



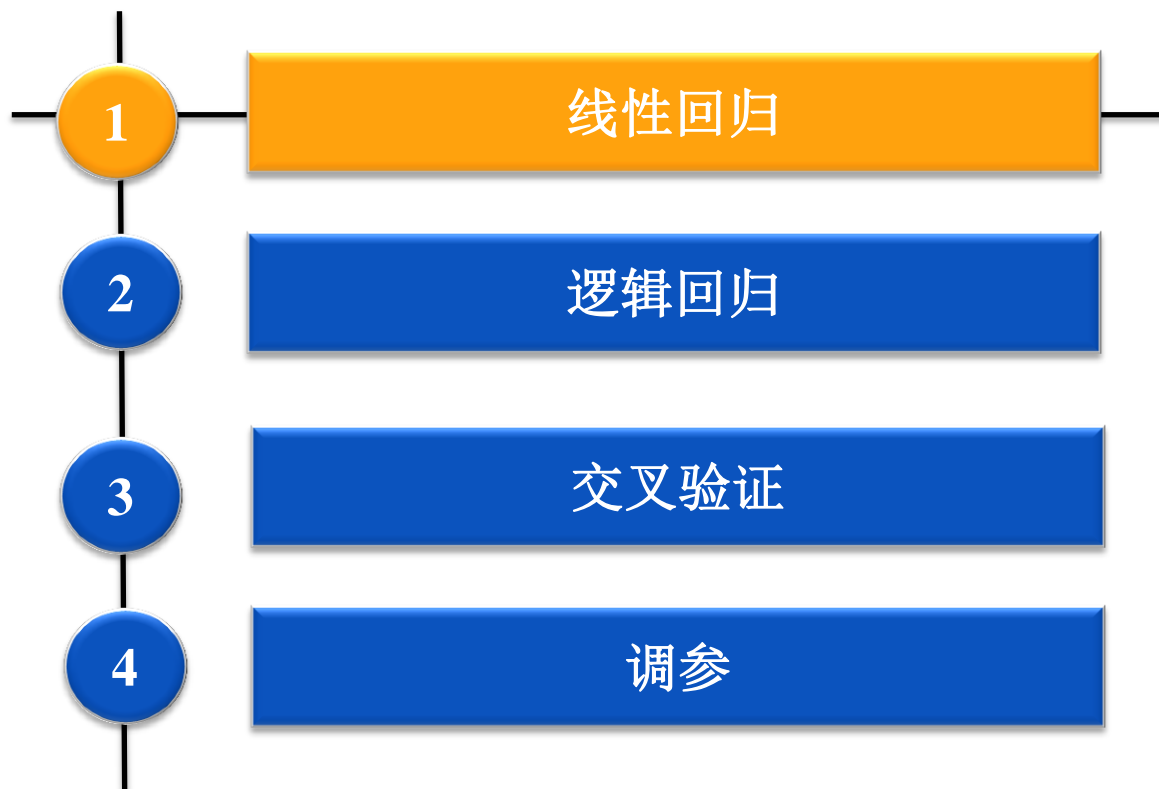
大数据，成就未来

# 回归分析



# 目录

---



# 线性回归

## 一元线性回归模型

➤ 问题：哪一条线对数据的预测更准确/更合适？

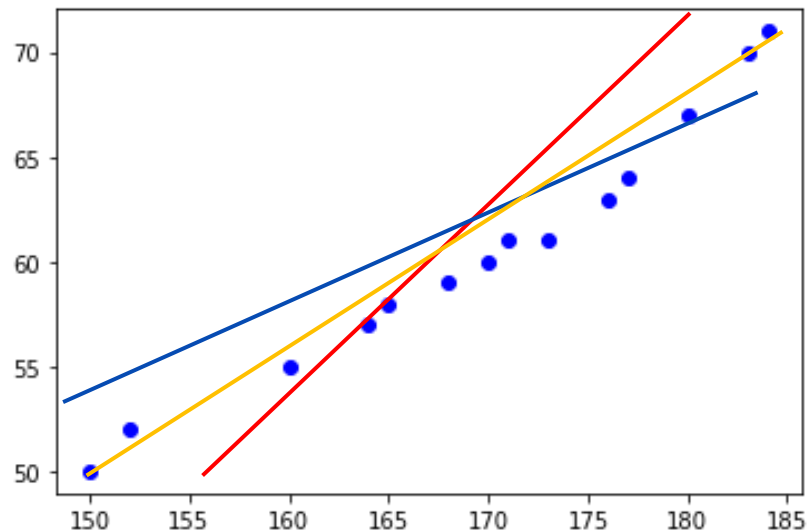
A. 红

B. 黄 ✓

C. 蓝

D. 一样

编号	身高(cm)	体重(kg)
1	150	50
2	152	52
3	160	55
4	164	57
5	165	58
6	168	59
7	170	60
8	171	61
9	173	61
10	176	63
11	177	64
12	180	67
13	183	70
14	184	?



# 线性回归

---

## 一元线性回归模型

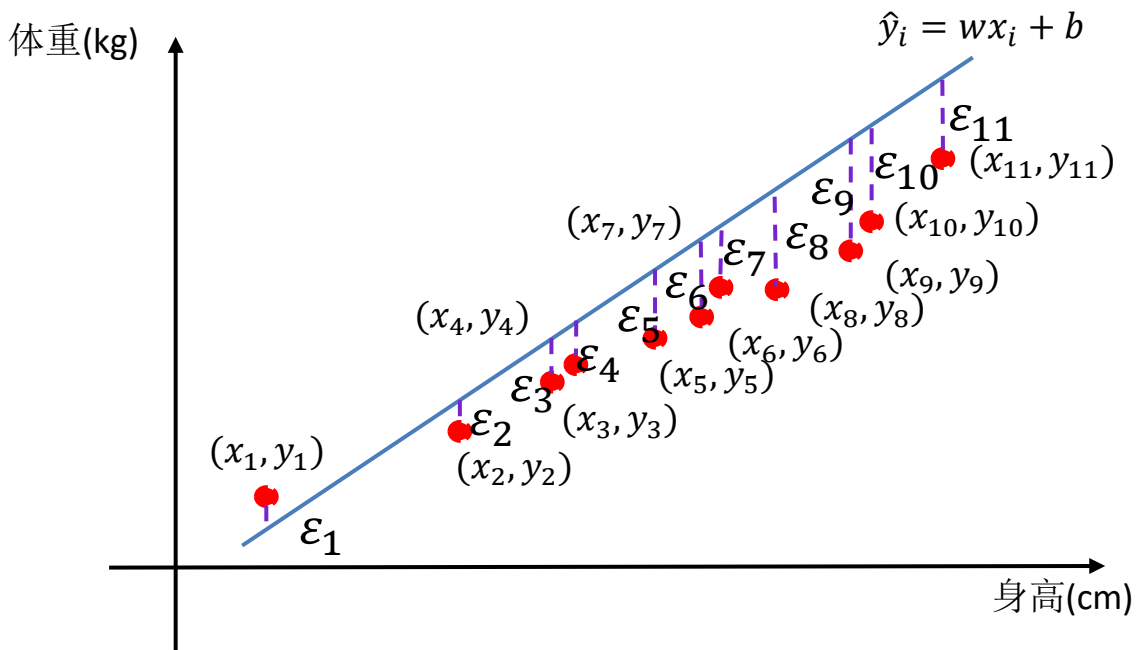
- 线性回归特点：简单、有效、可解释性
- 线性回归的应用：
  - 房价预测
  - 营收预测
  - 销量预测
  - 成绩预测

如何选择预测更准确的直线？

# 线性回归

## 1. 一元线性回归模型——目标函数

➤ 评估标准：离样本点越近的线条越适合，即误差越小越好



$$\hat{y}_i = wx_i + b \quad \text{vs} \quad y_i$$

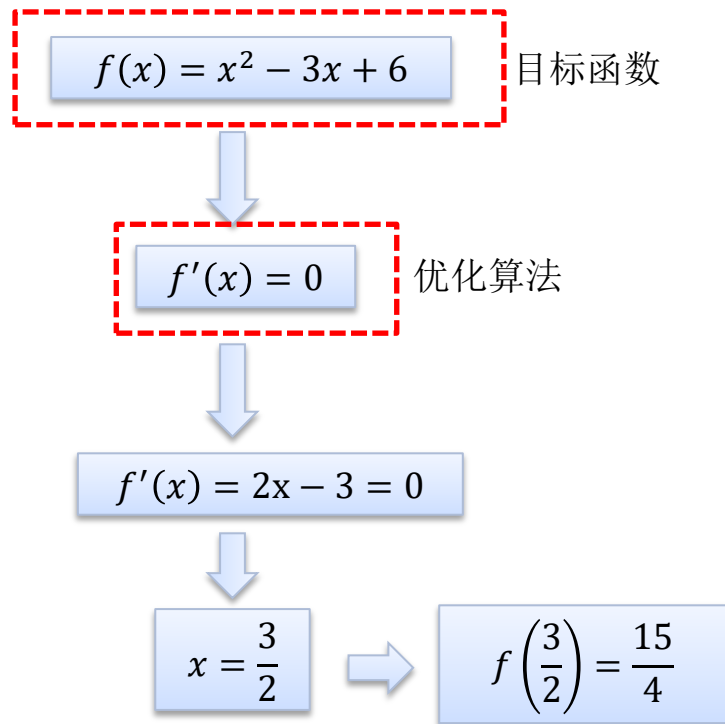
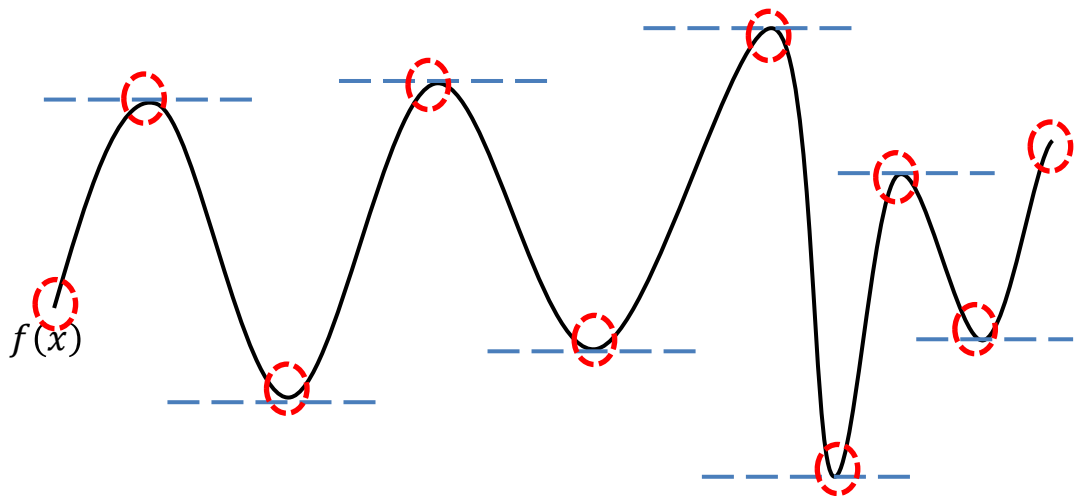
误差最小

$$l(w, b) = \sum_{i=0}^n (\hat{y}_i - y_i)^2 = \sum_{i=0}^n (wx_i + b - y_i)^2$$

# 线性回归

## 一元线性回归模型

- 回顾：一元函数最优解
- 问题：给定一个函数 $f(x)$ , 求出让 $f(x)$ 取最大值或最小值的 $x$



# 线性模型 (linear model)

## 基本形式

线性模型试图学得一个通过属性的线性组合来进行预测（目标属性）的函数

$x_1$	2.0	6.0	5.0	1.0	4.0
$x_2$	7.0	9.0	3.0	2.0	5.0
$y$	52.8	96.7	21.2	6.0	?



$$y = w_1 x_1 + w_2 x_2 + \dots + b$$

$$y = w^T x + b$$

# 线性模型 (linear model)

## 基本形式

- 形式简单，易于建模
- 蕴含机器学习的基本思想
- 是其他非线性模型的基础
- 权重体现出各属性重要性，可解释性强

$x_1$	2.0	6.0	5.0	1.0	4.0
$x_2$	7.0	9.0	3.0	2.0	5.0
$y$	52.8	96.7	21.2	6.0	?

$$y = w^T x + b$$



# 线性模型 (linear model)

---

线性回归

$x$	3	1	7	2	4
$y$	4.5	2.5	8.5	3.5	?

