乘法来求解重力加速度$g$和电阻$R$的。
- 例：波士顿房价预测任务中房间数目（RM）与房价（MEDV）之间用最小二乘线性回归模型建模（为了可视化，这里暂时只取一维特征）
  - 前5个训练样本：

| 样本索引 | 房间数目（RM） | 房价中位数（MEDV） |
| --- | --- | --- |
| 1 | 6.575 | 24 |
| 2 | 6.421 | 21.6 |
| 3 | 7.185 | 34.7 |
| 4 | 6.998 | 33.4 |
| 5 | 7.147 | 36.2 |

线性回归是用一条直线/一个超平面拟合输入特征（RM）与目标（MEDV）之间的关系。最小二乘线性回归是预测残差的平方和最小（样本点到直线/超平面的距离（下图中红色的线段长度）平方和最小）的线性模型。下面的python代码我们直接挑用scikit-learn中的LinearRegression训练最小二乘线性回归模型。

```
# 1、导入必要的工具包
import numpy as np  # 矩阵操作
import pandas as pd # 数据表处理

import matplotlib.pyplot as plt   #画图
# 图形出现在Notebook里而不是新窗口
%matplotlib inline

# 2、读取数据
data = pd.read_csv("boston_housing.csv", nrows = 20)

# 从原始数据中分离输入特征x和输出y
# 为了方便可视化，只取一维特征RM（房间数目），y为MEDV（房屋价格中值）
y_train = data['MEDV']
X_train = data['RM']

X_train = X_train.values.reshape(-1, 1)


# 3、训练模型
from sklearn.linear_model import LinearRegression

# 使用默认配置初始化
lr = LinearRegression()

# 训练模型参数
lr.fit(X_train, y_train)
print "The trained model is: y ={}x + {}".format(lr.coef_,
lr.intercept_)

# 模型可视化
#显示训练数据散点图
plt.scatter(X_train, y_train, label = 'Train Samples')

#4、显示预测结果
y_train_pred = lr.predict(X_train)
#plt.scatter(X_train, y_train_pred, label = 'LR Prediction')
plt.plot(X_train, y_train_pred,'g-',label = 'LR Prediction')

# 预测残差
for idx,x in enumerate(X_train):
    plt.plot([x,x],[y_train[idx], y_train_pred[idx]], 'r-')

plt.xlabel('Room Number')
```

```
plt.ylabel('Median value of owner-occupied homes')
plt.legend()
plt.tight_layout()
```

上述代码执行输出为：y =[ 10.35223355]x + -41.2444772871

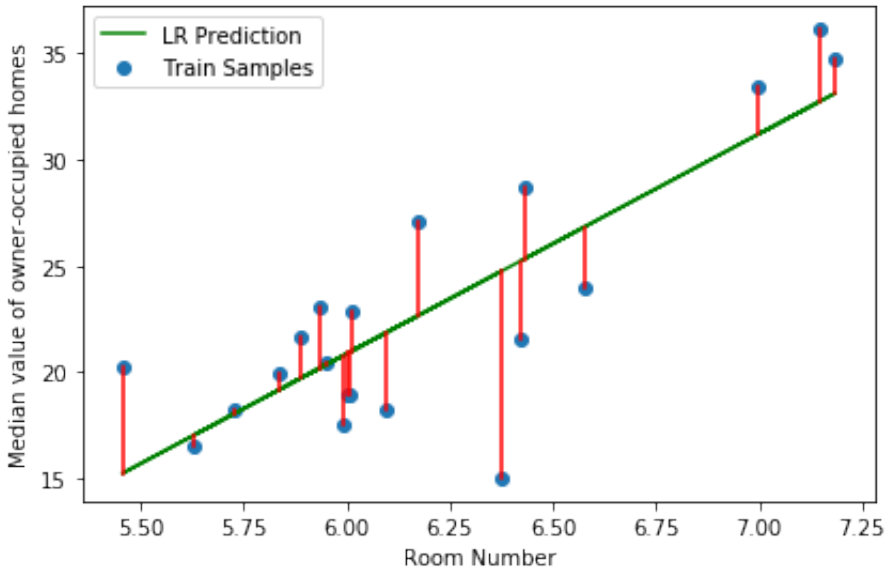



图1. 波士顿房价预测任务中根据前20个样本训练出的最小二乘线性回归模型。其中蓝色点为训练样本，绿色线为模型，红色线段为预测残差。

### 2. L2正则：岭回归（Ridge Regression）

- 正则项可以为L2正则，得到岭回归（Ridge Regression）模型：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda||\mathbf{w}||_2^2$$

### 3. L1正则：Lasso

- 正则项也可以选L1正则，得到Lasso模型：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda||\mathbf{w}||_1$$

  - 当λ取合适值时，Lasso（ least absolute shrinkage and selection operator）的结果是**稀疏**的（**w**的某些元素系数为0），起到**特征选择**作用。

