暨南大學
JINAN UNIVERSITY

机 器 学 习

第八章：回归分析

黄斐然

2022/4/11

目录

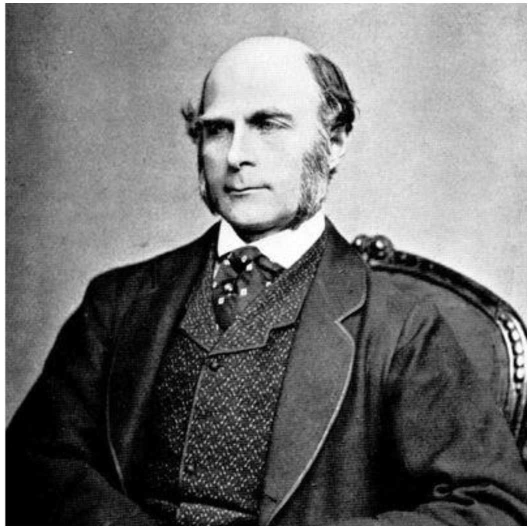
房价预测

装修	面积	销售价格（万元）
0	123	250
20	150	320
17	87	160
8	102	220
15	**112**	**???**

1 回归简介

回归的起源：

- “回归”是由英国著名生物学家兼统计学家高尔顿(Francis Galton,1822～1911.生物学家达尔文的表弟)在研究人类遗传问题时提出来的。为了研究父代与子代身高的关系，高尔顿搜集了1078对父亲及其儿子的身高数据。他发现这些数据的散点图大致呈直线状态，也就是说，总的趋势是父亲的身高增加时，儿子的身高也倾向于增加。



1 回归简介

■ 回归的起源：

- 高尔顿对试验数据进行了深入的分析，发现了一个很有趣的现象—回归效应。因为当父亲高于平均身高时，他们的儿子身高比他更高的概率要小于比他更矮的概率；父亲矮于平均身高时，他们的儿子身高比他更矮的概率要小于比他更高的概率。 它反映了一个规律，即儿子的身高，有向他们父辈的平均身高回归的趋势。
- 1855年，高尔顿将上述结果发表在论文《遗传的身高向平均数方向的回归》中，这就是统计学上“回归”定义的第一次出现。

