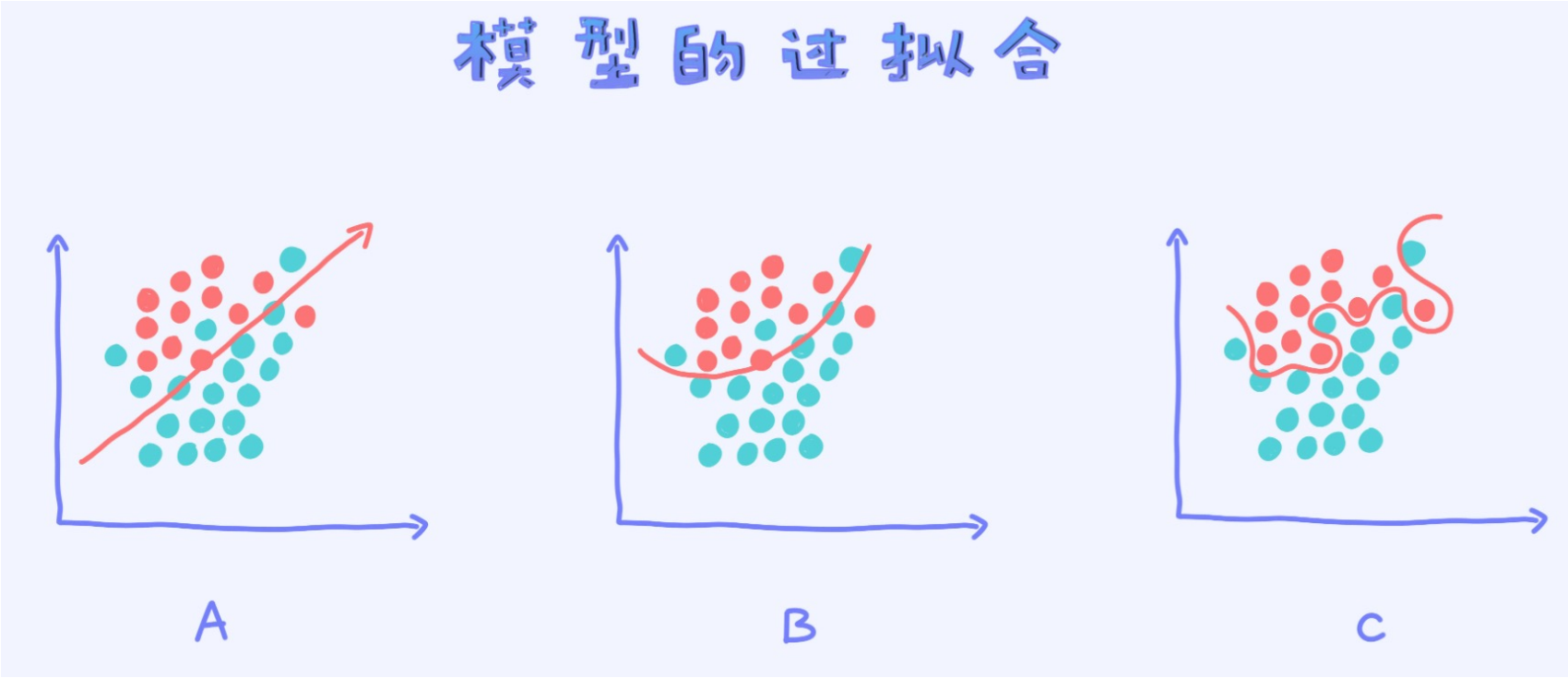
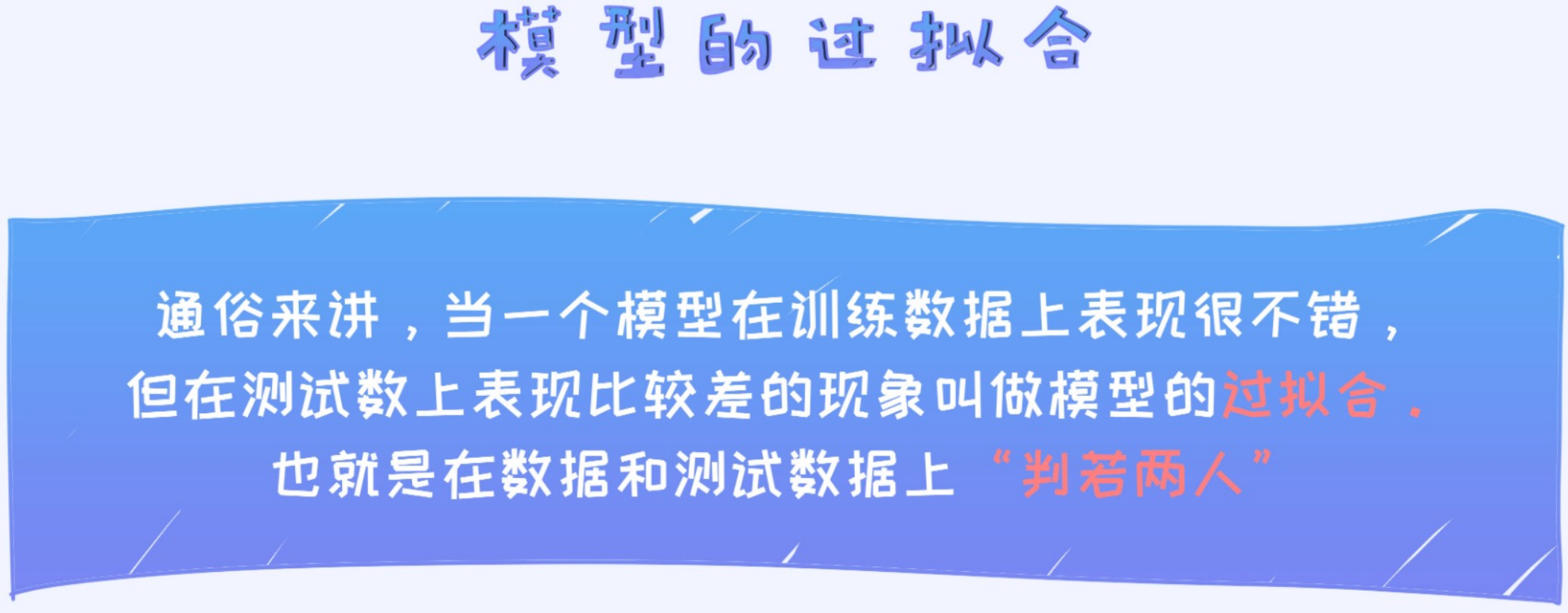
目录 | Contents

拟合状态



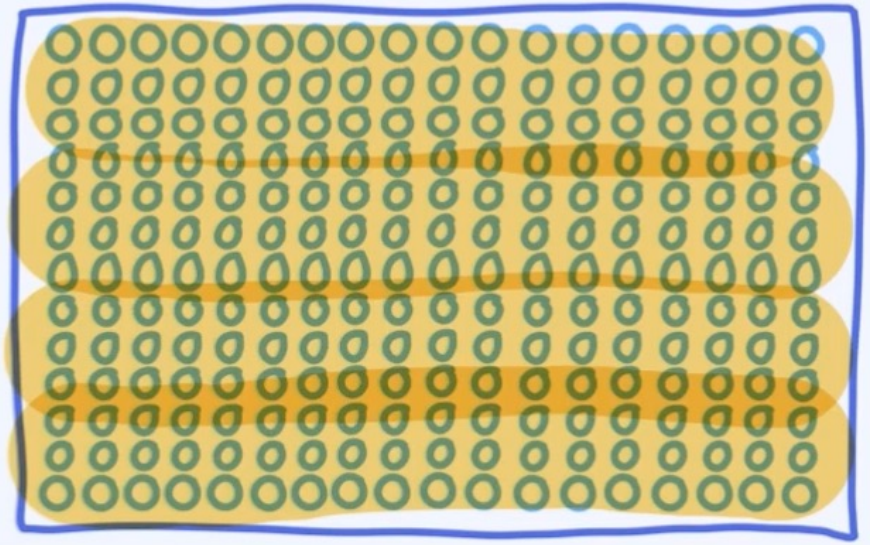
越复杂的模型越容易过拟合。这就是为什么深度神经网络会比较容易过拟合的主要原因。所以针对于复杂的模型，我们需要设计出好的机制来避免过拟合。

