

30

Regression Methods in Scikit-Learn

Scikit-Learn 回归

一元线性回归、二元线性回归、多项式回归



想象力比知识更重要，因为知识是有限的，而想象力概括世界上的一切，推动着进步，并且是知识进化的源泉。

Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution. It is, strictly speaking, a real factor in scientific research.

—— 阿尔伯特·爱因斯坦 (Albert Einstein) | 理论物理学家 | 1879 ~ 1955



- ◀ `sklearn.linear_model.LinearRegression` 线性回归模型类，用于建立和训练线性回归模型
- ◀ `sklearn.preprocessing.PolynomialFeatures` 特征预处理类，用于生成多项式特征，将原始特征的幂次组合以扩展特征空间，用于捕捉更复杂的非线性特征关系



本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

30.1 聊聊回归

回归分析是一种基础但很重要的机器学习方法，回归常用来研究变量之间的关系，并可以用来预测趋势。

本书前文第 27 章已经介绍过用 Statsmodels 库完成一元线性回归。一元线性回归是一种基本的统计分析方法，用于探究两个连续变量之间的关系。“一元”表示模型中只有一个**自变量** (independent variable)。自变量也叫**解释变量** (explanatory variable) 或**回归元** (regressor)、**外生变量** (exogenous variables)、**预测变量** (predictor variables)。本章后续还会介绍二元、多元回归。

而“线性回归”则表明模型假设自变量与因变量之间存在线性关系，如图 1 所示。

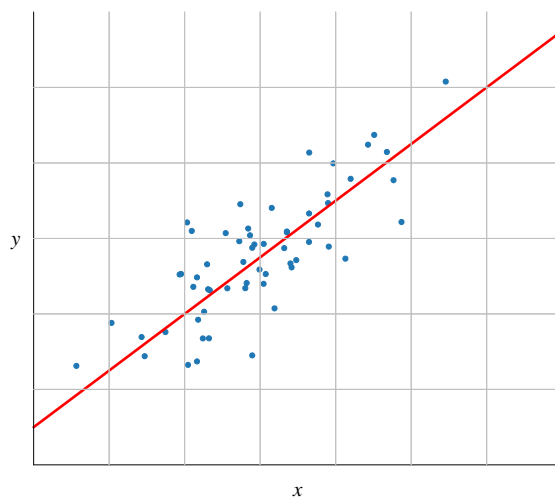


图 1. 平面上，一元线性回归

因变量 (dependent variable) 也叫**被解释变量** (explained variable)、或**回归子** (regressand)、**内生变量** (endogenous variable)、**响应变量** (response variable)。

在一元线性回归中，我们试图找到一条直线，该直线最好地拟合了自变量和因变量之间的数据关系。

具体来说，我们要找到一条直线，使得所有数据点到这条直线的垂直距离之差（残差）平方和最小化。**残差项** (residuals) 也叫**误差项** (error term)、**干扰项** (disturbance term) 或**噪音项** (noise term)。图 2 中灰色线段便代表残差。

如图 3 所示，残差平方和代表图中所有蓝色正方形的面积。这些蓝色正方形的边长便是残差。这种方法叫做最小二乘法 (Ordinary Least Square, OLS)。