1 回归简介

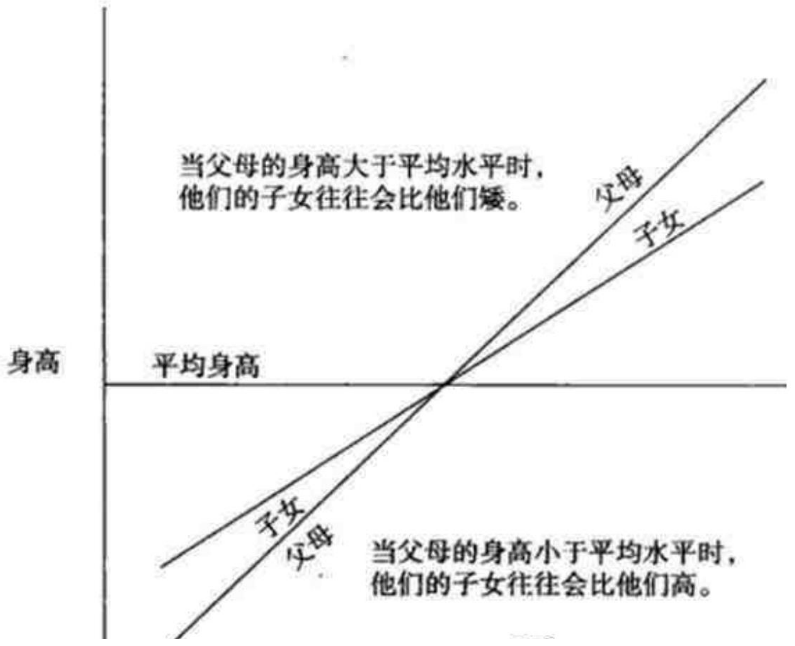
回归的起源：

- 可以拟合成一条直线：Y= 0.8567+0.516*X (单位为米);
- 这个拟合关系表明通过父亲的身高可以预测儿子（成年）的身高。

  假如父亲的身高为1.70米，则预测儿子的身高为1.734米。

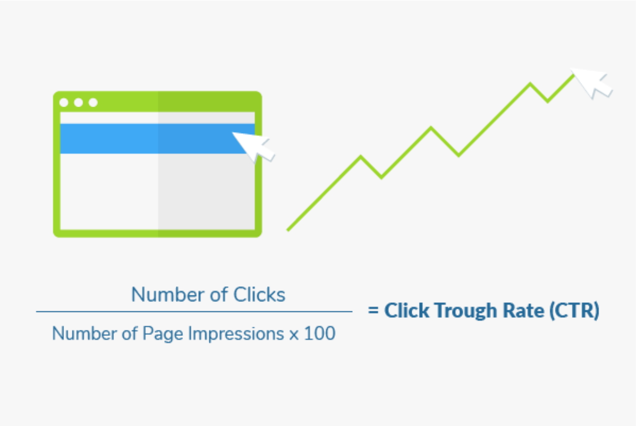
  假如父亲的身高为1.75米，则预测儿子的身高为1.760米。

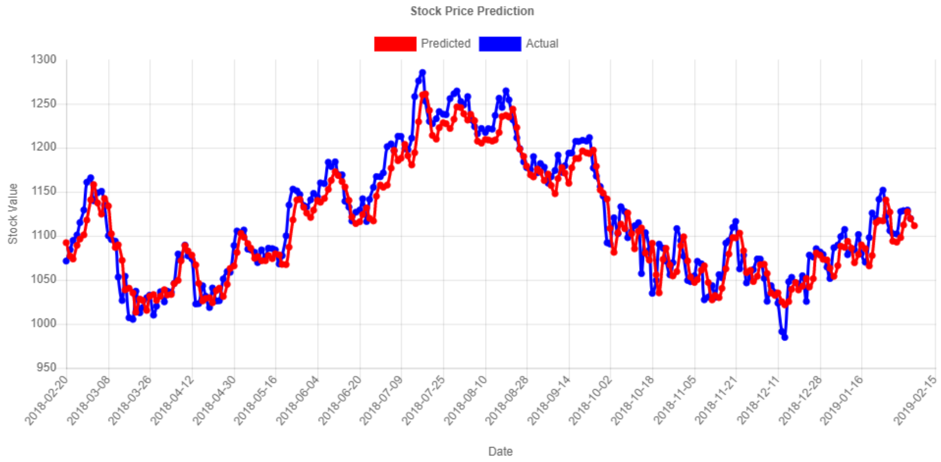
  假如父亲的身高为1.80米，则预测儿子的身高为1.786米

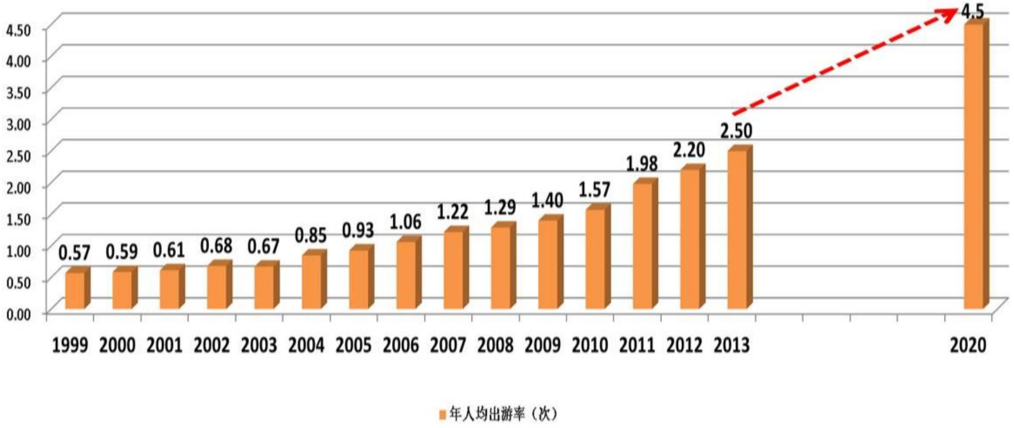


1 回归简介

■ 回归分析的应用：





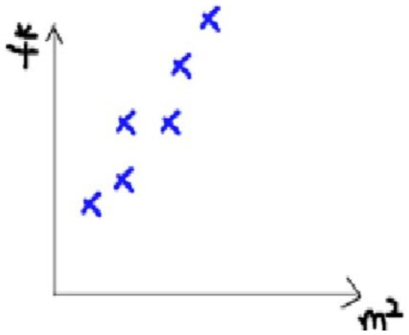


1 回归简介

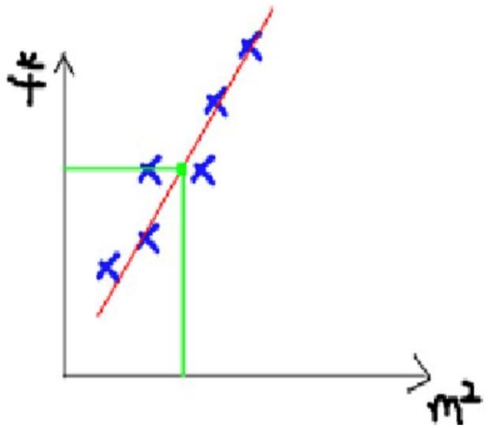
简单的回归例子：

- 假设有一个房屋销售的数据如下

面积	销售价格（万元）
123	250
150	320
87	160
102	220



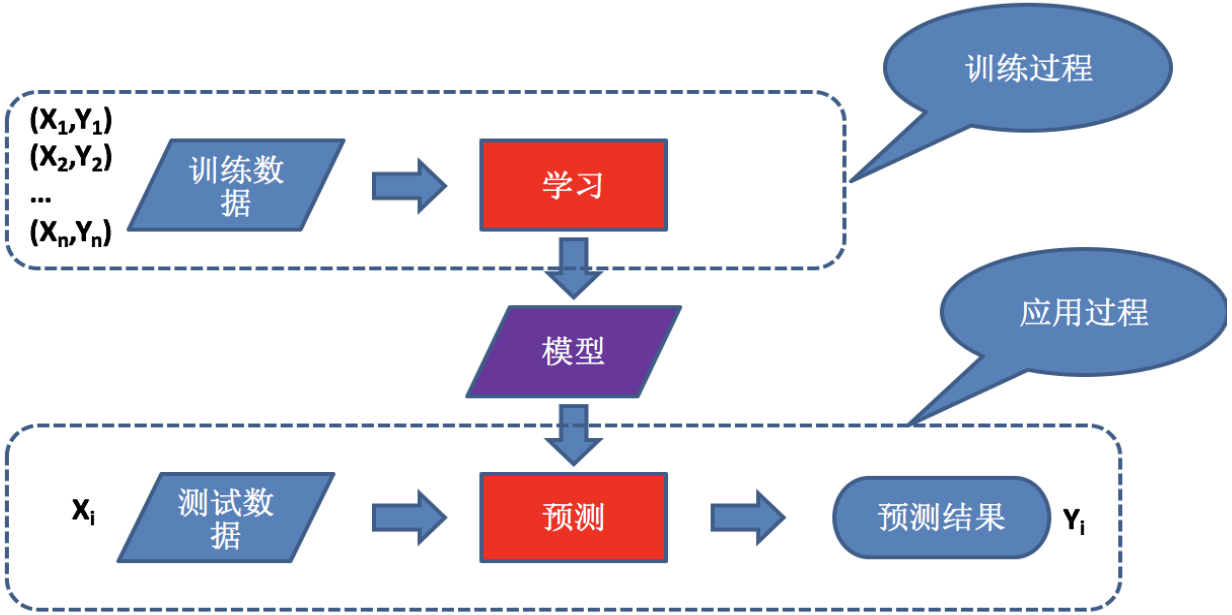
- 如果来了一个新的面积，假设在销售价钱的记录中没有的，怎么处理？
- 解决方法：用一条曲线去尽量准的拟合这些数据，如果有新的输入过来，我们可以在讲曲线上这个点对应的值返回。



