



人工智能与机器学习



讲师：曾江峰



E-mail: jfzeng@ccnu.edu.cn

A.I.



大数据，成就未来



回归

回归 vs 分类

回归与分类问题

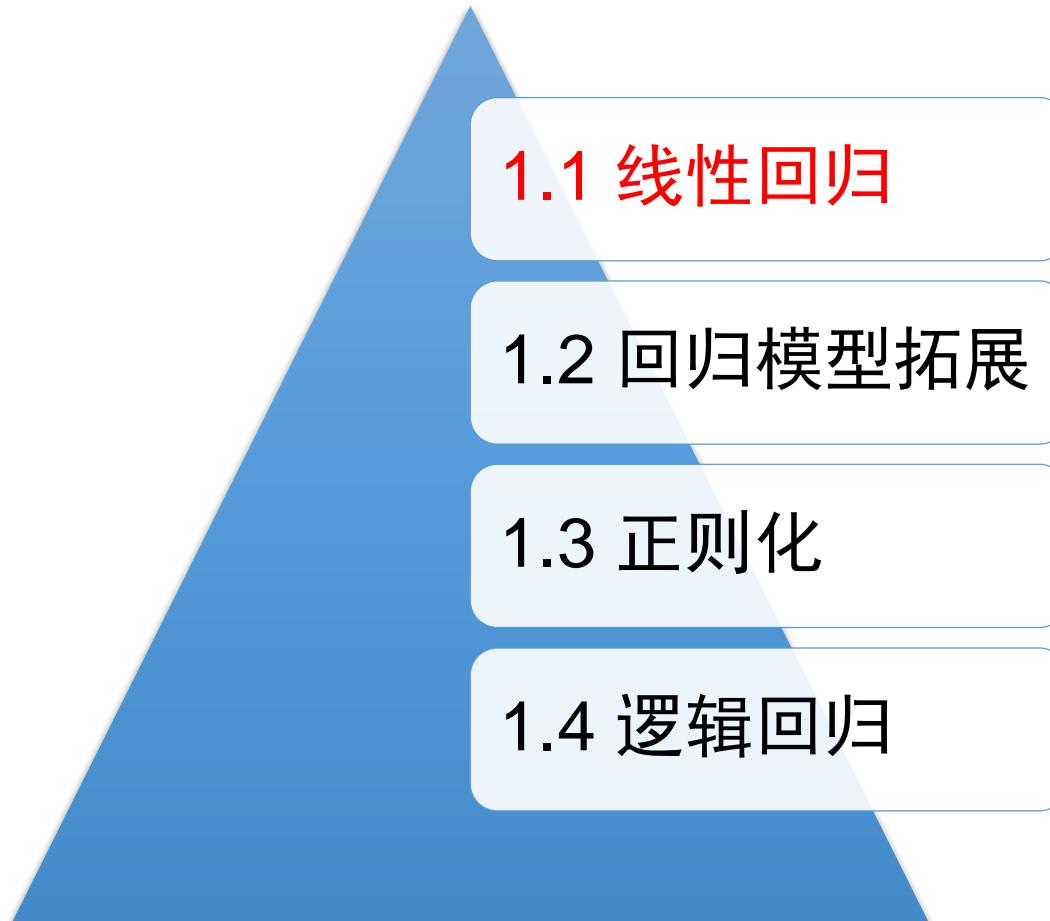
回归问题

输出是连续性数值，
比如温度，身高，
气温等

分类问题

输出是定性输出
比如阴或者晴，
好或者坏

outline



1.1 线性回归

1.2 回归模型拓展

1.3 正则化

1.4 逻辑回归



线性回归

- 线性回归研究单一因变量和一个或多个自变量之间的关系。
 - 自变量作为因变量发生的原因（正负相关）
- 线性回归是一种形式简单但包含机器学习主要建模思想的模型。

给定一组由输入 x 和输出 y 构成的数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ ，
其中， $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$ 。线性回归就是通过训练学习得到一个线性模型来
最大限度地根据输入 x 拟合输出 y 。



一个一元线性回归案例

问题描述

20岁男子标准体重

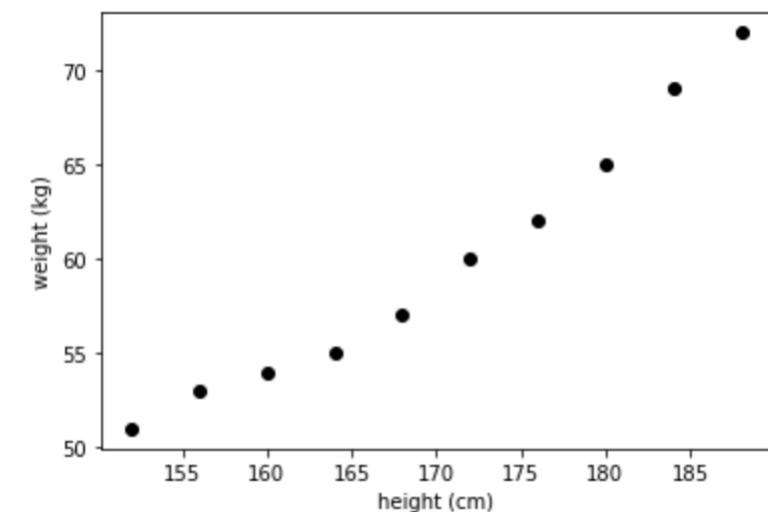
身高	体重
152	51
156	53
160	54
164	55
168	57
172	60
176	62
180	65
184	69
188	72

特征变量 x 目标变量 y

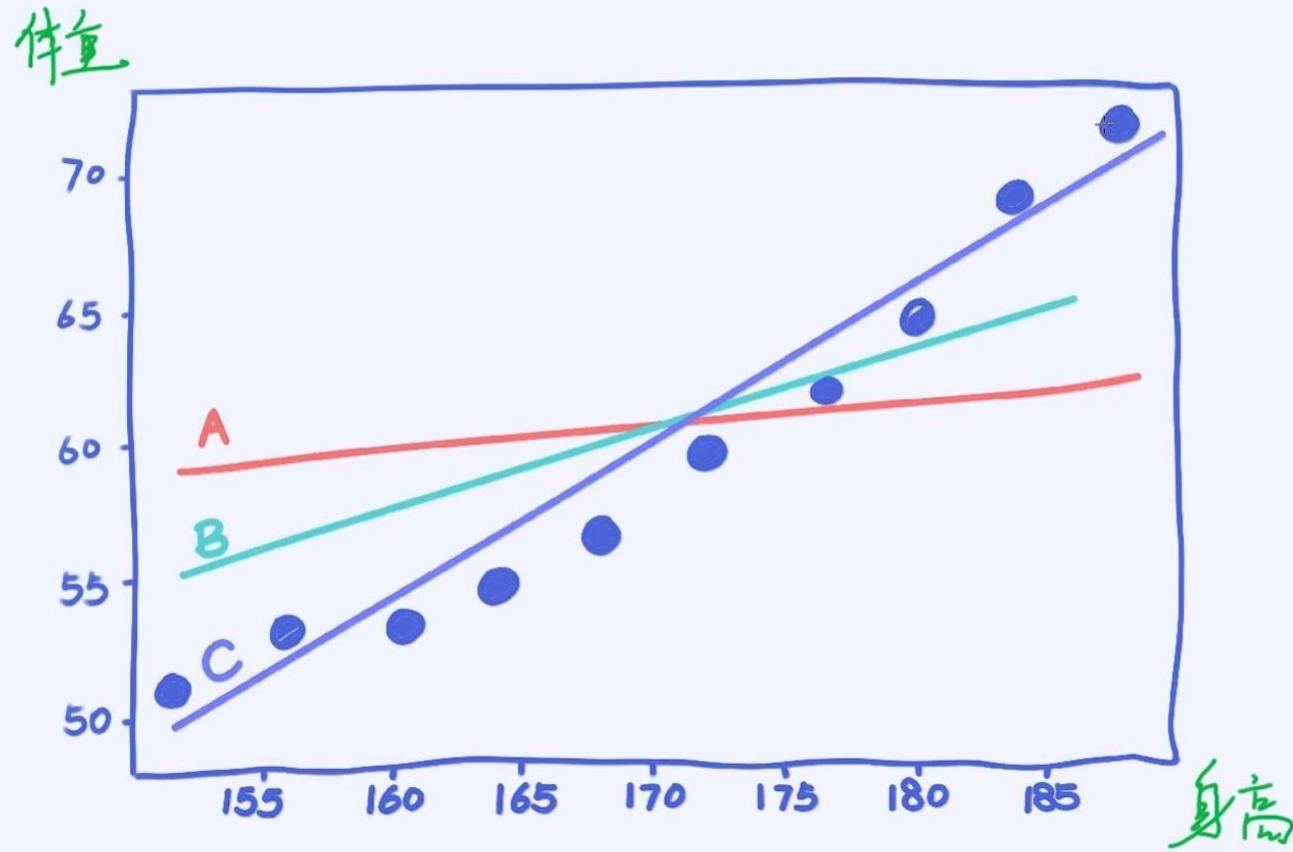
左边的图给出了20岁男子的平均体重，我们目标是针对没有在表格里列出来的身高值（比如185），预测其标准体重

可视化身高和体重

```
① 1 # 创建数据集，把数据写入到numpy数组
② 2 import numpy as np # 引用numpy库，主要用来做科学计算
③ 3 import matplotlib.pyplot as plt # 引用matplotlib库，主要用来画图
④ 4 data = np.array([[152,51],[156,53],[160,54],[164,55],
⑤ 5 [168,57],[172,60],[176,62],[180,65],
⑥ 6 [184,69],[188,72]])
⑦
⑧ 8 # 打印出数组的大小
⑨ 9 print(data.shape)
⑩
⑪
⑫ 12 # 从data中提取出身高和体重，分别存放在x，y变量中
⑬ 13 x, y = data[:,0].reshape(-1,1), data[:,1]
⑭
⑮ 15 # 在二维空间里画出身高和体重的分布图
⑯ 16 plt.scatter(x, y, color='black')
⑰ 17 plt.xlabel('height (cm)')
⑱ 18 plt.ylabel('weight (kg)')
⑲ 19 plt.show()
```



拟合数据



- Which one is the best?
- How to find?