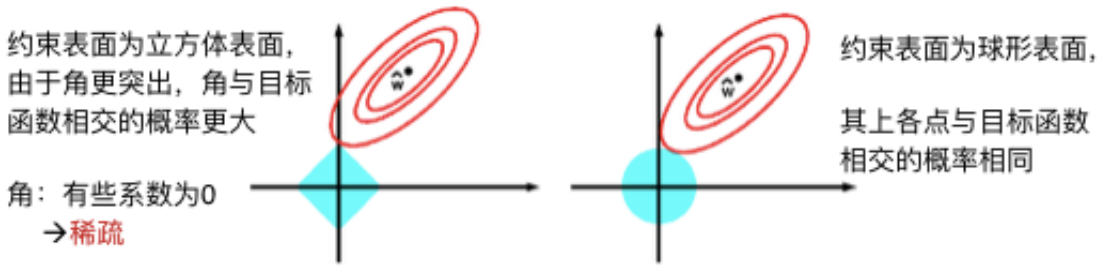
## 为什么$L_1$正则的解是稀疏的?

- 考虑两个优化问题：

$$\min_w RSS(\mathbf{w}) + \lambda||\mathbf{w}||_1$$
$$\min_w RSS(\mathbf{w}) + \lambda||\mathbf{w}||_2^2$$

- 分别等价于（连续的带约束的优化问题）

$$\min_w RSS(\mathbf{w}) \text{ s.t. } ||\mathbf{w}||_1 \leq B$$
$$\min_w RSS(\mathbf{w}) \text{ s.t. } ||\mathbf{w}||_2^2 \leq B$$




根据凸优化理论，上述问题的最优解为$RSS(\mathbf{w})$ 的等值线（红色，$\mathbf{w}$的二次函数，为椭圆曲线）与约束（蓝色）首次相切的位置。

- 例：如 $\mathbf{w} = (1,0)^T, (1/\sqrt{2}, 1/\sqrt{2})^T$ 的L2 模相同(1)，但L1模分别为1和$\sqrt{2}$。在损失相同（相同的红色等高线上）的情况下，L1正则选择L1模更小的模型。

## 线性回归模型的概率解释

- 最小二乘（线性）回归等价于极大似然估计
- 正则（线性）回归等价于高斯先验（L2正则）或Laplace先验（L1正则）下的贝叶斯估计

### 最小二乘线性回归等价于极大似然估计

- 假设：$y = f(\mathbf{x}) + \epsilon = \mathbf{w}^T\mathbf{x} + \epsilon$
- 其中$\epsilon$为线性预测和真值之间的**残差**
- 我们通常假设残差的分布为 $\epsilon \sim N(0, \sigma^2)$，均值为0，方差$\sigma^2$。对该残差分布的基础上，加上$y$的分布，因此线性回归可写成：

$$p(y|\mathbf{x}, \theta) \sim N(y|\mathbf{w}^T\mathbf{x}, \sigma^2)$$

其中 $\theta = (\mathbf{w}, \sigma^2)$。均值移动变化，方差没有变。

注意：由于假设残差为0均值的正态分布，最小二乘线性回归的残差和为0（图1.1中训练样本分列在模型两侧，即残差有+有-）

### Recall：极大似然估计

- 极大似然估计（Maximize Likelihood Estimator, MLE）定义为（即给定参数$\boldsymbol{\theta}$的情况下，数据$\boldsymbol{D}$出现的概率为$\boldsymbol{p}$，则MLE取使得$\boldsymbol{p}$最大的参数$\boldsymbol{\theta}$）

$$\hat{\theta} = \arg\max_{\theta} \log p(D|\theta)$$

其中（log）似然函数为

$$l(\theta) = \log p(D|\theta) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x_i}, \theta)$$

- 表示在参数为 $\theta$ 的情况下，数据 $D = \{\mathbf{x_i}, y_i\}_{i=1}^{N}$ 出现的概率.
- 极大似然：选择数据出现概率最大的参数。

## 线性回归的MLE

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) = (y_i|\mathbf{w}^T\mathbf{x}_i, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}(y_i - \mathbf{w}^T\mathbf{x}_i)^2)$$

- OLS的似然函数为

$$l(\theta) = \log p(D, \theta) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x_i}, \theta)$$

- 极大似然可等价地写成极小**负log似然损失**(negative log likelihood， **NLL**)(在sklearn中，叫做logloss)

$$\begin{aligned} NLL(\theta) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x_i}, \theta) &= \sum_{i=1}^{N} \log[(\frac{1}{2\pi\sigma^2})^{1/2}\exp(-\frac{1}{2\sigma^2}(y_i - \mathbf{w}^T\mathbf{x}_i)^2)] \\ &= \frac{N}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T x_i)^2 \end{aligned}$$