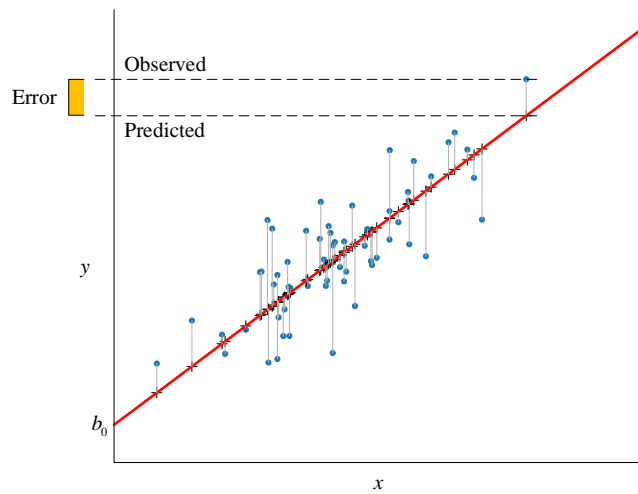


图 2. 一元线性回归中的残差

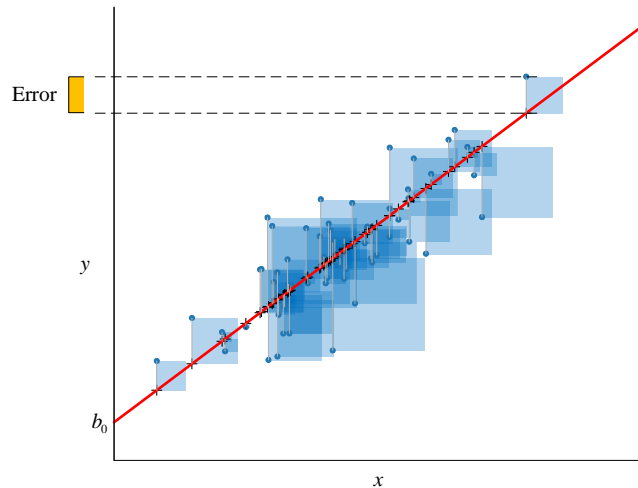


图 3. 残差平方和的几何意义

如图 4 所示，线性回归并不适合所有回归分析；很多时候，我们还需要非线性回归。

非线性回归是指自变量和因变量之间存在着非线性关系的回归模型。在非线性回归中，自变量和因变量的关系不再是简单的线性关系，而可能是多项式关系、指数关系、对数关系等其他非线性形式。非线性回归可以通过拟合曲线或曲面来捕捉数据的非线性关系。本章后续将会介绍多项式回归、逻辑回归两种非线性回归。

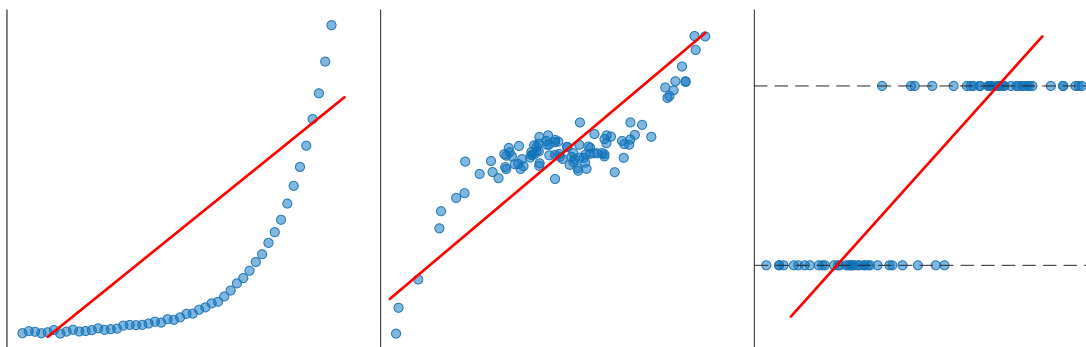


图 4. 线性回归并不适合所有回归分析

30.2 一元线性回归

本书第 27 章介绍用 `statsmodels.regression.linear_model.OLS()` 完成 OLS 一元线性回归。本节采用相同样本数据，但是 Scikit-Learn 函数完成线性回归。

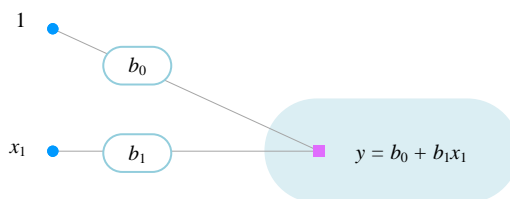


图 5. 一元 OLS 线性回归数据关系

a 从 `sklearn.linear_model` 导入 `LinearRegression`。`LinearRegression` 提供了许多方法和属性，使你能够创建、训练和使用线性回归模型。

b 创建了一个名为 `LR` 的 `LinearRegression` 对象，然后你可以使用这个对象来调用线性回归模型的方法，如拟合数据、进行预测以及评估模型性能等。

例如，可以使用 `LR.fit(X, y)` 方法来拟合训练数据，其中 `X` 是输入特征数据，`y` 是对应的目标输出数据。然后，可以使用 `LR.predict(X_new)` 来对新的输入特征数据 `X_new` 进行预测。

c 中 `LR` 对象调用 `fit(X, y[, sample_weight])` 来拟合模型。其中 `X` 为自变量的数据，`y` 为因变量的数据。该方法会求解最小二乘法的参数，拟合出一条线性回归模型，该模型可以用来预测新的数据。如果指定了 `sample_weight` 参数，则表示样本的权重，可以用于加权最小二乘法。

d 中 `coef_` 用来获取线性回归模型的系数。该属性返回一个数组，其中包含每个自变量对应的系数值，可以用于分析模型的特征重要性。

e 中 `intercept_` 用来获取线性回归模型的截距。该属性返回一个标量，表示线性回归模型的截距值。

f 中 `predict(X)` 用来对新的数据进行预测，其中 `X` 为自变量的数据。该方法会根据已经拟合的线性回归模型，对给定的自变量数据进行预测，返回对应的因变量数据。