任务描述

- 在这里我们使用一个线性回归模型来拟合身高-体重的数据，也就是希望通过身高来预测标准体重。在已创建的数据集里已经给出了部分样本，所以一旦我们训练出一个模型，则可以对任意的身高数据都可以给出一个体重的预测值。这里需要分成3个部分。
 - 第一部分是数据的创建；
 - 第二部分是利用线性回归来拟合数据；
 - 第三部分是利用已经训练好的模型去预测任意的体重并画出对任意体重的预测值，这里需要使用plot，并且指定颜色为蓝色(color="blue")。

提示一

- 1. 创建LinearRegression实例，这里需要调用sklearn的线性模型模块即可，暂时不需要指定任何的输入参数。
- 2. 对于模型的训练，可以直接使用模型自带的fit函数。
- 3. 画出拟合曲线。针对所有给定的x，使用predict函数计算出在这个模型下的预测值，并把x和预测值通过matplotlib库来去可视化。

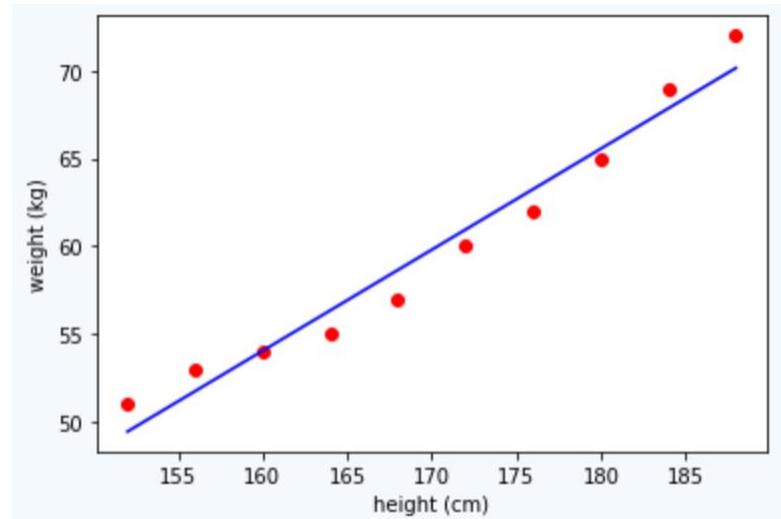
提示二

- 1. 在sklearn里线性回归模型在linear_model模块里。可以参考
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- 2. 对于训练数据 (x, y) ，通常通过sklearn中的`fit(x, y)`来实现模型的训练。这里 x 可以看做是特征部分， y 可以看做是真实的值或者标签。
- 3. 调用`plt.plot()`函数即可以画出一条线。

```

1 from sklearn import datasets, linear_model # 引用 sklearn库, 主要为了使用其中的线性回归模块
2
3 # 创建数据集, 把数据写入到numpy数组
4 import numpy as np # 引用numpy库, 主要用来做科学计算
5 import matplotlib.pyplot as plt # 引用matplotlib库, 主要用来画图
6 data = np.array([[152,51],[156,53],[160,54],[164,55],
7                 [168,57],[172,60],[176,62],[180,65],
8                 [184,69],[188,72]])
9
10 # 打印出数组的大小
11 print(data.shape)
12 x,y = data[:,0].reshape(-1,1), data[:,1]
13
14 # TODO 1. 实例化一个线性回归的模型
15 regr = linear_model.LinearRegression()
16
17 # TODO 2. 在x,y上训练一个线性回归模型。 如果训练顺利, 则regr会存储训练完成之后的结果模型
18 regr.fit(x, y)
19
20 # TODO 3. 画出身高与体重之间的关系
21 plt.scatter(x, y, color='red')
22
23 # 画出已训练好的线条
24 plt.plot(x, regr.predict(x), color='blue')
25
26 # 画x,y轴的标题
27 plt.xlabel('height (cm)')
28 plt.ylabel('weight (kg)')
29 plt.show() # 展示
30
31 # 利用已经训练好的模型去预测身高为163的人的体重
32 print ("Standard weight for person with 163 is %.2f"% regr.predict([[163]]))

```



Standard weight for person with 163 is 55.77

线性回归的数学推导

梯度下降 $w \leftarrow w - \alpha \frac{\partial L}{\partial w}$

$$b \leftarrow b - \alpha \frac{\partial L}{\partial b}$$

- 以一元线性回归为例

优化目标 $(w^*, b^*) = \arg \min \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \arg \min \sum_{i=1}^n (wx_i + b - y_i)^2$

最优参数求解 $\frac{\partial L(w, b)}{\partial w} = \frac{\partial}{\partial w} \left[\sum_{i=1}^n (wx_i + b - y_i)^2 \right] = 2 \cdot \left(w \sum_{i=1}^n x_i^2 - \sum_{i=1}^n y_i x_i + b \sum_{i=1}^n x_i \right)$

两种解法：

最小二乘法

梯度下降

$$\frac{\partial L(w, b)}{\partial b} = \frac{\partial}{\partial b} \left[\sum_{i=1}^n (wx_i + b - y_i)^2 \right] = 2 \cdot \left(nb - \sum_{i=1}^n (y_i - wx_i) \right)$$

最小二乘法

令导数为0

$$w^* = \frac{\sum_{i=1}^n y_i (x_i - \bar{x})^2}{\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2} \quad b^* = \frac{1}{n} \sum_{i=1}^n (y_i - wx_i) \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

线性回归的数学推导

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- 线性回归

$$\hat{\mathbf{w}} = (w; b) \quad \mathbf{L} = (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$$

优化目标 $\hat{\mathbf{w}}^* = \arg \min (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$

最优参数求解 $\frac{\partial \mathbf{L}}{\partial \hat{\mathbf{w}}} = \frac{\partial \mathbf{y}^T \mathbf{y}}{\partial \hat{\mathbf{w}}} - \frac{\partial \mathbf{y}^T \mathbf{X}\hat{\mathbf{w}}}{\partial \hat{\mathbf{w}}} - \frac{\partial \hat{\mathbf{w}}^T \mathbf{X}^T \mathbf{y}}{\partial \hat{\mathbf{w}}} + \frac{\partial \hat{\mathbf{w}}^T \mathbf{X}^T \mathbf{X}\hat{\mathbf{w}}}{\partial \hat{\mathbf{w}}}$

根据矩阵
微分公式 $\frac{\partial \mathbf{a}^T \mathbf{z}}{\partial \mathbf{z}} = \frac{\partial \mathbf{z}^T \mathbf{a}}{\partial \mathbf{z}} = \mathbf{a} \quad \frac{\partial \mathbf{z}^T \mathbf{A} \mathbf{z}}{\partial \mathbf{z}} = (\mathbf{A} + \mathbf{A}^T) \mathbf{z}$

$$\frac{\partial \mathbf{L}}{\partial \hat{\mathbf{w}}} = 2 \mathbf{X}^T (\mathbf{X}\hat{\mathbf{w}} - \mathbf{y})$$

当 $\mathbf{X}^T \mathbf{X}$ 为满秩矩阵或者正定矩阵时，令导数为 0

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

线性回归的数学推导

- 线性回归

当 $\mathbf{X}^T \mathbf{X}$ 不为满秩矩阵或者正定矩阵时，通过对 $\mathbf{X}^T \mathbf{X}$ 添加正则化项来使得该矩阵可逆。

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

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$



逆必须存在，则行列式不能为0



行列式不能为0，则要求矩阵必须满秩

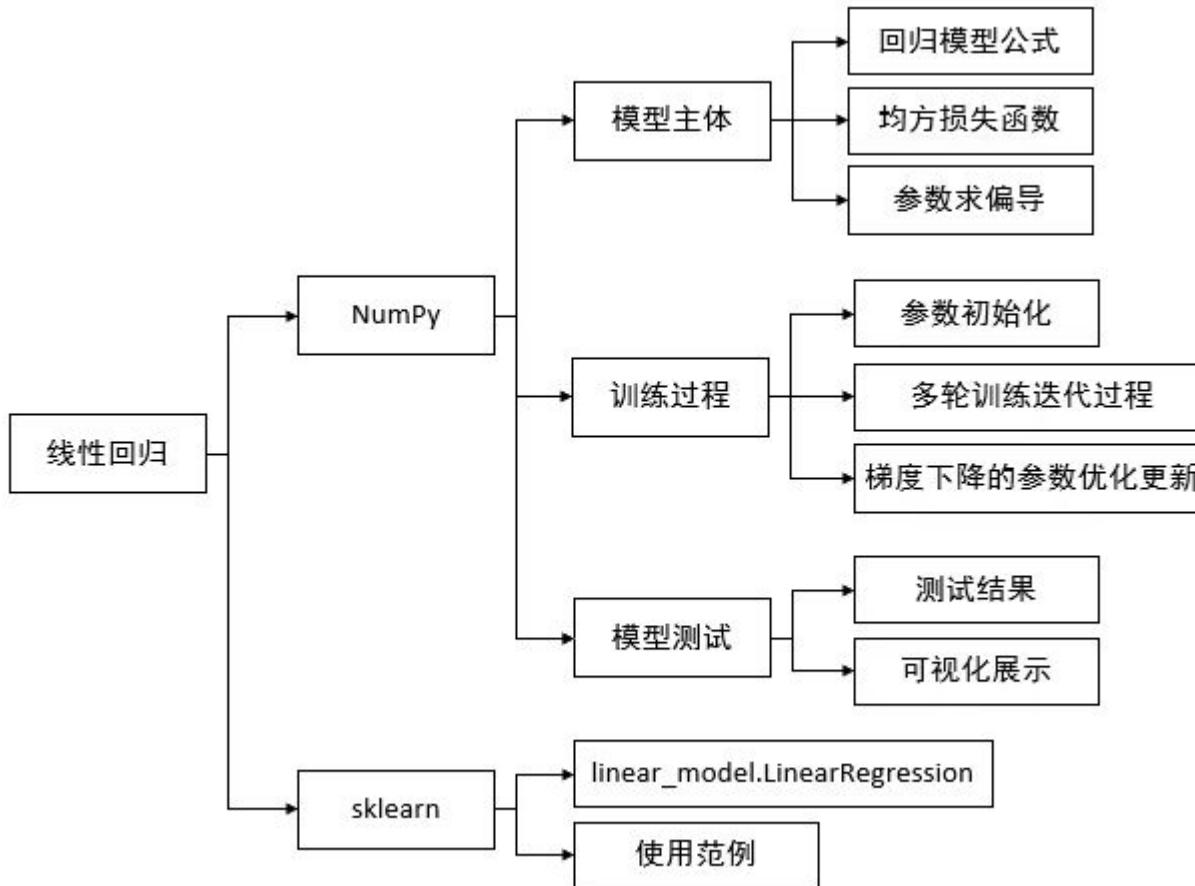


矩阵必须满秩，则特征之间不能存在多重共线性

线性回归的代码实现

- 按照机器学习三要素的原则，逐步搭建线性回归代码框架。
- 使用numpy实现一个线性回归模型——糖尿病指数预测
- 基于scikit-learn的模型实现

使用**numpy**实现一个线性回归模型——糖尿病指数预测



```

### 初始化模型参数
def initialize_params(dims):
    '',
    输入:
    dims: 训练数据变量维度
    输出:
    w: 初始化权重参数值
    b: 初始化偏差参数值
    '',
    # 初始化权重参数为零矩阵
    w = np.zeros((dims, 1))
    # 初始化偏差参数为零
    b = 0
    return w, b

```

```

### 定义模型主体部分
### 包括线性回归公式、均方损失和参数偏导三部分
def linear_regression(X, y, w, b):
    '',
    输入:
    X: 输入变量矩阵
    y: 输出标签向量
    w: 变量参数权重矩阵
    b: 偏差项
    输出:
    y_hat: 线性模型预测输出
    loss: 均方损失值
    dw: 权重参数一阶偏导
    db: 偏差项一阶偏导
    '',
    # 训练样本数量
    num_train = X.shape[0]
    # 训练特征数量
    num_feature = X.shape[1]
    # 线性回归预测输出
    y_hat = np.dot(X, w) + b
    # 计算预测输出与实际标签之间的均方损失
    loss = np.sum((y_hat - y) ** 2) / num_train
    # 基于均方损失对权重参数的一阶偏导数
    dw = np.dot(X.T, (y_hat - y)) / num_train
    # 基于均方损失对偏差项的一阶偏导数
    db = np.sum((y_hat - y)) / num_train
    return y_hat, loss, dw, db