上式中，观察第二项即可得知OLS的$\boldsymbol{RSS}$项与MLE是等价的关系（相差常数倍不影响目标函数取极值的位置）。

## 正则回归等价于贝叶斯估计

- 假设残差的分布为 $\epsilon \sim N(0, \sigma^2)$，线性回归可写成：

$$p(y_i|\mathbf{x}_i, \theta) \sim N(y_i|\mathbf{w}^T\mathbf{x}_i, \sigma^2)$$

- 若假设参数$\mathbf{w}$中每一维的先验分布为 $w_j \sim N(0, \tau^2)$，并假设$\mathbf{w}$中每一维独立，则

$$p(\mathbf{w}) = \Pi_{j=1}^{D} N(w_j|0,\tau^2) \propto \exp((-\frac{1}{2\tau^2}\sum_{j=1}^{D} w_j^2)$$

$$= \exp(-\frac{1}{2\tau^2}[\mathbf{w}^T\mathbf{w}])$$

  - $w_j$分布的均值为0，即我们偏向于较小的系数值，从而得到的曲线也比较平滑
  - 其中 $1/\tau^2$ 控制先验的强度

- 根据贝叶斯公式，参数的最大后验估计（MAP）为

$$\underset{\mathbf{w}}{argmax}(\sum_{i=1}^{N} logN(y_i|\mathbf{w}^T\mathbf{x}_i,\sigma^2) + \sum_{j=1}^{D} logN(w_j|0,\tau^2))$$

- 等价于最小目标函数

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \frac{\sigma^2}{\tau^2}\sum_{j=1}^{D} w_j^2$$

- 对比岭回归的目标函数

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda||\mathbf{w}||_2^2$$

- 惊喜！$\lambda = \frac{\sigma^2}{\tau^2}$
- L1正则等价于$w_j$先验为Laplace分布，即
  $p(\mathbf{w}) = \Pi_{j=1}^{D} Laplace(w_j|0,\frac{1}{\lambda}) \propto \exp(-\lambda\sum_{j=1}^{D}|w_j|)$。

# 小结之目标函数

- 线性回归模型也可以放到机器学习一般框架：损失函数+正则
-
  - 损失函数：L2损失...
  - 正则：无正则、L2正则、L1正则、L2正则+L1正则、...
- 正则回归模型可视为先验为正则、似然为高斯分布的贝叶斯估计
  - L2正则：先验分布为高斯分布
  - L1正则：先验分布为Laplace分布

# 优化求解

- 线性回归的目标函数

- 无正则的最小二乘线性回归（Ordinary Least Square，OLS）

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

- L2正则的岭回归（Ridge Regression）模型：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda||\mathbf{w}||_2^2$$

- L1正则的Lasso模型：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda||\mathbf{w}||_1$$

## 模型训练

- 模型训练是根据训练数据求解最优模型参数，即求目标函数取极小值的参数

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

- 根据模型的特点和问题复杂程度（数据、模型），可选择不同的优化算法
  - 一阶的导数为0：$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{0}$
  - 二阶导数>0：$\frac{\partial J^2(\mathbf{w})}{\partial \mathbf{w}^2} > \mathbf{0}$

## 1. OLS的优化求解（解析解）

- 将OLS的目标函数写成矩阵形式（为书写简单，省略因子$\frac{1}{N}$，不影响目标函数取极小值的位置）

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$
$$= （\mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T(\mathbf{X}^T\mathbf{y}) + \mathbf{w}^T(\mathbf{X}^T\mathbf{X})\mathbf{w}$$

- 只取与$\mathbf{w}$有关的项，得到

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

- 求导

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

- recall：**向量求导公式**：($\mathbf{w}$为$\mathbf{y}$,$\mathbf{A}$为$\mathbf{X}^T\mathbf{X}$，$\mathbf{X}^T\mathbf{X}$为对称矩阵，$\mathbf{A}^T$=$\mathbf{A}$)：

$$\frac{\partial}{\partial y}(\mathbf{y}^T\mathbf{A}\mathbf{y}) = (\mathbf{A}+\mathbf{A}^T)\mathbf{y}$$

$$\frac{\partial}{\partial \mathbf{a}}(\mathbf{b}^T\mathbf{a}) = \mathbf{b}$$

所以，得到

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

用伪逆矩阵实现OLS的python代码为：

```
#用伪逆矩阵
import numpy as np

from IPython.display import Latex
Latex(r"$\hat{\mathbf{W}}_{OLS}=
(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$")
def OLS_pinv(X, y):
    from numpy.linalg import pinv
    from numpy import transpose,dot
    inv = pinv(dot(X.transpose(), X))
    w_ols = dot(inv, X.transpose())
    w_ols = dot(w_ols, y.transpose())

    return  w_ols
```

实际应用中通常通过对输入$\mathbf{X}$进行**奇异值分解（singular value decomposition，SVD）**，求解更有效（scikit learn中采用的方式）。

对$\mathbf{X}$进行奇异值分解：

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

则

$$\mathbf{X}^T = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T$$
$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{\Sigma}^2$$

所以

$$\begin{aligned}\hat{\mathbf{w}}_{OLS} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= (\mathbf{\Sigma}^2)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{y} \\ &= \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{y}\end{aligned}$$

代码实现为：(注意numpy.linalg中 svd函数的输出为：$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$，即代码中的$\mathbf{V}$为公式中的$\mathbf{V}^T$)

```
# 采用SVD分解
import numpy as np
Latex(r"$\hat{\mathbf{W}}_{OLS}=
(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$")

def OLS_svd(X, y):
    from numpy.linalg import svd,inv
    from numpy import transpose,dot

    U,s,V = svd(X, full_matrices=False)
    S = np.diag(s)

    w_ols = dot(dot(transpose(V),inv(S)), dot(transpose(U),y))

    return  w_ols
```