1 回归简介

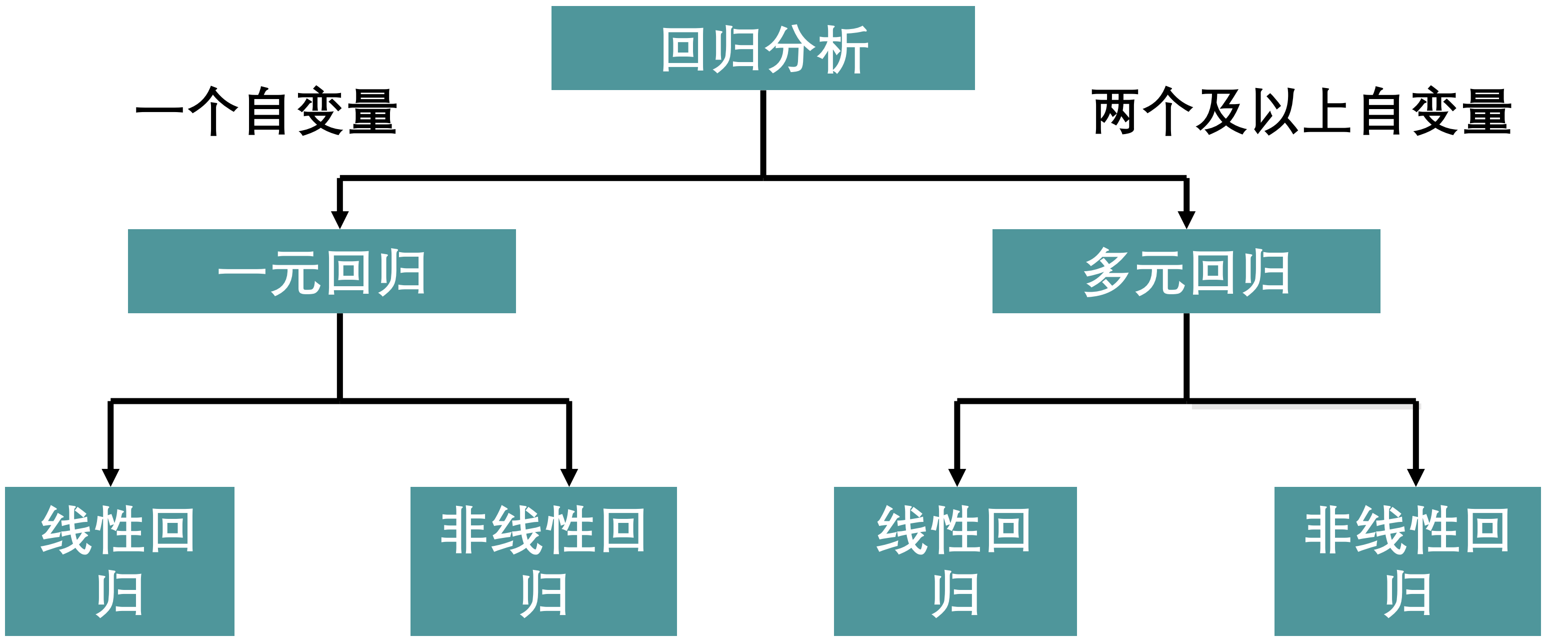
回归分析：

- 回归分析研究的主要是因变量（目标）和自变量（经验）之间的依存关系。按关系类型，又可分为线性回归分析和非线性回归分析。学习过程如下：



1 回归简介

■ 回归分析的分类：



目录

2 线性回归

■ 线性回归：

- 寻找X和Y之间的关系

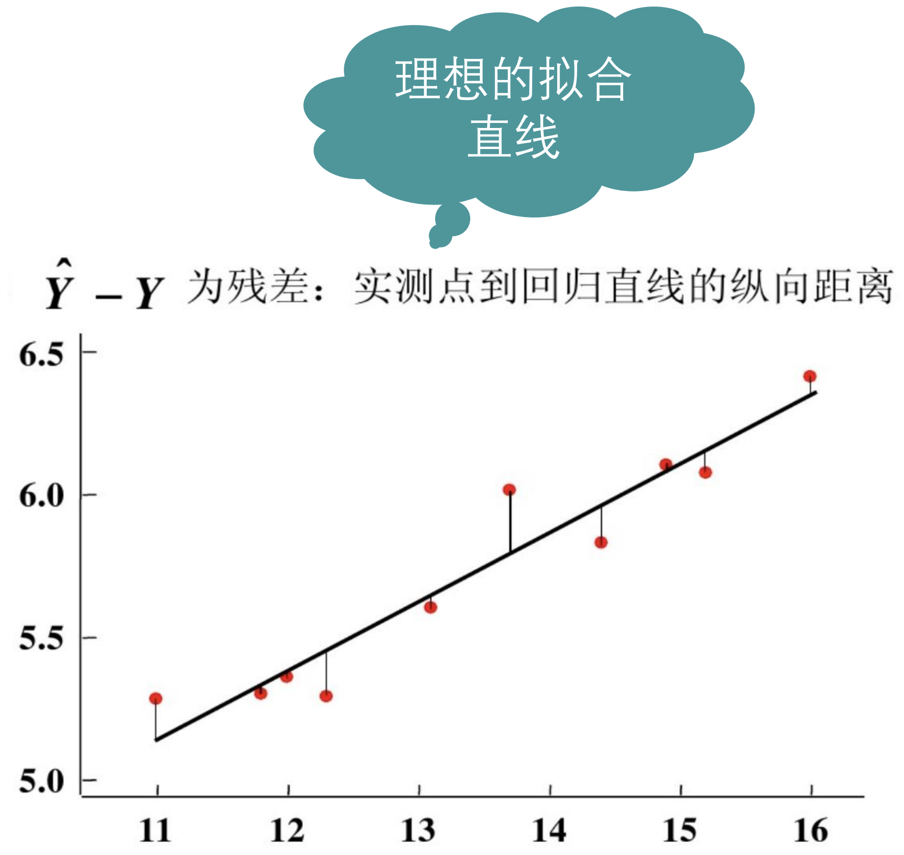
$$Y = \theta_0 + \theta_1 * x$$

- 如何确定参数 θ_0， θ_1呢？

常采用的策略是误差平方和最小化准则，即：

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{Y}^{(i)} - Y^{(i)})^2$$

$$min_\theta \ J_\theta$$



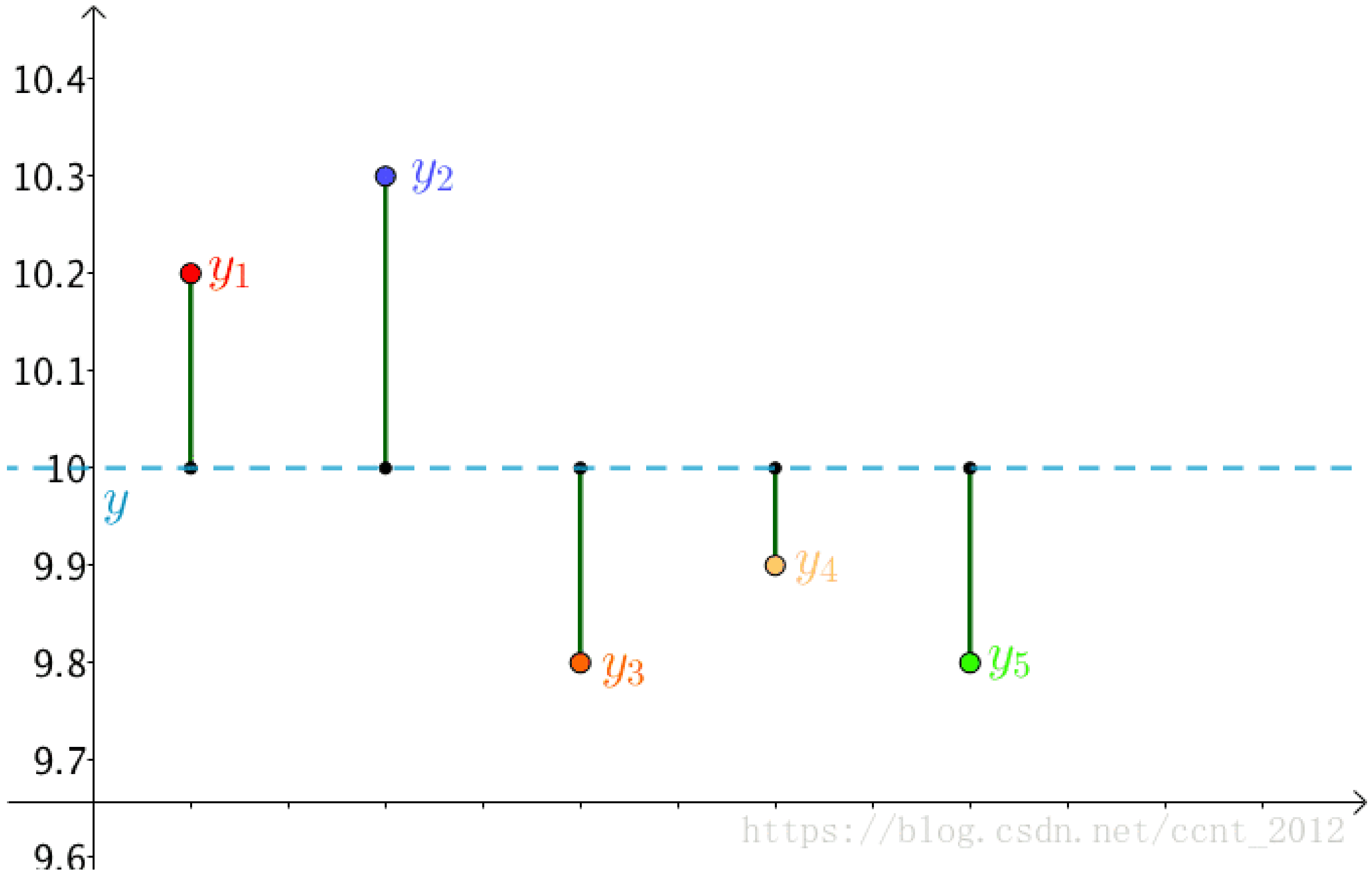
2 线性回归

■ 线性回归:

最小二乘:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\widehat{Y}^{(i)} - Y^{(i)})^2$$

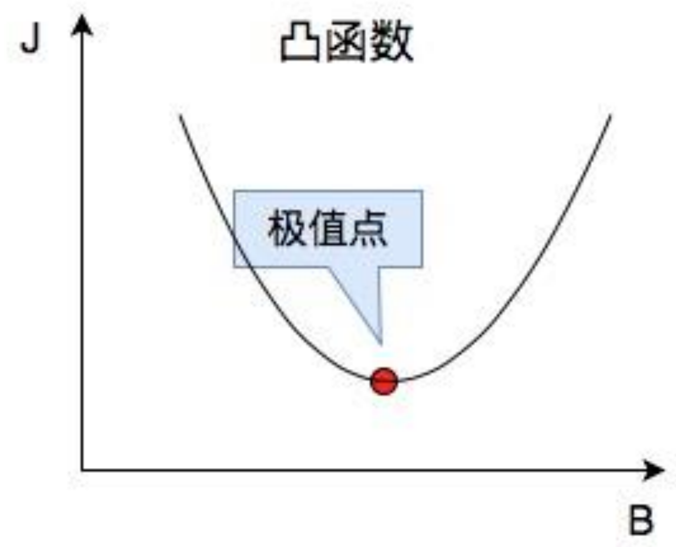
$$min_\theta \ J_\theta$$



2 线性回归

最小二乘算法：

- 现在问题就转化为求 J_θ 的最小值问题。
- 具体的做法是：
  - 1）对目标函数求导
  - 2）零其导数为0，求得极值
- 如果函数是凸函数，极值点就是最值点。这即是著名方法—最小二乘的基本思想。



2 线性回归

两个未知数，两个方程，可以解出 θ_0，θ_1

■ 最小二乘代数法：

$$J(\mathrm{B}) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1^{(i)} - \mathrm{y}^{(i)})^2$$

$$J'_{\theta_0} = \frac{1}{m}\sum_{i=1}^{\mathrm{m}}(\theta_0 + \theta_1 x_1^{(i)} - \mathrm{y}^{(i)}) = 0$$

$$\theta_0 + \theta_1\bar{x} - \bar{y} = 0$$

$$\Rightarrow \theta_0 = \bar{y} - \theta_1\bar{x}$$

$$J'_{\theta_1} = \frac{1}{m}\sum_{i=1}^{\mathrm{m}} x_1^{(i)}(\theta_0 + \theta_1 x_1^{(i)} - \mathrm{y}^{(i)}) = 0$$

$$\Rightarrow \frac{1}{m}\sum_{i=1}^{\mathrm{m}} x_1^{(i)}(\bar{y} - \mathrm{y}^{(i)} + \theta_1 x_1^{(i)} - \theta_1\bar{x}) = 0$$

$$\Rightarrow \bar{x}\bar{y} - \overline{xy} + \theta_1\overline{x^2} - \theta_1\bar{x}^2 = 0$$

$$\Rightarrow \theta_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}$$

2 线性回归

■ 最小二乘矩阵法:

$$J(\mathrm{B}) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 =$$

$$\frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_k x_k^{(i)} - \mathrm{y}^{(i)})^2$$

$$= \frac{1}{2m}(\boldsymbol{X\theta} - \boldsymbol{Y})^T(\boldsymbol{X\theta} - \boldsymbol{Y})$$