```

定义线性回归模型训练过程

```

def train(X, y, learning_rate=0.01, epochs=10000):
    '',
    输入:
    X: 输入变量矩阵
    y: 输出标签向量
    learning_rate: 学习率
    epochs: 训练迭代次数
    输出:
    loss_his: 每次迭代的均方损失
    params: 优化后的参数字典
    grads: 优化后的参数梯度字典
    '',
    # 记录训练损失的空列表
    loss_his = []
    # 初始化模型参数
    w, b = initialize_params(X.shape[1])
    # 迭代训练
    for i in range(1, epochs):
        # 计算当前迭代的预测值、损失和梯度
        y_hat, loss, dw, db = linear_regression(X, y, w, b)
        # 基于梯度下降的参数更新
        w += -learning_rate * dw
        b += -learning_rate * db
        # 记录当前迭代的损失
        loss_his.append(loss)
        # 每1000次迭代打印当前损失信息
        if i % 1000 == 0:
            print('epoch %d loss %f' % (i, loss))
        # 将当前迭代步优化后的参数保存到字典
        params = {
            'w': w,
            'b': b
        }
        # 将当前迭代步的梯度保存到字典
        grads = {
            'dw': dw,
            'db': db
        }
    return loss_his, params, grads

```

定义线性回归预测函数

```

def predict(X, params):
    '',
    输入:
    X: 测试数据集
    params: 模型训练参数
    输出:
    y_pred: 模型预测结果
    '',
    # 获取模型参数
    w = params['w']
    b = params['b']
    # 预测
    y_pred = np.dot(X, w) + b
    return y_pred

```

```

# 导入sklearn diabetes数据接口
from sklearn.datasets import load_diabetes
# 导入sklearn打乱数据函数
from sklearn.utils import shuffle
# 获取diabetes数据集
diabetes = load_diabetes()
# 获取输入和标签
data, target = diabetes.data, diabetes.target
# 打乱数据集
X, y = shuffle(data, target, random_state=13)
# 按照8/2划分训练集和测试集
offset = int(X.shape[0] * 0.8)
# 训练集
X_train, y_train = X[:offset], y[:offset]
# 测试集
X_test, y_test = X[offset:], y[offset:]
# 将训练集改为列向量的形式
y_train = y_train.reshape((-1, 1))
# 将验证集改为列向量的形式
y_test = y_test.reshape((-1, 1))
# 打印训练集和测试集维度
print("X_train's shape: ", X_train.shape)
print("X_test's shape: ", X_test.shape)
print("y_train's shape: ", y_train.shape)
print("y_test's shape: ", y_test.shape)
# 线性回归模型训练
loss_his, params, grads = train(X_train, y_train, 0.01, 200000)
# 打印训练后得到模型参数
print(params)
# 基于测试集的预测
y_pred = predict(X_test, params)
# 打印前五个预测值
print(y_pred[:5])
print(y_test[:5])

```

```

### 评价指标: 定义R2系数函数
def r2_score_selfdefine(y_test, y_pred):
    """
    输入:
    y_test: 测试集标签值
    y_pred: 测试集预测值
    输出:
    r2: R2系数
    """
    # 测试标签均值
    y_avg = np.mean(y_test)
    # 总离差平方和
    ss_tot = np.sum((y_test - y_avg)**2)
    # 残差平方和
    ss_res = np.sum((y_test - y_pred)**2)
    # R2计算
    r2 = 1 - (ss_res/ss_tot)
    return r2

```

```

### 可视化
import matplotlib.pyplot as plt
f = X_test.dot(params['w']) + params['b']
#绘制测试结果散点图
plt.scatter(range(X_test.shape[0]), y_test)
plt.plot(f, color = 'darkorange')
plt.xlabel('X_test')
plt.ylabel('y_test')
plt.show()
#绘制训练loss曲线
plt.plot(loss_his, color = 'blue')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()

```

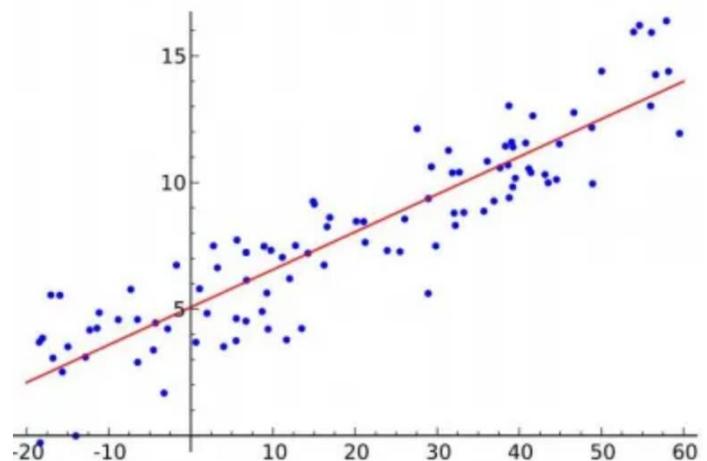
基于**scikit-learn**的模型实现

- 课后作业：基于sklearn的糖尿病指数预测
- 要求：
 - 数据集随机打乱，按照8/2分训练和测试；
 - 调用LinearRegression模型；
 - 在测试集上计算R2系数；
 - 打印出训练好模型的参数；
 - 在测试集上可视化（测试集的散点图和预测值的折线图）

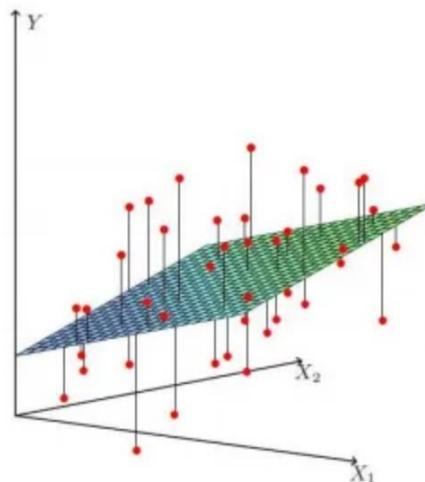
线性回归 vs 非线性回归

- 一元线性回归是指独立特征变量只有一个时的线性回归，独立特征变量为两个及以上的线性回归称为多元线性回归。
- 另外，当独立特征变量只有一个，但包含 x 的幂次方这种特征变量的次方项的线性回归叫做多项式线性回归。

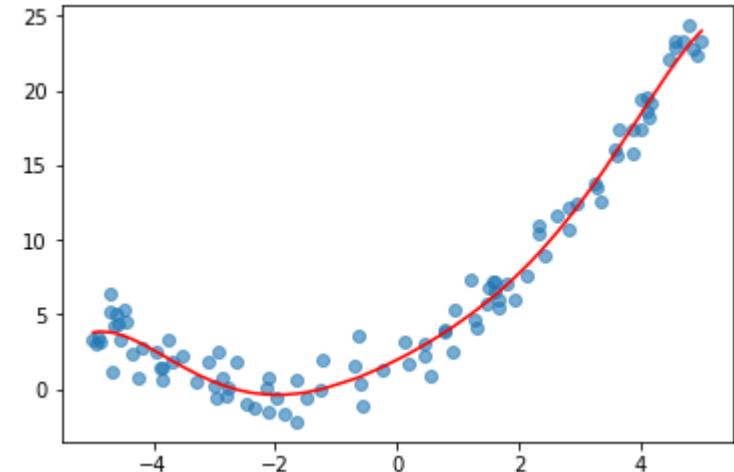
线性回归的种类	示例
一元回归	$y = w_1 x_1 + b$
多元回归	$y = w_1 x_1 + w_2 x_2 + b$
多项式回归	$y = w_1 x_1 + w_2 x_1^2 + w_3 x_1^3 + b$



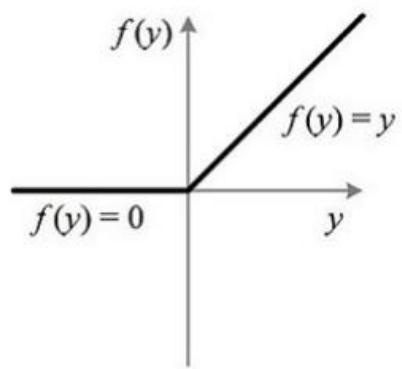
一元线性回归



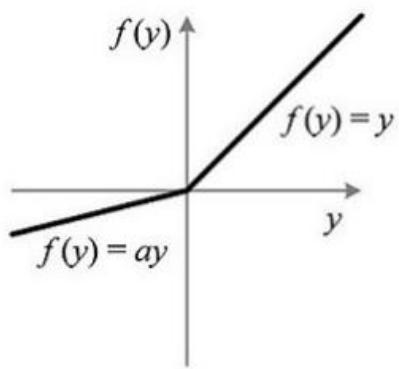
二元线性回归



多项式线性回归

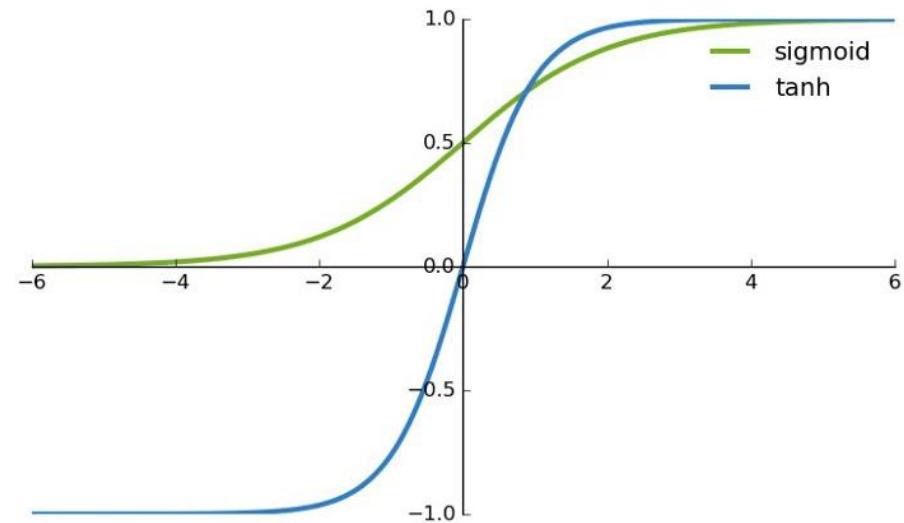


$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$



$$PReLU(x_i) = \begin{cases} x_i & \text{if } x_i > 0 \\ a_i x_i & \text{if } x_i \leq 0 \end{cases}$$

i 表示不同的通道



问题1

哪些数据线性回归不成功？



安斯库姆四重奏

Anscombe's Quartet

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.39

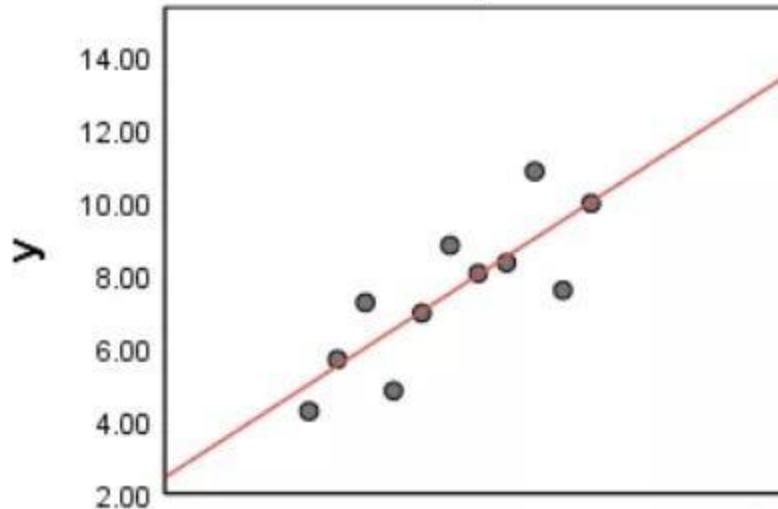
知源 @数据小兵

系数^a

group	模型		标准化系数				
			B	标准错误	Beta	t	显著性
1	1	(常量)	3.000	1.125		2.667	0.026
		x	0.500	0.118	0.816	4.241	0.002
2	1	(常量)	3.001	1.125		2.667	0.026
		x	0.500	0.118	0.816	4.239	0.002
3	1	(常量)	3.002	1.124		2.670	0.026
		x	0.500	0.118	0.816	4.239	0.002
4	1	(常量)	3.002	1.124		2.671	0.026
		x	0.500	0.118	0.817	4.243	0.002