图 6 中沿 y 轴方向的灰色线段代表误差，显然这些线段并不垂直红色线。如图 7 所示，如果代表误差的灰色线段绘制于红色线的话，这种回归模型叫正交回归 (orthogonal regression)。正交回归和前文介绍的主成分分析有关。正交回归的一种常见方法是主成分回归 (Principal Component Regression, PCR)，其中主成分分析 PCA 用于寻找数据中的主要方差方向，然后利用这些主成分进行回归。

鸢尾花书《数据有道》将专门介绍正交回归。

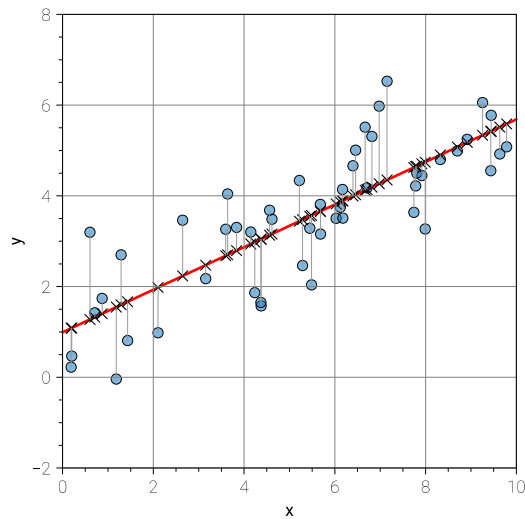


图 6. 一元 OLS 线性回归示例

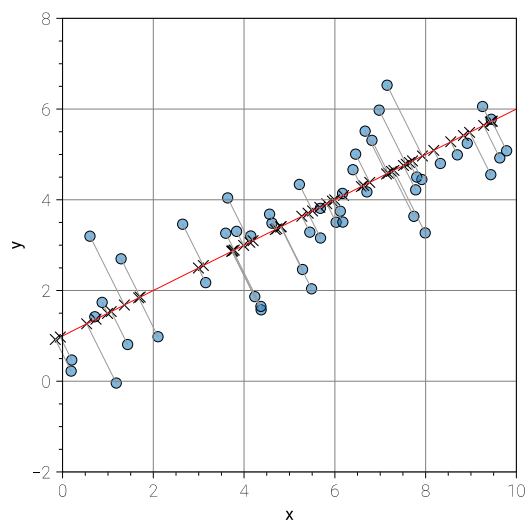


图 7. 一元正交线性回归示例



一元线性回归

```

a import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# 生成随机数据
num = 50
np.random.seed(0)
x_data = np.random.uniform(0, 10, num)
y_data = 0.5 * x_data + 1 + np.random.normal(0, 1, num)

x_data = x_data.reshape((-1, 1))
# 将x调整为列向量
data = np.column_stack([x_data, y_data])

# 创建回归对象并进行拟合
b LR = LinearRegression()
# 使用LinearRegression()构建了一个线性回归模型
c LR.fit(x_data, y_data)

d slope = LR.coef_ # 斜率
e intercept = LR.intercept_ # 截距

x_array = np.linspace(0, 10, 101).reshape((-1, 1))
# 预测
f predicted = LR.predict(x_array)

data_ = np.column_stack([x_data, LR.predict(x_data)])

fig, ax = plt.subplots()
ax.scatter(x_data, y_data)
ax.scatter(x_data, LR.predict(x_data),
           color = 'k', marker = 'x')
ax.plot(x_array, predicted,
        color = 'r')
ax.plot([i for (i, j) in data_], [i for (i, j) in data_],
        ([j for (i, j) in data_], [j for (i, j) in data_]),
        c=[0.6, 0.6, 0.6], alpha = 0.5)

ax.set_xlabel('x'); ax.set_ylabel('y')
ax.set_aspect('equal', adjustable='box')
ax.set_xlim(0, 10); ax.set_ylim(-2, 8)

```

图 8. 一元 OLS 线性回归，代码

30.3 二元线性回归

二元线性回归是一种线性回归模型，其中有两个自变量和一个因变量，它旨在分析两个自变量和因变量之间的线性关系。如图 10 所示，二元线性回归解析式在三维空间为一平面。