需要在左侧加一列1才能转为$\boldsymbol{X}$

$$J_\theta' = \boldsymbol{X}^T(\boldsymbol{X\theta} - \boldsymbol{Y}) = 0$$

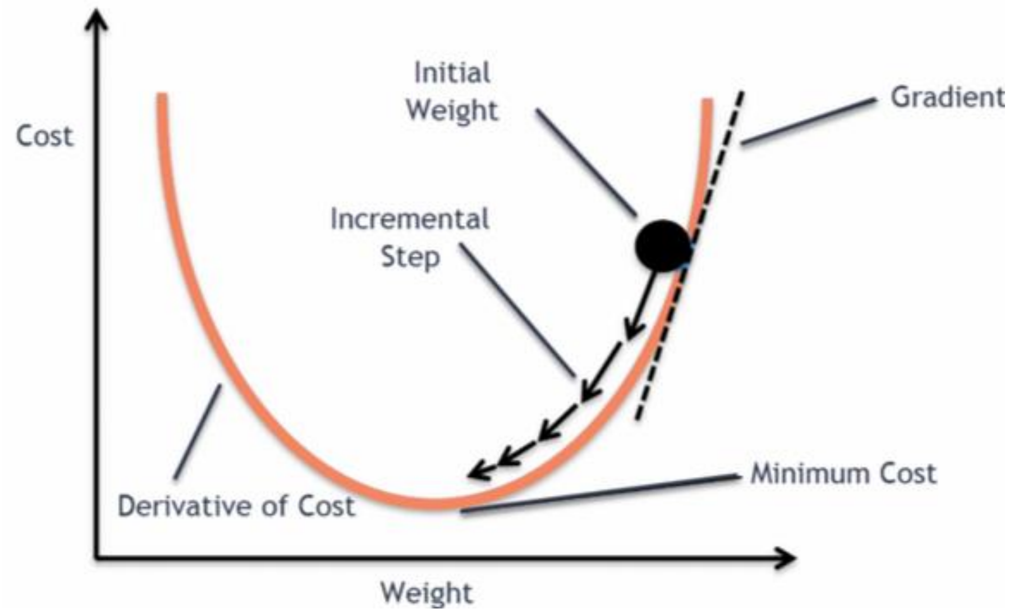
$$\hat{\theta} = (\mathrm{X}^T\mathrm{X})^{-1}X^TY$$

最小二乘法矩阵法缺陷：

- 首先，最小二乘法需要计算X^TX的逆矩阵，有可能它的逆矩阵不存在，这样就没有办法直接用最小二乘法了，此时梯度下降法仍然可以使用。
- 第二，当样本特征n非常的大的时候，计算X^TX的逆矩阵是一个非常耗时的工作，甚至不可行。此时以梯度下降为代表的迭代法仍然可以使用。如果超过10000个特征建议用迭代法吧。或者通过主成分分析降维后再用最小二乘法。
- 第三，如果拟合函数不是线性的，这时无法使用最小二乘法，需要通过一些技巧转化为线性才能使用，此时梯度下降仍然可以用。

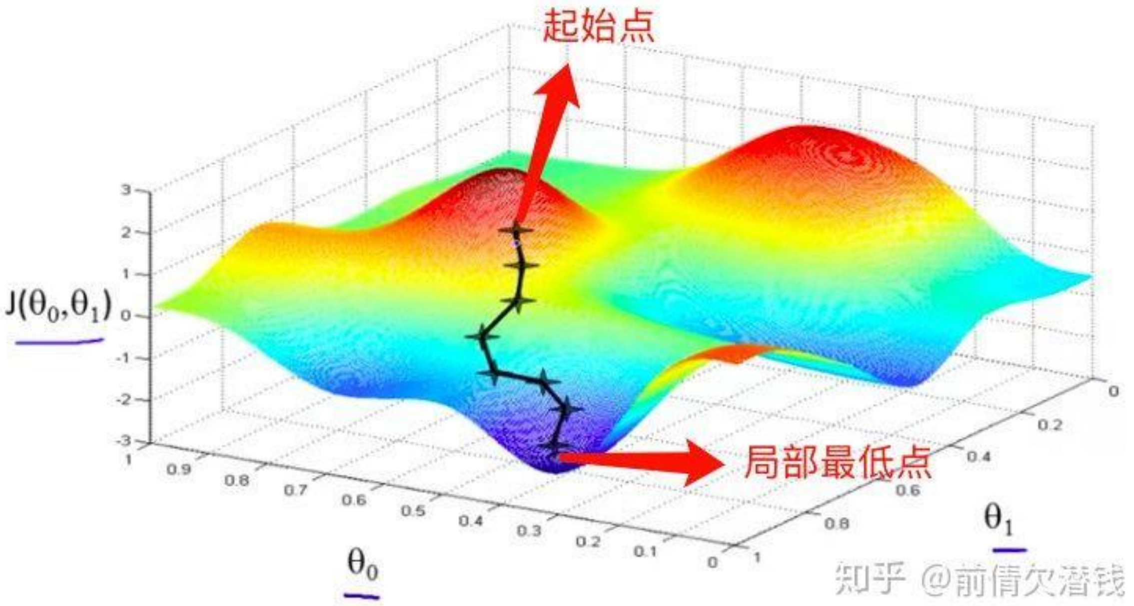
梯度：

- 梯度是一个向量，对于一个多元函数 f(x,y) 而言，在点 P(x,y) 的梯度是在这个点增大最快的方向，即以 f 在 P 上的偏导数为分量的向量。以二元函数 f(x,y) 为例，把 f(x,y) 的梯度记作 gradf(x,y)或者 $\nabla f(x,y)$ 。



2 线性回归

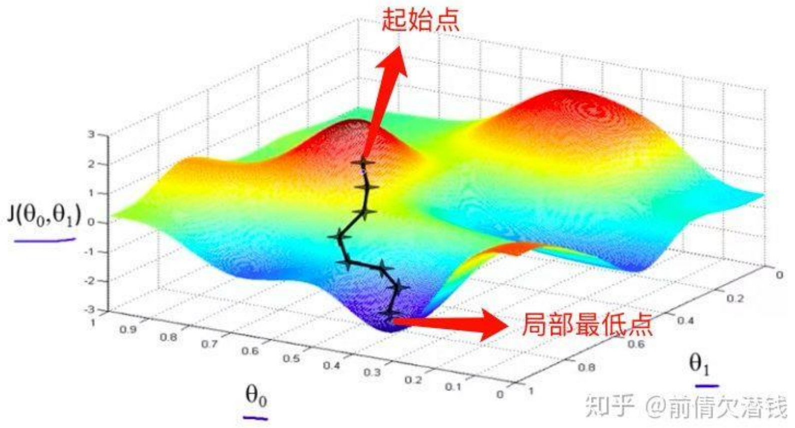
- 假设现在有一个人在山上，怎么下山?
  - 1、一定要沿着山高度下降的地方走，不然就不是下山而是上山了。
  - 2、最陡峭的方向，即山高度下降最快的方向。现在确定了方向，就要开始下山了。
  - 3、最开始选定的方向并不总是高度下降最快的地方。每走一段距离之后，重新确定当前所在位置的高度下降最快的地方。

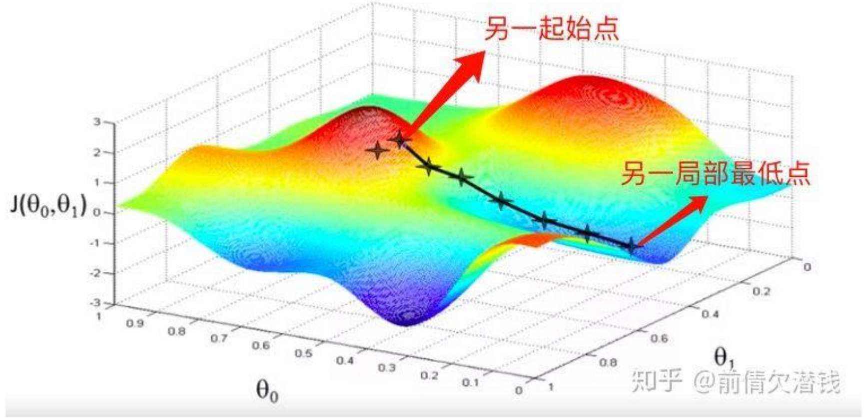


2 线性回归

- 假设现在有一个人在山上，怎么下山?

