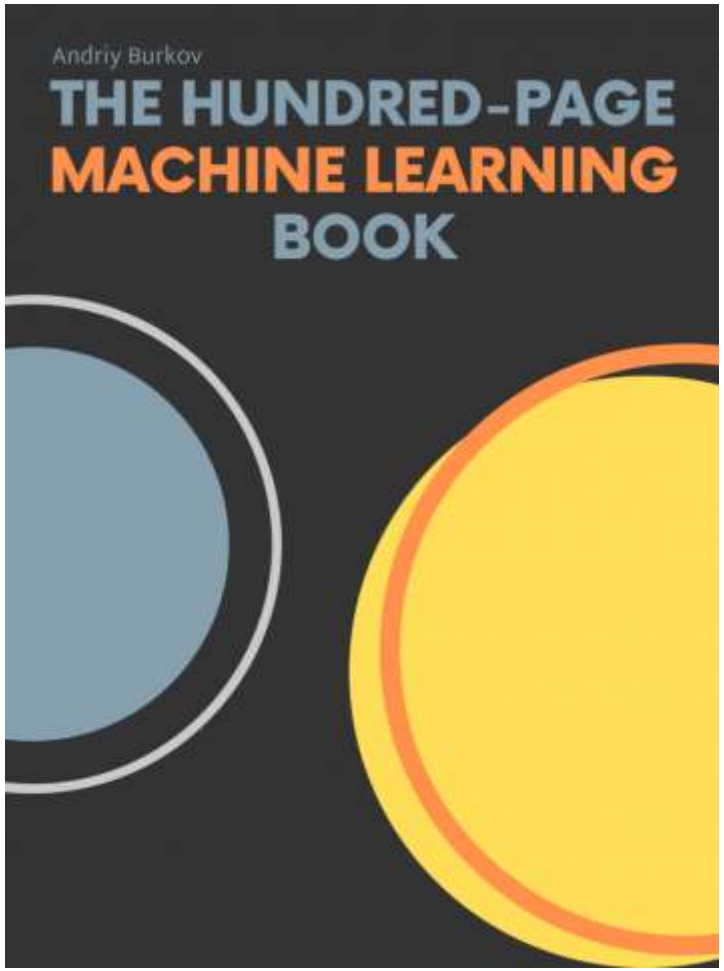
应用数据科学导论

Ht_song@163.com

2020.05

参考书


O'REILLY
TURING 图灵程序设计丛书
数据科学入门
Data Science from Scratch
[美] Joel Grus 著
高蓉 韩波 译
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