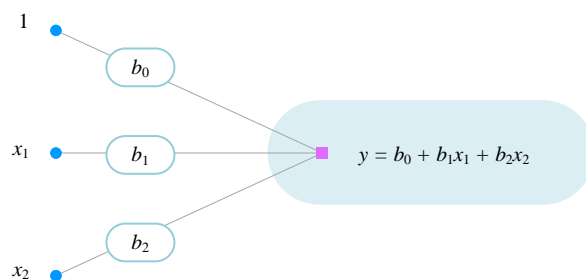


图 9. 二元 OLS 线性回归数据关系

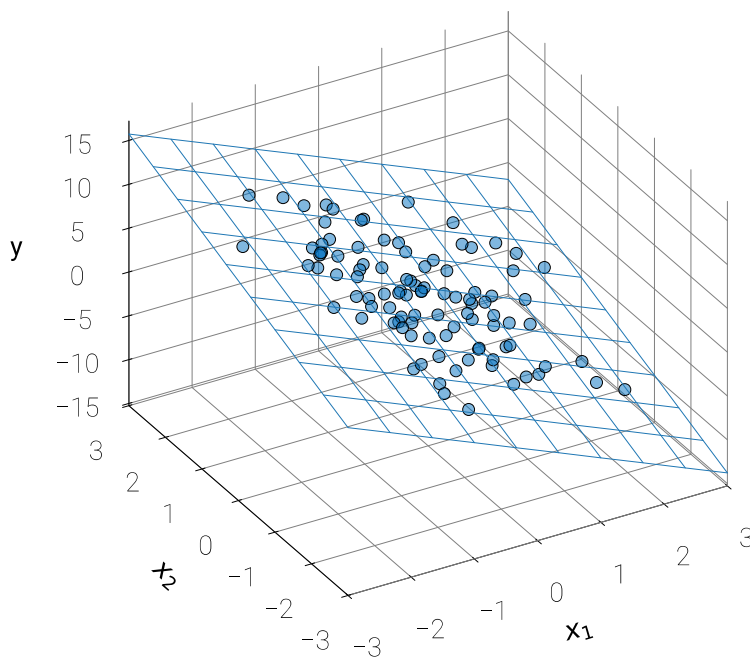


图 10. 二元线性回归示例

图 11 代码绘制图 10，下面介绍其中关键语句。

a 利用 `numpy.random.randn()` 生成自变量数据，两个特征，100 个样本点。

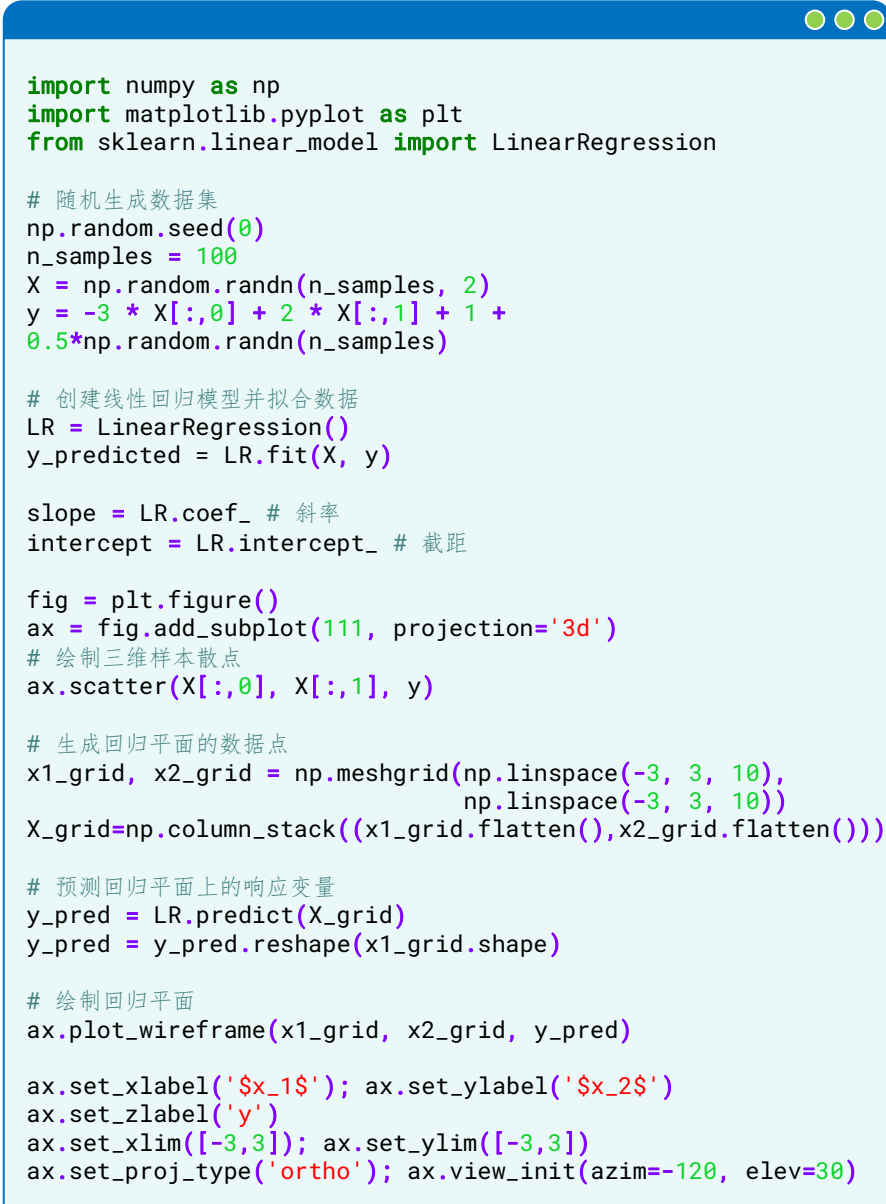
b 中 `fig` 是一个 Matplotlib 中的 Figure 对象，表示一个绘图窗口或画布，可以在这个画布上添加不同类型的子图图轴对象。`add_subplot(111, projection='3d')` 是在 `fig` 上添加一个子图的操作。其中，111 表示子图的布局。在这里，111 表示一个 1×1 的网格，即只有一个子图。`projection='3d'` 指定子图的投影方式为 3D 投影。这意味着，我们可以在该子图中创建一个三维的可视化场景，可以用于绘制三维数据点、曲线、表面等。