线性回归试图学得  $f(x_i) = wx_i + b$

使得  $f(x_i) \approx y_i$

如何确定  $w, b$  , 关键在于如何减小  $y$  与  $f(x_i)$  的差别

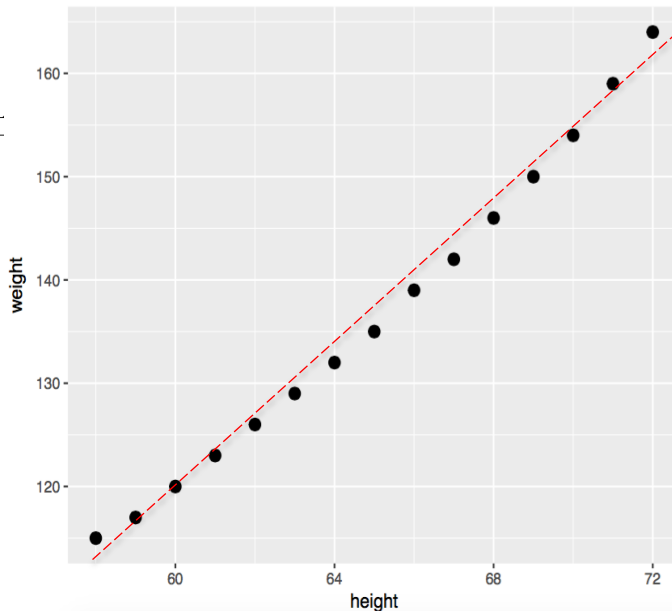
# 线性模型 (linear model)

目标函数 (单变量)

均方误差最小化(最小二乘法)

- 找到一条直线，使所有样本到直线上

$$\min_{i=1}^m \hat{a} (f(x_i) - y_i)^2$$
$$= \min_{i=1}^m \hat{a} (y_i - wx_i - b)^2$$



# 线性模型 (linear model)

目标函数 (单变量)

令目标函数对  $w$  和  $b$  的偏导为零可解得:

$$w = \frac{\sum_{i=1}^m y_i (x_i - \bar{x})}{\sum_{i=1}^m x_i^2 - \frac{1}{m} (\sum_{i=1}^m x_i)^2}$$

$$b = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i)$$

# 回归实现

## 线性回归

sklearn.linear\_model中的LinearRegression可实现线性回归

LinearRegression 的构造方法:

- LinearRegression(  
    fit\_intercept=**True**, #默认值为*True*, 表示 计算随机变量, *False* 表示不计算随机变量  
    normalize=**False**, #默认值为*False*, 表示在回归前是否对回归因子X进行归一化*True* 表示是,  
    copy\_X=**True**  
)
- [scikit-learn中文社区](https://scikit-learn.org.cn/): <https://scikit-learn.org.cn/>

# 回归实现

---

## 线性回归

LinearRegression 的常用方法有：

- `decision_function(X)` #返回  $X$  的预测值  $y$
- `fit(X,y[,n_jobs])` #拟合模型
- `get_params([deep])` #获取 LinearRegression 构造方法的参数信息
- `predict(X)` #求预测值 #同 `decision_function`

# 数据集划分

---

sklearn. model\_selection随机划分训练集和测试集

- train\_test\_split是交叉验证中常用的函数，功能是从样本中随机的按比例选取train data和testdata，形式为：
- `X_train,X_test, y_train, y_test =train_test_split(train_data,train_target,test_size=0.4, random_state=0)`

# 数据集划分

---

`train_test_split`参数解释：

- `train_data`: 所要划分的样本特征集
- `train_target`: 所要划分的样本结果
- `test_size`: 样本占比，如果是整数的话就是样本的数量
- `random_state`: 是随机数的种子。
- 随机数种子：其实就是该组随机数的编号，在需要重复试验的时候，保证得到一组一样的随机数。比如你每次都填1，其他参数一样的情况下你得到的随机数组是一样的。但填0或不填，每次都会不一样。
- 随机数的产生取决于种子，随机数和种子之间的关系遵从以下两个规则：
- 种子不同，产生不同的随机数；种子相同，即使实例不同也产生相同的随机数。

## 回归练习

使用Python实现下面输入与输出的线性回归

输入:  $[[0, 0], [1, 1], [2, 2]]$ ——两个输入

输出:  $[0, 1, 2]$

预测:  $[3, 3]$

```
from sklearn.linear_model import LinearRegression # 导入线性回归模型
import pandas as pd

x = [[0,0], [1,1], [2,2], [3,3]]
x = pd.DataFrame(x)
y = pd.Series([0,1,2])
clf = LinearRegression()
clf.fit(x.iloc[:-1, :], y)
clf.predict(x.iloc[-1:, :])

array([3.])
```



# 回归练习

## 使用Sklearn构建数据模型的万能模版

```
from sklearn. 1、算法位置 import 2、算法名
from sklearn.metrics import accuracy_score #用精确度评估模型
# 生成一个模型对象
3、模型名 = 2、算法名 ( 模型参数【选填】 )

# 训练模型
3、模型名 .fit(train_x,train_y)

# 在训练集上评估模型
pred1 = 3、模型名 .predict(train_x)
accuracy1 = accuracy_score(train_y,pred1)
print('在训练集上的精确度: %.4f'%accuracy1)

# 在测试集上评估模型
pred2 = 3、模型名 .predict(test_x)
accuracy2 = accuracy_score(test_y,pred2)
print('在测试集上的精确度: %.4f'%accuracy2)
```

# 波士顿房价数据集（Boston House Price Dataset）

---

## 数据说明

- 波士顿房价数据集（Boston House Price Dataset）包含对房价的预测，以千美元计，给定的条件是房屋及其相邻房屋的详细信息。
- 该数据集是一个回归问题。每个类的观察值数量是均等的，共有 506 个观察，13 个输入变量和1个输出变量。
- sklearn库的datasets包含该数据集（load\_boston）

# 波士顿房价数据集（Boston House Price Dataset）

---

## 变量名说明：

- **CRIM**: 城镇人均犯罪率。
- **ZN**: 住宅用地超过 25000 sq.ft. 的比例。
- **INDUS**: 城镇非零售商用土地的比例。
- **CHAS**: 查理斯河空变量（如果边界是河流，则为1；否则为0）。
- **NOX**: 一氧化氮浓度。
- **RM**: 住宅平均房间数。
- **AGE**: 1940 年之前建成的自用房屋比例。
- **DIS**: 到波士顿五个中心区域的加权距离。
- **RAD**: 辐射性公路的接近指数。
- **TAX**: 每 10000 美元的全值财产税率。
- **PTRATIO**: 城镇师生比例。
- **B**:  $1000 (B_k - 0.63)^2$ ，其中 **B<sub>k</sub>** 指代城镇中黑人的比例。
- **LSTAT**: 人口中地位低下者的比例。
- **MEDV**: 自住房的平均房价，以千美元计。

# 实现线性回归

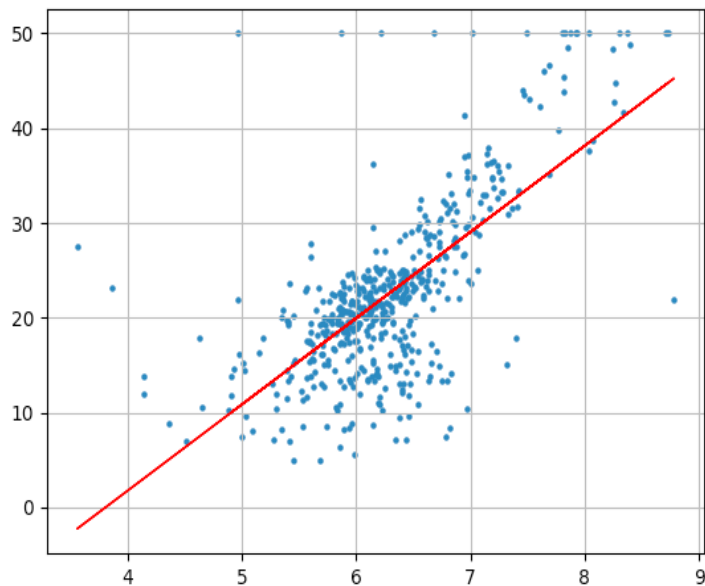
城镇人均犯罪率	.....	住宅平均房间数	.....	与五大就业中心的距离	.....
0.00632	.....	6.575	.....	4.09	.....
0.02731	.....	6.421	.....	4.9671	.....
0.02729	.....	7.185	.....	4.9671	.....
0.06905	.....	7.147	.....	6.0622	.....