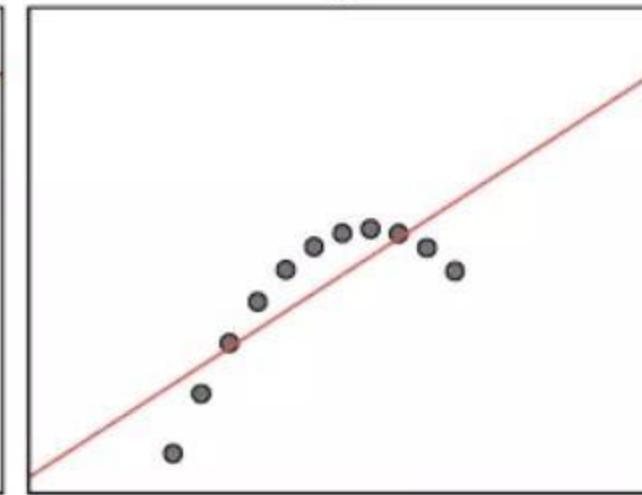
a. 因变量: y

数据分析之前，描绘数据所对应的可视化图形有多么的重要！

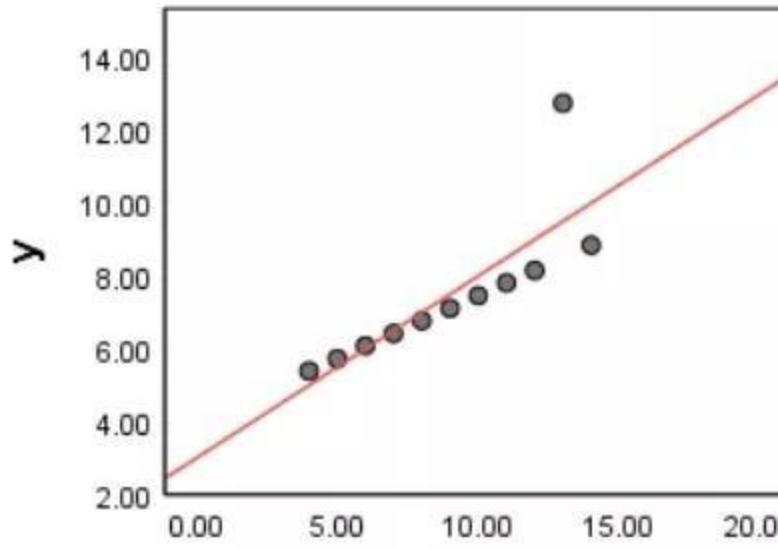


3

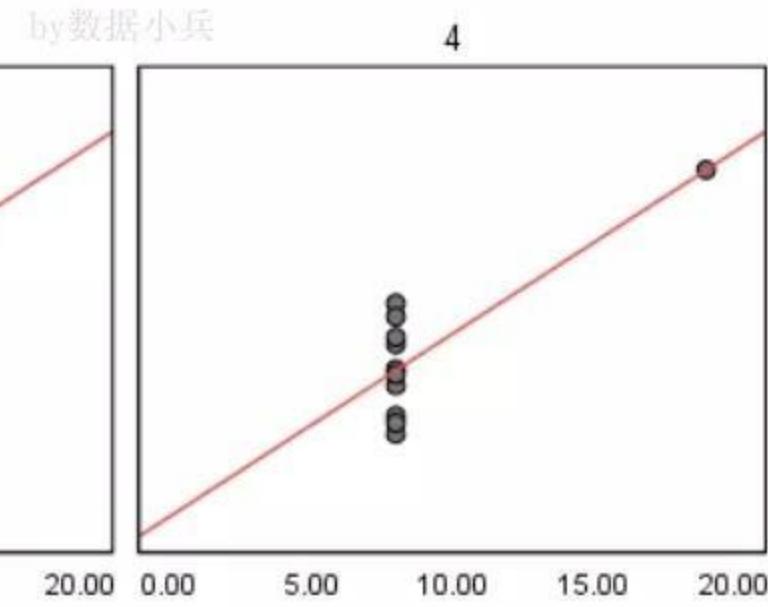
$$y = 3 + 0.5x$$



4



x



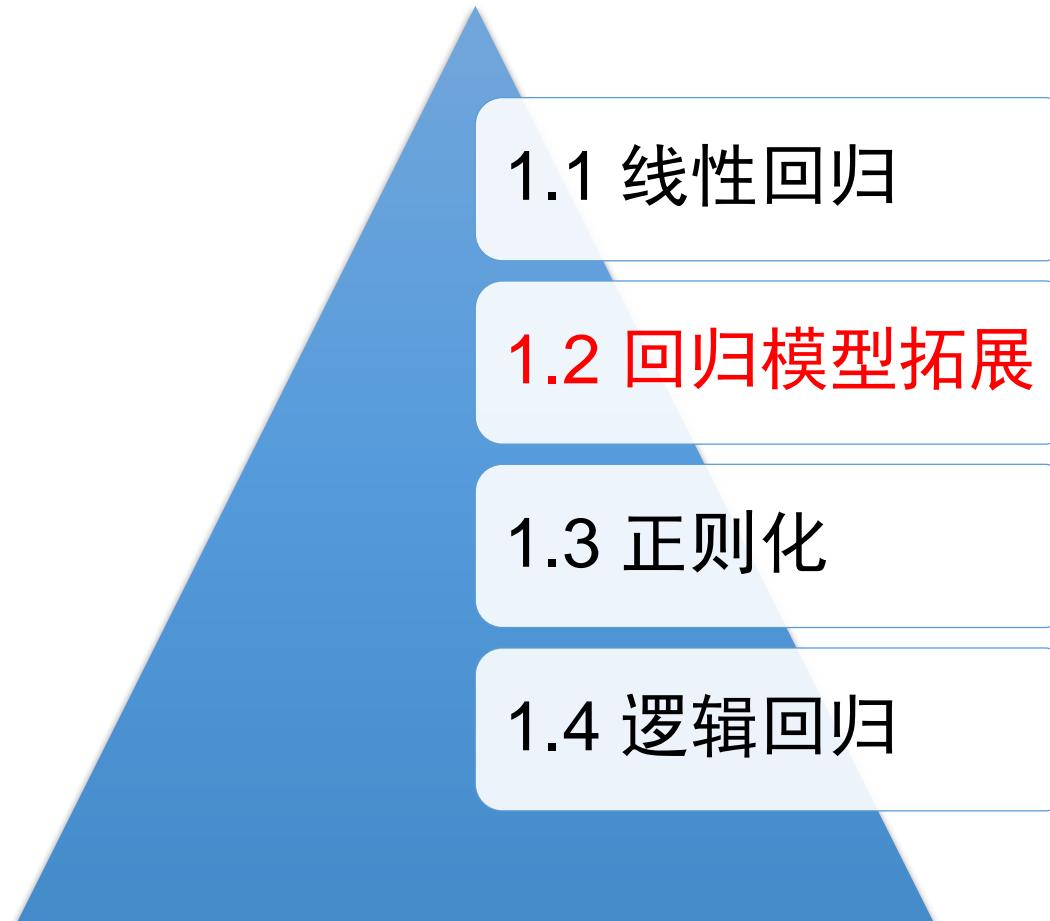
知乎 @数据小兵

1.1 线性回归

1.2 回归模型拓展

1.3 正则化

1.4 逻辑回归



问题2

如果自变量之间存在多重
共线性或者自变量个数多
于样本量应该怎么办？



多重共线性 (Multicollinearity)

- 多重共线性是一种统计现象，是指线性模型中的特征之间由于存在精确相关关系或高度相关关系，从而使得模型无法建立，或者估计失真。
- 多重共线性几乎是不可避免的，因为自变量之间总是会存在某种相关性，但多重共线性只有在自变量之间的线性关系达到一定高度时才会出现。
- 多重共线性使用指标方差膨胀因子(variance inflation factor, VIF)来进行衡量。VIF值越大，多重共线性越严重。一般认为VIF大于10时（严格是5），代表模型存在严重的共线性问题。

解决思路

- 从一组高度相关和具有多重共线性的自变量中删除某个变量，再建立回归模型，以逐步回归。
- 改变自变量的定义形式，将2个具有多重共线性的自变量合并成一个变量，或通过变量变换，改变变量的分布特征，或定义新的自变量代替具有高度多重共线性的变量。
- 通过搜集更多的观察值，增加样本容量，可避免或减少多重共线性的影响，但计算量会增大。
- 改进线性回归算法，进行回归系数的有偏估计，使其能够容忍特征列存在多重共线性的情况。原理是利用小部分回归系数估计值的偏差替代回归系数样本方差大幅度的降低。

解决思路

$$\mathbf{w} = \boxed{(\mathbf{X}^T \mathbf{X})^{-1}} \mathbf{X}^T \mathbf{y}$$



逆必须存在，则行列式不能为0

行列式不能为0，则要求矩阵必须满秩

矩阵必须满秩，则特征之间不能存在多重共线性



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

知识点回顾——监督机器学习的核心

- 正则化参数的同时最小化经验误差函数。

$$\min \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f)$$

- 最小化经验误差是为了极大的拟合训练数据，正则化参数是为了防止过分的拟合训练数据。

常用的正则化方法

- L0: L0 正则化也就是 L0 范数，即矩阵中所有非 0 元素的个数。L0 范数就是希望要正则化的参数矩阵 W 大多数元素都为 0。
- L1: L1 范数就是矩阵中各元素绝对值之和，L1 范数通常用于实现参数矩阵的稀疏性。
 - 在机器学习领域，L0 和 L1 都可以实现矩阵的稀疏性，但在实践中，L1 要比 L0 具备更好的泛化求解特性而广受青睐。
- L2: L2范数是指矩阵中各元素的平方和后的求根结果（很多模型中都采用元素的平方和）。采用L2范数进行正则化的原理在于最小化参数矩阵的每个元素，使其无限接近于0但又不像L1那样等于0。

$$L_0 \text{ 范数: } \|x\|_0 = |x_1|^0 + |x_2|^0 + \cdots + |x_n|^0$$

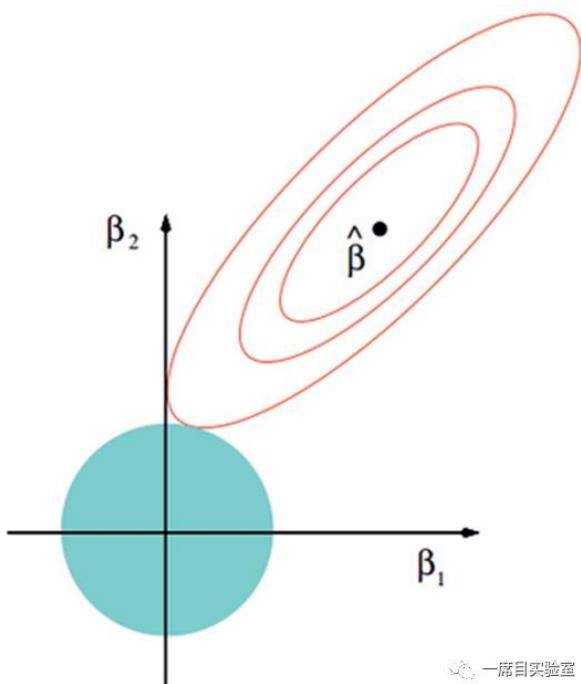
$$L_1 \text{ 范数: } \|x\|_1 = |x_1|^1 + |x_2|^1 + \cdots + |x_n|^1$$

$$L_2 \text{ 范数: } \|x\|_2 = (\|x_1\|^2 + \|x_2\|^2 + \cdots + \|x_n\|^2)^{\frac{1}{2}}$$