c 利用 `numpy.column_stack()` 将两个一维数组按列堆叠在一起，形成一个二维数组，代表了坐标。其中，`x1_grid.flatten()` 和 `x2_grid.flatten()` 将二维数组扁平化为一维数组。

d 将输入特征数据 `X_grid` 传递给已训练的线性回归模型 `LR`，然后获得预测输出值，这些预测输出值被存储在 `y_pred` 变量中。

e 利用 `numpy.reshape()` 调整之前计算得到的预测结果数组 `y_pred` 的形状，使其与另一个数组 `x1_grid` 具有相同的形状。

f 用 `plot_wireframe()` 绘制二元线性回归平面。



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

# 随机生成数据集
np.random.seed(0)
n_samples = 100
a X = np.random.randn(n_samples, 2)
y = -3 * X[:,0] + 2 * X[:,1] + 1 + 0.5*np.random.randn(n_samples)

# 创建线性回归模型并拟合数据
LR = LinearRegression()
y_predicted = LR.fit(X, y)

slope = LR.coef_ # 斜率
intercept = LR.intercept_ # 截距

b fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# 绘制三维样本散点
ax.scatter(X[:,0], X[:,1], y)

# 生成回归平面的数据点
x1_grid, x2_grid = np.meshgrid(np.linspace(-3, 3, 10),
                                np.linspace(-3, 3, 10))
c X_grid=np.column_stack((x1_grid.flatten(),x2_grid.flatten()))