直觉告诉我们：上表的第6列（住宅平均房间数）与最终房价一般是成正比的，具有某种线性关系。我们利用线性回归来验证想法。

同时，作为一个二维的例子，可以在平面上绘出图形，进一步观察图形。

# 线性回归例子

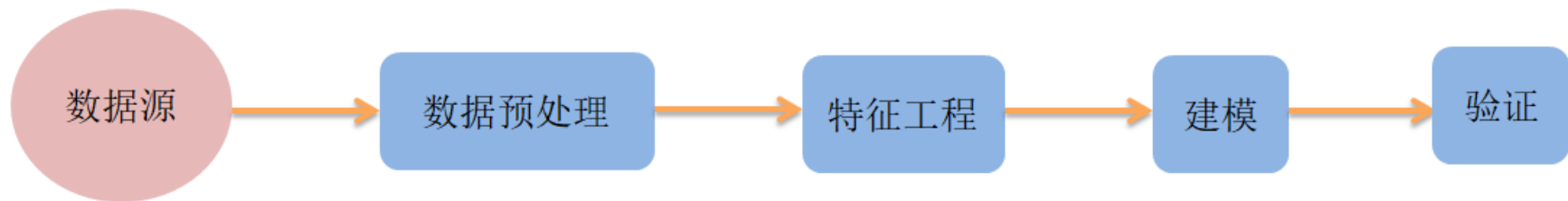
我们利用线性回归来验证想法。如下见图：



# 线性回归例子

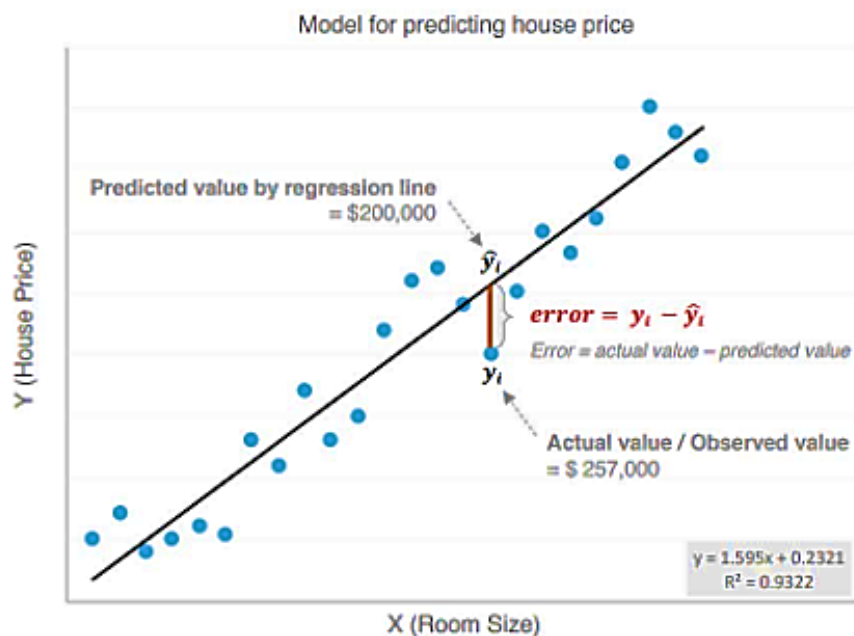
---

## ➤ 整体流程



# 性能度量

## 1. 回归问题评价方法和评价标准



**MAE**

Mean Absolute Error

**MSE**

Mean Squared Error

**RMSE**

Root Mean Squared Error

**MAPE**

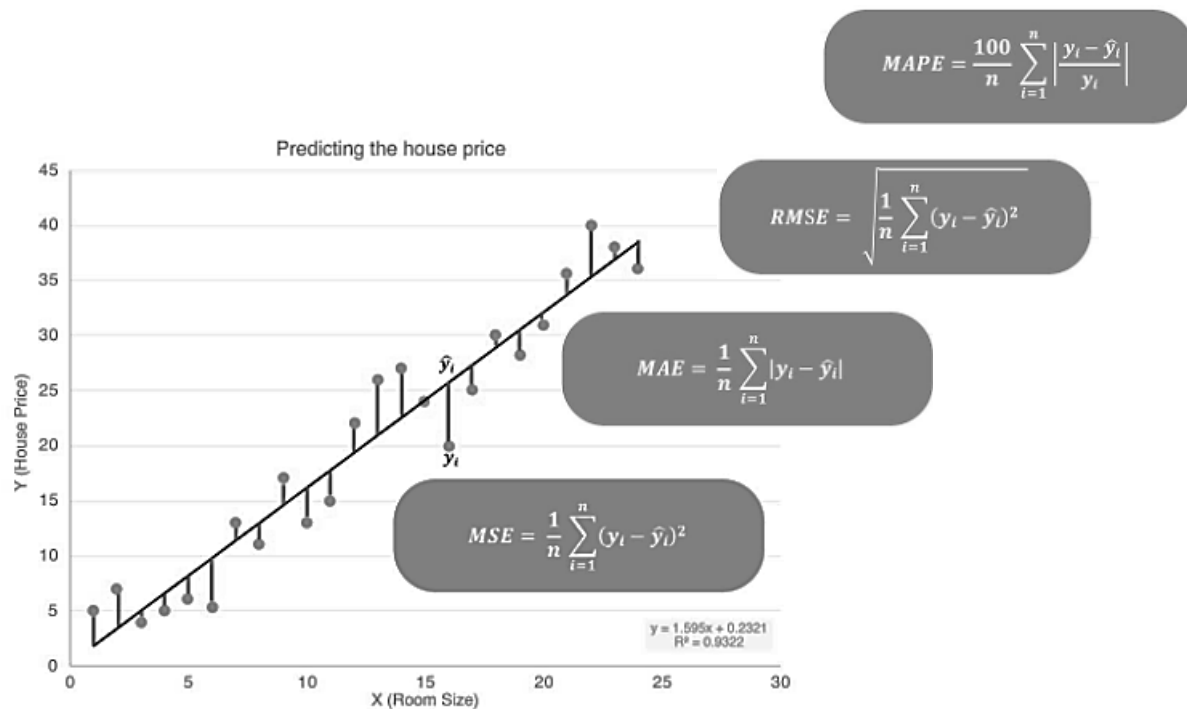
Mean Absolute Percentage Error

**$R^2$**

Coefficient of determination

# 性能度量

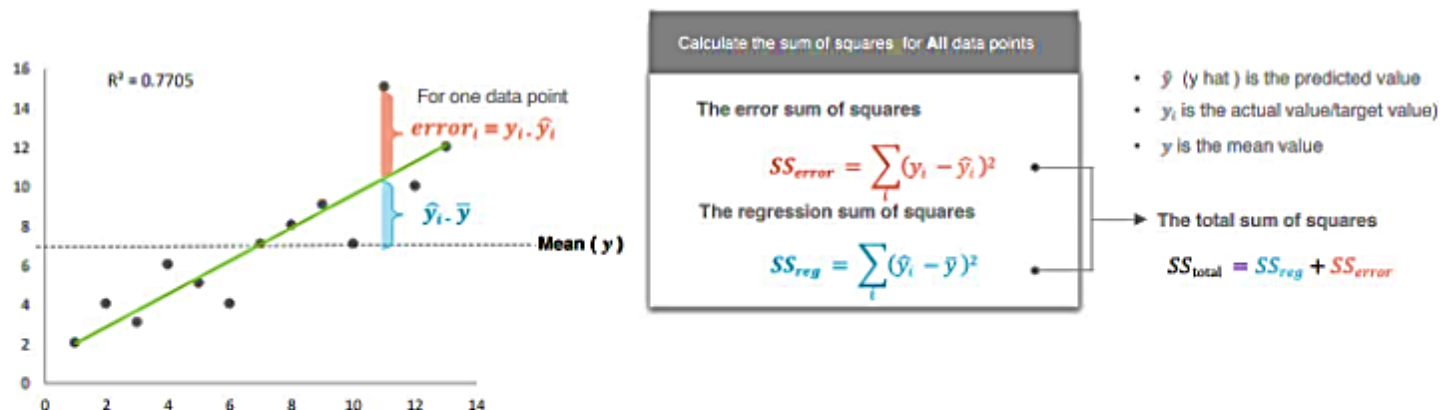
## 回归问题评价方法和评价标准





# 性能度量

## 回归问题评价方法和评价标准



$SS_{reg}$  quantifies how far the estimated sloped regression line  $\hat{y}_i$ , is from the horizontal "no relationship line," the sample mean  $\bar{y}$

$SS_{error}$  quantifies how much the data points  $y_i$  vary around the estimated regression line  $\hat{y}_i$ ,

$SS_{total}$  quantifies how much the data points,  $y_i$  vary around their mean  $\bar{y}$

$R^2$  is the regression sum of squares divided by the total sum of squares <sup>1</sup>.

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

# 多项式回归

- 观察数据，有时数据用一条直线（或者超平面）是不能拟合的，需要用一个多项式表示的曲线（或者超曲面）才能得到更好的拟合结果。
- 线性回归只能处理线性数据，若想解决非线性问题，可以使用多项式回归对线性回归进行改进。
- 让线性回归使用类似于升维的转换，将数据由低维非线性转换为高维线性，从而为线性回归赋予处理非线性数据的能力。

多项式回归可以写成下面这种形式：

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \cdots + \beta_k X_i^k$$



# 多项式回归

- 多项式变化
- 这是一种通过增加自变量上的次数将数据映射到高维空间的方法。只要我们设定一个自变量上的次数(大于1)，就可以相应地获得数据投影在高次方的空间中的结果。
- 简单的举个例子说就是将二维数据转为三维、四维空间数据。这种方法可以非常容易地通过 sklearn 中的 PolynomialFeatures 类来实现。

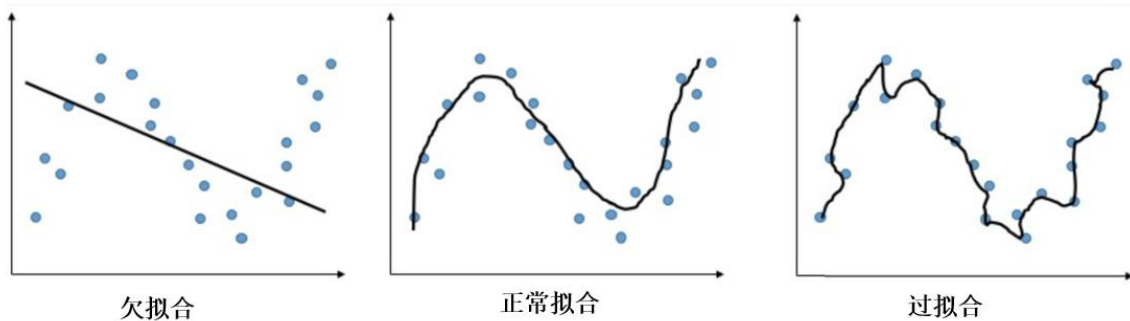
参数	含义
degree	多项式中的次数，默认为2
interaction_only	布尔值是否只产生交互项，默认为False
include_bias	布尔值，是否产出与截距项相乘的 $x_0$ ，默认True

```
array([[ 1.,  1.,  1.,  1.],  
       [ 1.,  2.,  4.,  8.],  
       [ 1.,  3.,  9., 27.]])
```

```
array([[ 1.,  0.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  1.],  
       [ 1.,  2.,  3.,  4.,  6.,  9.,  8., 12., 18., 27.],  
       [ 1.,  4.,  5., 16., 20., 25., 64., 80., 100., 125.]])
```