过拟合

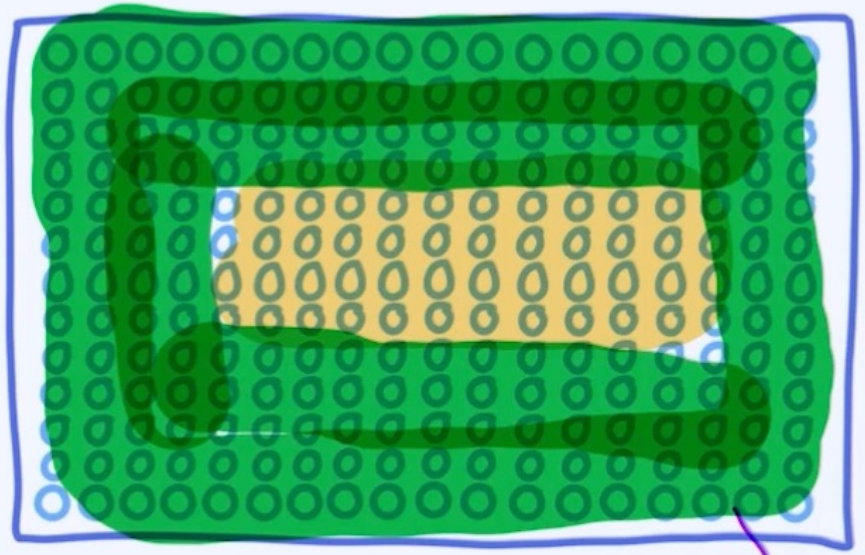

模型的过拟合
通俗来讲，当一个模型在训练数据上表现很不错，
但在测试数上表现比较差的现象叫做模型的过拟合。
也就是在数据和测试数据上“判若两人”

正则

正则的作用：对可行解空间（feasible region）的限制



加入正则之前



加入正则之后

复杂的模型

L2正则

原目标函数

$$\hat{w}_{MLE},\ \hat{b}_{MLE} = argmin_{w,b} - \sum_{i=1}^{n} \log p(y_i|x_i, w, b)$$

修改后的目标函数

$$\hat{w}_{MLE},\ \hat{b}_{MLE} = argmin_{w,b} - \sum_{i=1}^{n} \log p(y_i|x_i, w, b) + \lambda||\vec{w}||_2^2$$

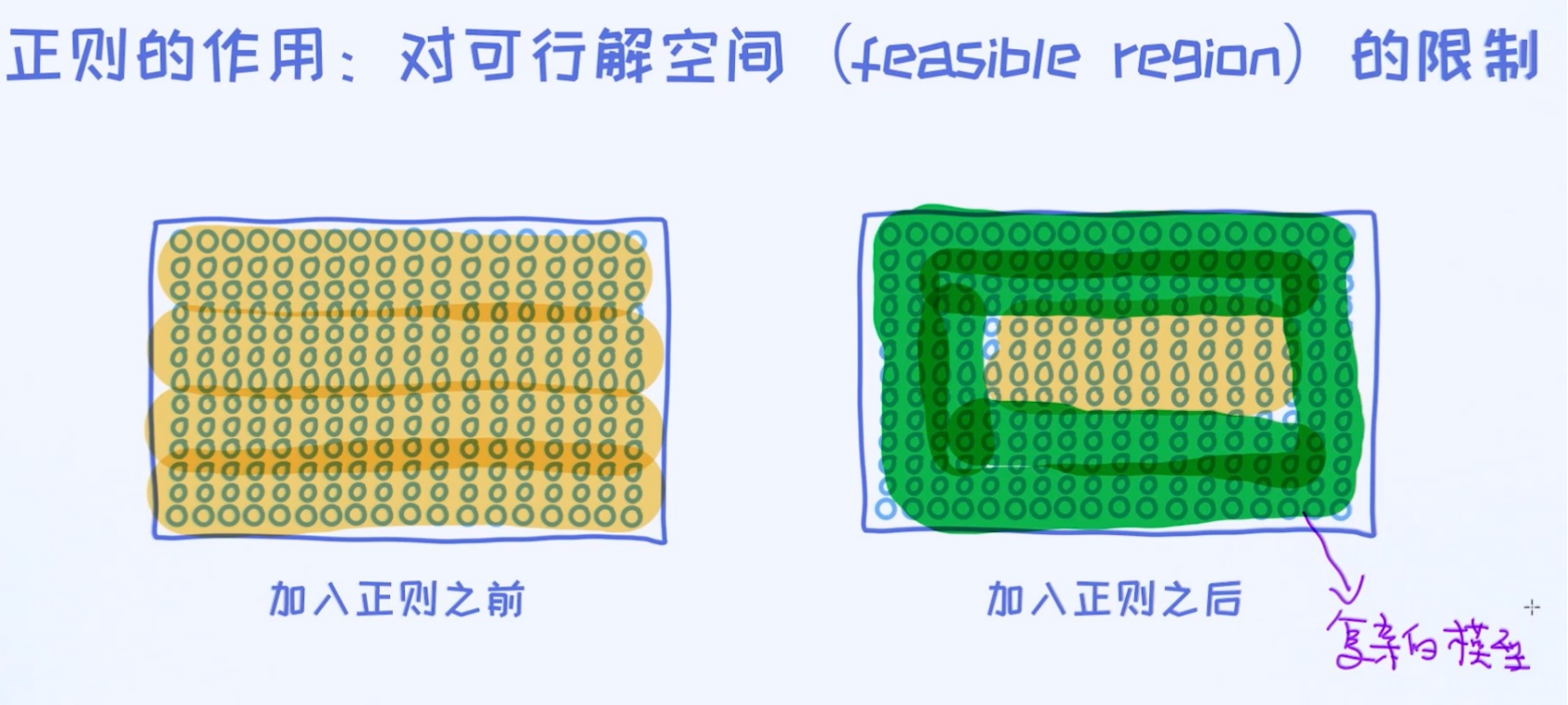
$$||w||_2^2 = w_1^2 + w_2^2 + \cdots + w_D^2$$

特征1　　特征2　　特征3　　特征4
0.5　　0.5　　0.5　　0.5

L2= 0.5**2 * 4 = 1

0　0　1　0

正则



通过正则我们可以缩小可行解空间，而且在这里被我们丢弃掉的可行解是比较容易产生过拟合的。我们知道正则的强度是由超参数lambda来控制，这个值越大，所选择的可行解空间也会相应变小。

L1和2正则

正则	$l(w) = f(w) + \lambda \|\|w\|\|_2^2$	$l(w) = f(w) + \lambda \|\|w\|\|_1$
名称	L2	L1
数学表达	$\|\|w\|\|_2^2 = w_1^2 + w_2^2 + \cdots + w_D^2$	$\|\|w\|\|_1 = \|w_1\| + \|w_2\| + \cdots + \|w_D\|$
作用	1.让参数变小	1.让参数变小 2.稀疏性：很多参数等于0

L1正则可以帮助做特征选择，因为不重要的特征慢慢的就变为0了

L1和2正则

从计算的角度来讲，L1范数的挑战要大很多。一旦目标函数里包含了L1的正则，则优化起来会比较麻烦。主要的原因是L1范数在0点不具备梯度，所以需要做一些特殊处理，比如使用subgradient来代替梯度。

另外，L1范数虽然有特征选择的功能，但也有一些不足。比如多个特征具有强相关性，那通过L1正则选出来的特征可能是这些特征里的任意个，但实际上特征还是有好有坏。即便特征之间有强相关性，但肯定其中有最好的特征。

为了弥补这个缺点，学者们提出了一个方案，就是联合L1和L2正则一起使用。如果我们在线性回归的基础上联合使用了此两个正则，这个模型就是非常著名的ElasticNet。

泛化能力

在建模中，我们最终的目的是要构建出一个泛化能力(generalization capability)很强的模型。所谓对泛化通俗的理解就是模型可以很好地适应新的环境。
这就类似于一个有能力的人，到什么岗位上其实都可以做出出色的成绩；在学习过程中，一个优秀的学生具备很强的举一反三的能力；一个好的模型不仅在训练数据上表现好，而且在测试或者线上环境里表现也比较稳定且出色。这就是对泛化的一个通俗的解释。