# 预测回归平面上的响应变量
d y_pred = LR.predict(X_grid)
e y_pred = y_pred.reshape(x1_grid.shape)

# 绘制回归平面
f ax.plot_wireframe(x1_grid, x2_grid, y_pred)

ax.set_xlabel('$x_1$'); ax.set_ylabel('$x_2$')
ax.set_zlabel('$y$')
ax.set_xlim([-3,3]); ax.set_ylim([-3,3])
ax.set_proj_type('ortho'); ax.view_init(azim=-120, elev=30)
```



二元线性回归

图 11. 二元 OLS 线性回归，代码

有了二元线性回归，理解多元线性回归就很容易了。如图 12 所示，多元线性回归是一种线性回归的扩展形式，用于建立一个预测模型来描述多个输入特征与一个连续的目标输出之间的线性关系。

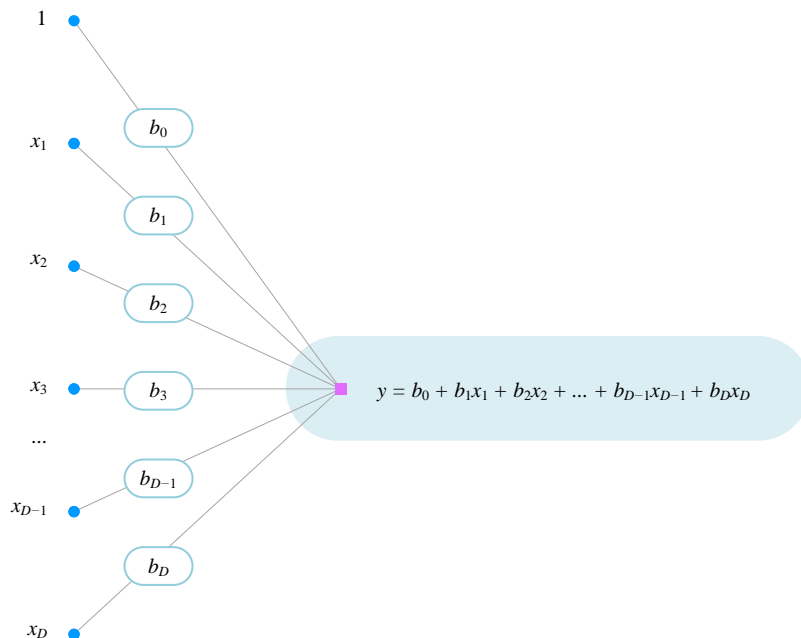


图 12. 多元 OLS 线性回归数据关系

30.4 多项式回归

多项式回归 (polynomial regression) 是一种线性回归的扩展，它允许我们通过引入多项式 (例如，二次、三次、四次等) 来建模非线性关系。如图 13 所示，在多项式回归中，我们不仅使用自变量的原始值，还将其不同阶数作为额外的特征，从而能够更好地拟合数据中非线性模式。

从函数图像角度来讲，如图 14 所示，多项式回归曲线好比若干曲线叠加的结果。

多项式回归的阶数影响着模型的灵活性。如图 15 所示，较低的阶数 (比如图 15 (a)、(b)) 可能无法很好地捕捉数据中的复杂关系，而较高的阶数 (比如图 15 (e)、(f)) 可能会导致过度拟合。阶数越高，模型越能够适应训练数据，但也越容易在测试数据或实际应用中表现不佳。

过拟合是指模型在训练数据上表现得很好，但在新数据上表现较差的现象。当多项式回归的阶数过高时，模型可能会过度适应训练数据中的噪声和细节，从而失去了泛化能力。这意味着模型对于新的、未见过的数据可能无法进行准确的预测，因为它在训练数据上“记住了”许多细微的变化，而这些变化可能在真实数据中并不存在。

图 16 中代码绘制图 15。下面介绍其中关键语句。

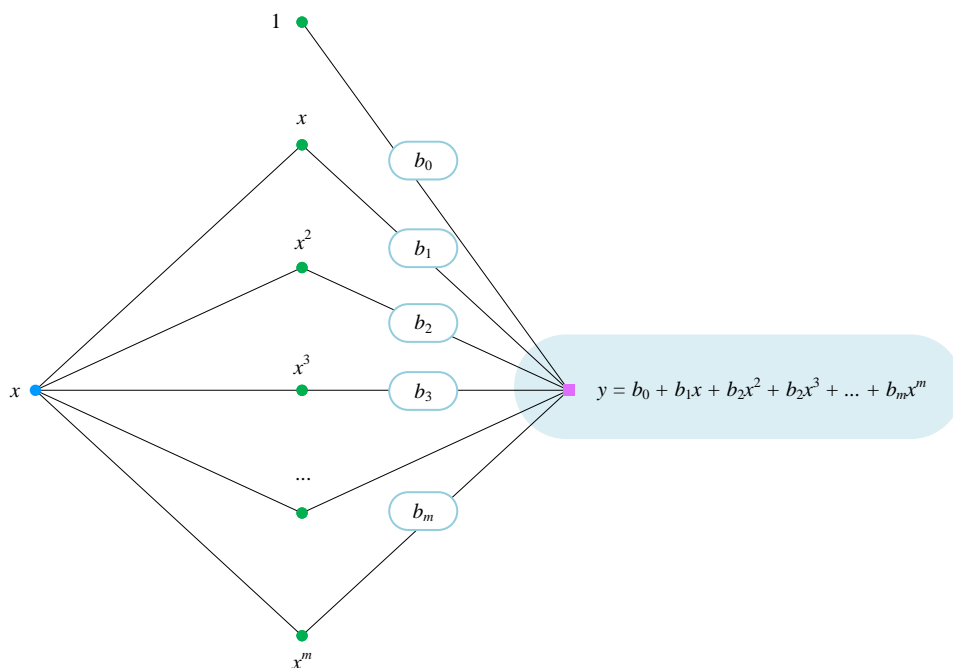


图 13. 多项式回归数据关系

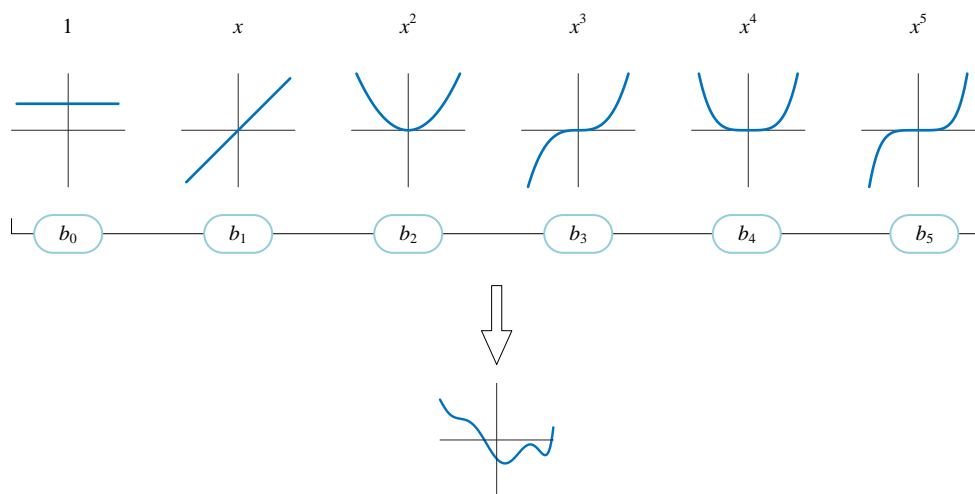


图 14. 一元五次函数可以看做是 6 个图像叠加的结果