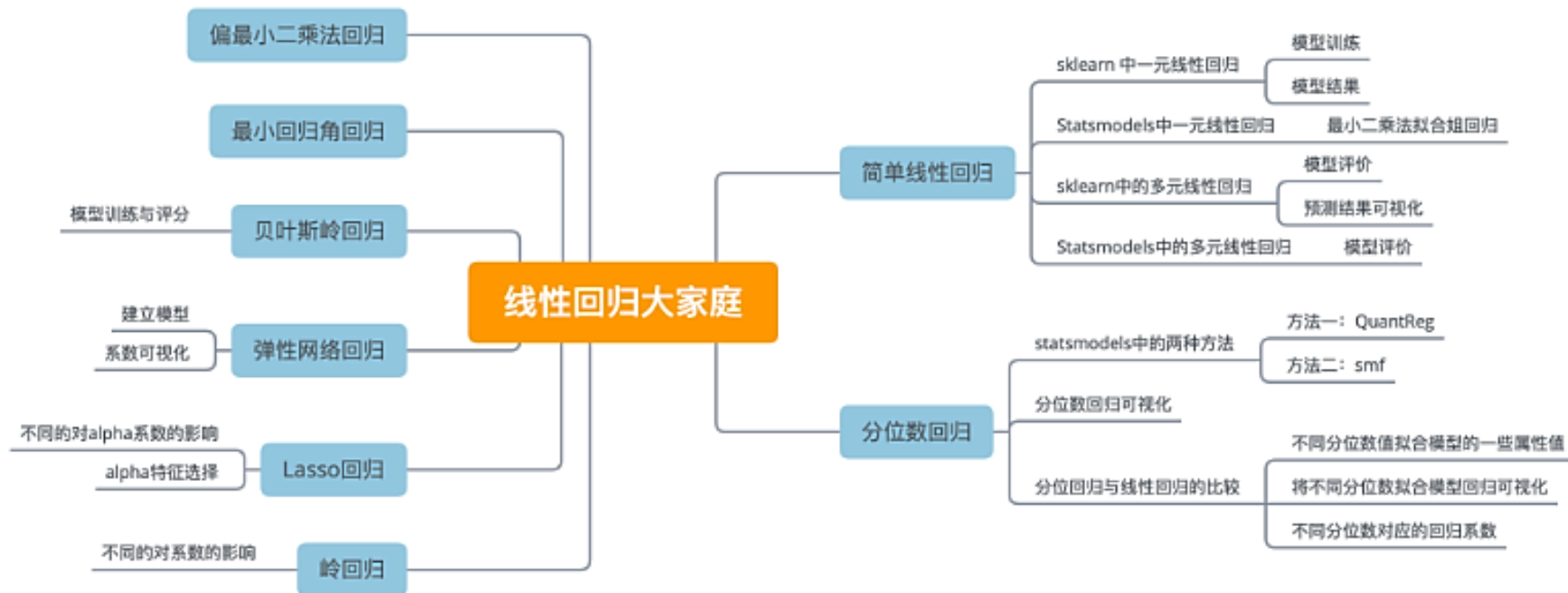
# 多项式回归

- 但是随着多项式的增大，极有可能出现overfitting（过拟合）的情况。



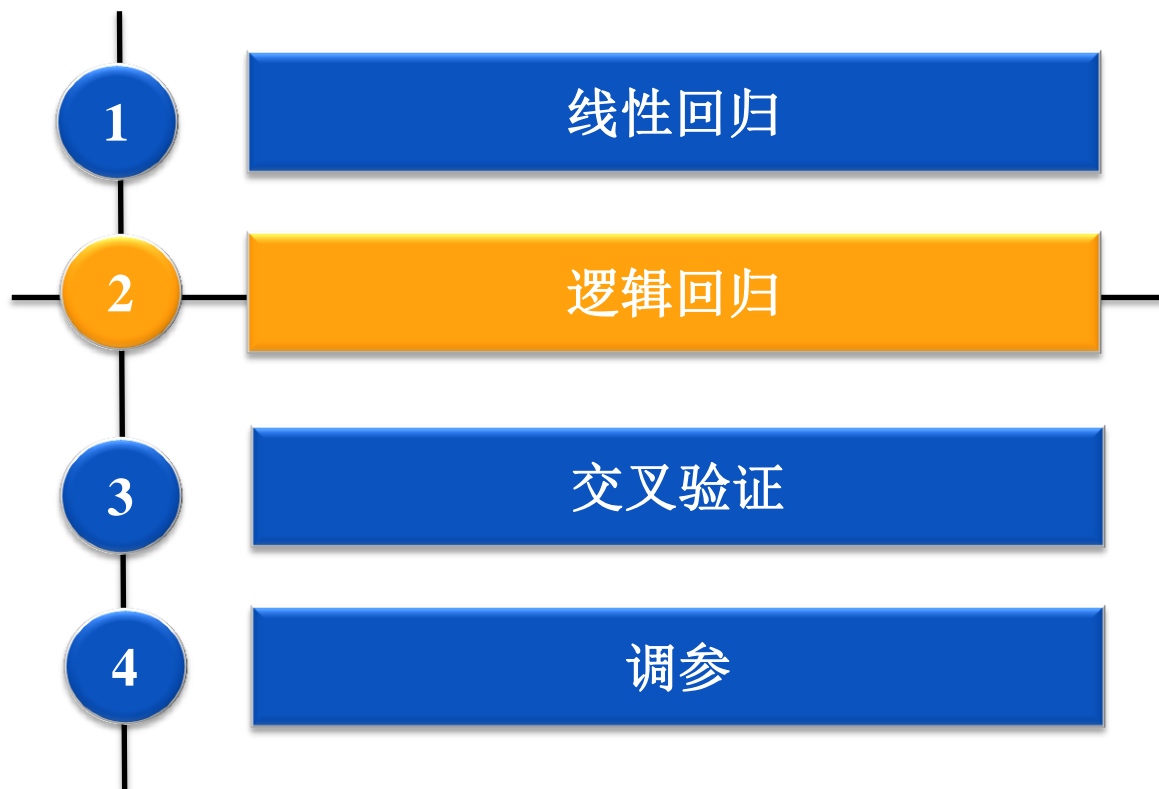
# 其他回归算法

- 除以下线性回归模型外，解决回归问题也可以使用决策树、SVR、KNN等其他算法
- 以下算法请看：八种线性回归算法总结.pdf



# 目录

---



# 逻辑回归

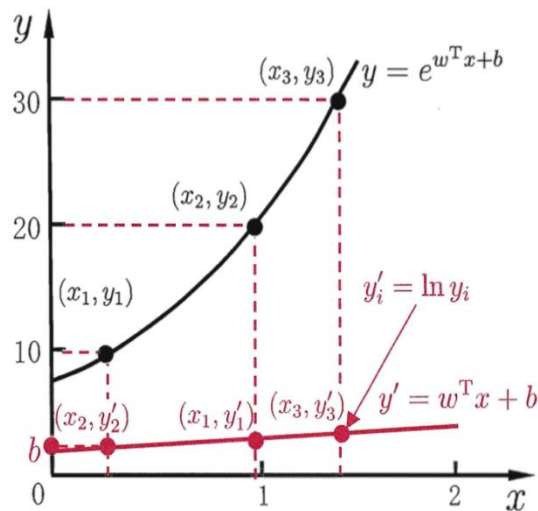
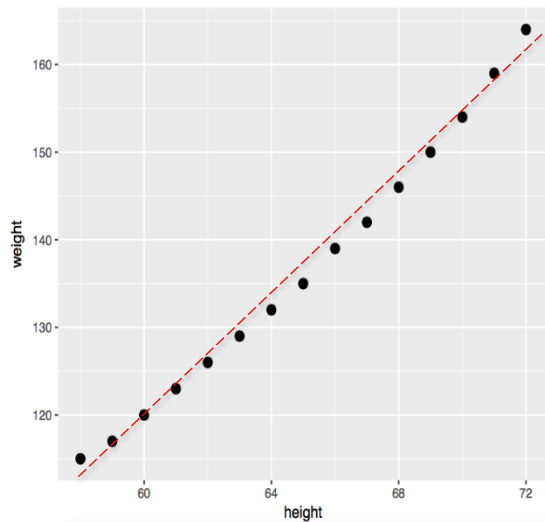
---

- 分类和回归二者不存在不可逾越的鸿沟。就波士顿房价预测作为例子：如果将房价按高低分为“高级”、“中级”和“普通”三个档次，那么这个预测问题也属于分类问题。
- 准确地说，逻辑回归（Logistic Regression）是对数几率回归，属于广义线性模型（GLM），它的因变量一般只有0或1。
- 需要明确一件事情：线性回归并没有对数据的分布进行任何假设，而逻辑回归隐含了一个基本假设：每个样本均独立服从于伯努利分布（0-1分布）。
- 伯努利分布属于指数分布族，这个大家庭还包括：高斯（正态）分布、多项式分布、泊松分布、伽马分布、Dirichlet分布等。

# 逻辑回归

## 对数几率回归 / 逻辑回归 (logistic regression)

- 对数线性回归  $\ln y = w^T x + b$
- 将线性回归模型的预测值和实际值关联起来





# 逻辑回归

## 对数几率回归 / 逻辑回归 (logistic regression)

- 更一般的形式：广义线性模型
- $g(x)$  称为联系函数  $y = g^{-1}(w^T x + b)$
- 二分类问题的理想联系函数：单位阶跃函数

$$y = \begin{cases} 0 & x < 0 \\ 0.5 & x = 0 \\ 1 & x > 0 \end{cases}$$

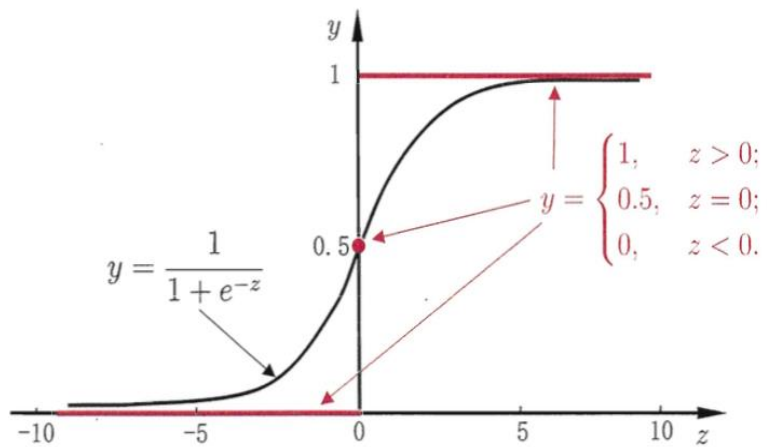
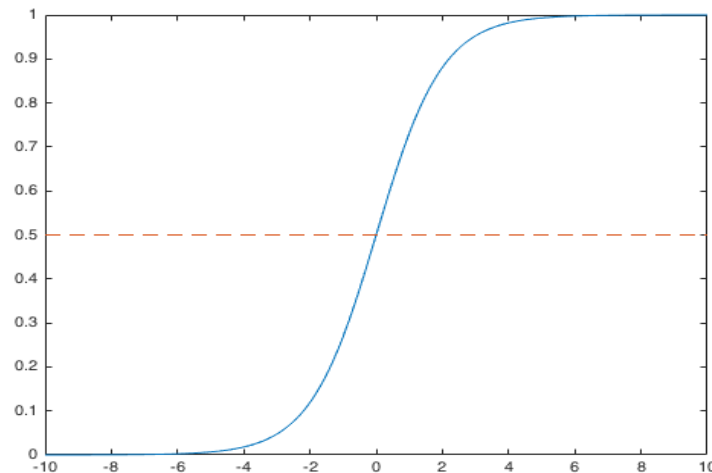


图 3.2 单位阶跃函数与对数几率函数

# 逻辑回归

对数几率回归 / 逻辑回归 (logistic regression)

- 阶跃函数的代替函数: Sigmoid函数  $y = \frac{1}{1 + e^{-z}}$
- 带入线性模型可得  $y = \frac{1}{1 + e^{-(w^T x + b)}}$   $\ln \frac{y}{1 - y} = w^T x + b$
- $\frac{y}{1 - y}$  称为几率, 表示样本取正例的可能性比例,  $\ln \frac{y}{1 - y}$  称为对数几率



# 逻辑回归

对数几率回归 / 逻辑回归 (logistic regression)

目标: 寻找合适的  $w, b$ , 使函数输出逼近真实类别

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \quad \ln \frac{y}{1 - y} = w^T x + b$$

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	T-class
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.5	3	5.8	2.2	?
6.2	2.9	4.3	1.3	?

# 逻辑回归

对数几率回归 / 逻辑回归 (logistic regression)

将 $y$ 视为类别取值为1或0的概率, 可得:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	T-class
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1

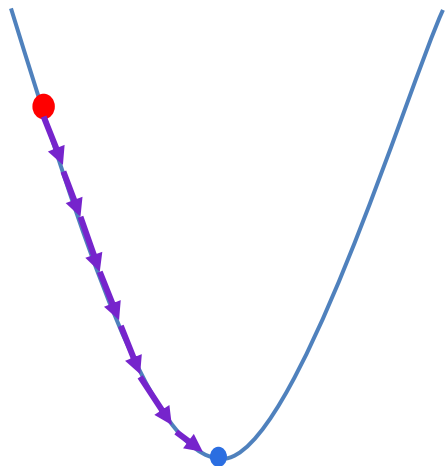
$$\ln \frac{y}{1-y} = w^T x + b \xrightarrow{\text{red dashed arrow}} \ln \frac{p(y=1|x)}{p(y=0|x)} = w^T x + b$$
$$\downarrow \text{red dashed arrow}$$
$$p(y=1|x) = \frac{e^{w^T x + b}}{1 + e^{w^T x + b}} \quad p(y=0|x) = \frac{1}{1 + e^{w^T x + b}}$$

则目标函数变为:  $\max_{w,b} \ell(w,b) = \sum_{i=1}^m \ln p(y_i | x_i; w, b)$

# 逻辑回归

## 逻辑回归的优化算法

- 解析解：最小二乘法
- 迭代法：梯度下降法



$$x^{t+1} = x^t - \eta \nabla f(x^t)$$

# 逻辑回归

对数几率回归 / 逻辑回归 (logistic regression)

目标函数求解方法：梯度下降法、牛顿法

$$\max_{w,b} \ell(w,b) = \sum_{i=1}^m \ln p(y_i | x_i; w, b)$$

- `model = LogisticRegression(penalty='l2', tol=0.0001, C=1.0, fit_intercept=True, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, n_jobs=1)`
- `penalty`: 正则化项的选择。正则化主要有两种：L1和L2，`LogisticRegression`默认选择L2正则化。  
‘liblinear’支持L1和L2，但 ‘newton-cg’, ‘sag’ 和 ‘lbfgs’ 只支持L2正则化。
- `tol`: float, default: 1e-4, 停止标准，误差不超过tol时，停止进一步的计算。

# 逻辑回归

## 对数几率回归 / 逻辑回归 (logistic regression)

### ➤ 参数说明

- **C**: float, default: 1.0, 正则化强度 (正则化系数 $\lambda$ ) 的倒数; 必须是大于0的浮点数。与支持向量机一样, 较小的值指定更强的正则化, 通常默认为1。
- **fit\_intercept**: bool (True、False), default: True, 是否存在截距, 默认存在。
- **random\_state**: int, RandomState instance or None, optional, default: None, 生成器的种子
- **solver**: 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga', default: liblinear
  - liblinear: 坐标轴下降法来迭代优化损失函数。
  - lbfgs: 拟牛顿法的一种, 利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。
  - newton-cg: 也是牛顿法家族的一种, 利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。
  - sag: 即随机平均梯度下降, 是梯度下降法的变种。
  - saga: 线性收敛的随机优化算法。
- **max\_iter**: int, default: 100, 仅适用于newton-cg, sag和lbfgs求解器。求解器收敛的最大迭代次数。

# 逻辑回归

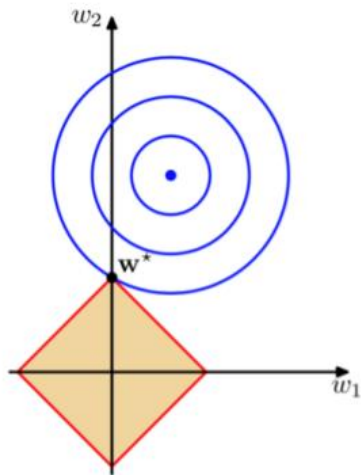
## L1、L2正则化

- 正则化是用来防止模型过拟合的过程，常用的有L1正则化和L2正则化两种选项，分别通过在损失函数后加上参数向量 $\theta$ 的L1范式和L2范式的倍数来实现。这个增加的范式，被称为“正则项”，也被称为“惩罚项”。
- 损失函数改变，基于损失函数的最优化来求解的参数取值必然改变，我们以此来调节模型拟合的程度。
- 原损失： $\min 1/N * \sum_{i=1}^N (y_i - \omega^T x_i)^2$  式子 (1)
- 加L1正则项： $\min 1/N * \sum_{i=1}^N (y_i - \omega^T x_i)^2 + C \|\omega\|_1$  式子 (2)
- 加L2正则项： $\min 1/N * \sum_{i=1}^N (y_i - \omega^T x_i)^2 + C \|\omega\|_2^2$  式子 (3)