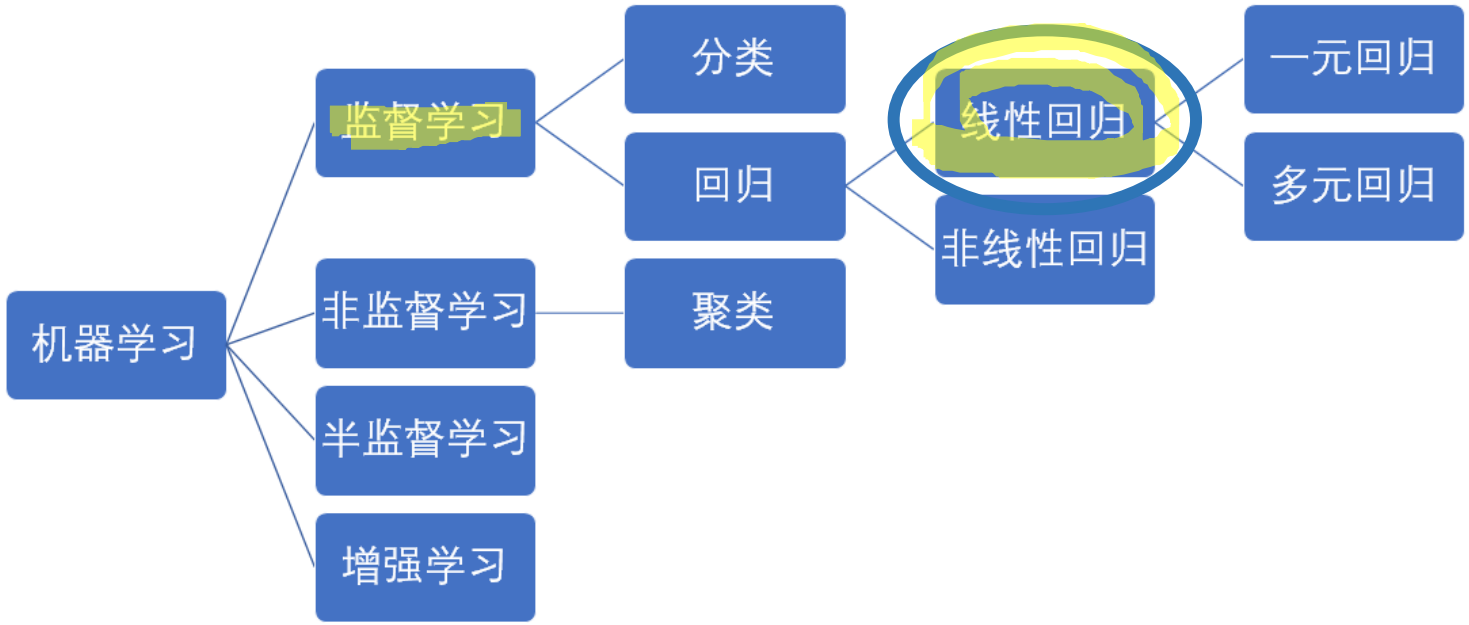

Andriy Burkov
THE HUNDRED-PAGE
MACHINE LEARNING
BOOK

主要内容

- 数据科学简介
- Python语言
- 数据统计可视化
- 数据分析方法——机器学习
- 分析项目实践

线性回归

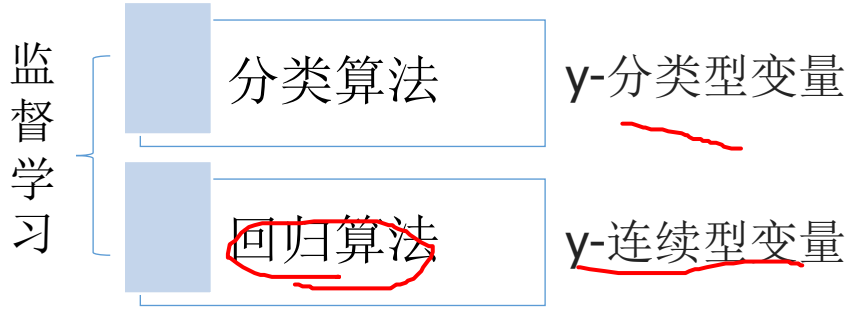
线性回归在机器学习中的位置

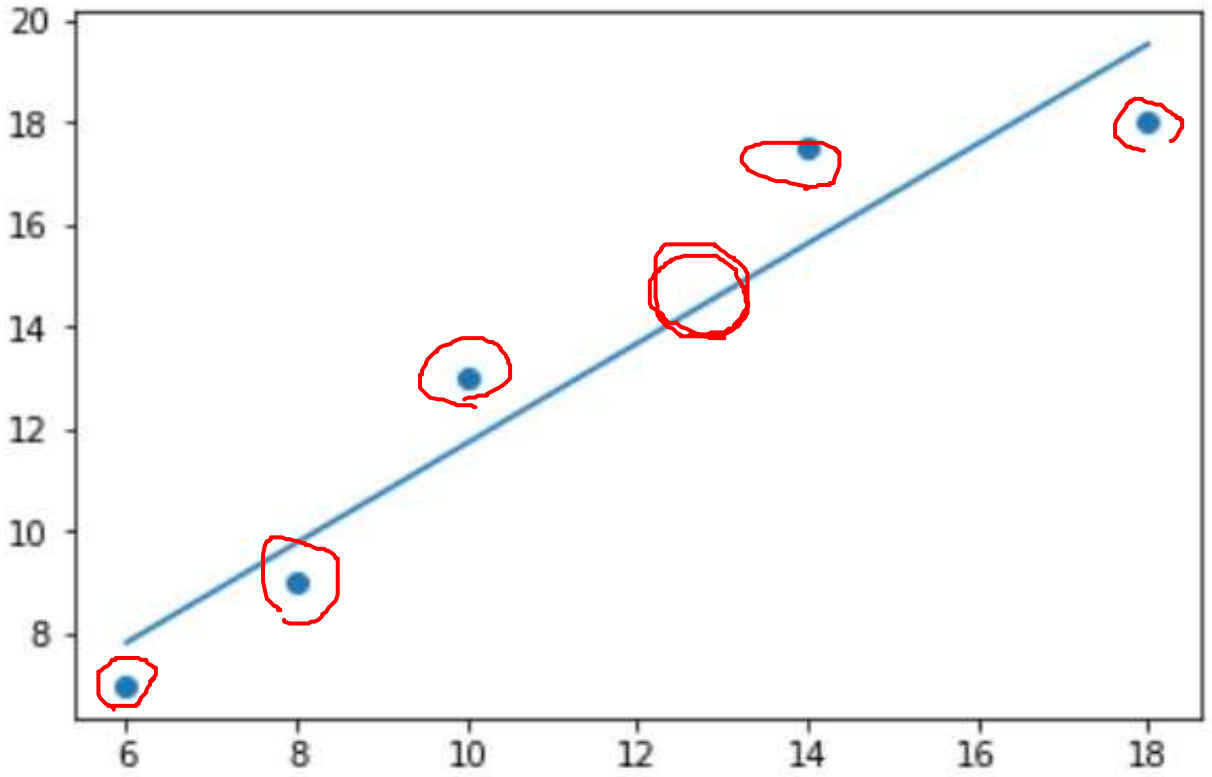


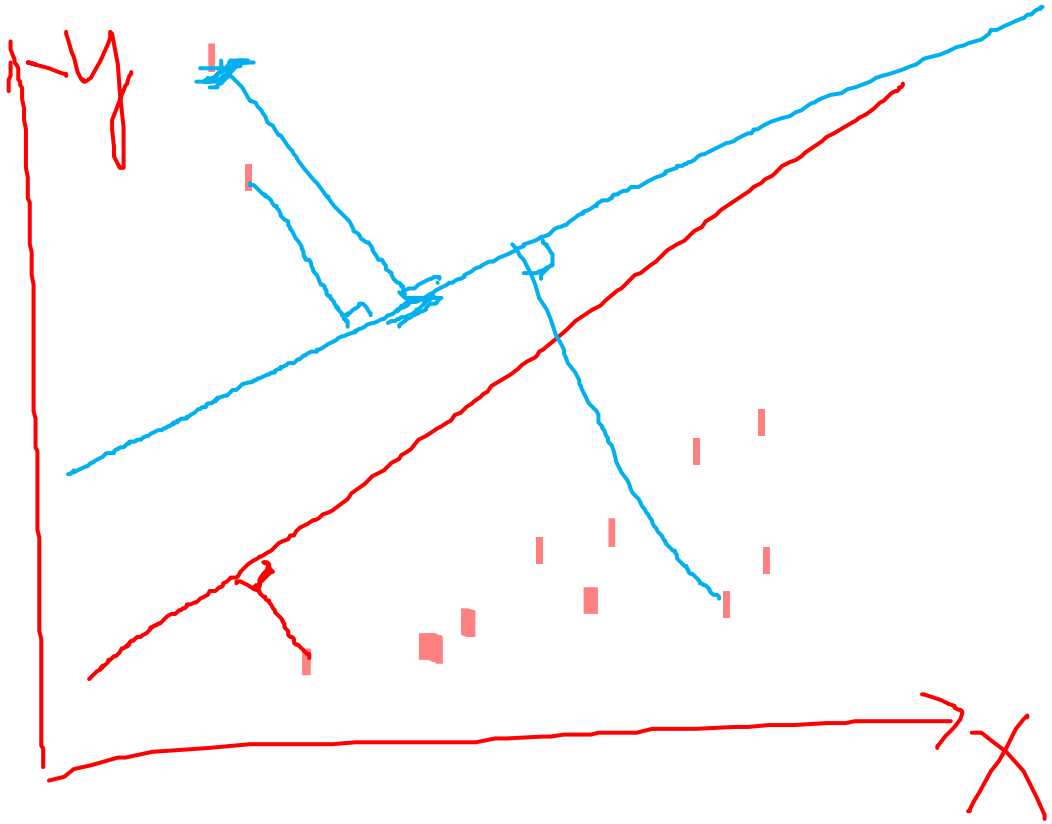
mandy

1.1 线性回归（简介）：

“回归”—计算因变量与自变量之间统计关系的一种方法。
“线性”—学习变量之间为线性关系。
“线性回归”:利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。