  - 在线性回归中，我们要找到参数 $\hat{\theta}$ 使得损失函数 $J(\mathrm{B}) = \frac{1}{2}(\boldsymbol{X\theta} - \boldsymbol{Y})^T(\boldsymbol{X\theta} - \boldsymbol{Y})$ 最小。如果把损失函数 看作是这座山，山底就是损失函数最小的地方，求解参数 $\hat{\theta}$ 的过程，就是人走到山底的过程。





梯度下降法：

$$J(B)=\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2$$

$$=\frac{1}{2m}\sum_{i=1}^{m}(\theta_0+\theta_1x_1^{(i)}-y^{(i)})^2$$

$$=\frac{1}{2m}(\boldsymbol{X\theta}-\boldsymbol{Y})^T(\boldsymbol{X\theta}-\boldsymbol{Y})$$

注意：这里需要给X水平拼接全1列，才能转换

梯度：

$$\nabla J_\theta'=\boldsymbol{X}^T(\boldsymbol{X\theta}-\boldsymbol{Y})$$

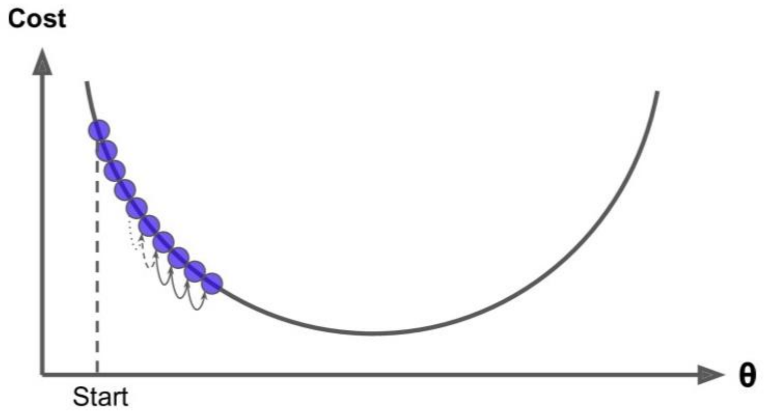
梯度下降：

$$\hat{\theta}:=\hat{\theta}-\alpha\cdot X^T(X\theta-Y)$$

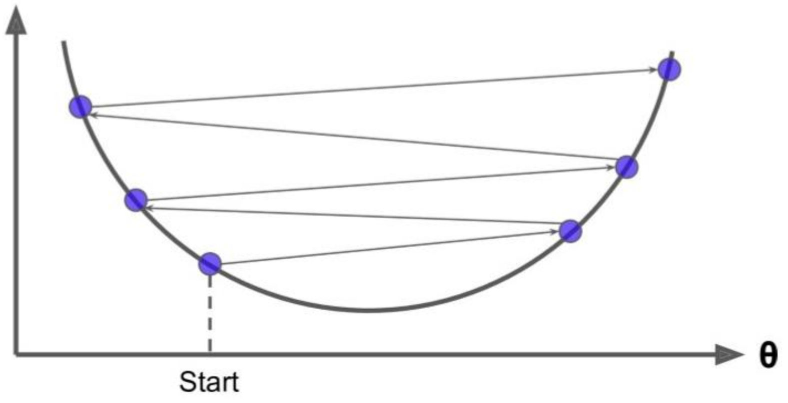
2 线性回归

■ 学习率的选择：

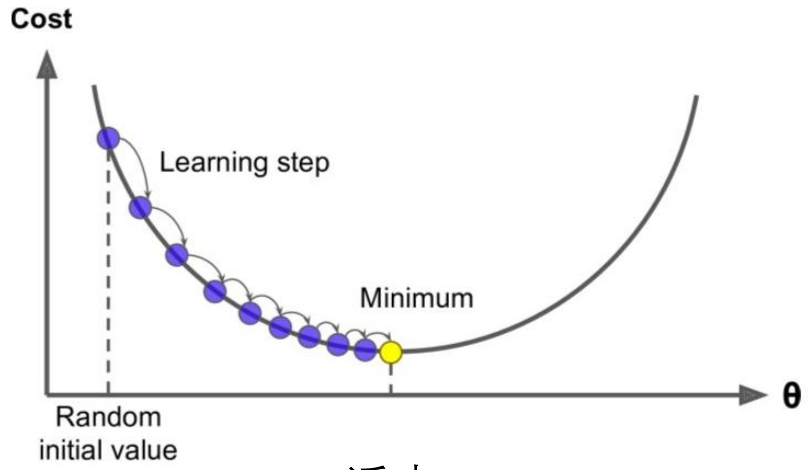
- 梯度下降中一个重要参数是每一步学习率。如果学习率太低，算法需要经过大量迭代才能收敛，这将耗费很长时间。如果学习率太高，可能会越过山谷直接到达山的另一边，甚至有可能比之前的起点还要高。这会导致算法发散，无法收敛。



太小



太大



适中

2 线性回归

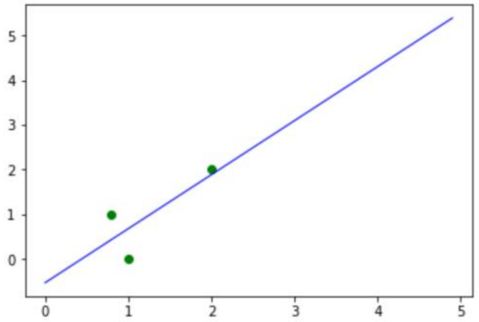
使用sklearn实现：

- from sklearn.linear_model import LinearRegression
- LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None)
- 重要参数：
  - fit_intercept： default True。是否计算截距，默认为计算。
  - normalize： 该参数在 fit_intercept 设置为False时自动忽略，表示是否在回归前对数据进行正则化，如果要对数据进行标准化，请使用sklearn.preprocessing.StandardScaler。
  - n_jobs： 使用线程数。

2 线性回归

■ 使用sklearn实现：

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

X_train = np.array([[1], [0.8], [2]])
y_train = np.array([0, 1, 2])
reg = LinearRegression(normalize=True).fit(X_train, y_train)

X_test = np.arange(0,5,0.1).reshape(-1,1)
y_pred = reg.predict(X_test)

plt.scatter(X_train, y_train,  color='g')
plt.plot(X_test, y_pred, color='b', linewidth=1)
plt.show()

print(reg.coef_)
print(reg.intercept_)
print('The learned regression model: y = %.2fx + %.2f' % (reg.coef_,reg.intercept_))

print('Mean squared error for training: ',mean_squared_error(y_train,reg.predict(X_train)))

```