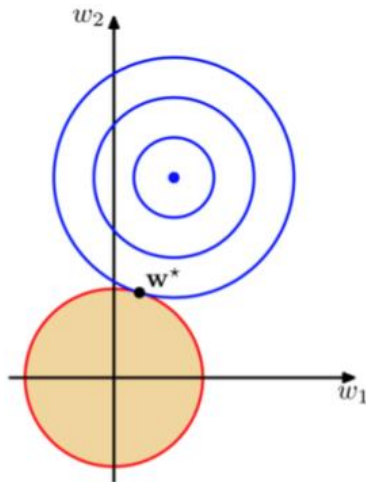
# 逻辑回归

## L1、L2正则化



$L_1$ 正则化是指在损失函数中加入权值向量 $w$ 的绝对值之和,  $L_1$ 的功能是使权重稀疏

### $L_1$ 正则化可以产生稀疏模型



在损失函数中加入权值向量 $w$ 的平方和,  $L_2$ 的功能是使权重平滑。

### $L_2$ 正则化可以防止过拟合

图上面中的蓝色轮廓线是没有正则化损失函数的等高线, 中心的蓝色点为最优解, 左图、右图分别为 $L_1$ 、 $L_2$ 正则化给出的限制。

可以看到在正则化的限制之下,  $L_2$ 正则化给出的最优解 $w^*$ 是使解更加靠近原点, 也就是说 $L_2$ 正则化能降低参数范数的总和。

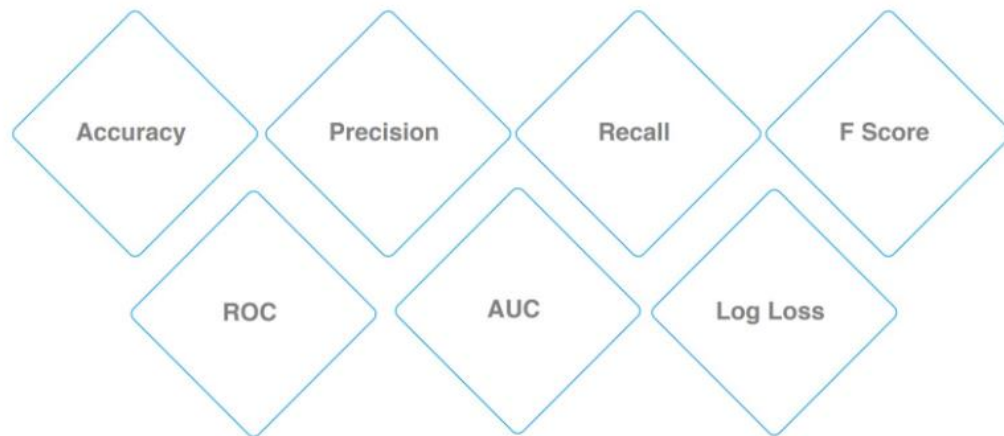
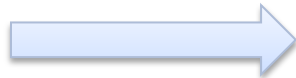
$L_1$ 正则化给出的最优解 $w^*$ 是使解更加靠近某些轴, 而其它的轴则为0, 所以 $L_1$ 正则化能使得到的参数稀疏化。

# 性能度量

## 1. 分类任务评价方法和评价标准

样本ID	Real	pre
1	男	男
2	女	女
3	男	男
4	男	女
5	女	女

结果如何评价?



# 性能度量

## 1. 分类任务评价方法和评价标准

➤ 错误率:  $E(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(x_i) \neq y_i) = \frac{FP+FN}{ALL}$

➤ 精度:  $acc(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(x_i) = y_i) = 1 - E(f; D) = \frac{TP+TN}{ALL}$

样本ID	Real	pre
1	男	男
2	女	女
3	男	男
4	男	女
5	女	女



错误率:  $E(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(x_i) \neq y_i) = \frac{1}{5}$

精度:  $acc(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(x_i) = y_i) = 1 - E(f; D) = \frac{4}{5}$

# 性能度量

## 2. 查准率与查全率

- 真正例 (True Positive, TP) : 预测为正例, 且正确
- 假正例 (False Positive, FP) : 预测为正例, 但错误
- 真反例 (True Negative, TN) : 预测为反例, 且正确
- 假反例 (False Negative, FN) : 预测为反例, 但错误

➤ 查准率P (精确率) :  $\frac{TP}{TP+FP}$ , 从所有预测为正例的样本中查出真的为正例的比例

➤ 查全率R (召回率) :  $\frac{TP}{TP+FN}$ , 从所有真的为正例的样本中查出预测为正例的比例

样本ID	Real	Pre
1	男	男
2	女	男
3	男	男
4	男	女
5	女	女

TP = 2  
FP = 1  
TN = 1  
FN = 1

真实情况	预测结果	
	正例	反例
正例	2	1
反例	1	1

混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP(真正例)	FN(假反例)
反例	FP(假正例)	TN(真反例)

混淆矩阵

查准率P = 0.667  
查全率R = 0.667

## 3. F1度量

- 综合查准率和查全率:  $F1 = \frac{2 \times P \times R}{P + R}$
- F1-score默认计算二分类的结果。Macro-F1和Micro-F1是相对于多标签分类而言的。
- Micro-F1, 计算出所有类别总的Precision和Recall, 然后计算F1。
- Macro-F1, 计算出每一个类的Precision和Recall后计算F1, 最后将F1平均。
- Weighted-F1, 综合考虑各类样本的权重, 计算加权的F1。




## 4. n个混淆矩阵上综合考察查准率和查全率

➤ macro-f1 是对每类F1分数计算算术平均值，该方法平等地对待所有类，而不考虑不同类别的重要性。

➤ 宏查准率:  $\text{macro-P} = \frac{1}{n} \sum_{i=1}^n P_i$

➤ 宏查全率:  $\text{macro-R} = \frac{1}{n} \sum_{i=1}^n R_i$

$$\text{宏F1: macro-F1} = \frac{2 \times (\text{macro-P}) \times (\text{macro-R})}{(\text{macro-P}) + (\text{macro-R})}$$

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
 Airplane	2	1	1	0.67	0.67	$2 * (0.67 * 0.67) / (0.67 + 0.67) = \mathbf{0.67}$
 Boat	1	3	0	0.25	1.00	$2 * (0.25 * 1.00) / (0.25 + 1.00) = \mathbf{0.40}$
 Car	3	0	3	1.00	0.50	$2 * (1.00 * 0.50) / (1.00 + 0.50) = \mathbf{0.67}$

# 性能度量




## 4. n个混淆矩阵上综合考察查准率和查全率

- micro-f1通过计算全局的TP、FN和FP来计算F1分数。先将所有类别的TP、FP和FN值相加，然后将它们代入F1方程中，得到micro-f1分数。

- 微查准率:  $\text{micro-P} = \frac{\overline{TP}}{\overline{TP} + \overline{FP}}$

$$\text{微F1: micro-F1} = \frac{2 \times (\text{micro-P}) \times (\text{micro-R})}{(\text{micro-P}) + (\text{micro-R})}$$




- 微查全率:  $\text{micro-R} = \frac{\overline{TP}}{\overline{TP} + \overline{FN}}$

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Micro-Averaged F1 Score
 Airplane	2	1	1	$\frac{TP}{TP + \frac{1}{2}(FP + FN)} = \frac{6}{6 + \frac{1}{2}(4 + 4)}$ $= 0.60$
 Boat	1	3	0	
 Car	3	0	3	
TOTAL	6	4	4	

# 性能度量

## 4. n个混淆矩阵上综合考察查准率和查全率

- weighted-f1考虑了不同类别的重要性，也就是把每个类别的样本数量作为权重，计算加权f1。

Label	Per-Class F1 Score	Support	Support Proportion	Weighted Average F1 Score
 Airplane	0.67	3	0.3	$(0.67 * 0.3) + (0.40 * 0.1) + (0.67 * 0.6) = \mathbf{0.64}$
 Boat	0.40	1	0.1	
 Car	0.67	6	0.6	
Total	-	10	1.0	



## 4. 代码演示[https://scikit-learn.org/stable/modules/model\\_evaluation.html#regression-metrics](https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics)

### ➤ 0. 导入必要模块:

```
from sklearn.metrics import precision_score  
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import classification_report
```

### ➤ 1. 构建数据:

```
y_true = [1, 2, 1, 1, 2] # 样本实际值
```

```
y_pred = [1, 1, 1, 2, 2] # 模型预测值
```

### ➤ 2. 计算准确率: `res = precision_score(y_true, y_pred, average=None)` # 准确率

### ➤ 3. 计算混淆矩阵: `res = confusion_matrix(y_true, y_pred)`

### ➤ 4. 生成模型评估报告: `res = classification_report(y_true, y_pred)`

## 5. ROC曲线 和 AUC值（一般用于二分类）

- 假阳性率（False Positive Rate, FPR）是指在所有实际为负例的样本中，模型错误地预测为正例的样本比例。假阳性率可以理解为所有阴性群体中被检测出来阳性的比率(误诊率)，因此FPR越接近0越好。它的计算公式如下：

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

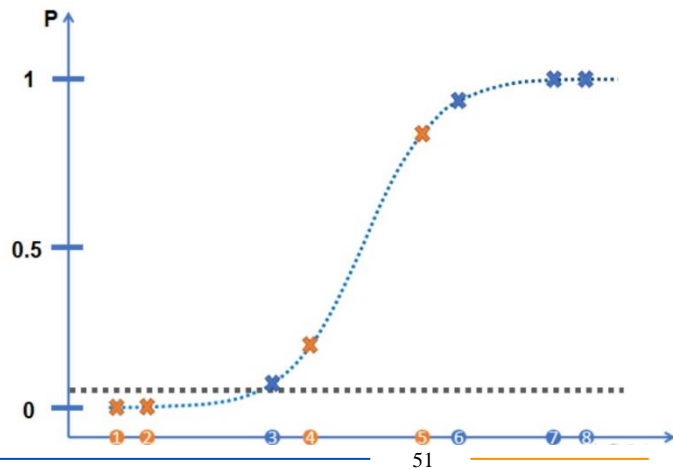
- 真阳性率（True Positive Rate, TPR）通常也被称为敏感性（Sensitivity）或召回率（Recall）。它是指分类器正确识别正例的能力。真阳性率可以理解为所有阳性群体中被检测出来的比率(1-漏诊率)，因此TPR越接近1越好。它的计算公式如下：

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

## 5. ROC曲线 和 AUC值

- ROC曲线和AUC值是用于评估二分类模型性能的两个重要指标。通过ROC曲线，可以直观地了解分类器在不同阈值下的性能；通过AUC值，可以对分类器的整体性能进行量化评估。
- ▣ ROC曲线（Receiver Operating Characteristic Curve）是一种描绘分类器性能的图形，它显示了在不同阈值下分类器的真阳性率（True Positive Rate, TPR）和假阳性率（False Positive Rate, FPR）之间的关系。
- ▣ AUC（Area Under the Curve）值表示ROC曲线下的面积，用于衡量分类器性能。AUC值越接近1，表示分类器性能越好；反之，AUC值越接近0，表示分类器性能越差。

	阈值								
	0.01	0.02	0.1	0.2	0.8	0.9	0.99	0.999	1
真阳性率TPR	1	1	1	0.75	0.75	0.75	0.5	0.25	0
假阳性率FPR	1	0.75	0.5	0.5	0.25	0	0	0	0

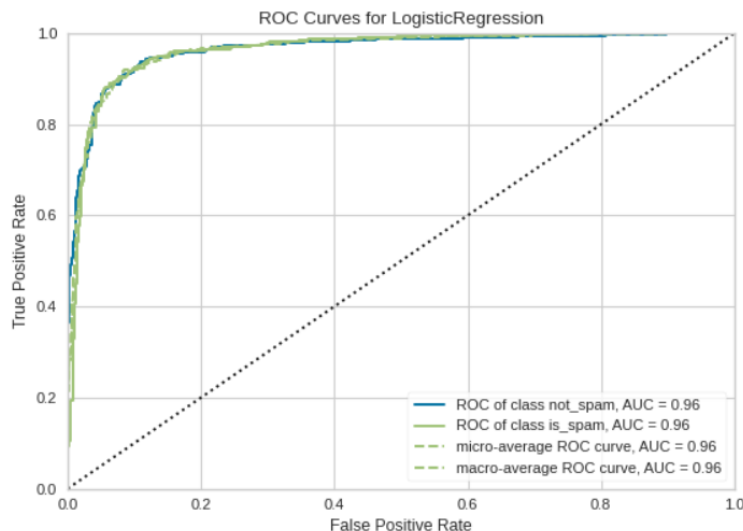


## 5. ROC曲线 和 AUC值

- ROC曲线所在的二维坐标轴空间的横轴为FPR，纵轴为TPR；其中

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$



- 相对来说，ROC曲线越靠近左上角表示分类器的效果越好

## 5. ROC曲线 和 AUC值

➤ `from sklearn.metrics import roc_curve` # 用于计算fpr, tpr

▣ `roc_curve`函数返回三个值:

1. fpr: 假阳性率 (False Positive Rate)
2. tpr: 真阳性率 (True Positive Rate)
3. thresholds: 计算得到的阈值