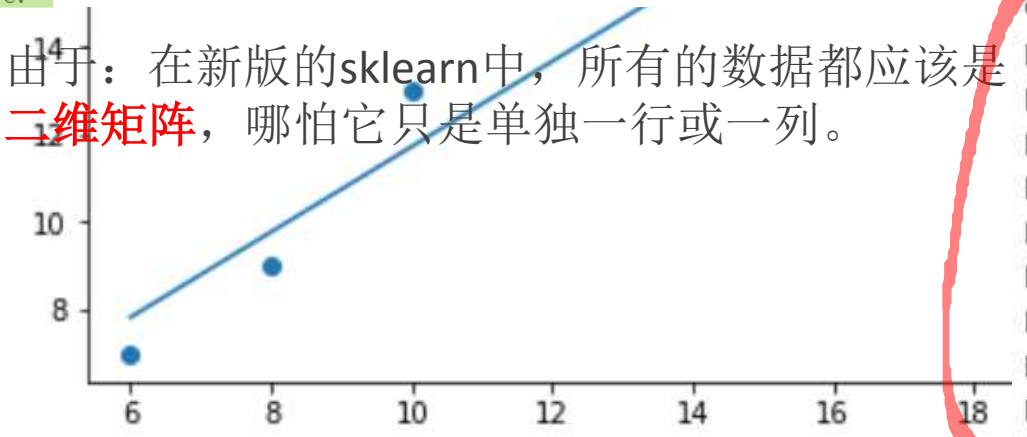
1.2 pizza案例（续）：

```
from sklearn import linear_model        #表示，可以调用sklearn中的linear_model模块进行线性回归。
import numpy as np
model = linear_model.LinearRegression()
model.fit(X, Y)
display(model.intercept_)  #截距
display(model.coef_)  #线性模型的系数
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-b68e817fa885> in <module>
      2 import numpy as np
      3 model = linear_model.LinearRegression()
----> 4 model.fit(X, Y)
      5 display(model.intercept_)  #截距
      6 display(model.coef_)  #线性模型的系数

ValueError: Expected 2D array, got 1D array instead:
array=[ 6  8 10 14 18].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) if it contains a single sampl
e.
```

由于：在新版的sklearn中，所有的数据都应该是二维矩阵，哪怕它只是单独一行或一列。




```
c = np.array([[1,2,3],[4,5,6]])
print('改成2行3列:')
print(c.reshape(2,3))
print('改成3行2列:')
print(c.reshape(3,2))
print('我也不知道几行，反正是1列:')
print(c.reshape(-1,1))
print('我也不知道几列，反正是1行：')
print(c.reshape(1,-1))
print('不分行列，改成1串')
print(c.reshape(-1))
```

1.2 pizza案例（续）：

```
from sklearn import linear_model          #表示，可以调用sklearn中的linear_model模块进行线性回归。
import numpy as np
model = linear_model.LinearRegression()
X=np.array(X).reshape(1,-1)
Y=np.array(Y).reshape(1,-1)
model.fit(X, Y)
display(model.intercept_)  #截距
display(model.coef_)  #线性模型的系数
```

```
array([ 7. ,  9. , 13. , 17.5, 18. ])

array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

```
from sklearn import linear_model          #表示，可以调用sklearn中的linear_model模块进行线性回归。
model = linear_model.LinearRegression()

X = [[6], [8], [10], [14], [18]]
Y = [[7.0], [9.0], [13.0], [17.5], [18.0]]

model.fit(X, Y)
display(model.intercept_)  #截距
display(model.coef_)  #线性模型的系数
```

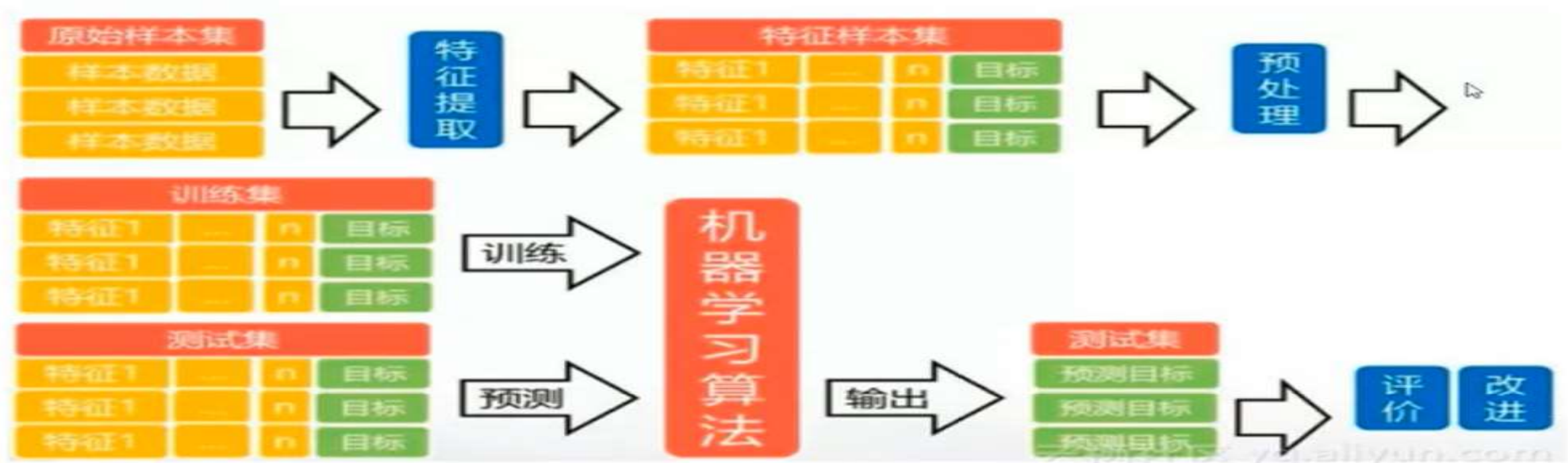
```
array([1.96551724])

array([[0.9762931]])
```

1.3 机器学习流程：

机器学习主要流程：

数据的特征提取 -> 数据预处理 -> 训练和测试建模 -> 评估、改进模型



1.4 波士顿房价预测案例：

波士顿数据集

CRIM: 城镇人均犯罪率

ZN: 住宅用地所占比例

INDUS: 城镇中非住宅用地所占比例

CHAS: 虚拟变量,用于回归分析

NOX: 环保指数

RM: 每栋住宅的房间数

AGE: 1940 年以前建成的自住单位的比例

DIS: 距离 5 个波士顿的就业中心的加权距离

RAD: 距离高速公路的便利指数

TAX: 每一万美元的不动产税率

PTRATIO: 城镇中的教师学生比例

B: 城镇中的黑人比例

LSTAT: 地区中有多少房东属于低收入人群

MEDV: 自住房屋房价中位数（也就是均价）

1.4 波士顿房价预测案例：