```
[1.20967742]
-0.5322580645161297
The learned regression model: y = 1.21x + -0.53
Mean squared error for training:  0.2634408602150537
```

目录

3 特征选择

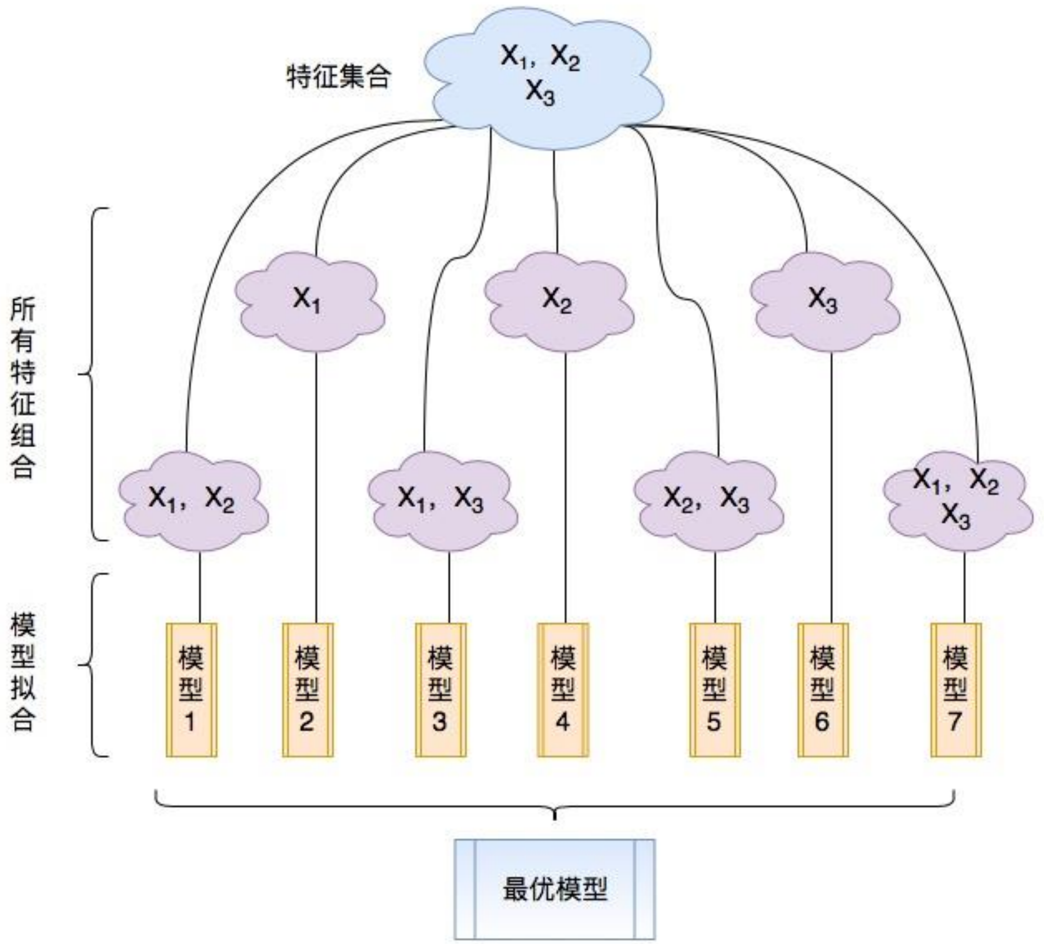
选择“最优回归方程”：

- 在多元线性回归中，并不是所用特征越多越好；选择少量、合适的特征既可以避免过拟合，也可以增加模型解释度。
- 既要选择对预测影响显著的自变量，又要使回归的损失很小，这样才有利于预测。
- 选择“最优回归方程”的方法有：
  - 最优子选择法（best subset selection）
  - 逐步选择法（stepwise selection）

3 特征选择

最优子选择法：

- 最优子集选择法（best subset selection），即对n个预测变量的所有可能组合（共有2n-1）分别进行拟合，然后选择出最优模型。



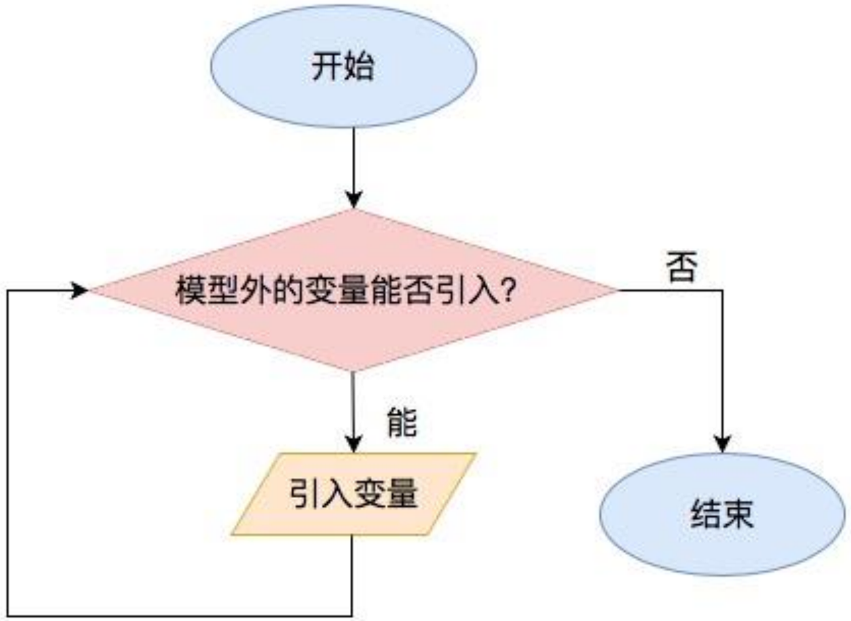
3 特征选择

逐步选择法：

- 逐步选择法按选择方式的不同，共分为三种：
  - 前向逐步选择法（Forward Stepwise Selection）
  - 后向逐步选择法（Backward Stepwise Selection）
  - 逐步回归法（Stepwise Regression）
- 基于最优子集回归方法的一些缺陷，逐步选择的优点是限制了搜索空间，从而提高了运算效率。

3 特征选择

- 前向逐步选择法：
  - 以零模型为起点，依次往模型中添加变量，直至加完所有的变量。
  - 但每次优先将能够最大限度地提升模型效果的变量加入模型。
  - 但无法保证找到的模型是所有2n-1个模型中最优的，且可能在前期将后来变得多余的变量纳入模型。

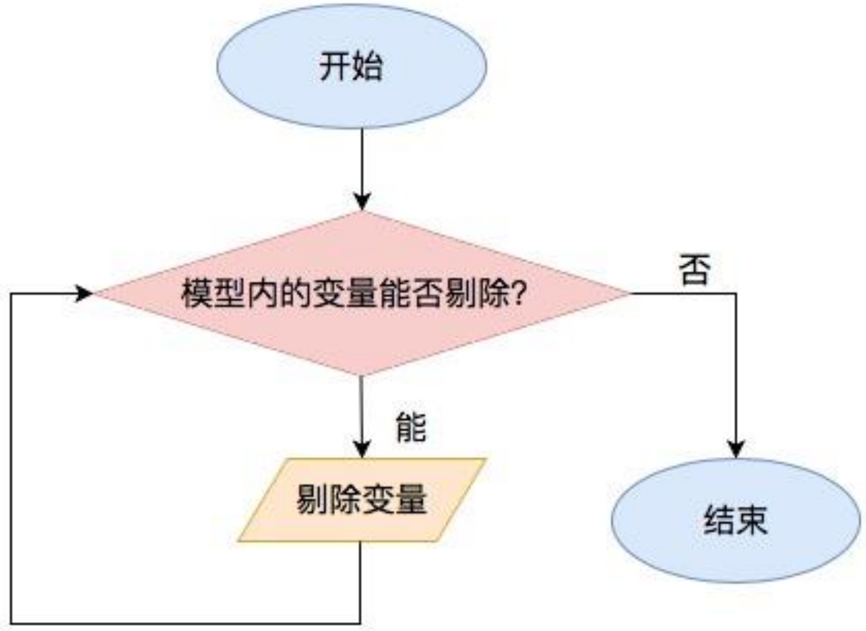


模型个数：[n(n+1)/2]+1

3 特征选择

■ 后向逐步选择法：

- 以全模型为起点，逐次迭代，每次移除一个对模型拟合结果最不利的变量。
- 需满足样本量m大于变量个数n（保证全模型被拟合）。而前向逐步选择即时在m<n的情况下也可以使用，适应于高维数据。

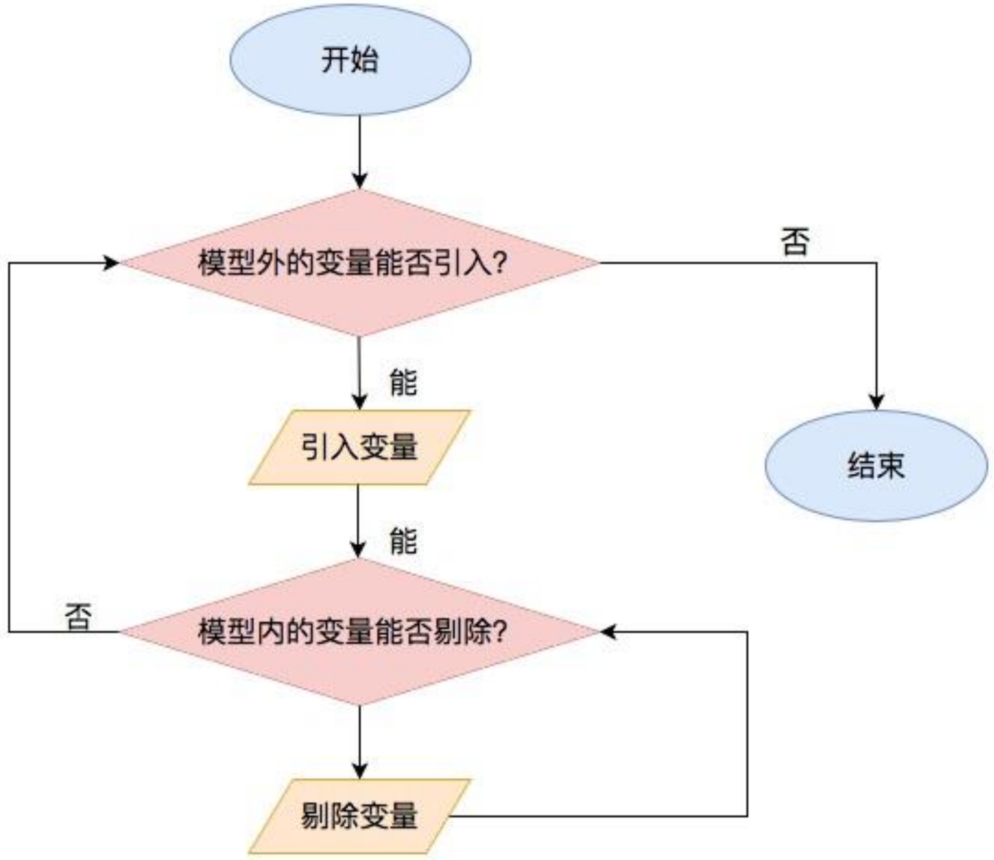


模型个数：[n(n+1)/2]+1

3 特征选择

■ 逐步回归法：

- 该方法将前向选择与后向选择进行了结合，试图达到最优子集选择效果的同时也保留了前向和后向逐步选择在计算上的优势。

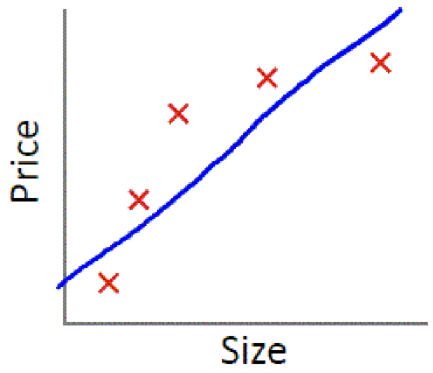


目录

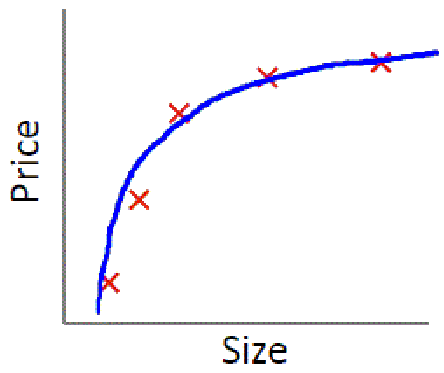
4 范数

■ 过拟合与欠拟合：

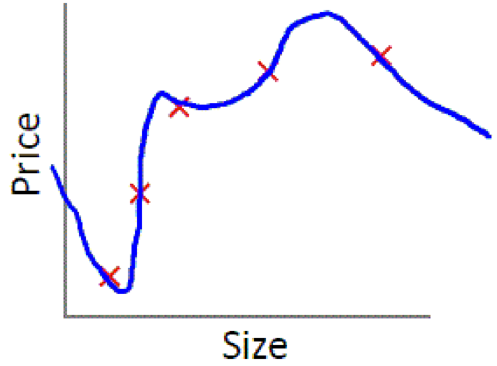