➤ **from** sklearn.metrics **import** roc\_auc\_score # 用于计算AUC面积

## 6. PR曲线

- PR 曲线在报告信息检索结果时特别有用。
- ▣ 例如，假设用户输入搜索查询“人工智能”。搜索引擎浏览数以百万计的文档去检索少量相关文档。与非相关文件数量相比，相关文件数量将非常少。
- ▣ 在这种情况下：
  - ✓  $TP$  = 实际相关的检索文档数（良好的结果）。
  - ✓  $FP$  = 检索到的实际上不相关的文档（虚假搜索结果）的数量。
  - ✓  $TN$  = 实际上不相关的未检索文档的数量。
  - ✓  $FN$  = 实际相关的未检索文档的数量（我们错过的好文档）。

## 6. PR曲线

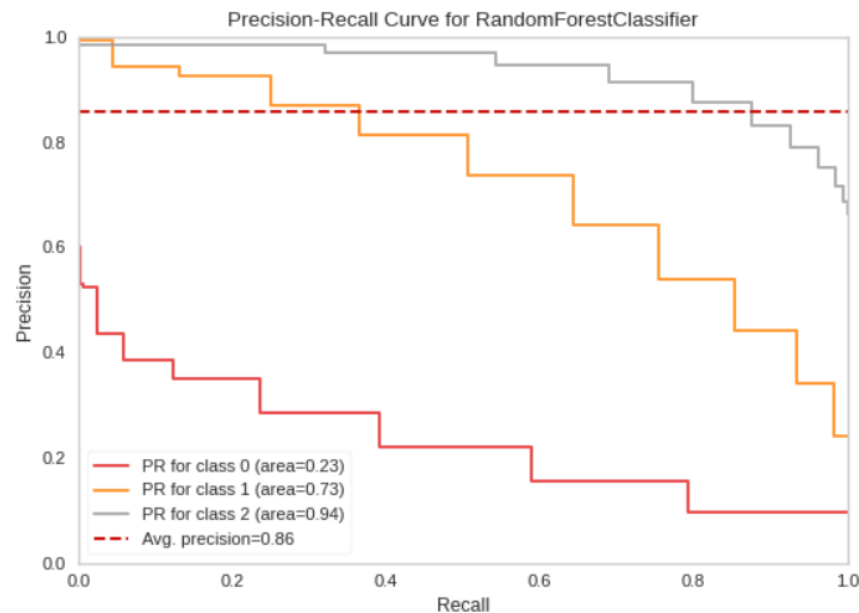
- ROC 曲线是包含 x 轴上的  $\text{Recall} = \text{TPR} = \text{TP}/(\text{TP}+\text{FN})$  和 y 轴上的  $\text{FPR} = \text{FP}/(\text{FP}+\text{TN})$  的图。
- 由于真阴性（即实际上不相关的未检索文档）数量大，FPR 变得微不足道。此外，FPR 并不能真正帮助我们很好地评估检索系统，因为我们希望更多地关注检索到的文档，而不是未检索到的文档。
- PR曲线有助于解决这个问题。PR 曲线在 x 轴上具有召回值 (TPR)，在 y 轴上具有精度  $= \text{TP}/(\text{TP}+\text{FP})$ 。精度有助于突出检索结果的相关性。

## 6. PR曲线

- 要评价信息检索系统的性能水平，就必须在一个检索系统中进行多次检索。每进行一次检索（每改变一次阈值后重新对样本进行分类），都计算其查准率和查全率，并以此作为坐标值，在平面坐标图上标示出来。通过大量检索，可以得到检索系统的性能曲线。
- PR曲线中的P代表的是Precision（精准率），R代表的是Recall（召回率），其代表的是精准率与召回率的关系，一般情况下，Precision设置为纵坐标，Recall设置为横坐标。

$$Precision = \frac{TP}{TP + FP}$$

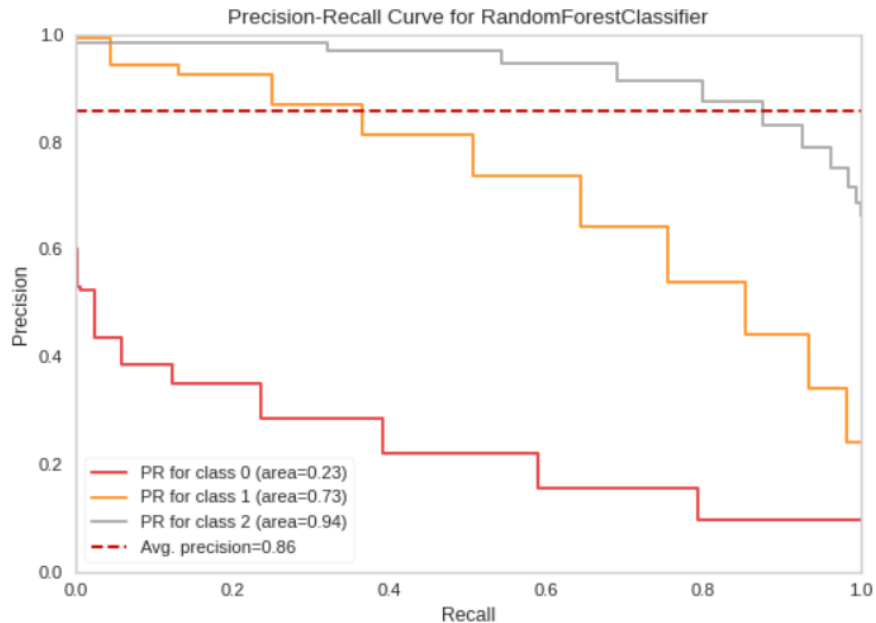
$$Recall = \frac{TP}{TP + FN}$$





## 6. PR曲线

- 对来说，PR曲线越靠近右上角，PR曲线下的面积越大效果越好。
- 在查全率和查准率之间存在着相反的相互依赖关系-如果提高输出的查全率，就会降低其查准率。



## PR曲线和ROC曲线的应用场景

- ROC曲线主要应用于**测试集中的样本分布的较为均匀的情况**，且当测试集中的正负样本的分布变化的时候，ROC曲线能够保持不变。这也是ROC曲线一个很好的特性。
- 但ROC曲线在出现类不平衡现象的数据集中时，即负样本比正样本多很多（或者相反），而且测试数据中的正负样本的分布也可能随着时间变化。ROC曲线是不敏感的，其曲线能够基本保持不变。
- **ROC的面对不平衡数据的一致性表明其能够衡量一个模型本身的预测能力**，而这个预测能力是与样本正负比例无关的。但是这个不敏感的特性使得其较难以看出一个模型在面临样本比例变化时模型的预测情况。此时ROC曲线最大的优点在面对不平衡数据集时便成为了它最大的一个缺点。

## PR曲线和ROC曲线的应用场景

- **PR曲线因为对样本比例敏感**，因此能够看出分类器随着样本比例变化的效果，而实际中的数据又是不平衡的，这样有助于了解分类器实际的效果和作用，也能够以此进行模型的改进。
- 在面对出现类不平衡现象数据集时，可以根据**PR曲线表现出来的结果衡量一个分类器面对不平衡数据进行分类时的能力**，从而进行模型的改进和优化。

## 如何选择PR曲线和ROC曲线？

- 在很多实际问题中，正负样本数量往往很不均衡。比如，计算广告领域经常涉及转化率模型，正样本的数量往往是负样本数量的1/1000，甚至1/10000。若选择不同的测试集，P-R曲线的变化就会非常大，而ROC曲线则能够更加稳定地反映模型本身的好坏。所以，**ROC曲线的适用场景更多**，被广泛用于排序、推荐、广告等领域。
- 但需要注意的是，选择P-R曲线还是ROC曲线是因实际问题而异的，如果希望更多地看到模型在特定数据集上的表现，P-R曲线则能够更直观地反映其性能。
- PR曲线比ROC曲线更加关注正样本，而ROC则兼顾了两者。AUC越大，反映出正样本的预测结果更加靠前。（推荐的样本更能符合用户的喜好）
- 当正负样本比例失调时，比如正样本1个，负样本100个，则ROC曲线变化不大，此时用PR曲线更加能比较出两个分类器性能的好坏。

## 总结

- 以上评估指标都可以用作评价标准，但有时候模型没有单纯的谁比谁好，选择模型还是要结合具体的使用场景。
- ▣ 1. 地震的预测
  - ✓ 对于地震的预测，我们希望的是RECALL非常高，也就是说每次地震我们都希望预测出来。这个时候我们可以牺牲PRECISION。情愿发出1000次警报，把10次地震都预测正确了；也不要预测100次对了8次漏了两次。
- ▣ 2. 嫌疑人定罪
  - ✓ 基于不错怪一个好人的原则，对于嫌疑人的定罪我们希望是非常准确的。及时有时候放过了一些罪犯（recall低），但也是值得的。

# 数据说明

通过分析不同的因素对研究生录取的影响来预测一个人是否会被录取。

数据的格式如下：

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3

- admit：表示是否被录取(目标变量)
- gre：标准入学考试成绩，预测变量
- gpa：学业平均绩点，预测变量
- rank：母校排名(预测变量)
- ——LogisticRegression.csv文件

# 算法实现

---

在入门时建议首先掌握scikit-learn中的逻辑回归实现算法。

算法实现代码如下：

- `import pandas as pd`
- `from sklearn.linear_model import LogisticRegression`
- `from sklearn.model_selection import train_test_split`
- `# 导入数据并观察`
- `data = pd.read_csv('LogisticRegression.csv')`
- `print(data.tail(5))` # 查看数据框的最后五行

# 算法实现

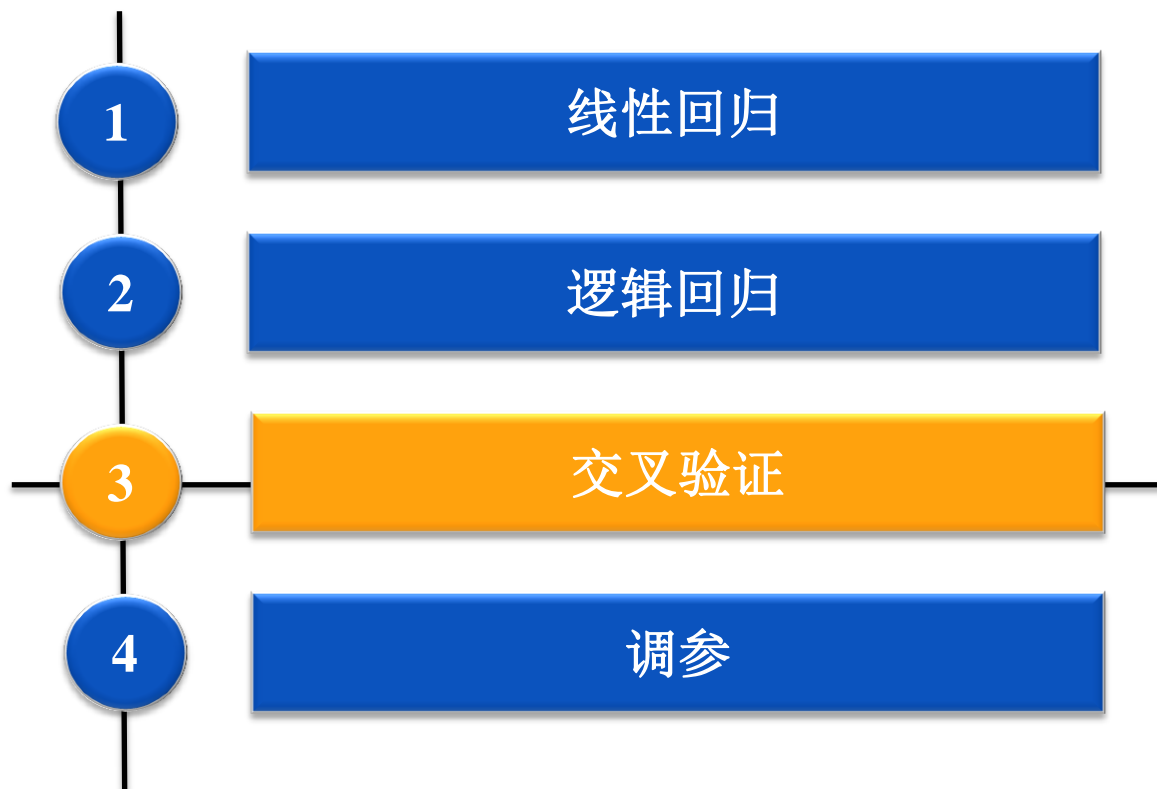
---

- # 切分训练集和测试集
- `X_train,X_test,y_train,y_test = train_test_split(data.iloc[:, 1:],data.iloc[:,0],test_size=.1,random_state=5)`
- `lr = LogisticRegression()` # 建立LR模型
- `lr.fit(X_train,y_train)` # 用处理好的数据训练模型
- `print('逻辑回归的准确率为: {0:.2f}%'.format(lr.score(X_test, y_test)*100))`