## 2. OLS的优化求解（梯度下降）

https://www.jiqizhixin.com/articles/2016-11-21-4

### 梯度下降

在求解机器学习算法的模型参数（无约束优化问题）时，**梯度下降**（Gradient Descent）是最常采用的方法之一。梯度下降法是一个一阶最优化算法，通常也称为**最速下降法**。

一元函数$f(x)$在$x$处的梯度为函数$f$在点$x$处的导数。对多元函数$f(x_1, x_2 \dots x_D)$在点$\mathbf{x} = (x_{1,2} \dots x_D)$处，共有$D$个偏导数，分别是$f$在该点关于$x_1$的偏导数，...，以及关于$x_D$的偏导数，将这$D$个偏导数组合成一个$D$维矢量，称为函数$f$在$\mathbf{x}$处的梯度。梯度一般记为$\nabla$或$grad$，即$\nabla f(x_1, x_2 \dots x_D) = g(x_1, x_2 \dots x_D) = [\partial f/\partial x_1, \dots, \partial f/\partial x_D]^T$。

函数$f$在某点梯度指向在该点函数值增长最快的方向。因此要使用梯度下降法找到一个函数的局部极小值，必须向函数上当前点对应**梯度的反方向（负梯度方向）**进行迭代搜索。梯度下降法的每一步通常会这么写

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}},$$

其中$\eta$为学习率（步长）。

**梯度下降的基本步骤:**

1. 确定学习率$\eta$，并初始化参数$\boldsymbol{\theta}^0$
2. 计算目标函数$J(\boldsymbol{\theta})$在当前参数值$\boldsymbol{\theta}^t$的梯度：$\nabla_{\boldsymbol{\theta}} = \frac{\partial J(\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}}$
3. 梯度下降更新参数：$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \nabla_{\boldsymbol{\theta}}$
4. 重复以上步骤, 直到收敛（目标函数的下降量小于某个阈值），得到目标函数的局部极小值$J(\boldsymbol{\theta}^{t+1})$及最佳参数值$\boldsymbol{\theta}^{t+1}$。

### OLS的梯度下降

OLS的目标函数为：

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

梯度为：

$$\nabla_{\mathbf{w}} = g(\mathbf{w}) = 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

参数更新公式为：

$$\begin{aligned}\mathbf{w}^{t+1} &= \mathbf{w}^t - \eta\nabla_{\mathbf{w}} \\ &= \mathbf{w}^t - 2\eta\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \mathbf{w}^t + 2\eta\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))\mathbf{x}_i\end{aligned}$$

或者写出标量形式为：

$$w_j^{t+1} = w_j^t + 2\eta\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))x_{ij}。$$

梯度下降的代码实现为：

```
import numpy as np
from numpy import transpose,dot
import matplotlib.pyplot as plt   #画图
%matplotlib inline
def OLS_bgd( X, y, eta=0.001,  numIterations=10000, epsilon = 0.00001):
    import numpy as np
    from numpy import transpose,dot
    import matplotlib.pyplot as plt   #画图
    %matplotlib inline
    N = len(X)  # number of samples
    D = len(X[0])  # dimension

    theta = np.zeros(D)  #Initilize the parameters
    J_list = []  # Object/cost

    X_transpose = transpose(X)
    for iter in range(0, numIterations):
        y_pred = np.dot(X, theta)
        res = y - y_pred
        J = np.sum(res ** 2) / N  # cost
        J_list.append(J)
        #print("iter %s | J: %.3f" % (iter, J))

        gradient = np.dot(X_transpose, res)
        theta += eta * gradient  # update,去掉了常数倍2
```

```
        if iter !=0 and abs(J_list[iter-1]-J)/J_list[iter-1] < epsilon:
            break;

    plt.plot(range(iter+1), J_list, "b-")
    return theta
```

注意：梯度下降的参数学习率和迭代次数需仔细选取。为了加快迭代的收敛速度，建议对数据进行标准化。若样本数$N$较多，梯度可以用样本数$N$归一化，以免梯度值（在所有样本上累加）太大。

## 3. OLS的优化求解（随机梯度下降，Stochastic Gradient Descent, SGD）

- 在上述梯度下降算法中，梯度 $g(\mathbf{w}) = \sum_{i=1}^{N} 2(f(\mathbf{x}_i) - y_i)\mathbf{x}_i$
  - 利用所有的样本，因此被称为“**批处理**梯度下降”
- 随机梯度下降：每次只用一个样本 $(\mathbf{x}_t, y_t)$

$$g(\mathbf{w}) = 2(f(\mathbf{x}_t) - y_t)\mathbf{x}_t$$

  - 通常收敛会更快，且不太容易陷入局部极值
  - 亦被称为**在线学习**(Online Learning)
  - 对大样本数据集尤其有效
- SGD的变形：
  - 可用于离线学习：每次看一个样本，对所有样本可重复多次循环使用（一次循环称为一个epoch）
  - 每次可以不止看一个样本，而是看一些样本（mini-batch）
  -