泛化能力

构建泛化能力强的模型

- 选择正确的数据
- 选择合适的模型
- 选择合适的优化算法
- 避免模型的过拟合

泛化能力

第一、需要正确的数据。不可能去期待使用一个错误的数据来构建一个泛化能力强的模型。比如数据里包含了大量的噪声，这很难让我们训练出有效模型出来。

第二、需要选择合适的模型。比如图像识别，我们都知道CNN是最合适的模型； 对于构建评分卡，可能集成模型是比较合适的。这里没有一个绝对的理论，很多都是通过不断尝试得到的经验。

第三、需要选择合适的优化算法。针对于复杂的模型，这一点尤其重要。比如面对复杂的深度学习模型，我们可以选择随机梯度下降法，也可以选择Adam等算法。但每一种不同类型的优化算法给我们带来不同质量的解，而且这种解带来的泛化能力也是不一样的。

第四、需要避免过拟合现象。在前面所讲的条件不变的情况下，过拟合是最核心的问题。我们需要通过一些手段来避免或者降低过拟合现象。

目录 | Contents

什么是回归

回归的目的是预测数值型的目标值。最直接的办法是依据输入写出一个目标值的计算公式。假如你想预测小姐姐男友汽车的功率，可能会这么计算：

HorsePower = 0.0015 * annualSalary +0.99 * hoursListeningToPublicRadio

写成中文就是：

小姐姐男友汽车的功率 = 0.0015 * 小姐姐男友年薪 + 0.99 * 收听公共广播的时间

这就是所谓的回归方程（regression equation），其中的0.0015和0.99称为回归系数（regression weights），求这些回归系数的过程就是回归。一旦有了这些回归系数，再给定输入，做预测就非常容易了。具体的做法是用回归系数乘以输入值，再将结果全部加在一起，就得到了预测值。