Ridge回归

- Ridge回归（岭回归）其本质就是在常规线性回归的损失函数后面加一个L2正则化约束。

$$L(w) = (y - wX)^2 + \lambda \|w\|_2$$



```

import numpy as np
import pandas as pd
# 导入线性模型模块
from sklearn.linear_model import Ridge
data = np.genfromtxt('example.dat', delimiter = ',')
# 选择特征与标签
x = data[:, 0:100]
y = data[:, 100].reshape(-1, 1)
# 加一列
X = np.column_stack((np.ones((x.shape[0], 1)), x))

# 划分训练集与测试集
X_train, y_train = X[:70], y[:70]
X_test, y_test = X[70:], y[70:]
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

# 创建Ridge模型实例
clf = Ridge(alpha=1.0)
# 对训练集进行拟合
clf.fit(X_train, y_train)
# 打印模型相关系数
print("sklearn Ridge intercept : ", clf.intercept_)
print("\nsklearn Ridge coefficients :\n", clf.coef_)

```

```

(70, 101) (70, 1) (31, 101) (31, 1)
sklearn Ridge intercept : [-0.40576153]

sklearn Ridge coefficients :
[[ 0.0000000e+00 -2.01786172e-01  5.45135248e-01  3.28370796e-01
   7.88208577e-02  8.63329630e-01 -1.28629181e-01  8.98548367e-01
  -4.15384520e-01  1.58905870e-01 -2.93807956e-02  6.32380717e-01
   4.21771945e-02  9.24308741e-01  1.20277300e-01 -3.85333806e-01
   1.63068579e-01  3.98963430e-01 -2.55902692e-02  8.88008417e-01
   3.69510302e-02  5.63702626e-04 -1.74758205e-01  4.51826721e-01
  -7.30107159e-02 -1.35017481e-01  5.39686001e-02 -4.02425081e-03
  -6.07507156e-02  3.75631827e-01  8.57162815e-02  1.45771573e-01
   1.44022204e-01  1.98972072e-01 -1.74729670e-01 -4.55411141e-02
   2.10931708e-01 -4.20589474e-02 -1.16955409e-01 -3.48704701e-01
   9.24987738e-02 -3.59919666e-02  3.12791851e-02  9.89341477e-02
  -3.20373964e-02  5.01884867e-04  2.52601261e-02 -1.43870413e-01
  -2.01630343e-01 -2.04659068e-02  1.39960583e-01 -2.40332862e-01
   1.64551174e-01  1.05411007e-02 -1.27446721e-01 -8.05713152e-02
   3.16799224e-01  2.97473607e-02 -3.62918779e-01 -4.33764143e-01
   1.85767035e-01  2.22954621e-01 -9.97451115e-02  3.27282961e-02
   2.41888947e-01 -2.56520012e-01 -9.21607311e-02 -1.32705556e-01
  -3.01710290e-01  3.25678251e-01  3.98328108e-01 -3.75685067e-02
   4.76284105e-01  4.66239153e-01  2.50059297e-01  3.35426970e-01
  -3.25276476e-04 -5.62721088e-02  3.05320327e-03  2.27021494e-01
   7.11869767e-02  1.96095806e-01 -4.35819139e-02 -1.69205809e-01
  -2.33710367e-02 -1.70079831e-01 -1.29346798e-01 -3.03112649e-02
   2.51270814e-01 -2.49230435e-01  6.83981071e-03 -2.30530011e-01
   4.31418878e-01  2.76385366e-01  3.30323011e-01 -7.26567151e-03
  -2.07740223e-01  2.47716612e-01  5.77447938e-02 -3.48931162e-01
   1.59732296e-01]]]

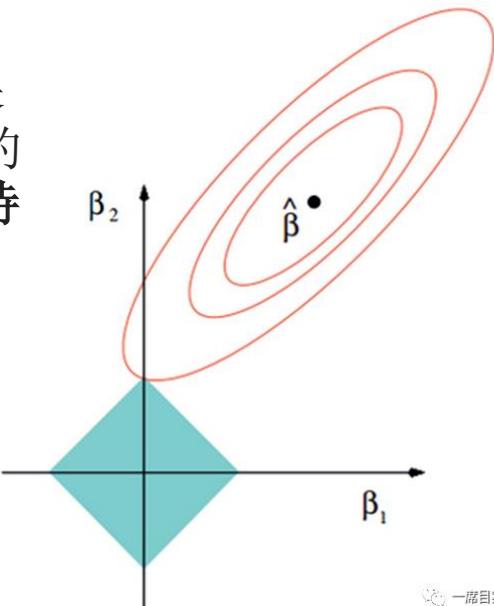
```

Lasso回归

- Lasso的全称叫做Least absolute shrinkage and selection operator，直译过来为最小收缩与选择算子。
- 其本质就是在常规线性回归的损失函数后面加一个L1正则化约束。

$$L(w) = (y - wX)^2 + \lambda \|w\|_1$$

Lasso回归是在岭回归的基础上发展起来的，当模型面临特征数量庞大的问题时，可以利用**Lasso**回归进行特征选择，[使模型便于解读](#)。



```

import numpy as np
# 导入线性模型模块
from sklearn.linear_model import Lasso
data = np.genfromtxt('example.dat', delimiter = ',')
# 选择特征与标签
x = data[:, 0:100]
y = data[:, 100].reshape(-1, 1)
# 加一列
X = np.column_stack((np.ones((x.shape[0], 1)), x))

# 划分训练集与测试集
X_train, y_train = X[:70], y[:70]
X_test, y_test = X[70:], y[70:]
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

# 创建Lasso模型实例
sk_lasso = Lasso(alpha=0.1)
# 对训练集进行拟合
sk_lasso.fit(X_train, y_train)
# 打印模型相关系数
print("sklearn Lasso intercept :", sk_lasso.intercept_)
print("\nsklearn Lasso coefficients :\n", sk_lasso.coef_)
print("\nsklearn Lasso number of iterations :", sk_lasso.n_iter_)

```

```

(70, 101) (70, 1) (31, 101) (31, 1)
sklearn Lasso intercept : [-0.23824815]

sklearn Lasso coefficients :
[ 0.          -0.          0.59804516  0.64230683  0.          1.0070841
-0.          0.81815409 -0.22756851  0.          0.          0.793601
 0.          0.74108335 -0.          -0.1250168 -0.          0.79407074
 0.          0.81867433  0.          0.          -0.          0.56664364
-0.          -0.          -0.          -0.          -0.          0.49526526
 0.          0.          0.          0.          -0.          -0.
-0.          -0.          -0.          -0.          0.          -0.
 0.          -0.          -0.0078254  0.          0.          -0.
-0.          0.01986066  0.          -0.          0.          -0.
 0.          -0.06797763  0.24581414  0.          -0.04180909 -0.
 0.10542471  0.03156005  0.          0.          0.          -0.
-0.          0.          -0.          0.12548825  0.2340209 -0.
 0.          0.16875552  0.          0.01596168  0.          -0.
 0.          0.          -0.          0.20050804 -0.          -0.
 0.          -0.04148499 -0.10729826 -0.          0.02385741 -0.10792259
-0.          -0.          0.12314032  0.          0.          -0.05876521
-0.          0.09361648 -0.          -0.17806356  0.06636851]

sklearn Lasso number of iterations : 24

```

总结

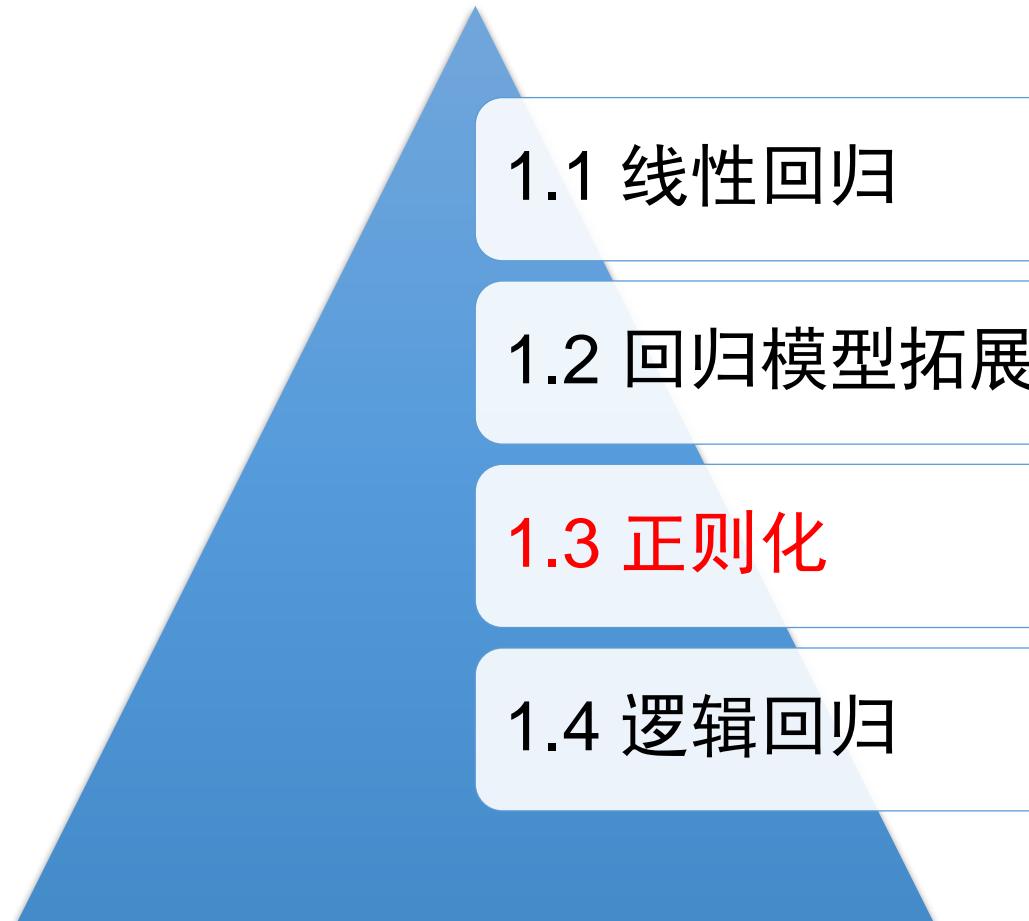
- 从数学的角度来看，Lasso和Ridge都是在 $X^T X$ 为不可逆矩阵的情况下，求解回归参数的一种妥协性方法。通过给常规的均方损失函数添加L1和L2正则化项，使得回归问题有可行解，但这种解是一种**有偏估计**。
- 从业务可解释性的角度来看，影响因变量的因素有很多，但关键因素永远只会是少数。**Lasso和Ridge回归通过对损失函数添加正则化项的方式**，使得回归建模过程中大量不重要的特征系数被压缩为0或接近0，从而找到对因变量有较强影响的关键特征。

1.1 线性回归

1.2 回归模型拓展

1.3 正则化

1.4 逻辑回归



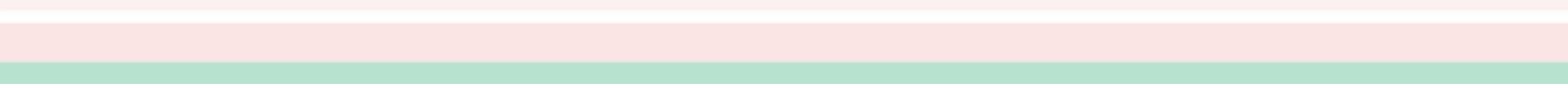
泛化误差 vs. 经验误差

泛化误差：在“未来”样本上的误差

经验误差：在训练集上的误差，亦称“训练误差”