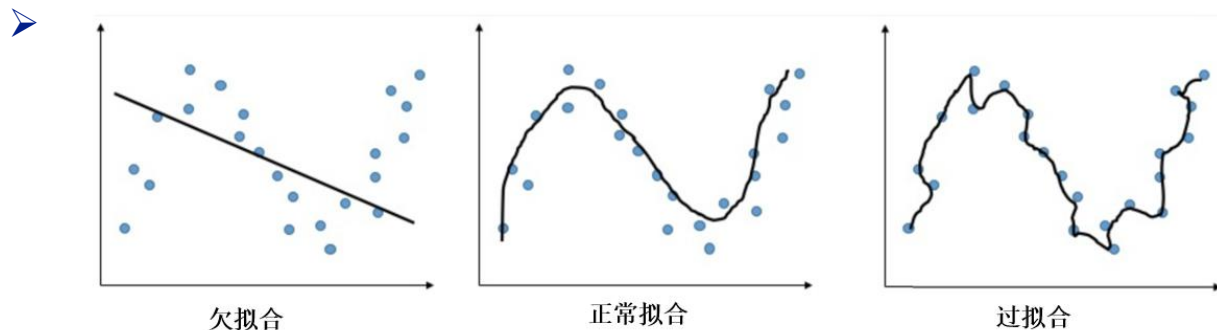
# 目录

---



# 交叉验证

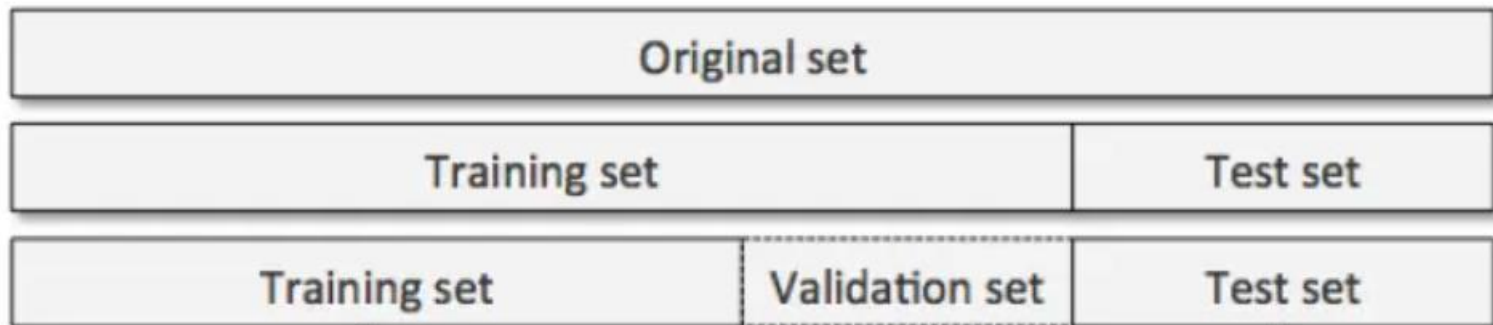
- 学习预测函数的参数，并在相同数据集上进行测试是一种错误的做法：一个仅给出测试用例标签的模型将会获得极高的分数，但对于尚未出现过的数据它则无法预测出任何有用的信息。这种情况称为 **overfitting**（过拟合）。
- 交叉验证的基本思想是把在某种意义下将原始数据(dataset)进行分组，一部分做为训练集(train set)，另一部分做为验证集(validation set or test set)，首先用训练集对分类器进行训练,再利用验证集来测试训练得到的模型(model)，以此来做为评价分类器的性能指标。
- 总的来说：交叉验证是一种预测模型拟合性能的方法。



# 交叉验证

## Holdout 验证

- 将原始数据随机分为两组，一组做为训练集，一组做为验证集，利用训练集训练分类器，然后利用验证集验证模型，记录最后的分类准确率为此分类器的性能指标。
- 在这种交叉验证技术中，整个数据集被随机划分为训练集和验证集。根据经验，整个数据集的近 70% 用作训练集，其余 30% 用作验证集。即sklearn中的`train_test_split`方法。



# 交叉验证

---

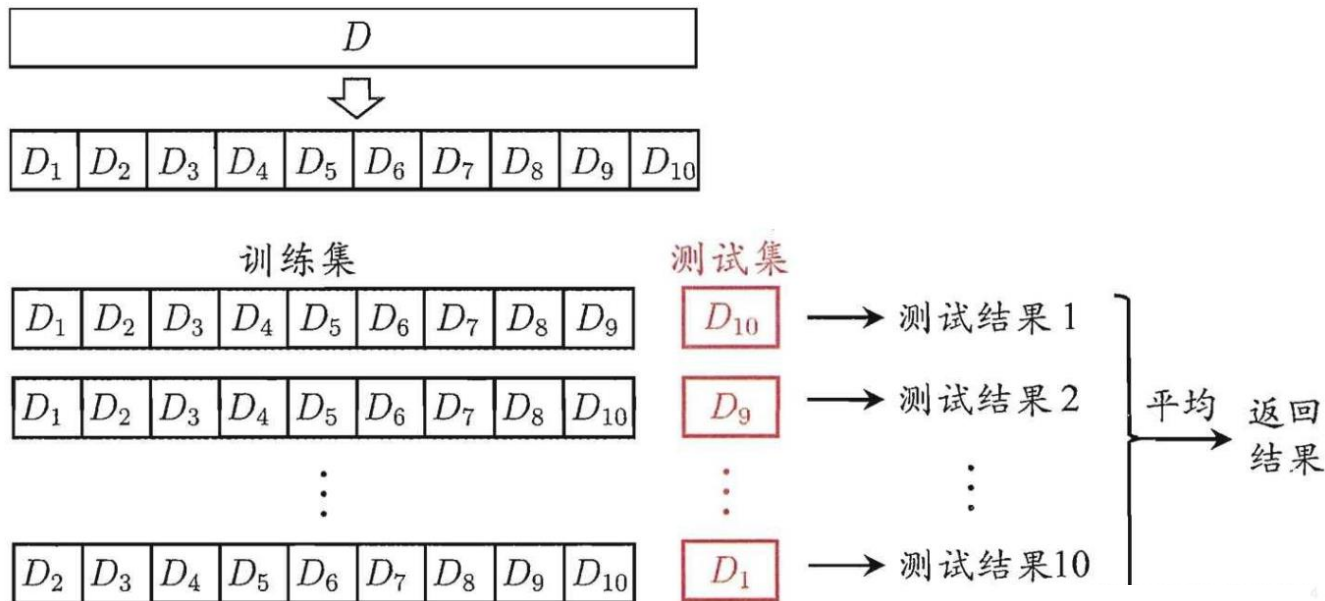
## K折交叉验证 (K-fold cross-validation)

- K次交叉验证，将训练集分割成K个子样本，一个单独的子样本被保留作为验证模型的数据，其他K-1个样本用来训练。交叉验证重复K次，每个子样本验证一次，平均K次的结果或者使用其它结合方式，最终得到一个单一估测。这个方法的优势在于，同时重复运用随机产生的子样本进行训练和验证，每次的结果验证一次，10次交叉验证是最常用的。
- 模型的最终精度是 验证数据的平均精度。

# 交叉验证

## K折交叉验证 (K-fold cross-validation)

- `sklearn.model_selection import cross_val_score, KFold`
- `cross_val_score`的参数`cv`没有设置，默认是3，可以修改`cv`为5或10，



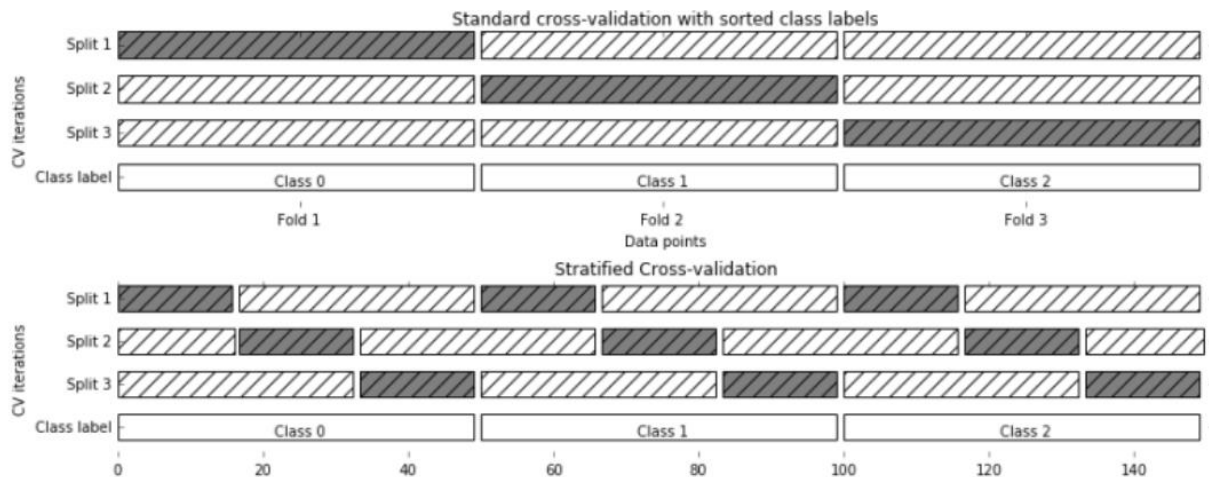
# 交叉验证

## 分层交叉验证 (Stratified k-fold cross validation)

- 分层 K-Fold 是 K-Fold 交叉验证的增强版本，主要用于不平衡的数据集。是针对非平衡数据的分层采样，在每一份子集中都保持原始数据集的类别比例。
- `from sklearn.model_selection import StratifiedKFold`

标准交叉验证（即K折交叉验证）：直接将数据分成几折；

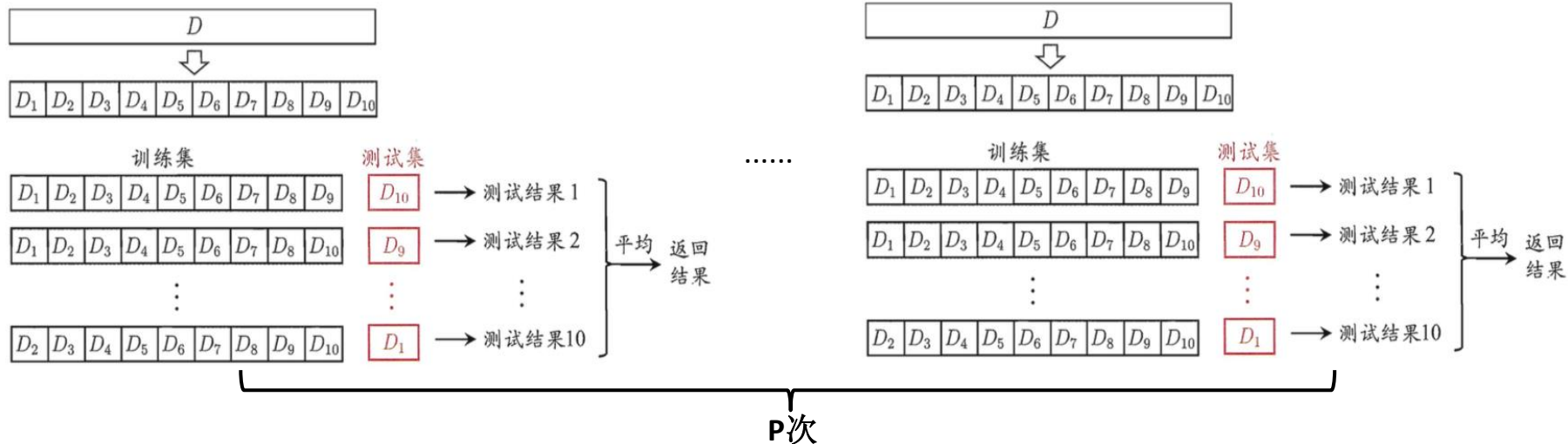
分层交叉验证：先将数据分类（class1,2,3），然后在每个类别中划分三折。



# 交叉验证

## 重复K折交叉验证 (RepeatedKFold)

- 重复 K-Fold n 次，在每次重复中产生不同的分割。
- `from sklearn.model_selection import RepeatedKFold`



# 交叉验证

## Leave P Out 交叉验证

- Leave P Out 交叉验证是一种详尽的交叉验证技术，其中  $p$  样本用作验证集，剩余的  $np$  样本用作训练集。
- 假设我们在数据集中有 100 个样本。如果使用  $p=1$ ，那么在每次迭代中，1 个值将用作验证集，其余 99 个样本将用作训练集。重复这个过程，直到整个数据集在训练集和验证集上都能被划分。
- `from sklearn.model_selection import LeavePOut`