线性回归

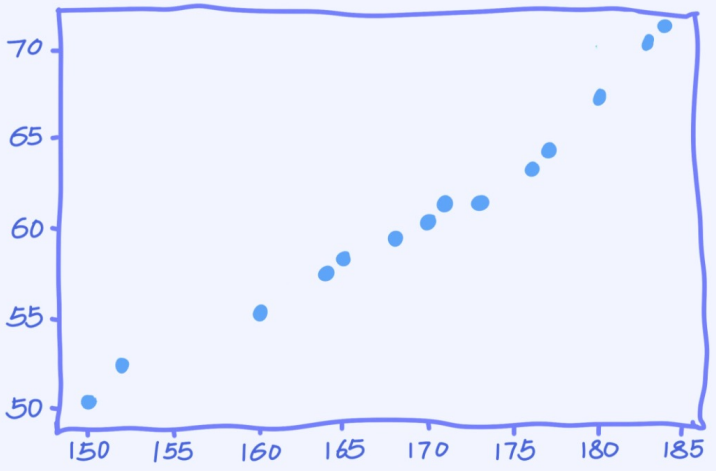
线性回归算法

- 解决回归问题
- 思想简单，实现容易
- 许多强大的非线性模型的基础
- 结果具有很好的可解释性
- 蕴含机器学习中的很多重要思想

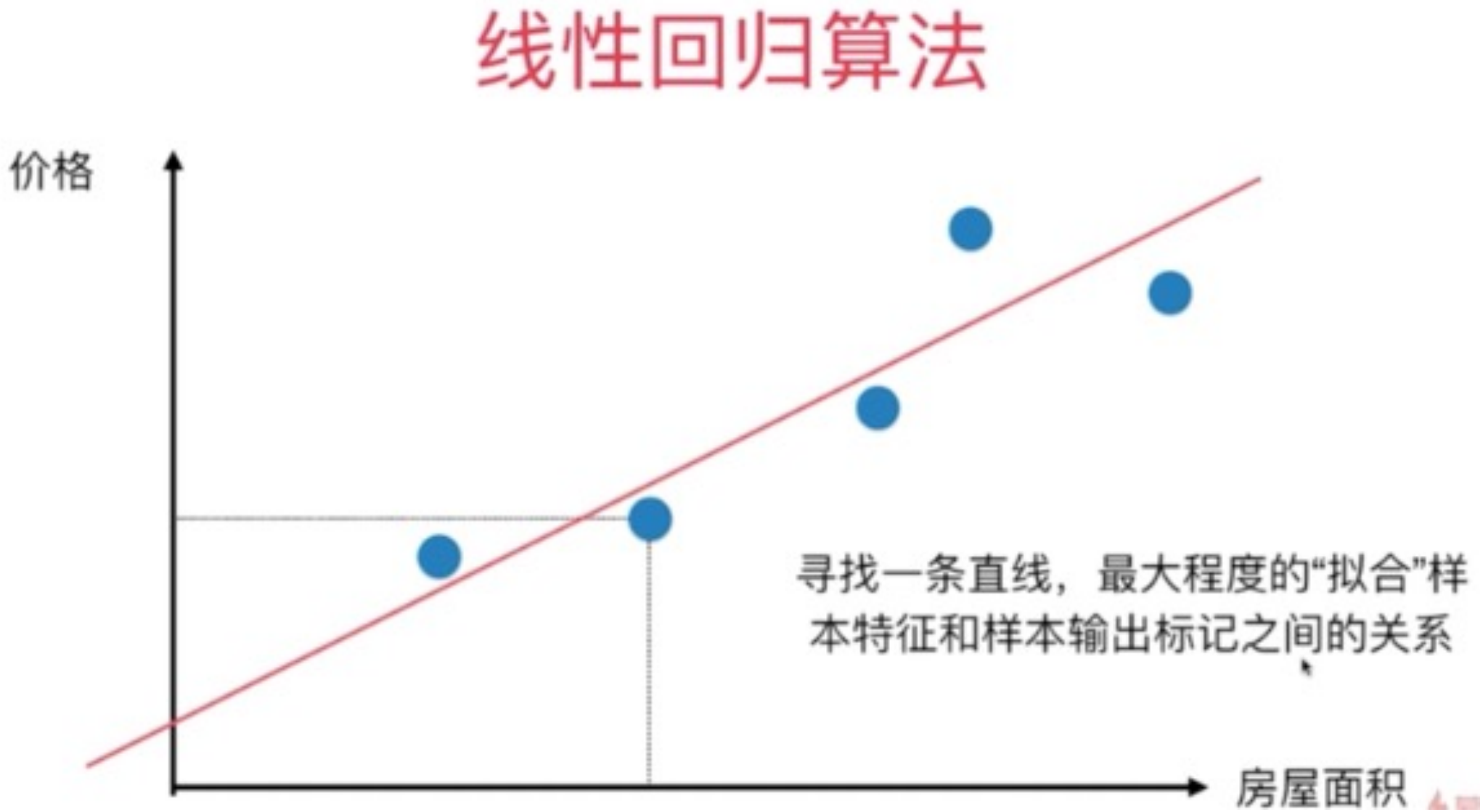
简单线性回归

使用线性回归的前提是数据本身具有一定的线性特点，不然很难拟合数据的。那数据到底有没有线性的特点呢？ 简单的方法就是通过可视化来观察数据。

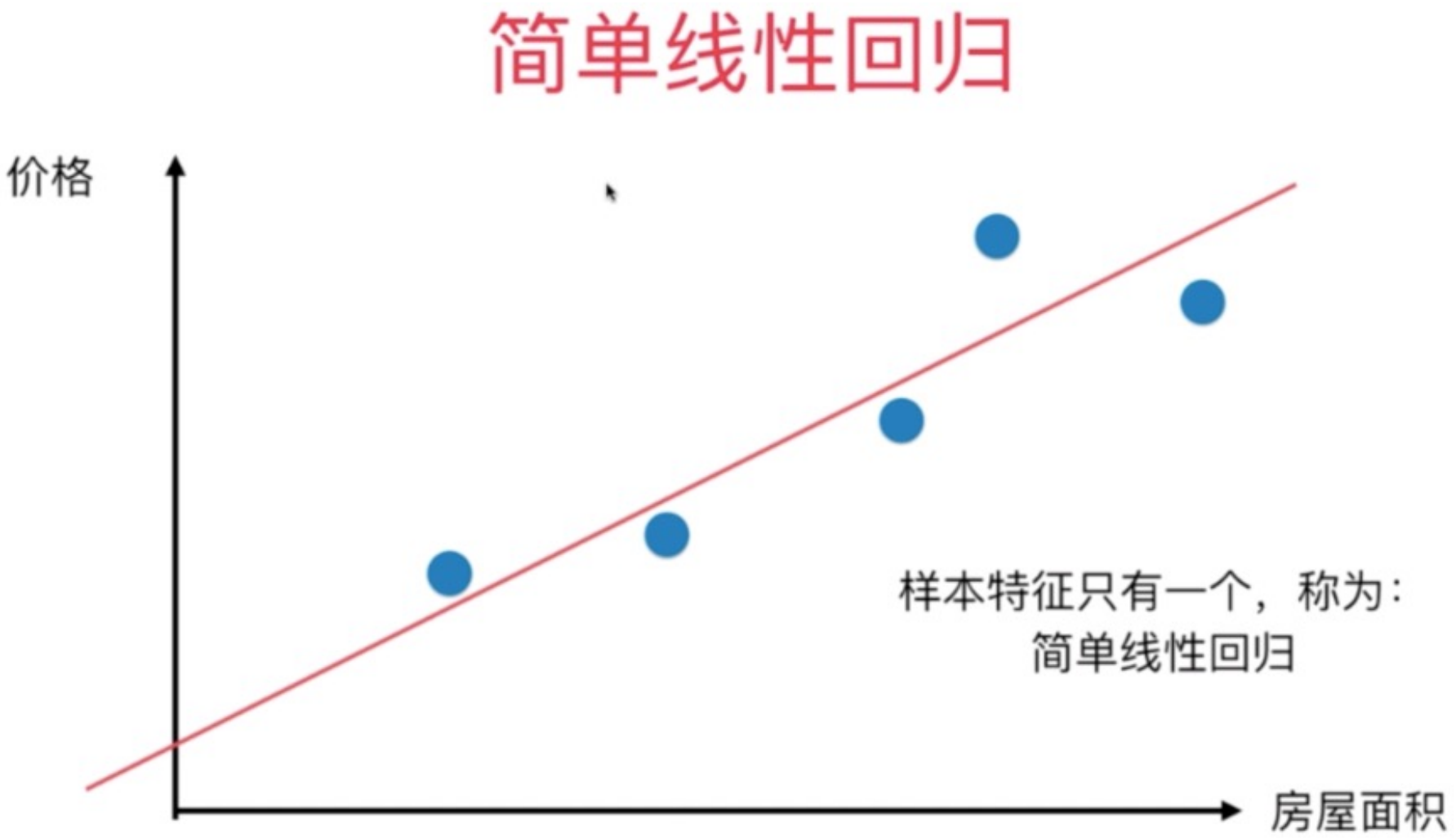
身高(cm)	体重(kg)
150	50
152	52
160	55
164	57
165	58
168	59
170	60
171	61
173	61
176	63
177	64
180	67
183	70
184	71



线性回归

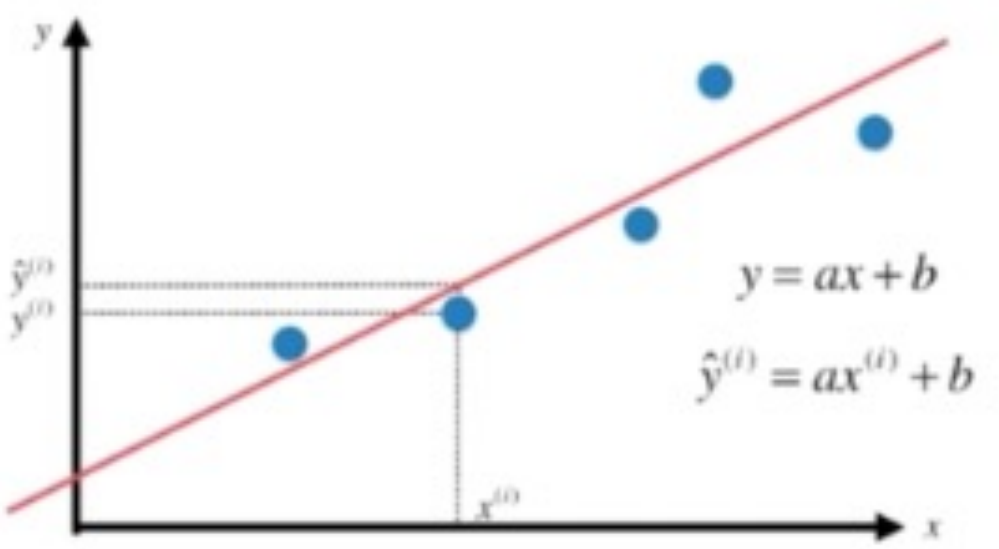


简单线性回归



线性回归

简单线性回归



假设我们找到了最佳拟合的直线方程：

$$y = ax + b$$

则对于每一个样本点 $x^{(i)}$

根据我们的直线方程，预测值为：

$$\hat{y}^{(i)} = ax^{(i)} + b$$

真值为：$y^{(i)}$

线性回归

简单线性回归

假设我们找到了最佳拟合的直线方程：

$$y = ax + b$$

则对于每一个样本点 $x^{(i)}$

根据我们的直线方程，预测值为：

$$\hat{y}^{(i)} = ax^{(i)} + b$$

真值为： $y^{(i)}$

我们希望 $y^{(i)}$ 和 $\hat{y}^{(i)}$ 的差距尽量小

表达 $y^{(i)}$ 和 $\hat{y}^{(i)}$ 的差距：

~~$y^{(i)} - \hat{y}^{(i)}$~~

~~$|y^{(i)} - \hat{y}^{(i)}|$~~

$$(y^{(i)} - \hat{y}^{(i)})^2$$

考虑所有样本： $\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$

第一个公式：差距有可能为0，比如第一个差距为100，第二个差距为-100

第二个公式：求导问题，导数不连续

线性回归

简单线性回归

目标：使 $\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2$ 尽可能小

$$\hat{y}^{(i)}=ax^{(i)}+b$$

目标：找到a和b，使得 $\sum_{i=1}^{m}(y^{(i)}-ax^{(i)}-b)^2$ 尽可能小

最小二乘法解决a,b

一个样本的误差 yi^ - yi
找到误差最小的时刻，为了去找到误差最小的时刻，
需要反复尝试，a和b。根据**最小二乘法**去求得误差。
反过来误差最小时刻的a，b就是最终最优解模型！

线性回归

一类机器学习算法的基本思路

目标：找到a和b，使得 $\sum_{i=1}^{m}(y^{(i)}-ax^{(i)}-b)^2$ 尽可能小

↓

损失函数(loss function)　效用函数(utility function)

通过分析问题，确定问题的损失函数或者效用函数；
通过最优化损失函数或者效用函数，获得机器学习的模型。

对于损失函数我们希望尽可能的小

对于效用函数我们希望尽可能的大

线性回归

一类机器学习算法的基本思路

通过分析问题，确定问题的损失函数或者效用函数；
通过最优化损失函数或者效用函数，获得机器学习的模型。

近乎所有参数学习算法都是这样的套路

线性回归	SVM
多项式回归	神经网络
逻辑回归	……

最小二乘法

假定x, y有如下数值：

y	1.00	0.90	0.90	0.81	0.60	0.56	0.35
x	3.60	3.70	3.80	3.90	4.00	4.10	4.20

将这些数值画图可以看出接近一条直线，故用y=ax+b表示，故将上面的数值代入表达式有：

3.6a+b−1.00=0
3.7a+b−0.90=0
3.8a+b−0.90=0
3.9a+b−0.81=0
4.0a+b−0.60=0
4.1a+b−0.56=0
4.2a+b−0.35=0

由于直线只有两个未知数a, b，理论上只需要两个方程就能求得，但是实际上是不可能的，因为所有点并没有真正的在同一条直线上，即不可能所有的数值都满足ax+b−y = 0