a 从 `sklearn.preprocessing` 导入 `PolynomialFeatures`。在机器学习中，有时候原始特征并不足够表达数据的复杂关系，这时可以引入多项式特征。多项式特征是原始特征的幂次组合，通过引入这些特征，可以更好地拟合数据的非线性关系。`PolynomialFeatures` 类的作用就是将原始特征转换为高次的多项式特征。它可以通过设置特定的阶数来生成不同阶数的多项式特征。

b 定义列表，列表中整数为指定的多项式回归阶数。

c 用 `PolynomialFeatures` 原始特征转换为高次的多项式特征。参数 `degree` 设置多项式的阶数。这个阶数决定了生成的多项式回归的最高阶数（次数）。

d 中 `X.reshape(-1, 1)` 将一维数据 `X` 进行形状变换，将其转换为一个二维数组，其中列数为 1。这是因为 `fit_transform` 方法接受的输入应该是一个二维数组，其中每行代表一个样本，每列代表一个特征。在运行代码时，请大家自行查看这一行结果，并用 `seaborn.heatmap()` 可视化结果。

e 创建一个 `LinearRegression` 类的实例，并将其赋值给变量 `poly_reg`。通过这个实例，可以访问回归模型的方法和属性，例如模型的拟合、预测等。

f 加载样本数据，训练回归模型。

g 使用已经训练好的线性回归模型对多项式特征转换后的数据进行预测。

h 这行代码连续完成了：多项式特征转换 + 模型预测。首先将输入数据 `x_array` 进行多项式特征转换，然后使用已经训练好的回归模型 `poly_reg` 对转换后的数据进行预测，并返回预测结果。

i 提取系数 b_1 、 b_2 、 $b_3 \dots$ **j** 提取截距 b_0 。

k 创建一个包含线性方程的字符串。这一句代码首先将截距插入到字符串中。其中，`{:.1f}` 是一个占位符，将用来插入一个浮点数，并保留一位小数。`.format(intercept)` 是 Python 字符串的 `.format()` 方法，用于将特定值插入到格式化字符串中的占位符。

l 利用 `for` 循环，将多项式回归系数项插入到字符串中。`'{:.1f}x^{j}'.format(coef[j], j)` 是一个格式化字符串，用于将系数的值 `coef[j]` 和次数 `j` 插入到字符串中的占位符位置。`{:.1f}` 表示插入一个浮点数，并保留一位小数；`^{j}` 表示插入一个整数。

m 用 `text()` 在子图上打印多项式回归解析式。

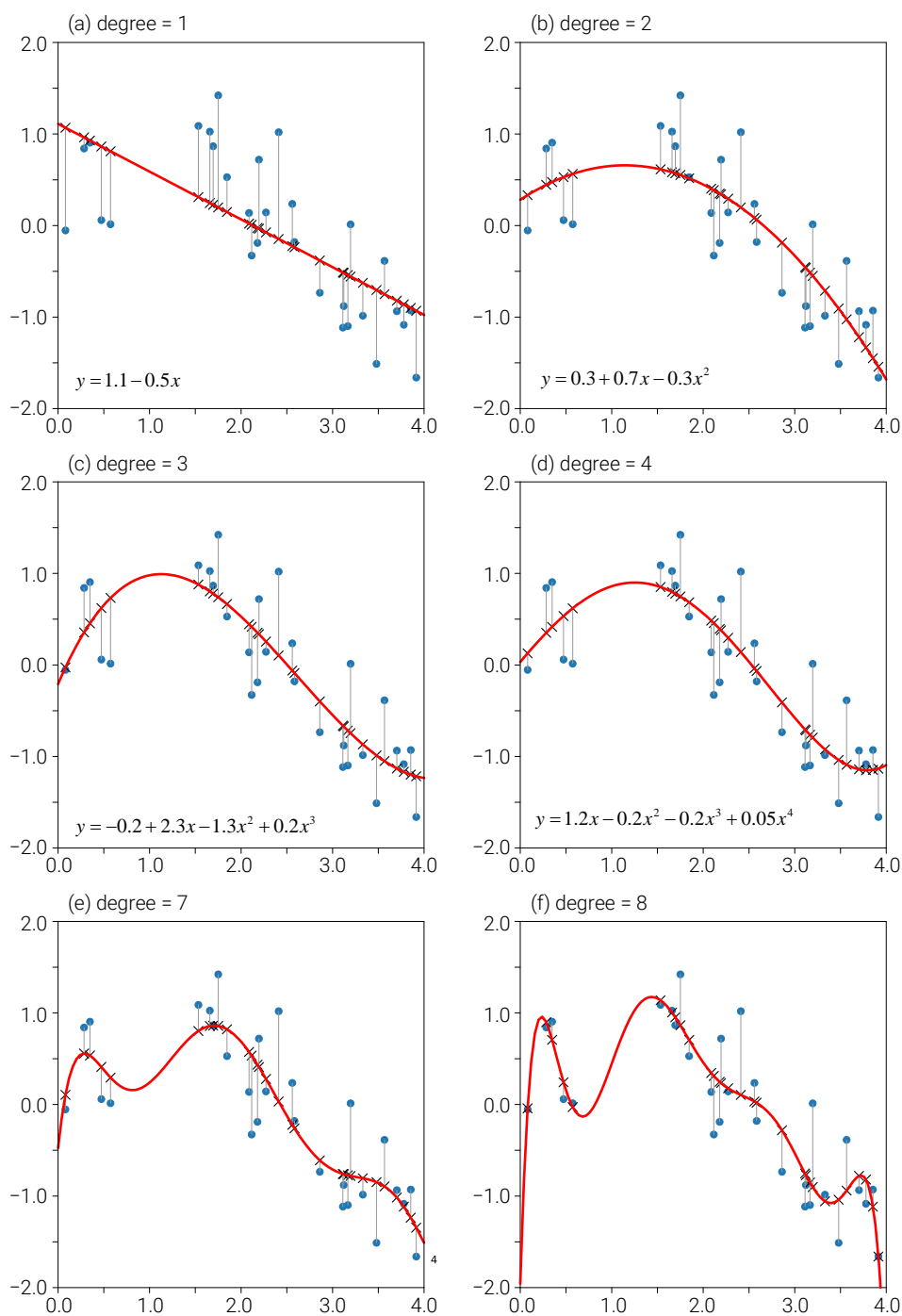


图 15. 阶数对多项式回归曲线影响



多项式回归