```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
```

	数据集名称	调用方式	适用算法	数据规模
小数据集	波士顿房价数据集	load_boston()	回归	506*13
	鸢尾花数据集	load_iris()	分类	150*4
	糖尿病数据集	load_diabetes()	回归	442*10
	手写数字数据集	load_digits()	分类	5620*64

TAT']

1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

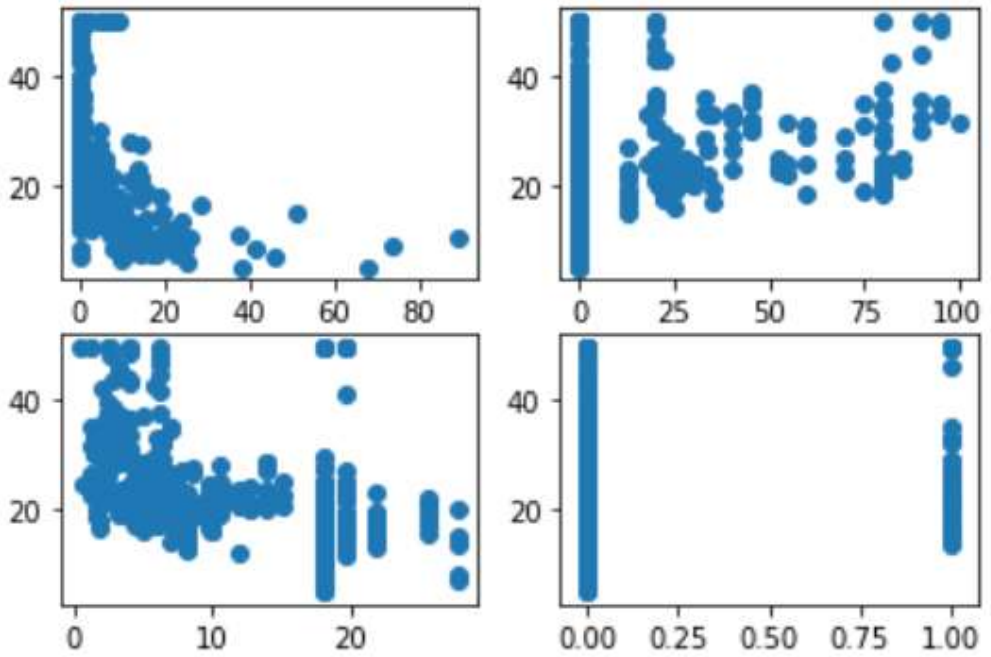
```
x=df[feature_cols]
y=df['MEDV']
```

1.4 波士顿房价预测案例：

绘制各个特征与房价的散点图

```
plt.subplot(221)
plt.scatter(x['CRIM'],y)
plt.subplot(222)
plt.scatter(x['ZN'],y)
plt.subplot(223)
plt.scatter(x['INDUS'],y)
plt.subplot(224)
plt.scatter(x['CHAS'],y)
```

```
<matplotlib.collections.PathCollection at 0x1b29a61f548>
```



1.4 波士顿房价预测案例：

构建训练集，测试集

```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)
```

X_train,X_test, y_train, y_test =sklearn.model_selection.train_test_split(train_data,train_target,test_size=0.4, random_state=0,stratify=y_train)

 # 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。**

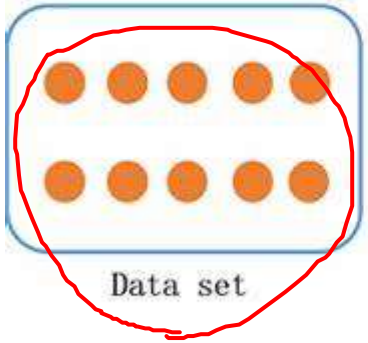
随机数种子：其实就是该组随机数的编号，在需要重复试验的时候，保证得到一组一样的随机数。比如你*每次都填1，其他参数一样的情况下你得到的随机数组是一样的。但填0或不填，每次都会不一样。*

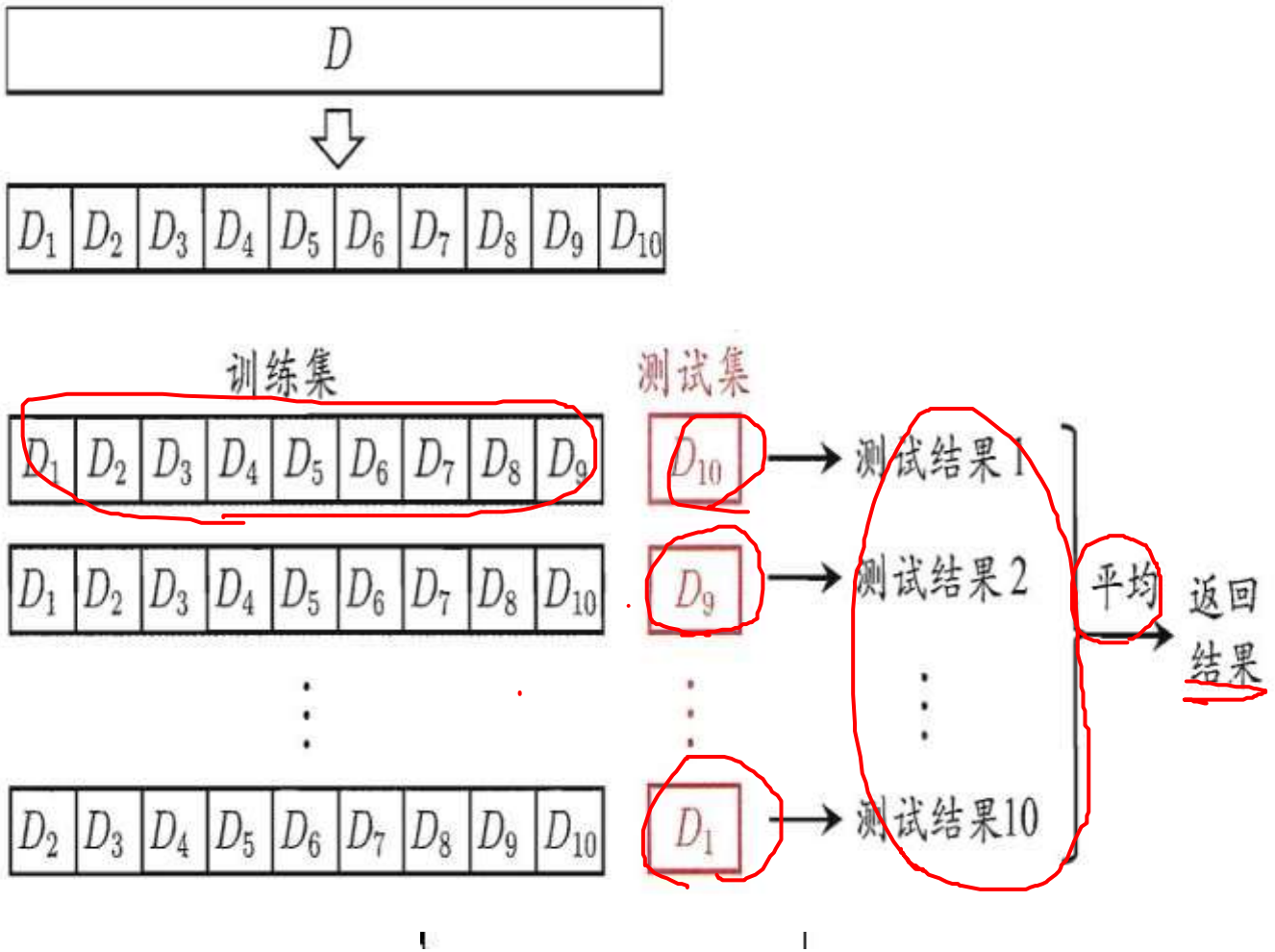
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(379, 13)
(379,)
(127, 13)
(127,)
```