故只需找到一对儿a、b，使得误差平方和

$$\sum(ax_i+b-y_i)^2=(ax_0+b-y_0)^2+(ax_1+b-y_1)^2+\ldots\ldots+(ax_n+b-y_n)^2$$

最小即可。
误差的平方即二乘方，故称之为最小二乘法。

线性回归

简单线性回归

目标：找到a和b，使得 $\sum_{i=1}^{m}(y^{(i)}-ax^{(i)}-b)^2$ 尽可能小

典型的最小二乘法问题：最小化误差的平方

$$a=\frac{\sum_{i=1}^{m}(x^{(i)}-\bar{x})(y^{(i)}-\bar{y})}{\sum_{i=1}^{m}(x^{(i)}-\bar{x})^2}$$

$$b=\bar{y}-a\bar{x}$$

$\bar{X}$ X的均值，y也是如此

求a，b的公式不用背，百度到处都有

推导

Loss$= \sum_{i=1}^{N}(wx_i + b - y_i)^2$对b求导

$= 2(z)1 = 0$ z对b求导，其中$z = \sum_{i=1}^{N} wx_i + b - y_i$,使得导数为0，最后一个1代表了z对b的导数

$$w\sum_{i=1}^{N} x_i + \sum_{i=1}^{N} b - \sum_{i=1}^{N} y_i = 0$$同时除以N

$$b = \bar{y} - w\bar{x}$$

A的推导过程

$$J(a,b)=\sum_{i=1}^{m}(y^{(i)}-ax^{(i)}-b)^2$$

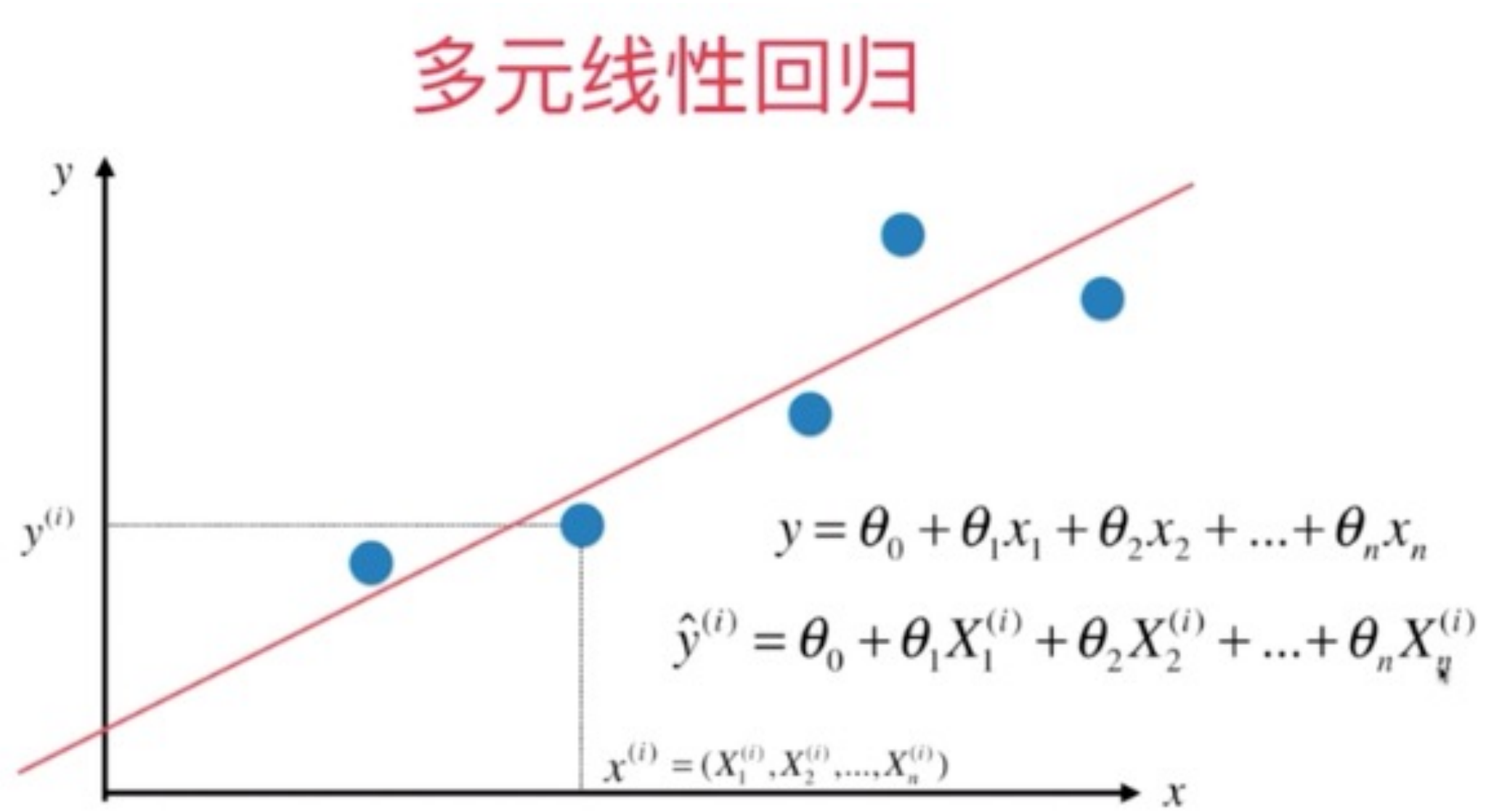
$$\frac{\partial J(a,b)}{\partial a}=0 \qquad b=\bar{y}-a\bar{x}$$

$$\frac{\partial J(a,b)}{\partial a}=\sum_{i=1}^{m}2(y^{(i)}-ax^{(i)}-b)(-x^{(i)})=0$$

$$\sum_{i=1}^{m}(y^{(i)}-ax^{(i)}-b)x^{(i)}=0$$

B带入到这个式子中。

多元线性回归



@0就是截距

多元线性回归

对于简单线性回归来说，求得a,b就可以了

对于多元线性回归来说，求得
@0~@n

多元线性回归

目标：使 $\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2$ 尽可能小

$$\hat{y}^{(i)}=\theta_0+\theta_1X_1^{(i)}+\theta_2X_2^{(i)}+...+\theta_nX_n^{(i)}$$

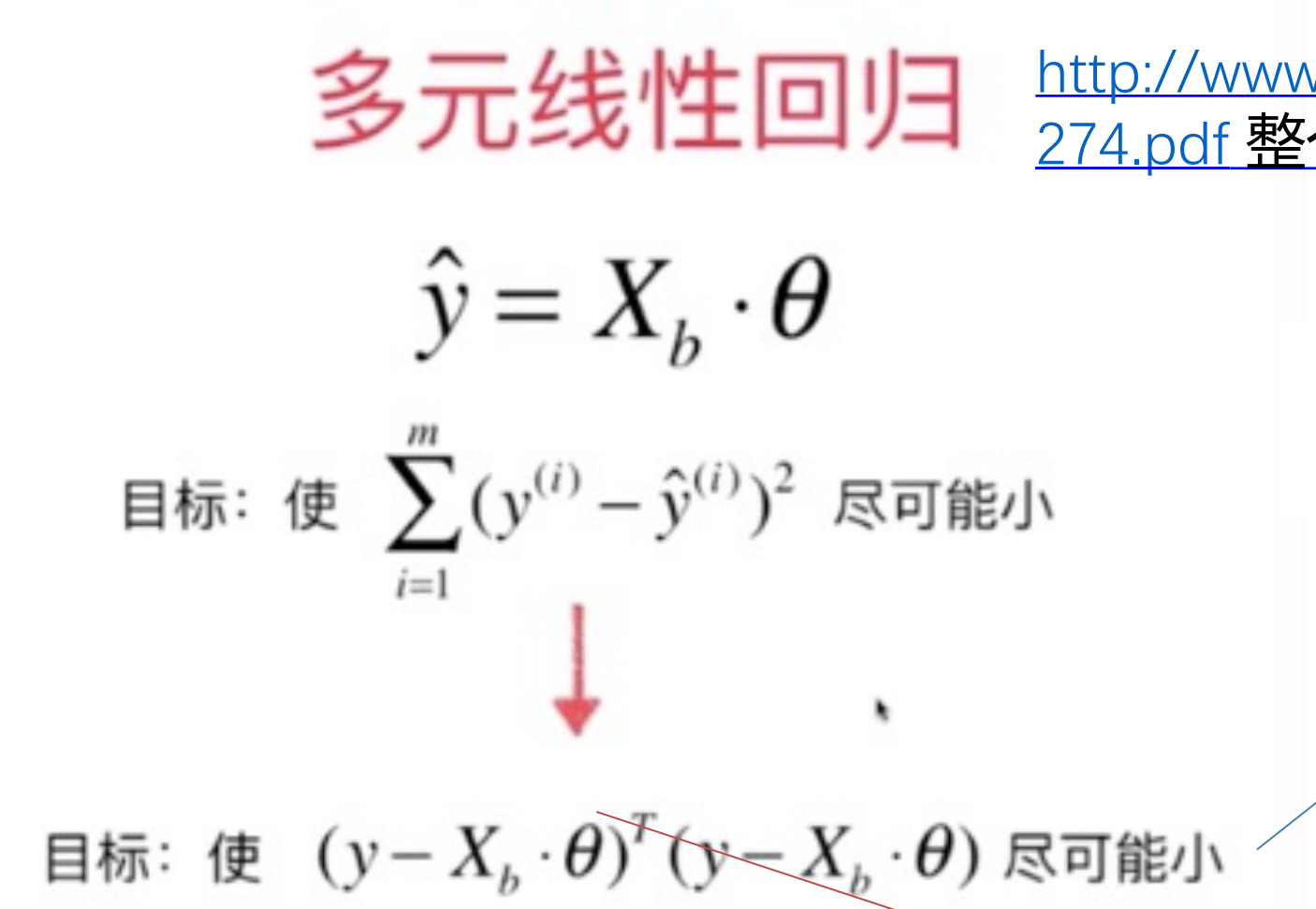
目标：找到 $\theta_0,\theta_1,\theta_2,...,\theta_n$ ，使得 $\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2$ 尽可能小