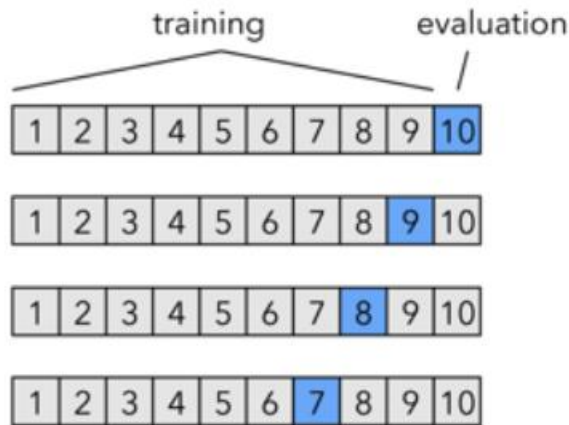
- 优点

所有数据样本都用作训练和验证样本。

- 缺点

计算时间长。

不适合不平衡数据集：





# 交叉验证

---

交叉验证后，最终的模型是什么？

- 交叉验证并不是为了去得到一个具体的模型或模型参数。
- 主要目的是为了去评估某个模型在整个数据集上的一些特性，最后决定是否使用这个模型。

# 交叉验证

## 交叉验证万能模版

```
from sklearn. 1、算法位置 import 2、算法名
from sklearn.metrics import accuracy_score #用精确度评估模型
# 生成一个模型对象
3、模型名 = 2、算法名 ( 模型参数【选填】 )

# 训练模型
3、模型名 .fit(train_x,train_y)

# 在训练集上评估模型
pred1 = 3、模型名 .predict(train_x)
accuracy1 = accuracy_score(train_y,pred1)
print('在训练集上的精确度: %.4f'%accuracy1)

# 在测试集上评估模型
pred2 = 3、模型名 .predict(test_x)
accuracy2 = accuracy_score(test_y,pred2)
print('在测试集上的精确度: %.4f'%accuracy2)
```

# 交叉验证

## 交叉验证万能模版

```
from sklearn. 1、算法位置 import 2、算法名
from sklearn.metrics import accuracy_score #用精确度评估模型
# 生成一个模型对象
3、模型名 = 2、算法名 ( 模型参数【选填】 )
```

# 训练模型

```
3、模型名 .fit(train_x,train_y)
```

第二步：加入交叉验证

```
scores1 = cross_val_score( 3、模型名 ,train_x,train_y,cv=n,scoring='accuracy')
```

# 输出精确度的平均值

```
print("训练集上的精确度: %0.2f " % scores1.mean())
```

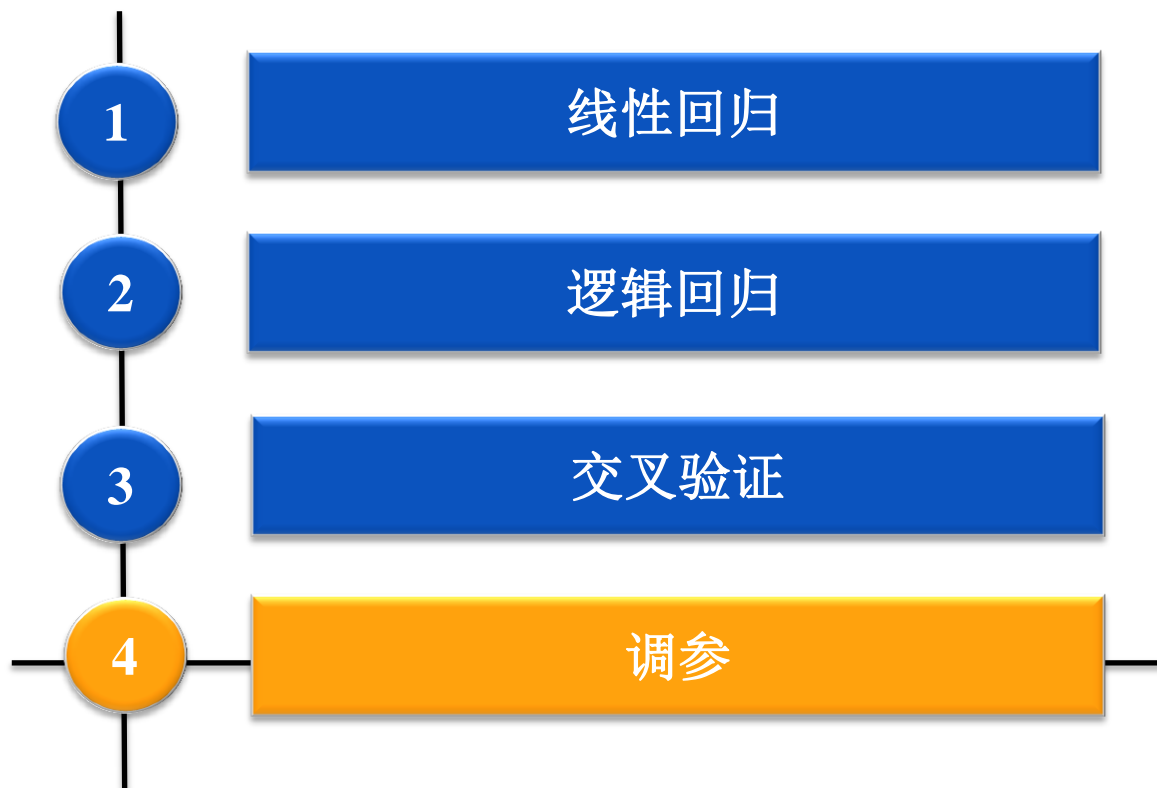
```
scores2 = cross_val_score( 3、模型名 ,test_x,test_y,cv=n,scoring='accuracy')
```

# 输出精确度的平均值

```
print("测试集上的精确度: %0.2f " % scores2.mean())
```

# 目录

---



# 调参

---

- **数据决定结果的上限**，如何更好的去达到模型上限取决于模型和模型的调参。
- 建模调参：每种模型的设计都有多种参数，使用何种参数搭配可以使得效果最好呢？
  1. 手动调参
  2. 算法调参（随机搜索、网格搜索、贝叶斯搜索）

## 网格搜索

### ➤ 网格搜索(Grid Search)

- 网格搜索是暴力搜索，在给定超参搜索空间内，尝试所有超参组合，最后搜索出最优的超参组合。表现最好的参数就是最终结果（暴力搜索）。
- 它存在的意义就是自动调参，只要把参数输进去，就能给出最优化的结果和参数。但是这个方法适合于小数据集，一旦数据的量级上去了，很难得出结果。
- 缺点很明显，就是慢！指数级的慢！

## 随机搜索

➤ 在超参的搜索分布中随机搜索超参进行尝试，其搜索策略为：

1. 对于搜索范围是distribution的超参数，根据给定的distribution进行随机采样
  2. 对于搜搜范围是list的超参数，在给定的list中等概率采样
  3. 对1，2两步中得到的n\_iter组采样结果，进行遍历。若给定的搜索范围均为list，则不放回抽样n\_iter次。
- 随机搜索的好处在于搜索速度快，但是容易错过一些重要的信息这种方法存在精度较差的问题，但是找到近似最优解的效率高于网格搜索。

# 调参

## 调参模版

```
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn. 1、算法位置 import 2、算法名

3、模型名 = SVC()

params = [
    {' 模型参数1 ': [ 选择1 , 选择2 , 选择3 ], ' 模型参数2 ': [ 选择1 , 选择2 , 选择3 ]},
    {' 模型参数1 ': [ 选择1 , 选择2 ], ' 模型参数2 ': [ 选择1 , 选择2 ]}
    .....
]

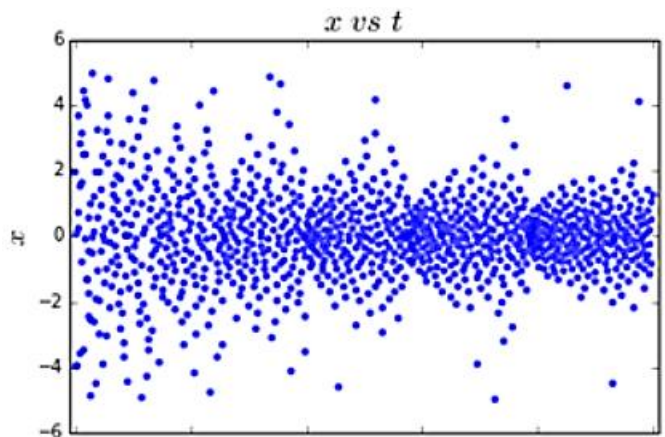
best_model = GridSearchCV( 3、模型名 , param_grid=params, cv = 5, scoring='accuracy')
best_model.fit(train_x, train_y)
```



# 调参

## 贝叶斯调参

- 贝叶斯优化就是通过初期进行超参尝试后，会逐步学习（随着从目标函数获得更多的反馈），对初始搜索空间不同部分进行调整和采样。简单的说，就是考虑上一次参数的信息，从而更好的调整当前的参数。



## 贝叶斯调参

➤ 与常规的网格搜索或者随机搜索的区别是：

1. 贝叶斯调参采用高斯过程，考虑之前的参数信息，不断地更新先验；
2. 网格搜索未考虑之前的参数信息贝叶斯调参迭代次数少，速度快；
3. 网格搜索速度慢，参数多时易导致维度爆炸；
4. 贝叶斯调参针对非凸问题依然稳健；网格搜索针对非凸问题易得到局部最优。
5. 搜索的效率更高，但同时避免了随机搜索会遗漏重要信息的影响。

## Optuna框架

- optuna是一个为机器学习，深度学习特别设计的自动超参数优化框架，具有脚本语言特性的用户API。因此，optuna的代码具有高度的模块特性，并且用户可以根据自己的希望动态构造超参数的搜索空间。
- 框架特点
  1. 小巧轻量，通用且与平台无关
  2. python形式的超参数空间搜索
  3. 高效的优化算法
  4. 写法简单，可以并行
  5. 快速可视化
  6. Optuna拥有许多非常先进的调参算法(如贝叶斯优化，遗传算法采样等)，这些算法往往可以在几十上百次的尝试过程中找到一个不可微问题的较优解

## Optuna框架

➤ optuna是一个使用python编写的超参数调节框架。optuna 的优化程序中只有三个最核心的概念：

- ① **objective**: 定义待优化函数并指定参/超参数数范围，根据目标函数的优化Session,由一系列的trail组成。
- ② **trial**: 根据目标函数作出一次执行。
- ③ **study**: 负责管理优化，决定优化的方式，总试验的次数、试验结果的记录等功能。根据多次trail得到的结果发现其中最优的超参数。

## 搜索方式

- optuna根据需要，支持多种不同的搜索方式(含有low和high值的都是左闭右开):
  - 选择型搜索方式: `trial.suggest_categorical('max_iter',[A',B'])`
  - 整型搜索方式: `trial.suggest_int('num_layers',1,3)`
  - 连续均匀采样搜索方式: `trial.suggest_uniform('dropout_rate',0.0,1.0)`
  - 对数均匀采样方式: `trial.suggest_loguniform('learning_rate',1e-5,1e-2)`
  - 离散均匀采样方式: `trial.suggest_discrete_uniform('drop_path_rate',0.0,1.0,0.1)`

# 调参

---

## 调参算法

➤ Optuna支持的调参算法主要包括以下这些：

- `optuna.samplers.GridSampler()` —— 网格搜索采样
- `optuna.samplers.RandomSampler()` —— 随机搜索采样
- `optuna.samplers.TPESampler()` —— 贝叶斯优化采样
- `optuna.samplers.NSGAIIISampler()` —— 遗传算法采样
- `optuna.samplers.CmaEsSampler()` —— 协方差矩阵自适应演化策略采样，非常先进的优化算法

# 调参

## 执行过程

```
def objective(trial):  
    global x, y  
    X_train, X_test, y_train, y_test=train_test_split(x, y, train_size=0.3)# 数据集划分  
    param = {  
        "n_estimators": trial.suggest_int('n_estimators', 5, 20),  
        "criterion": trial.suggest_categorical('criterion', ['gini','entropy'])  
    }  
  
    dt_clf = RandomForestClassifier(**param)  
    dt_clf.fit(X_train, y_train)  
    pred_dt = dt_clf.predict(X_test)  
    score = (y_test==pred_dt).sum() / len(y_test)  
    return score
```

1 定义函数

```
study=optuna.create_study(direction='maximize')  
n_trials=20 # try50次  
study.optimize(objective, n_trials=n_trials)  
print(study.best_value)  
print(study.best_params)
```

2 执行



大数据，成就未来



# Thank you!