1.4 波士顿房价预测案例：

构建训练集，





1.4 波士顿房价预测案例：

利用训练集进行线性拟合

```
linreg = LinearRegression()
linreg.fit(x_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

显示拟合系数

```
print(linreg.intercept_)
print(linreg.coef_)
```

```
45.192515388288445
[-1.14428903e-01  5.71299780e-02  3.83002824e-02  2.42854641e+00
 -2.12326236e+01  2.87723416e+00  6.91118094e-03 -1.47158266e+00
  3.05784197e-01 -1.06750361e-02 -9.96138270e-01  6.27746234e-03
 -5.57414427e-01]
```

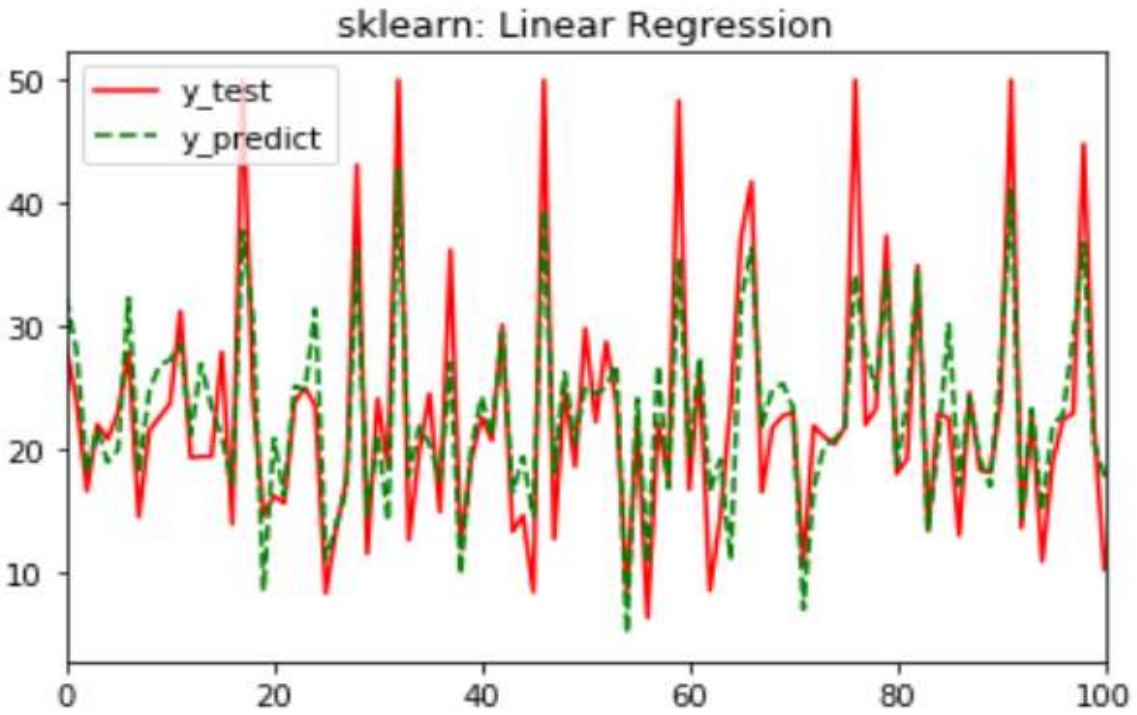
利用测试集进行预测

```
y_pred = linreg.predict(x_test)
```

1.4 波士顿房价预测案例：

图形化预测结果

```
'''----------图形化预测结果----------'''
#只显示前100个预测结果，太多的话看起来不直观
plt.xlim([0,100])
plt.plot( range(len(y_test)), y_test, 'r', label='y_test')
plt.plot( range(len(y_pred)), y_pred, 'g--', label='y_predict' )
plt.title('sklearn: Linear Regression')
plt.legend()
#plt.savefig('lr/lr-13.png')
plt.show()
```



1.4 波士顿房价预测案例：

```
'''----------评价模型----------'''
print("MSE:",metrics.mean_squared_error(y_test,y_pred))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
MSE: 21.89776539604953
RMSE: 4.679504823808768
```