多元线性回归

$$X_b = \begin{pmatrix} 1 & X_1^{(1)} & X_2^{(1)} & \cdots & X_n^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} & \cdots & X_n^{(2)} \\ \cdots & & & & \cdots \\ 1 & X_1^{(m)} & X_2^{(m)} & \cdots & X_n^{(m)} \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \cdots \\ \theta_n \end{pmatrix}$$

$$\hat{y} = X_b \cdot \theta$$

多元线性回归



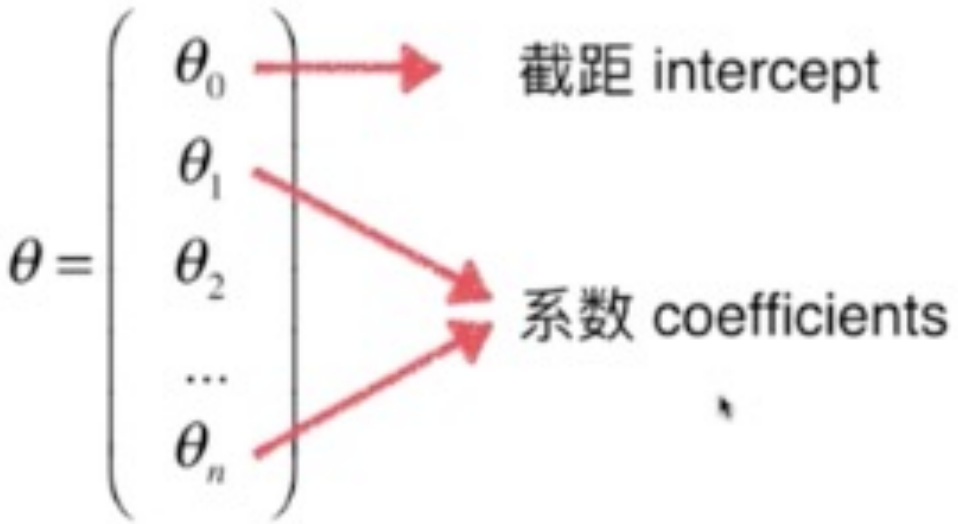
http://www2.imm.dtu.dk/pubdb/edoc/imm3274.pdf 整个求解可以参考

$$\theta = (X_b^T X_b)^{-1} X_b^T y$$

推导过于复杂。。。略

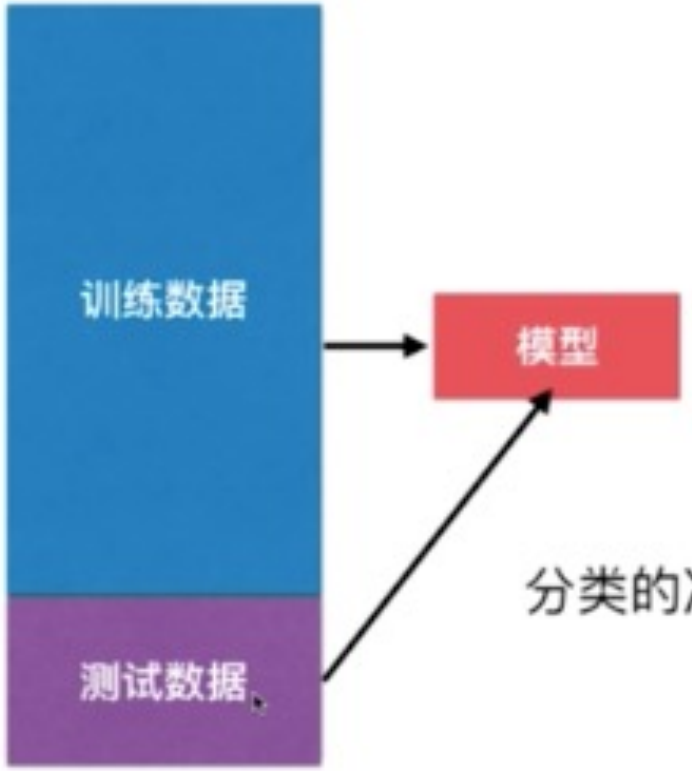
列向量转为行向量

多元线性回归


$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \cdots \\ \theta_n \end{pmatrix}$
截距 intercept
系数 coefficients

回归算法的评价

回归算法的评价



目标：找到a和b，使得 $\sum_{i=1}^{m}(y_{train}^{(i)}-ax_{train}^{(i)}-b)^2$ 尽可能小

$$\hat{y}_{test}^{(i)}=ax_{test}^{(i)}+b$$

$$\sum_{i=1}^{m}(y_{train}^{(i)}-\hat{y}_{train}^{(i)})^2$$

衡量标准：$\sum_{i=1}^{m}(y_{test}^{(i)}-\hat{y}_{test}^{(i)})^2$

分类的准确度：accuracy

回归算法的评价

衡量标准：$\sum_{i=1}^{m}(y_{test}^{(i)}-\hat{y}_{test}^{(i)})^2$

问题：和m相关?

M的问题就不好衡量了，可能测试集的数量引起的问题所以需要除以m！

$\frac{1}{m}\sum_{i=1}^{m}(y_{test}^{(i)}-\hat{y}_{test}^{(i)})^2$

均方误差 MSE
(Mean Squared Error)

问题：误差的量纲和y不同

$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_{test}^{(i)}-\hat{y}_{test}^{(i)})^2}=\sqrt{MSE_{test}}$

均方根误差 RMSE
(Root Mean Squared Error)

回归算法的评价

$$\frac{1}{m}\sum_{i=1}^{m}\left| y_{test}^{(i)} - \hat{y}_{test}^{(i)} \right|$$

平均绝对误差 MAE
(Mean Absolute Error)

评价指标

RMSE vs MAE

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_{test}^{(i)} - \hat{y}_{test}^{(i)})^2}$$

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_{test}^{(i)} - \hat{y}_{test}^{(i)}|$$

RMSE会放大错误率，所以如果优化RMSE的话，使之变小意义就很大。

放大错误率的原因是 最大的误差 开平方了

回归最佳评价指标

RMSE vs MAE

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_{test}^{(i)} - \hat{y}_{test}^{(i)})^2} \qquad MAE = \frac{1}{m}\sum_{i=1}^{m}|y_{test}^{(i)} - \hat{y}_{test}^{(i)}|$$