- 泛化误差越小越好
- 经验误差是否越小越好？

NO! 因为会出现“过拟合”(overfitting)



过拟合 (overfitting) vs. 欠拟合 (underfitting)

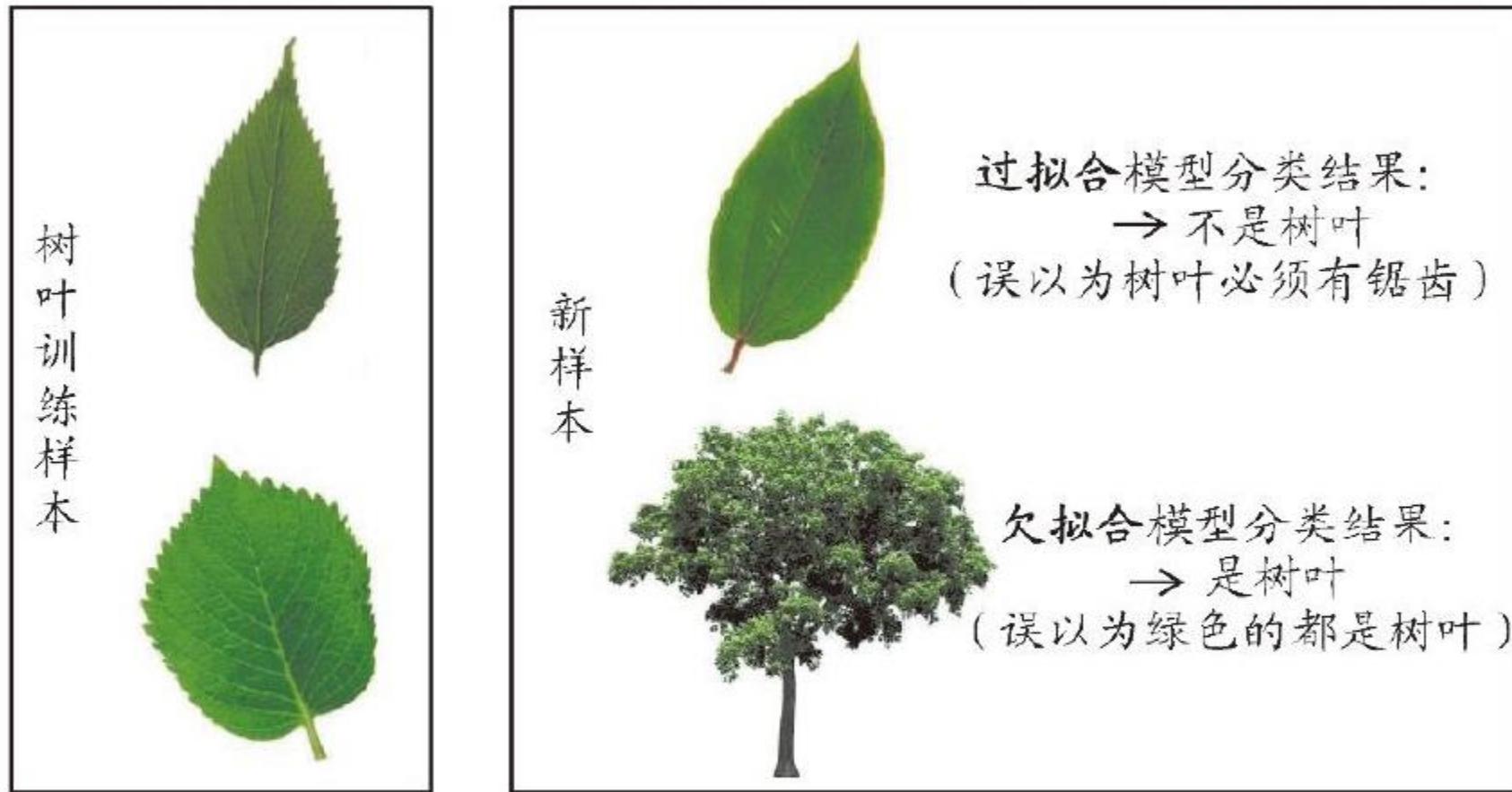
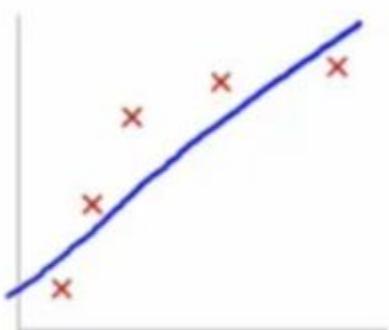
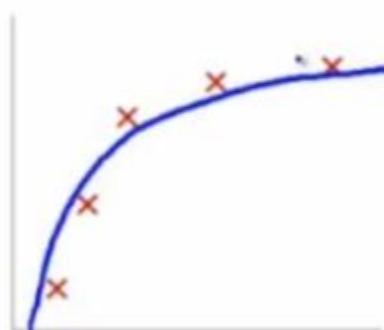


图 2.1 过拟合、欠拟合的直观类比

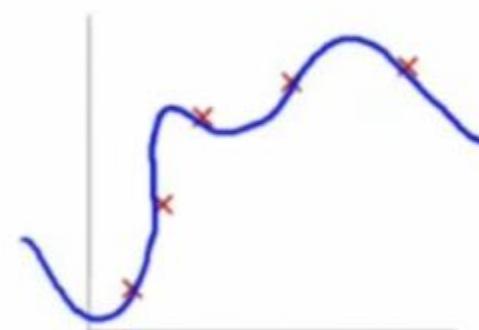
回归 (Regression) 问题中三种拟合状态



欠拟合



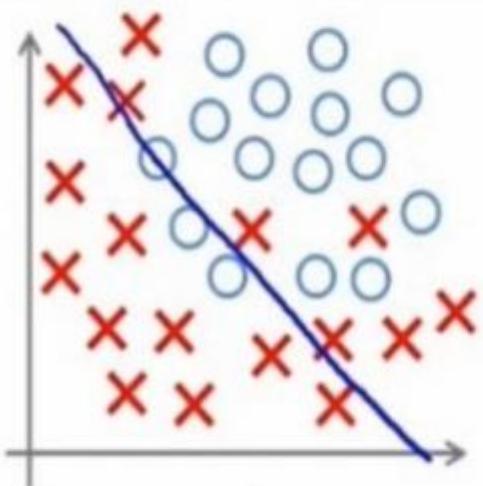
好的拟合



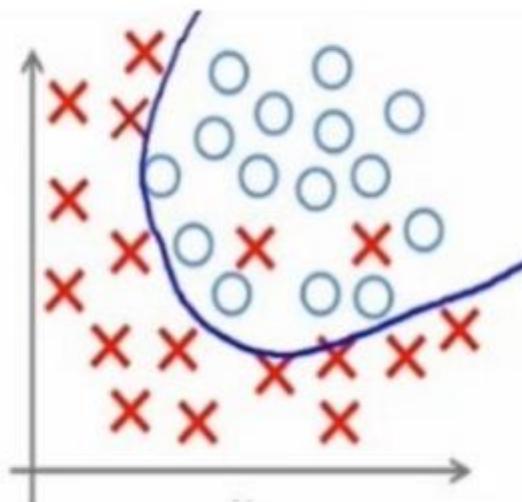
过拟合

<https://blog.csdn.net/xuaho0907>

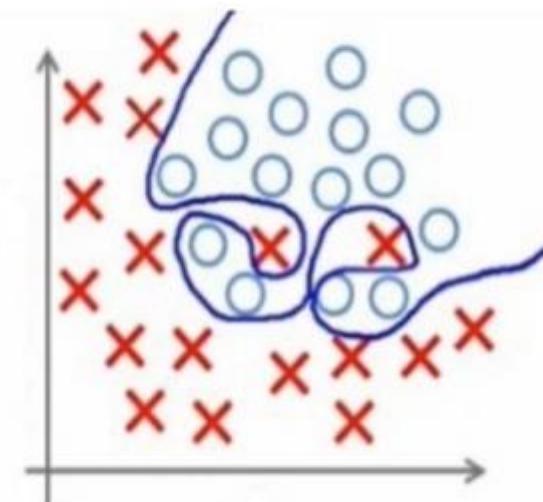
分类 (Classification) 问题中三种拟合状态



欠拟合



好的拟合



过拟合

<https://blog.csdn.net/xuaho0907>

过拟合：太过自信，夜郎自大（纸上谈兵-赵括）

欠拟合：太不自信，随机预测

好的拟合：既不要妄自尊大，也不要妄自菲薄

- 过拟合就好比死读书，平时书背得滚瓜烂熟，每次考试却不太理想！（模型太复杂，记住所有样本）
- 欠拟合就好比不读书，平时贪玩不爱学习，每次考试肯定不理想！（模型太简单。）
- 好的拟合就好比爱读书勤思考，平时喜欢举一反三，每次考试顶呱呱！

正则化

- 正则化是防止过拟合的一种方法，用于机器学习模型的训练阶段。通过向损失函数增加惩罚项的方式对模型施加制约，降低模型的复杂度，有望提高模型的泛化能力。
- 过拟合是模型在验证数据集上产生的误差比在训练数据集上产生的误差大得多的现象。
- 过拟合的一个重要原因是机器学习模型过于复杂。

正则化的形象化比喻

- 菲尔普斯成功法则：越聪明的人，越懂得“挫折演习”。



奇葩的训练方式



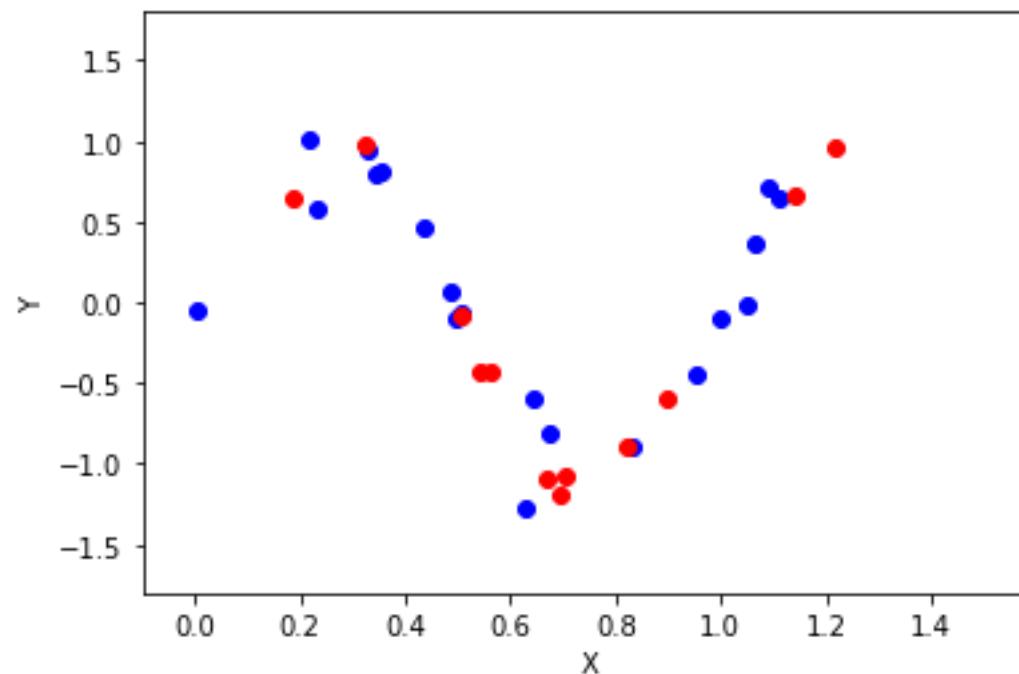
锻炼抗干扰能力



从容应对各种异常情况

多项式回归拟合 $\sin 2\pi x$

$\sin 2\pi x$



多项式回归 (无正则化)