```
print(linreg.score(x_train,y_train))
print(linreg.score(x_test,y_test))
```

```
0.7168057552393374
0.7789410172622855
```

model.score(predict,label_test) 根据预测值和真实值计算评分

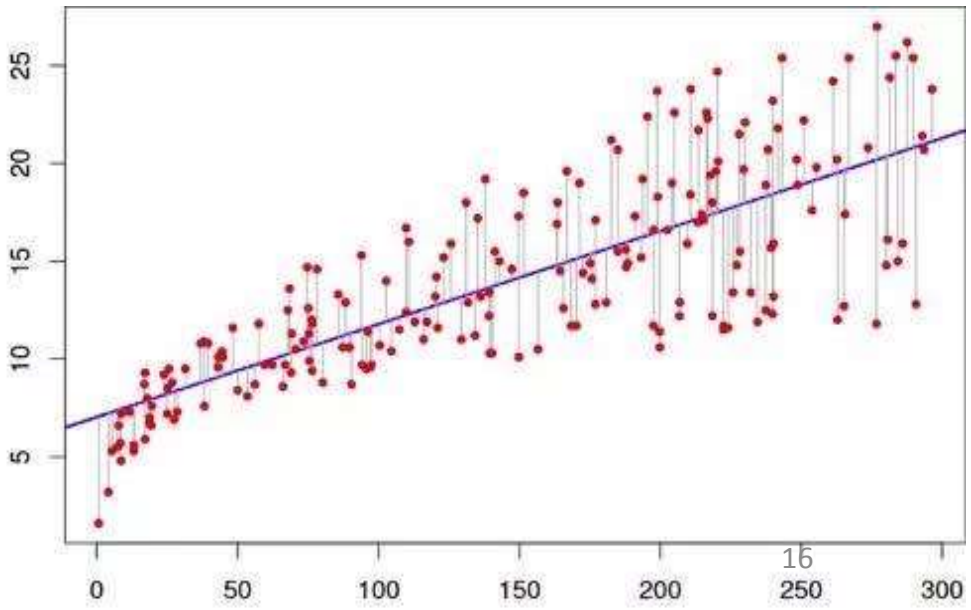
score()对不同类型模型评价标准是不一样的。

- 回归模型：使用“决定系数”评分（coefficient of Determination）
- 分类模型：使用“准确率”评分（accuracy）

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

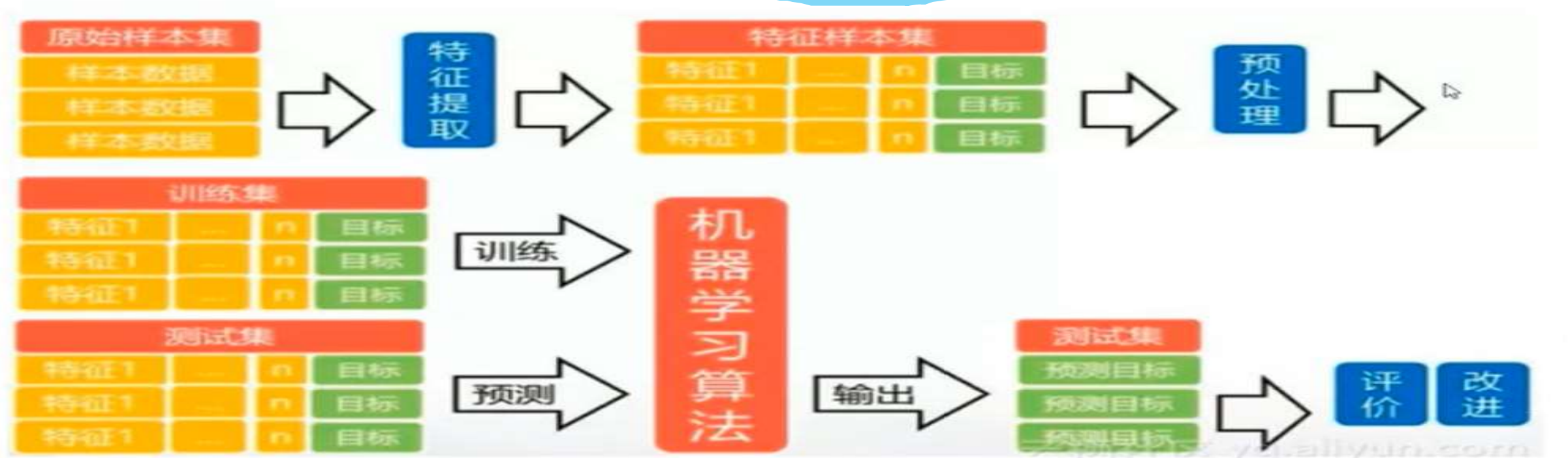
$$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i-\hat{y}_i)^2}$$

和变量的数量级有关系，
比如单位选取之类都会影响它的大小。



机器学习主要流程：

数据的特征提取 -> 数据预处理 -> 训练和测试建模 -> 评估、改进模型



1.5 用Python来实现线性回归（案例）：

1、sklearn.linear_model.LinearRegression

2、NumPy.linalog.lstsq

3、SciPy.stats.linregress

4、Pandas.ols

1.4.2 NumPy.linalog.lstsq案例

- 这是通过矩阵分解计算线性方程组的最小二乘解的基本方法。来自NumPy包的简便线性代数模块。在该方法中，通过计算欧几里德2-范数||b-ax||2最小化的向量x来求解等式ax = b。
- 该方程可能有无数解、唯一解或无解。如果a是方阵且满秩，则x（四舍五入）是方程的“精确”解。
- 可以使用这个方法做一元或多元线性回归来得到计算的系数和残差。一个小诀窍：在调用函数之前必须在x数据后加一列1来计算截距项。

- 优：这被证明是更快速地解决线性回归问题的方法之一。

numpy.linalg.lstsq

numpy.linalg.**lstsq**(*a*, *b*, *rcond=-1*)¶

Return the least-squares solution to a linear matrix equation.

Solves the equation *a x = b* by computing a vector *x* that minimi
determined (i.e., the number of linearly independent rows of *a* c
columns). If *a* is square and of full rank, then *x* (but for round-oft

Parameters:

a : *(M, N) array_like*
"Coefficient" matrix.

b : *{(M,), (M, K)} array_like*
Ordinate or "dependent variable" values.
columns of *b*.

rcond : *float, optional*

D NumPy.linalg函数和属性：

函数	说明
det(ndarray)	计算矩阵列式
eig(ndarray)	计算方阵的本征值和本征向量
inv(ndarray) pinv(ndarray)	计算方阵的逆 计算方阵的Moore-Penrose伪逆
qr(ndarray)	计算qr分解
svd(ndarray)	计算奇异值分解svd
solve(ndarray)	解线性方程组Ax = b，其中A为方阵
lstsq(ndarray)	计算Ax=b的最小二乘解

1.4.3 SciPy.stats.linregress案例

- 这是一个高度专业化的线性回归函数，可以在SciPy的统计模块中找到。除了拟合的系数和截距项之外，它还返回基本统计量，如R2系数和标准差。

- 缺：然而因为它仅被用来优化计算两组测量数据的最小二乘回归，所以其灵活性相当受限。因此，不能使用它进行广义线性模型和多元回归拟合。
- 优：但是，由于其特殊性，它是简单线性回归中最快速的方法之一。由于其简单，即使多达1000万个数据点，stats.linregress和简单的矩阵求逆还是最快速的方法。

scipy.stats.linregress

scipy.stats.linregress(*x*, *y=None*)

Calculate a linear least-squares regression for two sets

Parameters: **x, y** : *array_like*
Two sets of measurements. Both
a two-dimensional array where or
splitting the array along the lengtl

Returns: **slope** : *float*
slope of the regression line
intercept : *float*
intercept of the regression line
rvalue : *float*
correlation coefficient

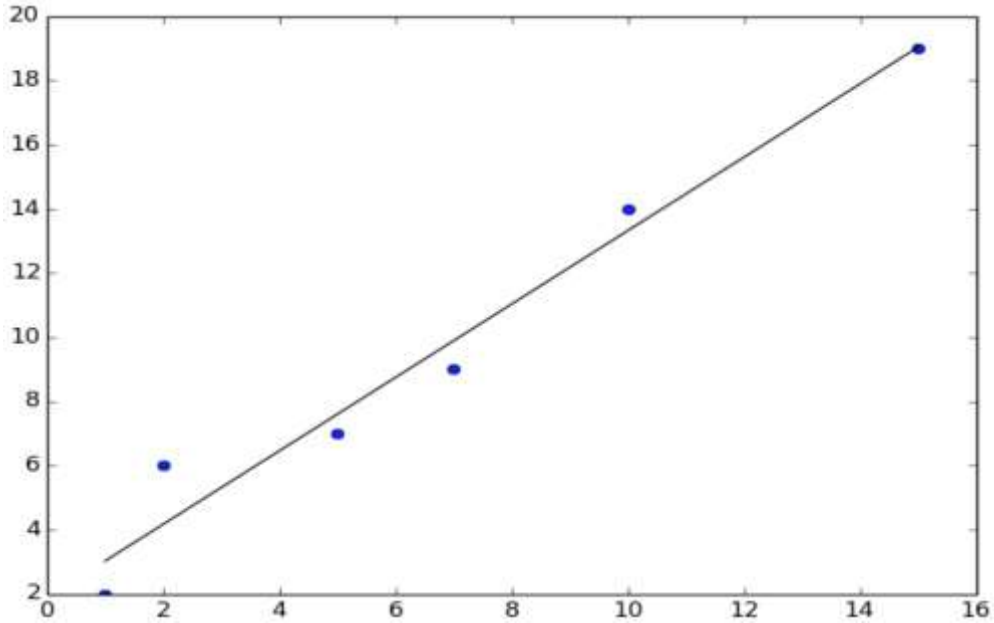
```
from scipy import stats
import numpy as np
import pylab