随机梯度下降的代码为:

```
# 随机梯度下降
def OLS_sgd(X, y, eta = 0.001, epochs =10000, epsilon = 0.00001):
    import numpy as np
    from numpy import transpose,dot
    import matplotlib.pyplot as plt   #画图
    %matplotlib inline
    N = len(X)     # number of samples
    D = len(X[0])  # dimension

    theta = np.zeros(D)  #Initilize the parameters
    J_list = []  # Object/cost

    for iter in range(0, epochs):
        J = 0.0
        #将训练样本打散顺序
```

```
        kk = np.random.permutation(N)
        for t in range(N):
        #每次一个训练样本
            y_pred = np.dot(X[t], theta)
            res = y[t] - y_pred

            gradient = np.dot(X[t], res)
            theta += eta * gradient  # update

            J += res * res;

        J_list.append(J)
        #print("iter %s | J: %.3f" % (iter, J))

        if iter !=0 and abs(J_list[iter-1]-J)/J_list[iter-1] < epsilon:
            break;

    plt.plot(range(iter+1), J_list, "b-")
    return theta
```

注意：scikit learn建议样本数目超过10000采用梯度下降。

## 4. 岭回归的优化求解

- 岭回归的目标函数与OLS只相差一个正则项（也是**w**的二次函数），所以类似可得：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda||\mathbf{w}||_2^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$$

- 求导，得到

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

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

计算时也可以采用对$\mathbf{X}$进行SVD得到：

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$
$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}\mathbf{U}^T\mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{V}\mathbf{D}^2\mathbf{V}^T$$
$$\hat{\mathbf{W}}_{Ridge} = \mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{y}$$

岭回归的优化实现代码很容易从OLS的优化推导，故此省略。

# 5. Lasso的优化求解——坐标轴下降法

Lasso的优化求解推荐采用**坐标下降法**。

## 坐标轴下降法

坐标轴下降法顾名思义，是沿着坐标轴的方向去下降。为了找到一个函数的局部极小值，坐标轴下降法在每次迭代中可以在当前点处沿一个坐标方向进行一维搜索。在整个过程中循环使用不同的坐标方向。一个周期的一维搜索迭代过程相当于一个梯度迭代。 而梯度下降是根据目标函数的导数（梯度）来确定搜索方向，沿着梯度的负方向下降。该梯度方向可能不与任何坐标轴平行。不过梯度下降和坐标轴下降的共性是都是迭代法，通过启发式的方式一步步迭代求解函数的最小值其实，坐标下降法在稀疏矩阵上的计算速度非常快，同时也是Lasso回归最快的解法。

坐标轴下降法的数学依据主要是如下结论（此处不做证明）：一个可微的凸函数$J(\boldsymbol{\theta})$, 其中 $\boldsymbol{\theta}$ 是$D$维向量，即有$D$个维度。如果在某一点$\overline{\boldsymbol{\theta}}$，使得$J(\boldsymbol{\theta})$在每一个坐标轴$\overline{\boldsymbol{\theta}}_j(j=1,\ldots,D)$, 上都是最小值，那么$J(\overline{\boldsymbol{\theta}})$就是一个全局的最小值。于是我们的优化目标就是在$\boldsymbol{\theta}$的$D$个坐标轴上(或者说向量的方向上)对损失函数做迭代的下降，当所有的坐标轴上的$\boldsymbol{\theta}_j(=1,\ldots,D)$都达到收敛时，我们的损失函数最小，此时的$\boldsymbol{\theta}$即为我们要求的结果。

算法过程如下：

1. 初始化$\boldsymbol{\theta}$为一随机初值。记为$\boldsymbol{\theta}^{(0)}$，上标括号中的数字表示迭代次数；
2. 对第$t$轮迭代，我们依次计算$\theta_j^{(t)}(=1,\ldots,D)$：

$$\theta_j^{(t)} \in \underbrace{argmin}_{\theta_j} J(\theta_1^{(t)},\theta_2^{(t)},\ldots\theta_{i-1}^{(t)},\theta_i,\theta_{i+1}^{(t-1)},\ldots,\theta_D^{(t-1)})$$

   也就是说$\theta_j^{(t)}$是使$J(\theta_1^{(t)},\theta_2^{(t)},\ldots\theta_{i-1}^{(t)},\theta_i,\theta_{i+1}^{(t-1)},\ldots,\theta_D^{(t-1)})$ 最小化的$\theta_j$的值。此时$J(\boldsymbol{\theta})$ 中只有$\boldsymbol{\theta}_j^{(t)}$是变量，其余均为常量，因此最小值容易通过求导求得。
3. 检查向量$\boldsymbol{\theta}^{(t)}$ 和$\boldsymbol{\theta}^{(t-1)}$向量在各个维度上的变化情况，如果在所有维度上变化都足够小，那么$\boldsymbol{\theta}^{(t)}$ 即为最终结果，否则转入第2步，继续第$t+1$轮的迭代。