- 模型学的“太好”，把样本自身的一些特点当作所有潜在样本都会具有的一般性质，泛化性能很差，称为“过拟合”。正则化可以很好的解决这一问题。
- 与“过拟合”相对的是“欠拟合”，这是指对训练样本的一般性质尚未学好。



$\theta_0 + \theta_1 x$



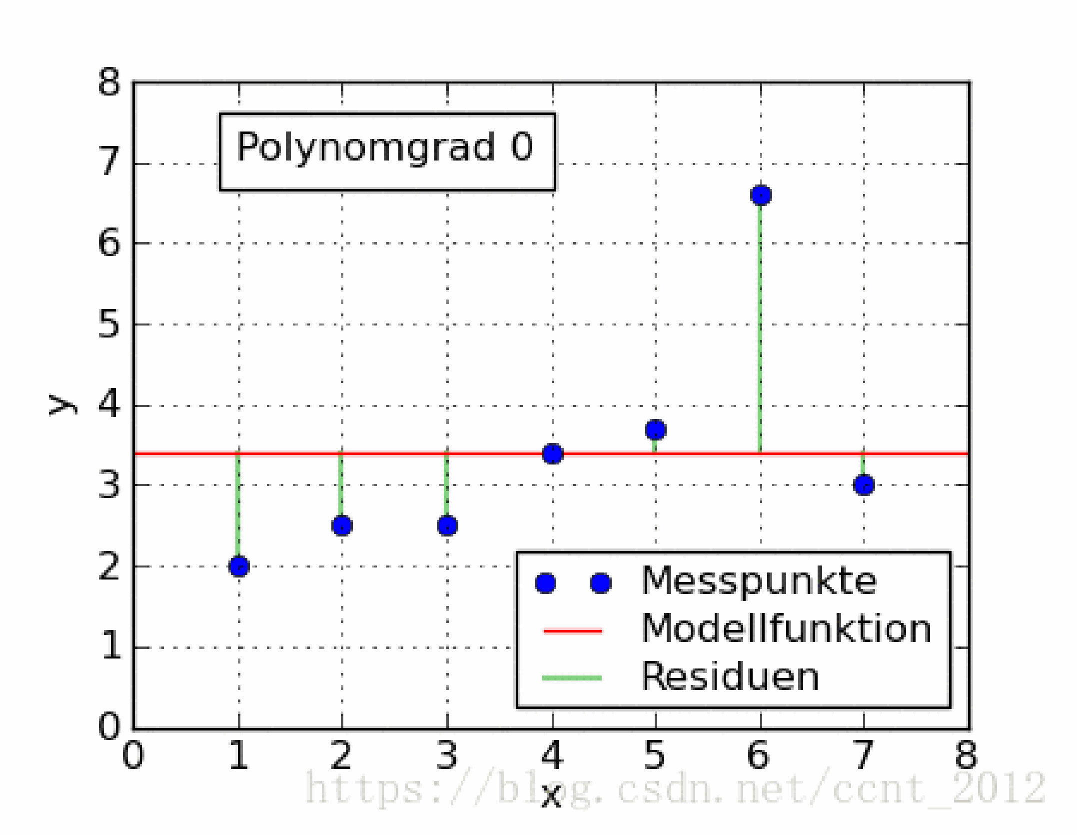
$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

4 范数

■ 过拟合与欠拟合：



Polynomgrad 0
Messpunkte
Modellfunktion
Residuen
y
x
0
1
2
3
4
5
6
7
8
https://blog.csdn.net/ccnt_2012

4 范数

L1范数与L2范数：

- 范数：范数是衡量某个向量空间（或矩阵）中的每个向量的长度或大小。范数的一般化定义如下（实数p>=1）：

$$\|x\|_p := \left(\sum_{i=0}^{n} |x_i|^p\right)^{\frac{1}{p}}$$

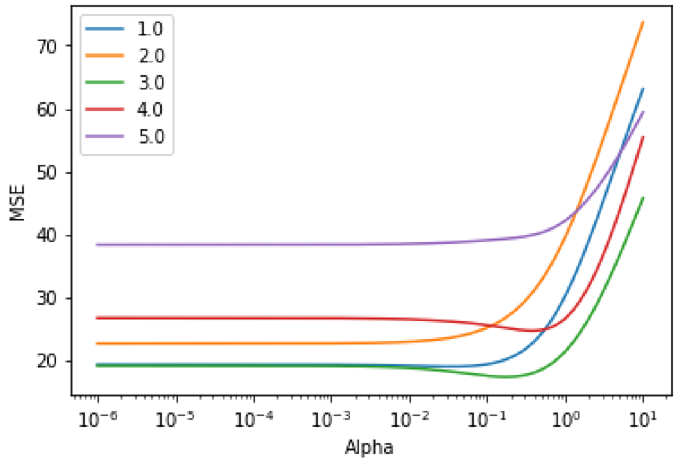
- L1范数：当p=1时，是L1范数，表示某个向量中所有元素的绝对值之和。
- L2范数：当p=2时，是L2范数，表示某个向量中所有元素的平方和再开根号。

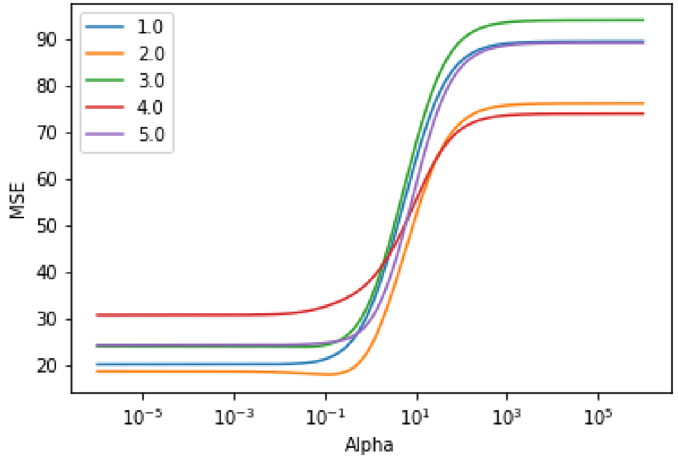
4 范数

岭回归：

- 正则化项是参数的L2范数时，整个回归方法就叫做岭回归。相应损失函数：

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 + \alpha\sum_{\mathrm{j}}^{\mathrm{n}}\theta_{\mathrm{j}}^2$$





4 范数

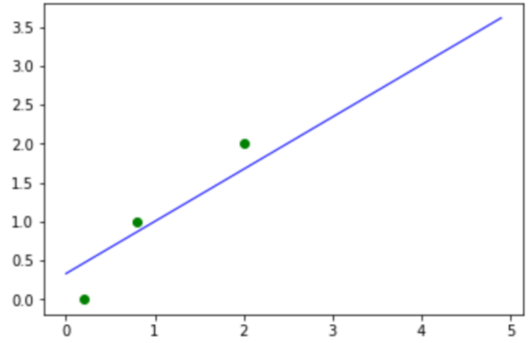
使用sklearn实现：

- from sklearn.linear_model import Ridge
- Ridge(alpha=1.0, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
- 重要参数：
  - alpha：二范数权值。默认1.0。
  - fit_intercept：default True。是否计算截距，默认为计算。
  - normalize：该参数在 fit_intercept 设置为False时自动忽略，表示是否在回归前对数据进行正则化，如果要对数据进行标准化，请使用sklearn.preprocessing.StandardScaler。
  - solver：优化器。默认选择适合数据的。可使用 'auto'，'svd'，'cholesky'，'lsqr'，'sparse_cg'，'sag'，'saga'

4 范数

■ 使用sklearn实现：