问题：分类的准确度：1最好，0最差

但是上述这两个指标是不能给出的。

R Squared

R Squared

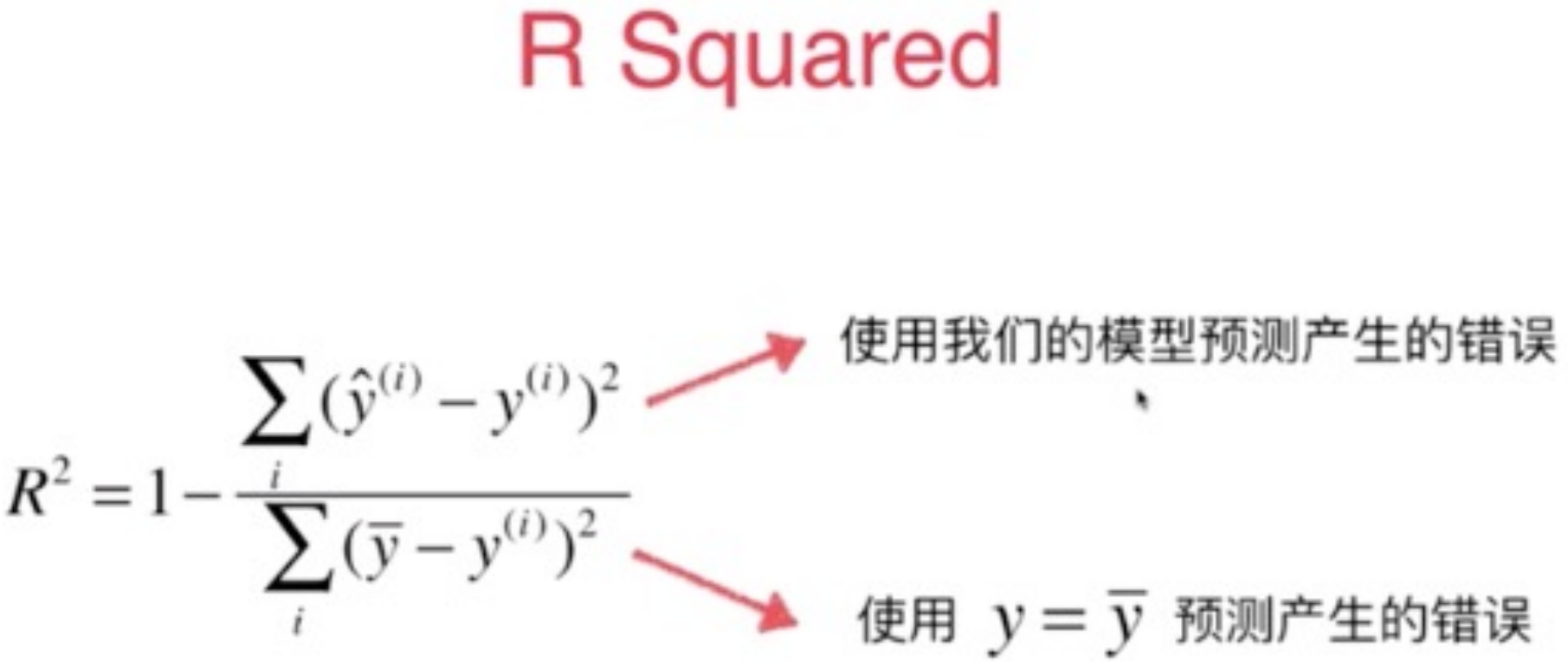
$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

(Residual Sum of Squares)

(Total Sum of Squares)

$$R^2 = 1 - \frac{\sum_i (\hat{y}^{(i)} - y^{(i)})^2}{\sum_i (\bar{y} - y^{(i)})^2}$$

R Squared



Baseline Model（产生非常多的错误）：所有的数据应该等于均值，上面的公式我们产生的错误会比较少。

R Squared

R Squared

$$R^2 = 1 - \frac{\sum_i (\hat{y}^{(i)} - y^{(i)})^2}{\sum_i (\bar{y} - y^{(i)})^2}$$

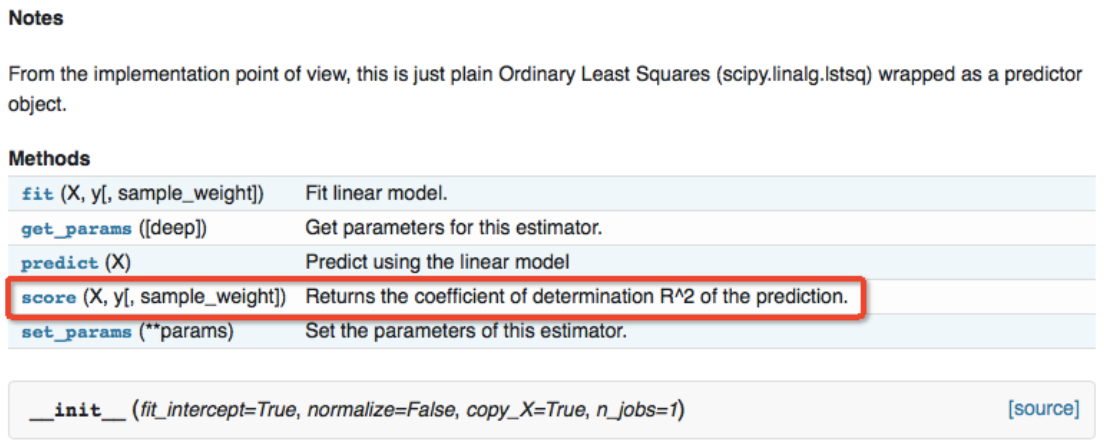
- R^2 <= 1
- R^2 越大越好。当我们的预测模型不犯任何错误是，R^2得到最大值1
- 当我们的模型等于基准模型时，R^2为0
- 如果R^2 < 0，说明我们学习到的模型还不如基准模型。此时，很有可能我们的数据不存在任何线性关系。

R Squared-sklearn计算

```
from sklearn.metrics import r2_score

r2_score(X_test,y_test)
```

在sklearn中的linearRegression里有一个方法score所返回的就是 R2这个计算方式。



Notes

From the implementation point of view, this is just plain Ordinary Least Squares (scipy.linalg.lstsq) wrapped as a predictor object.

Methods

`fit` (X, y[, sample_weight])	Fit linear model.
`get_params` ([deep])	Get parameters for this estimator.
`predict` (X)	Predict using the linear model
`score` (X, y[, sample_weight])	Returns the coefficient of determination R^2 of the prediction.
`set_params` (**params)	Set the parameters of this estimator.

`__init__` (*fit_intercept=True, normalize=False, copy_X=True, n_jobs=1*) [source]