### Lasso优化求解之坐标轴下降法

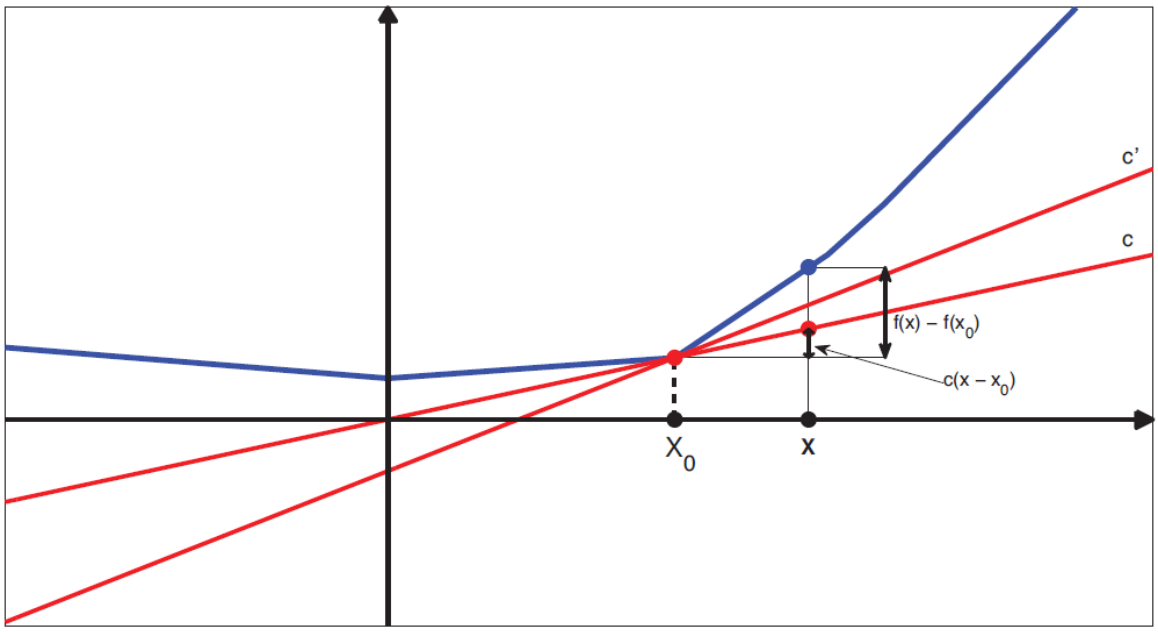
**次梯度**

- Lasso的目标函数为 $J(\mathbf{w},\lambda)=RSS(\mathbf{w})+\lambda||\mathbf{w}||_1$
- 但项 $||\mathbf{w}||_1$ 在 $\mathbf{w}_j=0$ 处不可微（不平滑优化问题）
- 为了处理不平滑函数，扩展导数的表示，定义一个 (凸)函数 $f$ 在点处 $x_0$ 的*次梯度*(subgradient)或次导数(subderivative)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。
- 定义区间[a,b]的子梯度集合为

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




- 所有次梯度的区间称为函数$f$在处$x_0$的次微分(subdifferential)，用 $\partial f(x)|_{x_0}$ 表示
- 例：绝对值函数 $f(x)=|x|$，其次梯度为

$$\partial f(x)=\begin{cases}\{-1\} & \text{if } x<0\\ [-1,+1] & \text{if } x=0\\ \{+1\} & \text{if } x>0\end{cases}$$

- 如果函数处处可微，$\partial f(x)=\frac{df(x)}{dx}$
- 同标准的微积分类似，可以证明当且仅当 $0\in\partial f(x)|_{\hat{x}}$ 时，为 $f$ 的局部极值点。

**将上述结论代入Lasso问题**

- 目标函数为：

$$J(\mathbf{w},\lambda)=RSS(\mathbf{w})+\lambda||\mathbf{w}||_1=\sum_{i=1}^{N}(y_i-\mathbf{w}^T\mathbf{x_i})^2+\lambda||\mathbf{w}||_1$$

- 可微项$RSS(\mathbf{w})$的梯度：

$$
\begin{aligned}
\frac{\partial}{\partial w_j} RSS(\mathbf{w}) &= \frac{\partial}{\partial w_j} \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x_i})^2 \\
&= \frac{\partial}{\partial w_j} \sum_{i=1}^{N} (y_i - (\mathbf{w}_{-j}^T \mathbf{x}_{i,-j} + w_j x_{i,j}))^2 \\
&= -2 \sum_{i=1}^{N} (y_i - \mathbf{w}_{-j}^T \mathbf{x}_{i,-j} - w_j x_{i,j}) x_{i,j} \\
&= 2 \sum_{i=1}^{N} x_{i,j}^2 w_j - 2 \sum_{i=1}^{N} (y_i - \mathbf{w}_{-j}^T \mathbf{x}_{i,-j}) x_{i,j} \\
&= a_j w_j - c_j
\end{aligned}
$$

其中$\mathbf{w}_{-j}$表示向量$\mathbf{w}$中除了第$j$维的其他$D-1$个元素，$\mathbf{x}_{i,-j}$表示向量$\mathbf{x}_i$中除了第$j$维的其他$D-1$个元素，$x_{i,j}$表示向量$\mathbf{x}_i$中第$j$维元素，

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

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

公式中$\mathbf{w}_{-j}^T \mathbf{x}_{i,-j}$为$D$维特征中去掉第$j$维特征的其余$D-1$维特征对应的预测，则$(y_i - \mathbf{w}_{-j}^T \mathbf{x}_{i,-j})$为对应的预测残差，$c_j$为第$j$维特征与该预测残差的相关性。