```

import numpy as np
import matplotlib.pyplot as plt
a from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 生成随机数据
np.random.seed(0)
num = 30
X = np.random.uniform(0,4,num)
y = np.sin(0.4*np.pi * X) + 0.4 * np.random.randn(num)
data = np.column_stack([X,y])

b x_array = np.linspace(0,4,101).reshape(-1,1)
degree_array = [1,2,3,4,7,8]
fig, axes = plt.subplots(3,2,figsize=(10,20))
axes = axes.flatten()

for ax, degree_idx in zip(axes,degree_array):
c     poly = PolynomialFeatures(degree = degree_idx)
d     X_poly = poly.fit_transform(X.reshape(-1, 1))

    # 训练线性回归模型
e     poly_reg = LinearRegression()
f     poly_reg.fit(X_poly, y)
g     y_poly_pred = poly_reg.predict(X_poly)
    data_ = np.column_stack([X,y_poly_pred])

h     y_array_pred = poly_reg.predict(
        poly.fit_transform(x_array))

    # 绘制散点图
    ax.scatter(X, y, s=20)
    ax.scatter(X, y_poly_pred, marker = 'x', color='k')

    ax.plot([i for (i,j) in data_], [i for (i,j) in data_],
            ([j for (i,j) in data_], [j for (i,j) in data_]),
            c=[0.6,0.6,0.6], alpha = 0.5)

    ax.plot(x_array, y_array_pred, color='r')
    ax.set_title('Degree = %d' % degree_idx)

    # 提取参数
i     coef = poly_reg.coef_
j     intercept = poly_reg.intercept_

    # 回归解析式
k     equation = '$y = {:.1f}'.format(intercept)
l     for j in range(1, len(coef)):
        equation += ' + {:.1f}x^{j}'.format(coef[j], j)
    equation += '$'

m     equation = equation.replace("+ -", "-")
    ax.text(0.05, -1.8, equation)
    ax.set_aspect('equal', adjustable='box')
    ax.set_xlim(0,4)
    ax.grid(False)
    ax.set_ylim(-2,2)

```

图 16. 多项式回归，代码