次数	训练误差	测试误差
1	0.3898864531381092	0.7248224844711412
2	0.31903850162432246	0.3669343772338131
3	0.04725607424320766	0.3931535984865149
4	0.04021367045248463	0.1658909370304751
5	0.03234310856923899	0.023022399037513824
6	0.027393435066204952	0.382304745480778

多项式回归 (有正则化)

次数	训练误差	测试误差
1	0.6092102473744997	0.580703932776699
2	0.5688772324653796	0.5536641304301287
3	0.4510799565946028	0.4479167338100943
4	0.33232225246361835	0.32972573679077133
5	0.25012193591898596	0.24601544536925166
6	0.204398856515735	0.20302352405968585

代码实现

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

def regression_test(d, train_X, train_y, test_X, test_y):
    # d次线性回归(次数d=d)
    poly = PolynomialFeatures(d) # 次数为d
    train_poly_X = poly.fit_transform(train_X.reshape(train_size, 1))
    test_poly_X = poly.fit_transform(test_X.reshape(test_size, 1))
    model = Ridge(alpha=1.0)
    model.fit(train_poly_X, train_y)
    train_pred_y = model.predict(train_poly_X)
    test_pred_y = model.predict(test_poly_X)
    return mean_squared_error(train_pred_y, train_y), mean_squared_error(test_pred_y, test_y)

### 主程序入口 ###
if __name__ == "__main__":
    train_size = 20
    test_size = 12
    train_X = np.random.uniform(low=0, high=1.2, size=train_size)
    test_X = np.random.uniform(low=0.1, high=1.3, size=test_size)
    train_y = np.sin(train_X * 2 * np.pi) + np.random.normal(0, 0.2, train_size)
    test_y = np.sin(test_X * 2 * np.pi) + np.random.normal(0, 0.2, test_size)
    print(test_X)
    print(test_y)

    # 绘制训练数据和测试数据的散点图
    plt.scatter(train_X, train_y, color='blue')
    plt.scatter(test_X, test_y, color='red')
    plt.xticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4])
    plt.yticks([-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5])
    plt.xlim(-0.1, 1.6)
    plt.ylim(-1.8, 1.8)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()

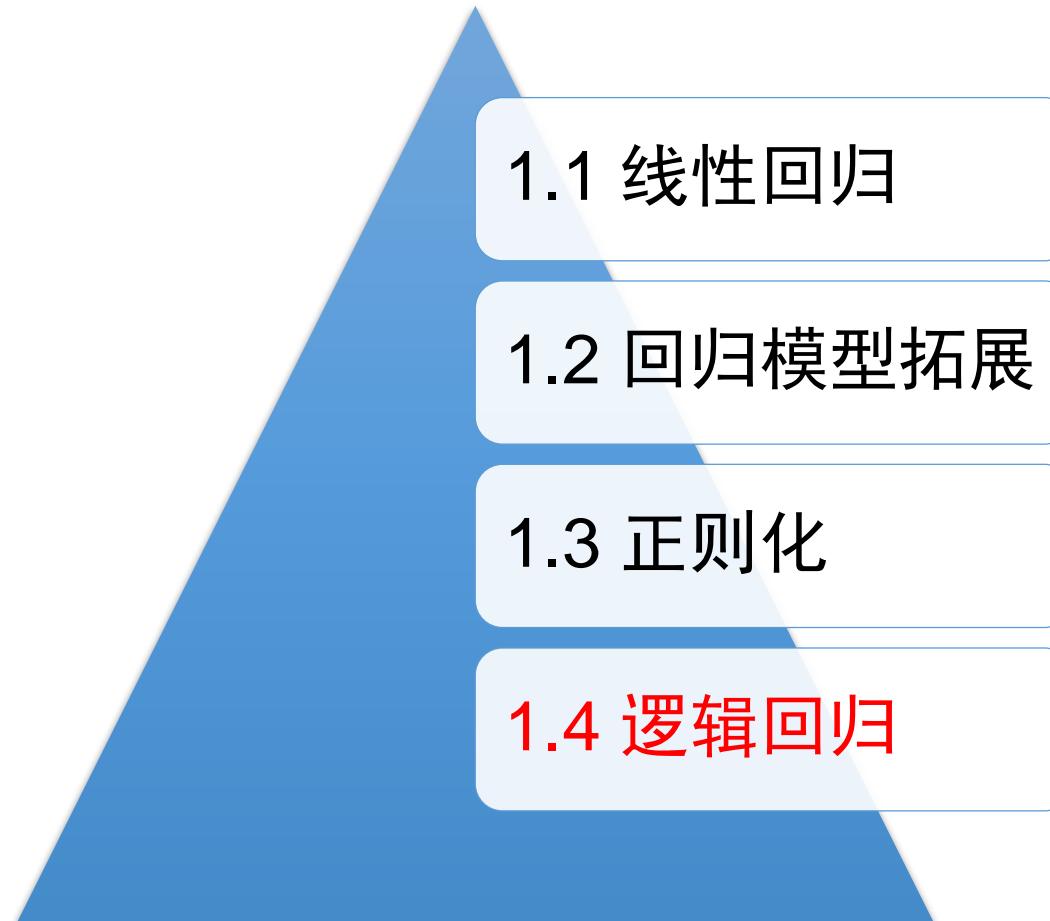
for i in range(1, 7):
    train_error, test_error = regression_test(i, train_X, train_y, test_X, test_y)
    print(str(i)+"次线性回归的训练误差和测试误差分别为: "+str(train_error)+"和"+str(test_error))
```

1.1 线性回归

1.2 回归模型拓展

1.3 正则化

1.4 逻辑回归



问题3

线性回归针对的是标签为
连续值的任务，那能否用
线性模型来做分类任务？



逻辑回归（对数几率回归）

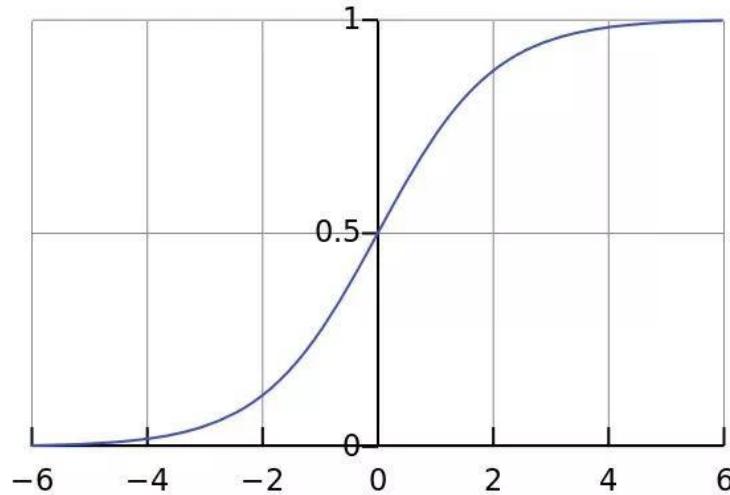
- **Logistic Regression** 直译为逻辑回归，其实本质上应该叫对数几率回归，是线性回归的一种推广，是一种线性分类模型，所以在统计学上也称之为广义线性模型。
- 逻辑回归是一种学习某个事件发生概率的算法，利用这个概率，可以对某个事件进行二元分类，也可以用于三种类别以上的分类问题。
- 与线性回归不同的是，为了计算概率，逻辑回归的输出范围必须限制在0到1之间。

逻辑回归（对数几率回归）

- 因此，逻辑回归为了执行分类任务，就需要找到一个单调可微函数，将分类任务的真实标签与线性回归模型的预测值进行映射。
- 逻辑回归使用Sigmoid函数，它单调可微，取值范围为 $(0,1)$ ，且具有较好的求导特性（其导数可以用其本身来表达）。

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$



逻辑回归模型的数学推导（二分类）

- 由 sigmoid 函数可知逻辑回归模型的基本形式为：

$$y = \frac{1}{1+e^{-W^T x + b}}$$

- 两边取对数并转换为：

$$\ln \frac{y}{1-y} = W^T x + b$$

对数几率回归

对数几率回归是用线性回归的结果来拟合真实标签的对数几率

$$\ln \frac{p(y=1|x)}{p(y=0|x)} = W^T x + b$$

$$p(y=1|x) = \frac{e^{W^T x + b}}{1+e^{W^T x + b}} = \hat{y}$$

$$p(y=0|x) = \frac{1}{1+e^{W^T x + b}} = 1 - \hat{y}$$

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

经典的二分类交叉熵损失函数

$$L = -\ln p(y|x) = -\frac{1}{n} \sum_{i=1}^n (\mathbf{y} \ln \hat{\mathbf{y}} + (1-\mathbf{y}) \ln (1-\hat{\mathbf{y}}))$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W} = -\frac{1}{n} \sum_{i=1}^n x(\hat{y} - y)$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = -\frac{1}{n} \sum_{i=1}^n (\hat{y} - y)$$

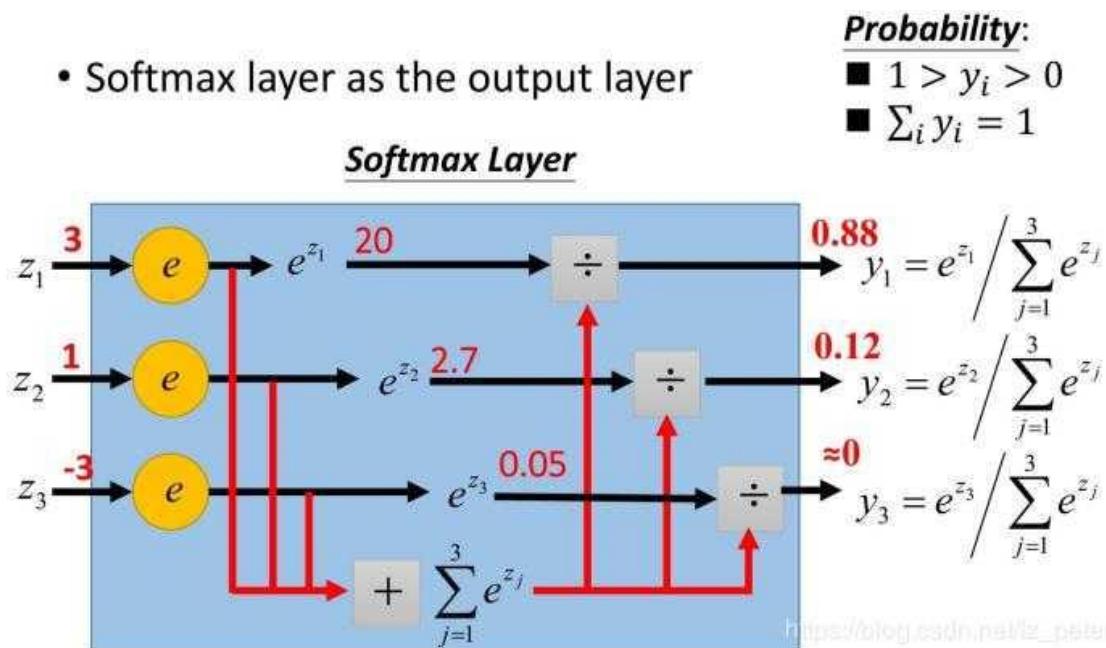
多分类问题的交叉熵损失函数

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

$$x_i = (x_{i1}, x_{i2}, \dots, x_{id})$$

$$\boldsymbol{c} \in (\boldsymbol{c}_1, \boldsymbol{c}_2, \dots, \boldsymbol{c}_m)$$

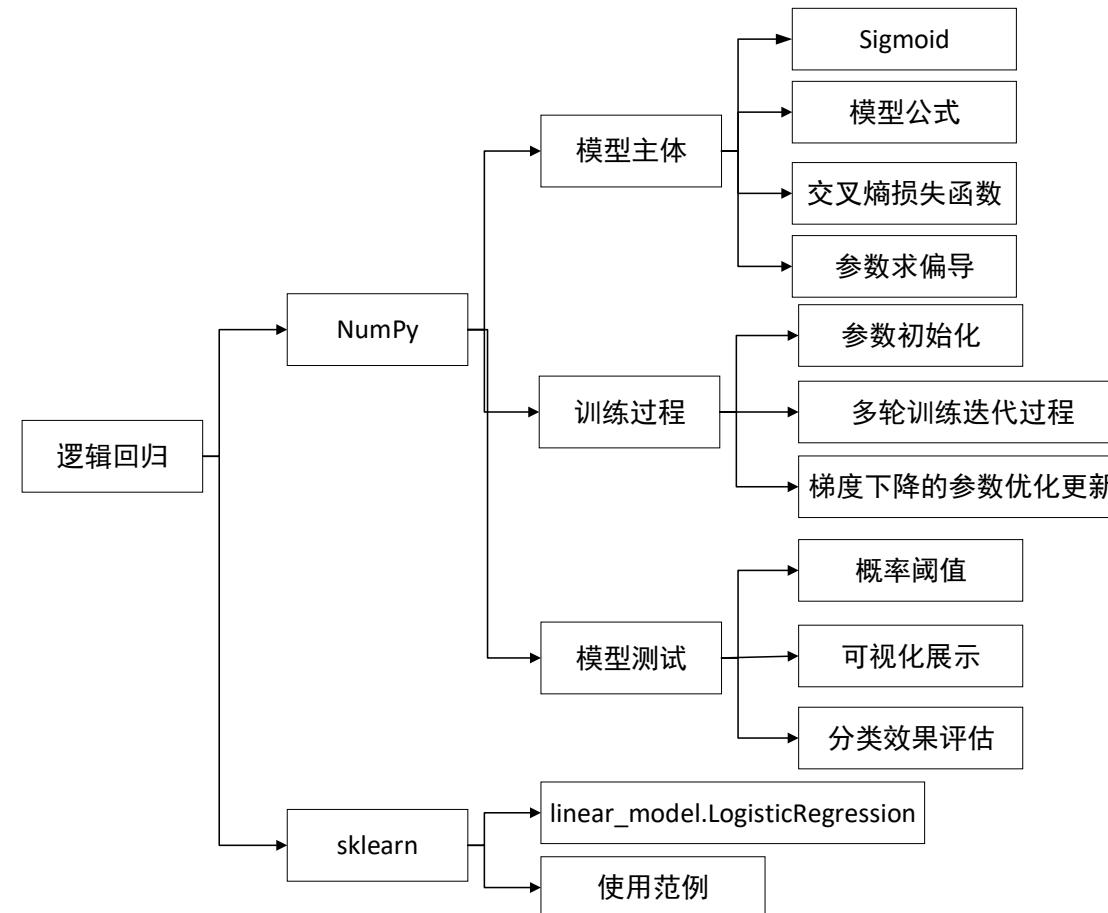
- 交叉熵损失函数满足两个特性： $c \in (c_1, c_2, \dots)$
 - (1) 非负性；
 - (2) 输出接近目标值时，代价函数接近于0。
 - 二分类问题中的激活函数是sigmoid()，多分类问题中的激活函数是softmax()。



多分类问题的交叉熵损失函数

$$L = -\ln p(y|x) = -\frac{1}{n} \sum_{i=1}^n \sum_{c=1}^m y_c \ln \hat{y}_c$$

使用numpy实现一个逻辑回归模型