上面是坐标轴下降算法中对第$j$维坐标参数$w_j$进行分析。

- 将上述结论代入，得到目标函数的子梯度(subgradient )

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

当0属于目标函数的子梯度时，目标函数取极小值，即

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

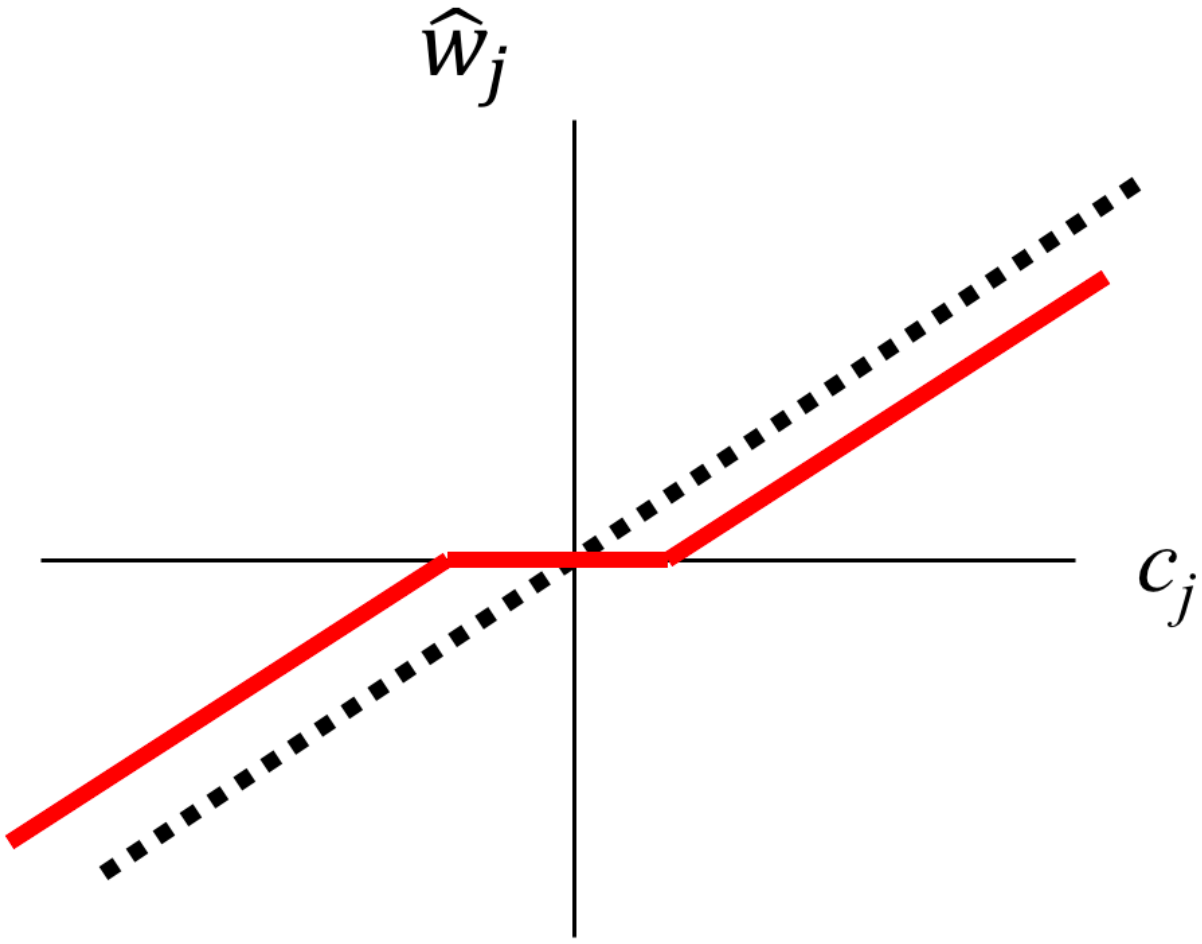
上式可简写为：

$$
\hat{w}_j = \text{soft}(\frac{c_j}{a_j}; \frac{\lambda}{a_j})
$$

其中

$$
\text{soft}(a; \delta) \triangleq \text{sgn}(a)(|a| - \delta)_+
$$

称为**软阈值（soft thresholding）**。其中$x_+ = max(x, 0)$为$x$的正的部分，sgn为符号函数。软阈值如下图所示，图中黑色虚线为最小二乘结果$\hat{w}_j = c_j / a_j$，红色实线为L1正则结果$\hat{w}_j(c_j)$，根据情况将最小二乘结果往下平移$\lambda$（$c_j > \lambda$）、将最小二乘结果往上平移$\lambda$（$c_j < -\lambda$）、或者置为0（$-\lambda < c_j < \lambda$）。




**因此$\mathbf{w}$的坐标下降优化过程为：**

- 预计算

$$a_j = 2 \sum_{i=1}^{N} x_j^2$$

- 初始化参数$\mathbf{w}$（全0或随机）
- 循环直到收敛:
- - for $j = 1, \dots D$
  - - 计算

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

    - 选择变化幅度最大的维度或者轮流更新$w_j$:

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

Lasso优化的坐标轴下降实现代码为：

```
def lasso_cd(X, y, lambd =1, niter = 10000, threshold =0.0001):
    ''' 通过坐标下降(coordinate descent)法获取LASSO回归系数
    '''
    # 计算残差平方和
    rss = lambda X, y, w: (y - X*w).T*(y - X*w)

    # 初始化回归系数w.
    N, D = X.shape
    w = np.matrix(np.zeros((n, 1)))
    r = rss(X, y, w)

    # 使用坐标下降法优化回归系数w
    for j in range(D):  # 常量值a_j
        a[j] = X[:,j].transpose().X[:,j]

    for it in niter:
        for j in range(D):
            # 计算c_j
            c_j = 0
            for i in range(N):
                c_j += X[i, j]*(y[i, 0] - sum([X[i,k]*w[k, 0] for k in
range(D) if k != j]))
            if c_j < -lambd/2:
                w_j = (c_j + lambd/2)/a[j]
            elif p_j > lambd/2:
                w_j = (c_j - lambd/2)/a[j]
            else:
                w_j = 0
            w[j, 0] = w_j
        r_prime = rss(X, y, w)
        delta = abs(r_prime - r)[0, 0]
        r = r_prime
        print('Iteration: {}, delta = {}'.format(it, delta))
        if delta < threshold:
            break
    return w
```

# 小结：线性回归之优化求解

- 线性回归模型比较简单
- 当数据规模较小时，可直接解析求解
    - scikit learn中的实现采用SVD分解实现
- 当数据规模较大时，可采用随机梯度下降
    - Scikit learn提供一个SGDRegression类
- 岭回归求解类似OLS，采用SVD分解实现
- Lasso优化求解采用坐标轴下降法

# 模型评估与模型选择

模型训练好后，需要在校验集上采用一些度量准则检查模型预测的效果，并根据模型评估结果进行模型选择（选择最佳的模型超参数）。

- 线性回归的目标函数
    - 无正则的最小二乘线性回归（Ordinary Least Square，OLS）

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

    - L2正则的岭回归（Ridge Regression）模型：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda||\mathbf{w}||_2^2$$

    - L1正则的Lasso模型：

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 + \lambda|\mathbf{w}|$$

对最小二乘线性回归，模型没有需要调整的超参数。岭回归模型和Lasso模型的**超参数为正则系数λ**。

- 校验集

通常给定一个任务时只给了训练集，没有专门的校验集。因此我们需要从训练集中分离一部分样本作为校验。

当训练样本较多时，可以直接从训练集中分离一部分样本作为校验。在scikit-learn中，调用**train_test_split**得到新的训练集和校验集。

当训练样本没那么多时，可以采用**交叉验证**方式，每个样本轮流作为训练样本和校验样本。

对岭回归和Lasso，scikit-learn提供了高效的广义留一交叉验证（leave-one-out cross-validation，LOOCV），即每次留出一个样本做校验，相当于$N$折交叉验证，类名称分别为RidgeCV和LassoCV。

对一般的模型，Scikit learn将交叉验证与网格搜索合并为一个函数：
sklearn.model_selection.GridSearchCV

## 评价准则

对回归模型，模型训练好后，可用一些度量准则检查模型拟合的效果：

- 开方均方误差（rooted mean squared error，RMSE）：

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

- 平均绝对误差（mean absolute error，MAE）：

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

- R2 score：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线性回归模型的缺省评价准则）

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

- 也可以检查残差的分布：回归建模时我们假设残差的分布为0均值的正态分布$N(0, \sigma^2)$。

## Scikit learn中的回归评价指标

| **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 |

## 线性回归中的模型选择

sklearn.model_selection

- Scikit learn中的model selection模块提供模型选择功能
- - 对于线性模型，留一交叉验证（*N*折交叉验证，亦称为leave-one-out cross-validation，LOOCV）有更简便的计算方式，因此Scikit learn提供了RidgeCV类和LassoCV类
  - 后续课程将讲述一般模型的交叉验证和参数调优GridSearchCV

## RidgeCV

- RidgeCV中超参数λ用alpha表示
- RidgeCV(*alphas**=(0.1, 1.0, 10.0), *fit_intercept**=True, *normalize**=False, *scoring**=None, *cv**=None, *gcv_mode**=None, **store_cv_values**=False)*

```
from sklearn.linear_model import  RidgeCV

alphas = [0.01, 0.1, 1, 10,100]

reg = RidgeCV(alphas=alphas, store_cv_values=True)
reg.fit(X_train, y_train)
```

## LassoCV

- LassoCV的使用与RidgeCV类似
- Scikit learn还提供一个与Lasso类似的LARS（least angle regression，最小角回归），二者仅仅是优化方法不同，目标函数相同。
- 当数据集中特征维数很多且存在共线性时，LassoCV更合适。

# 小结：线性回归之模型选择

- 采用交叉验证评估模型预测性能，从而选择最佳模型
- - 回归性能的评价指标
  - 线性模型的交叉验证通常直接采用广义线性模型的留一交叉验证进行快速模型评估
  - - Scikit learn中对RidgeCV和LassoCV实现该功能