```
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

X_train = np.array([[0.2], [0.8], [2]])
y_train = np.array([0, 1, 2])
reg = Ridge(alpha=1).fit(X_train, y_train)

X_test = np.arange(0,5,0.1).reshape(-1,1)
y_pred = reg.predict(X_test)

plt.scatter(X_train, y_train,  color='g')
plt.plot(X_test, y_pred, color='b', linewidth=1)
plt.show()

print(reg.coef_)
print(reg.intercept_)
print('Learned regression model: y = %.2fx + %.2f' % (reg.coef_,reg.intercept_))

print('Mean Squared Error for training: ',mean_squared_error(y_train,reg.predict(X_train)))
```




```
[0.67164179]
0.32835820895522405
Learned regression model: y = 0.67x + 0.33
```

4 范数

Lasso回归:

- lasso回归：参数范数为L1范数

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2+\beta\sum_{\mathrm{j}}^{\mathrm{n}}|\theta_{\mathrm{j}}|$$

- 优势：不仅可以解决过拟合问题，而且可以在参数缩减过程中，将一些重复或不重要的参数直接缩减为零（删除），有提取有用特征的作用。
- 劣势：计算过程复杂，毕竟L1范数不是连续可导的。

4 范数

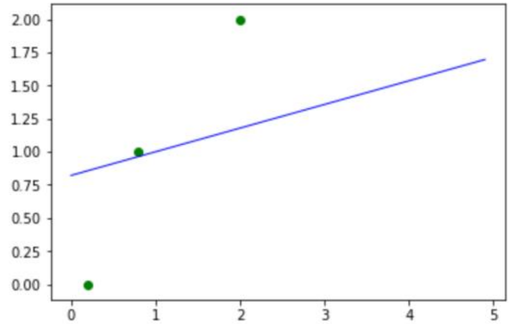
使用sklearn实现：

- from sklearn.linear_model import Lasso
- 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')
- 重要参数：
    - alpha：一范数权值。默认1.0。
    - fit_intercept：default True。是否计算截距，默认为计算。
    - normalize：该参数在 fit_intercept 设置为False时自动忽略，表示是否在回归前对数据进行正则化，如果要对数据进行标准化，请使用sklearn.preprocessing.StandardScaler。

4 范数

- 使用sklearn实现：

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

X_train = np.array([[0.2], [0.8], [2]])
y_train = np.array([0, 1, 2])
reg = Lasso(alpha=0.5).fit(X_train, y_train)

X_test = np.arange(0,5,0.1).reshape(-1,1)
y_pred = reg.predict(X_test)

plt.scatter(X_train, y_train,  color='g')
plt.plot(X_test, y_pred, color='b', linewidth=1)
plt.show()

print(reg.coef_)
print(reg.intercept_)
print('Learned regression model: y = %.2fx + %.2f' % (reg.coef_,reg.intercept_))

print('Mean Squared Error for training: ',mean_squared_error(y_train,reg.predict(X_train)))
```




```
[0.17857143]
0.8214285714285714
Learned regression model: y = 0.18x + 0.82
Mean Squared Error for training:  0.4702380952380952
```

4 范数

Elastic Net：

- 另一种回归方法叫Elastic Net，它同时采用了L1和L2正则，以综合Ridge Regression和Lasso Regression两者的优点。

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 + \alpha\sum_{\mathrm{j}}^{\mathrm{n}}|\theta_{\mathrm{j}}| + +\beta\sum_{\mathrm{j}}^{\mathrm{n}}\theta_{\mathrm{j}}^2$$

- 既能稀疏化模型权重，又能保持岭回归的稳定性。

4 范数

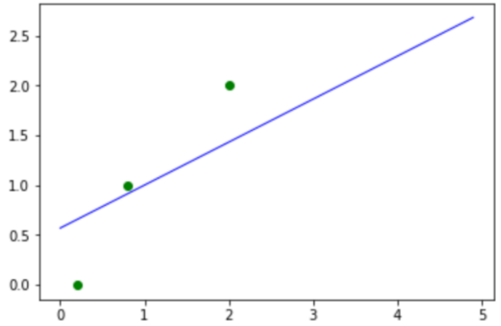
使用sklearn实现：

- from sklearn.linear_model import ElasticNet
- 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')
- 重要参数：
  - alpha：范数权值。默认1.0。
  - l1_ratio：一范数的比例。默认0.5。
  - fit_intercept ：default True。是否计算截距，默认为计算。
  - normalize ：该参数在 fit_intercept 设置为False时自动忽略，表示是否在回归前对数据进行正则化，如果要对数据进行标准化，请使用sklearn.preprocessing.StandardScaler。

4 范数

- 使用sklearn实现：

```python
# simple example for lasso regression
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

X_train = np.array([[0.2], [0.8], [2]])
y_train = np.array([0, 1, 2])
reg = ElasticNet(alpha=0.5).fit(X_train, y_train)

X_test = np.arange(0,5,0.1).reshape(-1,1)
y_pred = reg.predict(X_test)

plt.scatter(X_train, y_train,  color='g')
plt.plot(X_test, y_pred, color='b', linewidth=1)
plt.show()

print(reg.coef_)
print(reg.intercept_)
print('Learned regression model: y = %.2fx + %.2f' % (reg.coef_,reg.intercept_))

print('Mean Squared Error for training: ',mean_squared_error(y_train,reg.predict(X_train)))
```




```
[0.43209877]
0.5679012345679013
Learned regression model: y = 0.43x + 0.57
```

4 范数

非线性模型：

- 对于非线性关系问题，自然的方法是将标准线性模型换成一个多项式函数：

$$y_i = b_0 + b_1 * x_i^1 + b_2 * x_i^3 + \cdots + b_n * x_i^n$$

称为多项式回归。

- 对于一般的非线性模型，本质上是在对X的函数或变换进行建模：

$$y_i = b_0 + b_1 * \beta_1(x_i) + b_2 * \beta_2(x_i) + \cdots + b_n * \beta_n(x_i)$$

这里$\beta_n(x_i)$的形式是事先确定好的。

4 范数

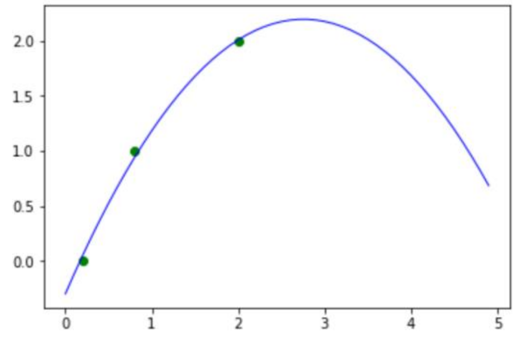
使用sklearn实现特征转换：

- from sklearn.preprocessing import PolynomialFeatures
- PolynomialFeatures(degree=2, interaction_only=False, include_bias=True, order='C')
- 如：$[x_1,x_2]$ 转换为 $[1,x_1,x_2,x_1^2,x_1x_2,x_2^2]$
- 重要参数：
  - degree ：多项式特征的次数。
  - interaction_only：是否只生成交互特征，默认否。如果值为是的话，像x[1] ** 2, x[0] * x[2] ** 3这样的特征不会有。
  - order：顺序，默认C的顺序。

4 范数

- 使用sklearn实现：

In [367]:
```python
# simple example for Polynomial regression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

X_train = np.array([[0.2], [0.8], [2]])
y_train = np.array([0, 1, 2])

reg = Pipeline([('poly', PolynomialFeatures(degree=2)),
                  ('linear', Ridge(alpha=0.01))])

reg.fit(X_train, y_train)

X_test = np.arange(0,5,0.1).reshape(-1,1)
y_pred = reg.predict(X_test)

plt.scatter(X_train, y_train,  color='g')
plt.plot(X_test, y_pred, color='b', linewidth=1)
plt.show()

print('Mean Squared Error for training: ',mean_squared_error(y_train,reg.predict(X_train)))
```




```
Mean Squared Error for training:  0.002064563346534875
```

问题？

暨南大學
JINAN UNIVERSITY