```
def sigmoid(x):
    z = 1 / (1 + np.exp(-x))
    return z
```

```
def initialize_params(dims):
    W = np.zeros((dims, 1))
    b = 0
    return W, b
```

```
### 定义逻辑回归模型主体
def logistic(X, y, W, b):
```

输入:
X: 输入特征矩阵
y: 输出标签向量
W: 权值参数
b: 偏置参数
输出:
a: 逻辑回归模型输出
cost: 损失
dW: 权值梯度
db: 偏置梯度
,,,

```
# 训练样本量
num_train = X.shape[0]
# 训练特征数
num_feature = X.shape[1]
# 逻辑回归模型输出
a = sigmoid(np.dot(X, W) + b)
# 交叉熵损失
cost = -1/num_train * np.sum(y*np.log(a) + (1-y)*np.log(1-a))
# 权值梯度
dW = np.dot(X.T, (a-y))/num_train
# 偏置梯度
db = np.sum(a-y)/num_train
# 压缩损失数组维度
cost = np.squeeze(cost)
return a, cost, dW, db
```

```
### 定义逻辑回归模型训练过程
def logistic_train(X, y, learning_rate, epochs):
    """
    输入:
    X: 输入特征矩阵
    y: 输出标签向量
    learning_rate: 学习率
    epochs: 训练轮数
    输出:
    cost_list: 损失列表
    params: 模型参数
    grads: 参数梯度
    """

    # 初始化模型参数
    W, b = initialize_params(X.shape[1])
    # 初始化损失列表
    cost_list = []

    # 迭代训练
    for i in range(epochs):
        # 计算当前次的模型计算结果、损失和参数梯度
        a, cost, dW, db = logistic(X, y, W, b)
        # 参数更新
        W = W - learning_rate * dW
        b = b - learning_rate * db
        # 记录损失
        if i % 100 == 0:
            cost_list.append(cost)
        # 打印训练过程中的损失
        if i % 100 == 0:
            print('epoch %d cost %f' % (i, cost))

    # 保存参数
    params = {
        'W': W,
        'b': b
    }

    # 保存梯度
    grads = {
        'dW': dW,
        'db': db
    }

    return cost_list, params, grads
```

```

### 定义预测函数
def predict(X, params):
    """
    输入:
    X: 输入特征矩阵
    params: 训练好的模型参数
    输出:
    y_prediction: 转换后的模型预测值
    """

    # 模型预测值
    y_prediction = sigmoid(np.dot(X, params['W']) + params['b'])
    # 基于分类阈值对概率预测值进行类别转换
    for i in range(len(y_prediction)):
        if y_prediction[i] > 0.5:
            y_prediction[i] = 1
        else:
            y_prediction[i] = 0

    return y_prediction

```

```

def accuracy(y_test, y_pred):
    correct_count = 0
    for i in range(len(y_test)):
        for j in range(len(y_pred)):
            if y_test[i] == y_pred[j] and i == j:
                correct_count += 1

    accuracy_score = correct_count / len(y_test)
    return accuracy_score

```

```

# 导入matplotlib绘图库
import matplotlib.pyplot as plt
# 导入生成分类数据函数
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score, classification_report
# 生成100*2的模拟二分类数据集
X, labels = make_classification(
    n_samples=100,
    n_features=2,
    n_redundant=0,
    n_informative=2,
    random_state=1,
    n_clusters_per_class=2)
# 设置随机数种子
rng = np.random.RandomState(2)
# 对生成的特征数据添加一组均匀分布噪声
X += 2 * rng.uniform(size=X.shape)
# 标签类别数
unique_labels = set(labels)
# 根据标签类别数设置颜色
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
# 绘制模拟数据的散点图
for k, col in zip(unique_labels, colors):
    x_k = X[labels == k]
    plt.plot(x_k[:, 0], x_k[:, 1], 'o', markerfacecolor=col, markeredgecolor="k",
             markersize=14)
plt.title('Simulated binary data set')
plt.show()

labels = labels.reshape((-1, 1))
data = np.concatenate((X, labels), axis=1)
print(data.shape)

# 训练集与测试集的简单划分
offset = int(X.shape[0] * 0.9)
X_train, y_train = X[:offset], labels[:offset]
X_test, y_test = X[offset:], labels[offset:]
y_train = y_train.reshape((-1, 1))
y_test = y_test.reshape((-1, 1))

cost_list, params, grads = logistic_train(X_train, y_train, 0.01, 1000)
y_pred = predict(X_test, params)
print(classification_report(y_test, y_pred))
accuracy_score_train = accuracy(y_train, y_train_pred)
print(accuracy_score_train)
accuracy_score_test = accuracy(y_test, y_prediction)
print(accuracy_score_test)

```

第10章 回归——良性肿瘤二分类

```
# 导入pandas与numpy工具包。
import pandas as pd
import numpy as np
# 使用sklearn.model_selection里的train_test_split模块用于分割数据。
from sklearn.model_selection import train_test_split
# 从sklearn.preprocessing里导入StandardScaler。
from sklearn.preprocessing import StandardScaler
# 从sklearn.linear_model里导入LogisticRegression
from sklearn.linear_model import LogisticRegression
# 从sklearn.metrics里导入classification_report模块。 分类报告
from sklearn.metrics import classification_report

def create_dataset():
    # 创建特征列表。
    column_names= ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size',
                   'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size',
                   'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class']

    # 使用pandas.read_csv函数从互联网读取指定数据。
    data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                       names = column_names)
    # 将?替换为标准缺失值表示。
    data = data.replace(to_replace='?', value=np.nan)
    # 丢弃带有缺失值的数据（只要有一个维度有缺失）。
    data = data.dropna(how='any')

    # 随机采样25%的数据用于测试，剩下的75%用于构建训练集合。
    X_train, X_test, y_train, y_test = train_test_split(
        data[column_names[1:10]], #特征：第0列id去掉, 第10列分类结果去掉
        data[column_names[10]], #第10列的分类结果作为目标分类变量
        test_size=0.25,
        random_state=33)

    return X_train, X_test, y_train, y_test
```

```

if __name__ == "__main__":
    X_train, X_test, y_train, y_test = create_dataset()
    # 标准化数据，保证每个维度的特征数据方差为1，均值为0。使得预测结果不会被某些维度过大的特征值而主导。
    ss = StandardScaler()
    X_train = ss.fit_transform(X_train)
    X_test = ss.transform(X_test)

    # 初始化LogisticRegression
    # 逻辑回归
    lr = LogisticRegression()

    # 调用LogisticRegression中的fit函数/模块用来训练模型参数。
    lr.fit(X_train, y_train)
    # 使用训练好的模型lr对X_test进行预测，结果储存在变量lr_y_predict中。
    lr_y_predict = lr.predict(X_test)
    # 使用逻辑斯蒂回归模型自带的评分函数score获得模型在测试集上的准确性结果。准确性：预测和真实值比较，预测正确的百分比，Accuracy。
    print('Accuracy of LR Classifier:', lr.score(X_test, y_test))
    # 利用classification_report模块获得LogisticRegression其他三个指标的结果。
    print(classification_report(y_test, lr_y_predict, target_names=['Benign', 'Malignant']))

```

Accuracy of LR Classifier: 0.9883040935672515

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

Benign	0.99	0.99	0.99	100
Malignant	0.99	0.99	0.99	71

accuracy			0.99	171
macro avg	0.99	0.99	0.99	171
weighted avg	0.99	0.99	0.99	171

A.I.



大数据，成就未来



Thank you!
感谢您！