x = np.array([1, 2, 5, 7, 10, 15])
y = np.array([2, 6, 7, 9, 14, 19])

slope, intercept, r_value, p_value, slope_std_error = stats.linregress(x, y)

predict_y = intercept + slope * x
pred_error = y - predict_y
degrees_of_freedom = len(x) - 2
residual_std_error = np.sqrt(np.sum(pred_error**2) / degrees_of_freedom)
```

```
# Plotting
pylab.plot(x, y, 'o')
pylab.plot(x, predict_y, 'k-')
pylab.show()
```



1.4.4 pandas.ols案例

优：pandas提供了一些很方便的功能，比如最小二乘法(OLS)，可以用来计算回归方程式的各个参数。同时pandas还可以输出类似ANOVA的汇总信息，比如决定系数(R平方), F 统计量等。

```
sudo pip install numpy
sudo pip install pandas

sudo pip install statsmodels #安装pandas的依赖包(使用pandas.ols函数就必须安装这个依赖包)
```

Pandas主要的两个数据结构：Series与DataFrame

```
import numpy as np
import pandas as pd
x = np.array([2,3,4,6])
xx = pd.DataFrame({"k": x})
yy = pd.Series([22,33,44,66])
res = pd.ols(y=yy, x=xx)
res
```

Series是一种类似于一维数组的对象，它由一组数据(各种Numpy数据类型)以及一组与之相关的数据标签(即索引)组成。仅由一组数据即可产生最简单的Series

```
res.beta
k 1.100000e+01
intercept -2.131628e-14
dtype: float64

res.beta[0]
```

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。
DataFrame既有行索引也有列索引，它可以被看做由Series组成的字典（共同用一个索引）。
在DataFrame中的数据时以一个或多个二维块进行存放的（而不是列表、字典或别的一维数据结构）。
DataFrame是以二维结构保存数据的，但是可以仍然可以轻松地将其表示为更高维度的数据。

```
Summary of Regression Analysis
Formula: Y ~ <k> + <intercept>
R-squared: 1.0000
-----------------------------------------------------------------------
k 11.0000 0.0000 1404889085527666.75 0.0000 11.0000 11.0000
intercept -0.0000 0.0000 -0.68 0.5691 -0.0000 0.0000
End of Summary
```

假设现在有个 multiple factor model

```
y = 0.7 * x1 - 1.2 * x2 + 0.3 * x3 + random value
```

按照这个模型创建一个portfolio, 然后我们再拿实际得到的值来跟这3个factor来做下回归分析，看得到的系数是不是跟上面的这个model比较接近。

创建1000只股票，股票代码(5个字符）通过随机方式生成。

```
In [29]: import string

In [32]: import random

In [33]: random.seed(0)

In [34]: N = 1000

In [35]: def rands(n):
   ....:     choices = string.ascii_uppercase
   ....:     return ''.join([random.choice(choices) for _ in xrange(n)])
   ....:

In [36]: tickers = np.array([rands(5) for x in xrange(N)])
```

创建三个随机数组（每个大小都为1000， 对应刚才创建的1000只股票），分别为fac1,fac2,和fac3。

```
In [58]: from numpy.random import rand

In [59]: fac1, fac2, fac3 = np.random.rand(3, 1000)

In [62]: ticker_subset = tickers.take(np.random.permutation(N)[:1000])
```

用选择的1000只股票按照上面的model创建portfolio, 得到的一组值也就是因变量y.

```
In [64]: port = Series(0.7*fac1 - 1.2*fac2 + 0.3*fac3 + rand(1000),
index=ticker_subset)
```

用实际得到y和x1/x2/x3来做下回归。 首先把三个factors 构建成DataFrame.

```
In [65]: factors = DataFrame({'f1':fac1, 'f2':fac2, 'f3':fac3}, index=ticker_subset)
```

直接调用**pd.ols**方法来进行回归

```
In [70]: pd.ols(y=port, x=factors)
```

```
-----------------------Summary of Regression Analysis-----------------------

Formula: Y ~ <f1> + <f2> + <f3> + <intercept>

Number of Observations:         1000
Number of Degrees of Freedom:   4

R-squared:         0.6867
Adj R-squared:     0.6857

Rmse:              0.2859

F-stat (3, 996):   727.6383, p-value:     0.0000

Degrees of Freedom: model 3, resid 996

---------------------Summary of Estimated Coefficients----------------------
      Variable       Coef    Std Err      t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
            f1     0.6968     0.0311       22.44     0.0000     0.6359     0.7577
            f2    -1.2672     0.0312      -40.64     0.0000    -1.3283    -1.2061
            f3     0.3345     0.0310       10.80     0.0000     0.2738     0.3952
     intercept     0.5018     0.0275       18.28     0.0000     0.4480     0.5557
--------------------------------End of Summary--------------------------------
```

如果只关注每个系数，
可以直接读取beta.

```
In [71]: pd.ols(y=port, x=factors).beta
Out[71]:
f1            0.696817
f2           -1.267172
f3            0.334505
intercept     0.501836
dtype: float64
```