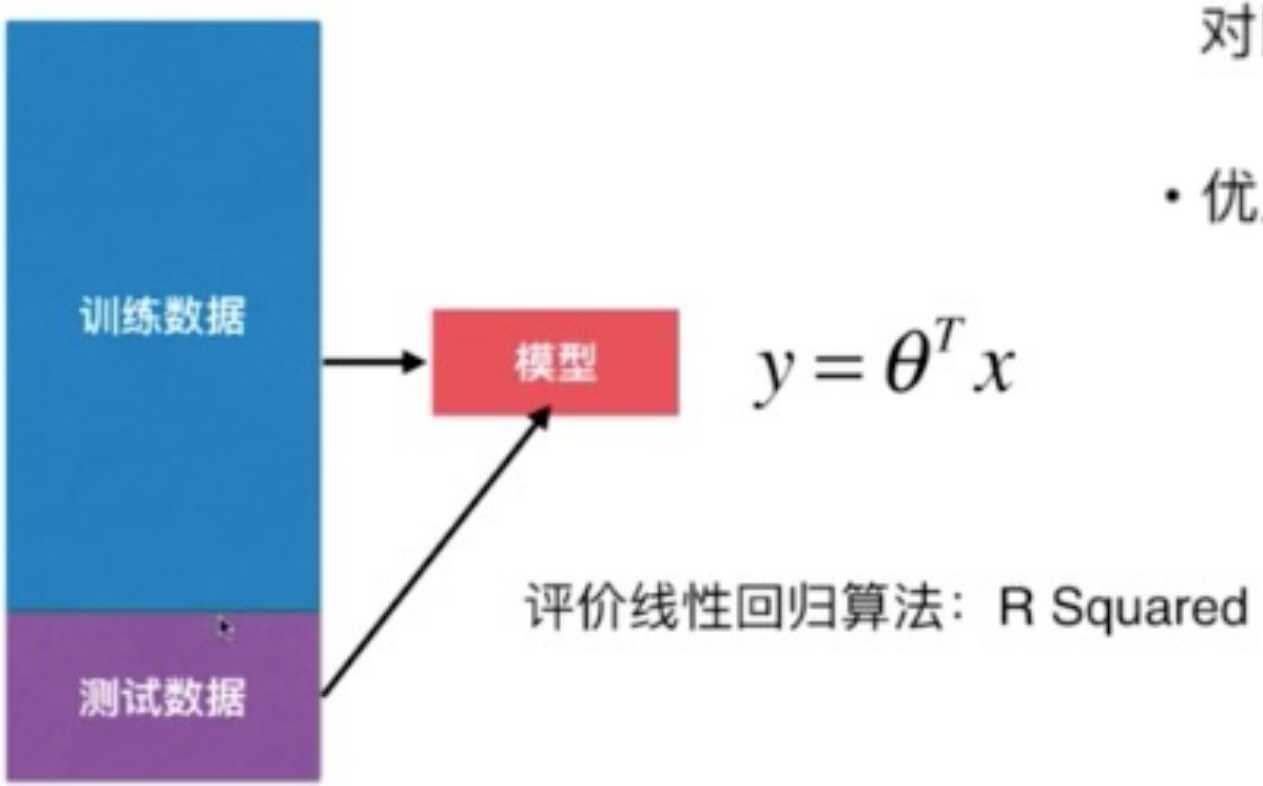
总结

线性回归算法总结

- 对数据有假设：线性

  对比kNN 对数据没有假设

- 优点：对数据具有强解释性



岭回归与lasso

岭回归与Lasso回归的出现是为了解决线性回归出现的过拟合以及在通过正规方程方法求解θ的过程中出现的(X^TX)不可逆这两类问题的，这两种回归均通过在损失函数中引入正则化项来达到目的。

在日常机器学习任务中，如果数据集的特征比样本点还多， $(X^TX)^{-1}$的时候会出错。岭回归最先用来处理特征数多于样本数的情况，现在也用于在估计中加入偏差，从而得到更好的估计。这里通过引入λ限制了所有θ^2之和，通过引入该惩罚项，能够减少不重要的参数，这个技术在统计学上也叫作缩减（shrinkage）。和岭回归类似，另一个缩减LASSO 也加入了正则项对回归系数做了限定。

为了防止过拟合(θ过大)，在目标函数$J(\theta)$后添加复杂度惩罚因子，即正则项来防止过拟合。正则项可以使用L1-norm(Lasso)、L2-norm(Ridge)，或结合L1-norm、L2-norm(Elastic Net)。

Lasso：使用L1-norm正则

$$J(\theta) = \frac{1}{2}\sum_{i}^{m}(y^{(i)} - \theta^T x^{(i)})^2 + \lambda \sum_{j}^{n} |\theta_j|$$

Ridge：使用L2-norm正则

$$J(\theta) = \frac{1}{2}\sum_{i}^{m}(y^{(i)} - \theta^T x^{(i)})^2 + \lambda \sum_{j}^{n} \theta_j^2$$

ElasticNet：结合l1-norm、l2-norm进行正则

$$J(\theta) = \frac{1}{2}\sum_{i}^{m}(y^{(i)} - \theta^T x^{(i)})^2 + \lambda(\rho \sum_{j}^{n} |\theta_j| + (1-\rho) \sum_{j}^{n} \theta_j^2)$$

目的

简单的理解正则化：

- 正则化的目的：防止过拟合
- 正则化的本质：约束（限制）要优化的参数
- 关于第1点，过拟合指的是给定一堆数据，这堆数据带有噪声，利用模型去拟合这堆数据，可能会把噪声数据也给拟合了，这点很致命，一方面会造成模型比较复杂，另一方面，模型的泛化性能太差了，遇到了新的数据让你测试，你所得到的过拟合的模型，正确率是很差的。
- 关于第2点，本来解空间是全部区域，但通过正则化添加了一些约束，使得解空间变小了，甚至在个别正则化方式下，解变得稀疏了。

岭回归实现

- **from sklearn.linear_model import Ridge**
- **from *sklearn.datasets* import load_boston**
- **from *sklearn.model_selection* import train_test_split**
- boston = load_boston()
- X = boston.data
- y = boston.target
- # 把数据分为训练数据集和测试数据集(20%数据作为测试数据集)
- X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
- model = Ridge(alpha=0.01, normalize=**True**)
- model.fit(X_train, y_train)
- test_score = model.score(X_test, y_test) # 模型对测试集的准确
- print(test_score)

调优

Ridge Classifier

Ridge regression is a penalized linear regression model for predicting a numerical value.

Nevertheless, it can be very effective when applied to classification.

Perhaps the most important parameter to tune is the regularization strength (*alpha*). A good starting point might be values in the range [0.1 to 1.0]

- **alpha** in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

岭回归实现

- **from sklearn.linear_model import RidgeCV**
- **from *sklearn.datasets* import load_boston**
- **from *sklearn.model_selection* import train_test_split**
- boston = load_boston()
- X = boston.data
- y = boston.target
- # 把数据分为训练数据集和测试数据集(20%数据作为测试数据集）
- X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
- model = RidgeCV(alphas=[1.0, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001], normalize=**True**)
- model.fit(X_train, y_train)
- print(model.alpha_)

lasso实现

- from sklearn.linear_model import Lasso
- lasso_reg = Lasso(alpha=0.1)
- lasso_reg.fit(x,y)

弹性网络

- 弹性网络是介于岭回归和Lasso回归之间的中间地带，大多数情况下，应该避免使用纯线性回归，岭回归是个不错的默认选择，但是如果实际用到的特征只有少数几个，那就应该更倾向于Lasso回归或者是弹性网络，因为它们会将无用的特征的权重降为0.一般来说弹性网络优于Lasso回归。因为特征数量超过训练实例数又或者是几个特征强相关的时候lasso回归的表现可能不稳定。
- from sklearn.linear_model import ElasticNet
- e_net = ElasticNet(alpha=0.1,l1_ratio=0.5)

参考代码

- **from sklearn.datasets import load_boston**
- **from *sklearn.linear_model* import LassoCV, ElasticNetCV**
- boston = load_boston()
- # Find the optimal alpha value for Lasso regression
- lscv = LassoCV(alphas=(1.0, 0.1, 0.01, 0.001, 0.005, 0.0025, 0.001, 0.00025), normalize=**True**)
- lscv.fit(boston.data, boston.target)
- print('Lasso optimal alpha: %.3f' % lscv.alpha_)
- # Find the optimal alpha and l1_ratio for Elastic Net
- encv = ElasticNetCV(alphas=(0.1, 0.01, 0.005, 0.0025, 0.001), l1_ratio=(0.1, 0.25, 0.5, 0.75, 0.8), normalize=**True**)
- encv.fit(boston.data, boston.target)
- print('ElasticNet optimal alpha: %.3f and L1 ratio: %.4f' % (encv.alpha_, encv.l1_ratio_))

美国标准地址

- 美国的标准地址为 第一行 街道号码跟街道名字，第二行为房间号，公寓号码，仓库号码等。第三行为城市，州，邮政编码。其中有一些地址，第二行是没有信息的。
-
- 490 2nd St,
-
- Suite 300
-
- San Francisco, CA 94107

labelEncoder

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

sklearn.preprocessing.LabelEncoder

class sklearn.preprocessing.LabelEncoder [source]

Encode target labels with value between 0 and n_classes-1.

This transformer should be used to encode target values, *i.e.* `y`, and not the input `X`.

Read more in the User Guide.

New in version 0.12.

Attributes:

classes_ : *ndarray of shape (n_classes,)*
Holds the label for each class.

See also:

`OrdinalEncoder`
Encode categorical features using an ordinal encoding scheme.

`OneHotEncoder`
Encode categorical features as a one-hot numeric array.