```
y = 0.5018 + 0.6968 * f1 - 1.2672 * f2 + 0.3345 * f3
```

```
y = 0.7 * x1 - 1.2 * x2 + 0.3 * x3 + random value
```

1.6 用Python来实现线性回归（简介+总结）：

Scipy.polyfit( ) or numpy.polyfit( )：

最基本的最小二乘多项式拟合函数（least squares polynomial fit function），接受数据集和任何维度的多项式函数（由用户指定），并返回一组使平方误差最小的系数。

Optimize.curve_fit( )

这与Polyfit方法是一致的，但本质上更具一般性。这个强大的函数来自scipy.optimize模块，可以通过最小二乘最小化将任意的用户自定义函数拟合到数据集上。

不言而喻，它也适用于多元回归，并返回最小二乘度量最小的函数参数数组以及协方差矩阵。

Statsmodels.OLS ( )

Statsmodels是一个小型的Python包，它为许多不同的统计模型估计提供了类和函数，还提供了用于统计测试和统计数据探索的类和函数。每个估计对应一个泛结果列表。可根据现有的统计包进行测试，从而确保统计结果的正确性。

对于线性回归，可以使用该包中的OLS或一般最小二乘函数来获得估计过程中的完整的统计信息。

1.7 线性回归优势：

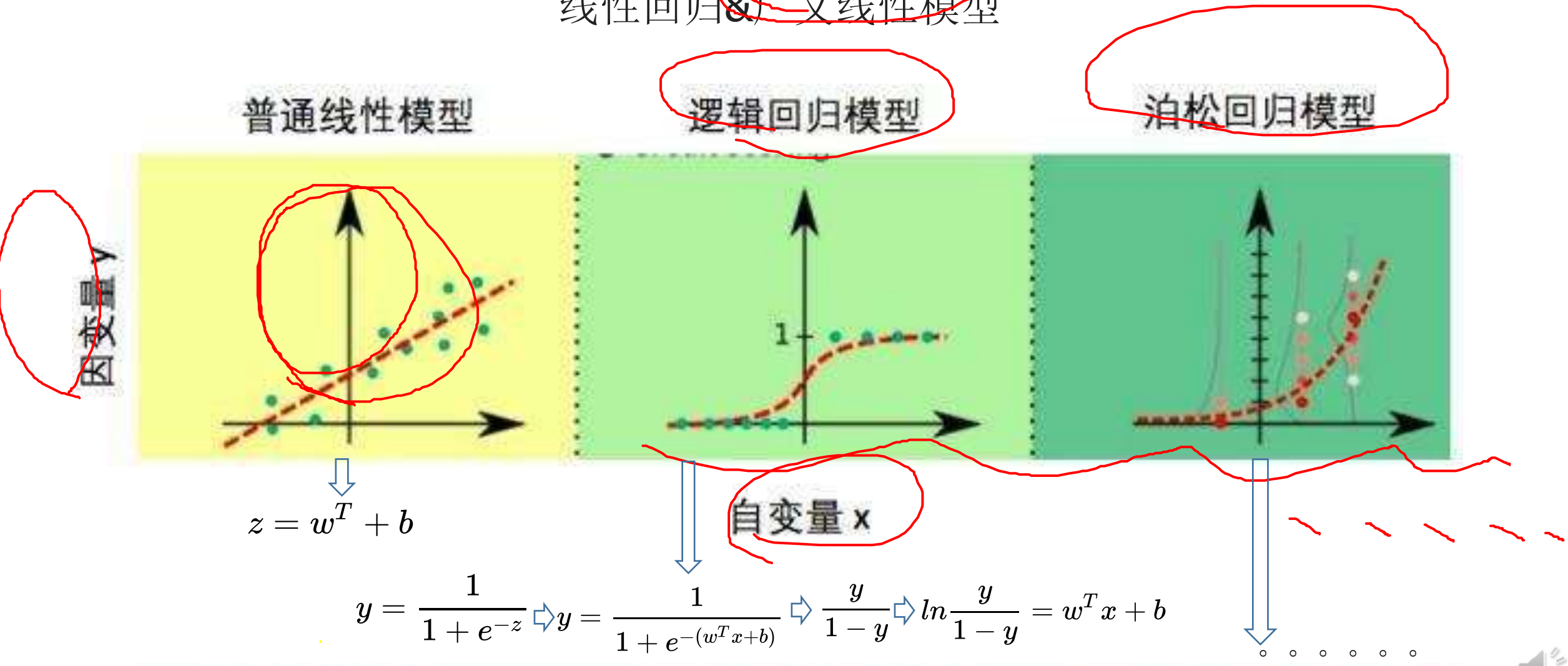
为什么需要学习线性回归?

时至今日，深度学习早已成为数据科学的新宠。即便往前推10年，SVM、boosting等算法也能在准确率上完爆线性回归。
那么，为什么我们还需要线性回归呢？

- 线性回归所能够模拟的关系其实远不止线性关系。线性回归中的“线性”指的是系数的线性，而通过对特征的非线性变换，以及广义线性模型的推广，输出和特征之间的函数关系可以是高度非线性的。
- 更为重要的一点是：且线性模型的易解释性使得它在物理学、经济学、商学等领域中占据了难以取代的地位。

1.8 线性回归与广义线性模型：

线性回归&广义线性模型



1.8 线性回归与广义线性模型（GLMs）：

1.1.1. **普通最小二乘法**：LinearRegression拟合一个带有系数 w = (w1, ..., w_p) 的线性模型，使得数据集实际观测数据和预测数据（估计值）之间的残差平方和最小。

1.1.2. **岭回归**：Ridge回归通过对系数的大小施加惩罚来解决 普通最小二乘法的一些问题。岭系数最小化的是带罚项的残差平方和。

1.1.3. **Lasso**：Lasso是拟合稀疏系数的线性模型。

1.1.4. **多任务 Lasso**：MultiTaskLasso是一个估计多元回归稀疏系数的线性模型。

1.1.5. **弹性网络**：弹性网络是一种使用 L1，L2 范数作为先验正则项训练的线性回归模型。

1.1.6. **多任务弹性网络**：MultiTaskElasticNet 是一个对多回归问题估算稀疏参数的弹性网络。

1.1.7. **最小角回归**：最小角回归 （LARS） 是对高维数据的回归算法。

1.1.8. **LARS Lasso**：LassoLars 是一个使用 LARS 算法的 lasso 模型，不同于基于坐标下降法的实现，它可以得到一个精确解，也就是一个关于自身参数标准化后的一个分段线性解。

1.1.9. **正交匹配追踪法（OMP）：**OrthogonalMatchingPursuit (正交匹配追踪法)orthogonal_mp使用了 OMP 算法近似拟合了一个带限制的线性模型，该限制影响于模型的非 0 系数。

1.1.10. **贝叶斯回归：**贝叶斯回归可以用于在预估阶段的参数正则化: 正则化参数的选择不是通过人为的选择，而是通过手动调节数据值来实现。

1.1.11. **logistic回归：**logistic 回归，虽然名字里有 “回归” 二字，但实际上是解决分类问题的一类线性模型。【下一章节具体介绍】

1.1.12. **随机梯度下降，SGD** Thank you guys~ 而高效的方法。在样本量（和特征数）很大时尤为有用。

1.1.13. **Perceptron（感知器）：**Perceptron 是适用于大规模学习的一种简单算法。不需要设置学习率（learning rate）、不需要正则化处理、仅使用错误样本更新模型。

1.1.14. **Passive Aggressive Algorithms(被动攻击算法)：**与感知器类似，但多出正则化参数 C 。

1.1.15. **稳健回归（Robustness regression）：**适用于回归模型包含损坏数据（corrupt data）的情况，如离群点或模型中的错误。处理离群点（outliers）和模型错误。

1.1.16. **多项式回归：**用基函数展开线性模型。机器学习中一种常见的模式，是使用线性模型训练数据的非线性函数。

数据，是信息时代的真相！