机器学习（L05）：非线性回归模型和模型选择

授课教师：王贝伦 / 助教：张嘉琦，黄旭，谈笑，徐浩卿

1 基本非线性回归问题

线性回归模型形式简单，可解释性强，有着大量的理论支撑，但是在实际问题中，很多关系往往不能用线性模型简单地概括。我们解决数学问题时，往往会将新问题转化为旧问题解决。同样的，在这一章节中，我们将学习一些典型的非线性回归模型，通过变量代换，我们可以将其转化为已经学过的线性回归模型来解决。

1.1 多项式回归（Polynomial Regression）

1.1.1 多项式回归函数的表达形式

通常多项式回归模型函数的形式为

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \cdots + \theta_m x_i^m \quad (i = 1, 2, ..., n) \tag{1}$$

为了方便表达，我们可以将 (1) 式写为矩阵形式：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 & x_1^2 & \dots & x_1^m \\ x_2 & x_2^2 & \dots & x_2^m \\ x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix}$$

$$\boldsymbol{y} = \boldsymbol{\varphi\theta} \tag{2}$$

1.1.2 多项式回归的函数图像

在线性回归中，变量之间成线性关系，即 $\hat{\boldsymbol{y}} = \boldsymbol{\theta x}$，回归拟合函数为直线：

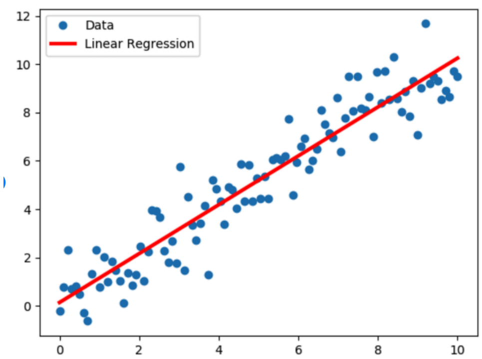

Figure 1: 样本特征空间 $\chi = \mathbb{R}$ 时的线性回归函数

但是在多项式回归中，由于含有指数不为 1 的自变量的项，图像呈现为一条曲线：

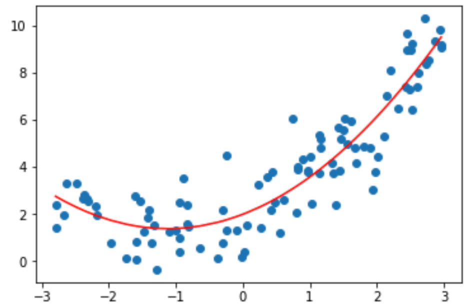

Figure 2: 样本特征空间 $\chi = \mathbb{R}$ 时的多项式回归函数

1.1.3 标准方程法

在多项式回归中，由于自变量的指数增大，写出损失函数后，对每个参数进行求导等操作的运算量也随之变大，对此我们希望能够将一些线性回归的知识迁移运用，简化计算形式，减少计算成本。

由 1.1.1 可知其表达形式，借此我们可以写出它的损失函数：

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - \boldsymbol{\varphi}_i^T\boldsymbol{\theta})^2 \tag{3}$$

整合之前学过的知识，我们要解决的凸优化问题为 $\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$，若损失函数图像具有强凸性，则 $\varphi^T\varphi$ 为正定矩阵，那么最小二乘法同样适用，具体推导可见讲义 L03 第 4 节，最优解为：

$$\boldsymbol{\theta}^* = (\boldsymbol{\varphi}^T\boldsymbol{\varphi})^{-1}\boldsymbol{\varphi}^T\boldsymbol{y} \tag{4}$$

1.1.4 代码实现

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
# 生成含高斯噪声的训练集，含50个样本

X = 5 * np.random.rand(50, 1) - 2.5
# 这里减去2.5是为了方便我们看到整个散点图轮廓
noise = np.random.rand(50, 1)
y = 0.6 * X ** 2 + 2 * X + 1 + noise
plt.scatter(X, y)

'''
从散点图判断用二次函数或可拟合，
我们要用原有的数据集X生成方便非线性转为线性解决的新数据集 ，
这里我们可以用sklearn库中现成的polynomialfeatures模块函数，也
    可以自己构建
'''

pf = PolynomialFeatures(degree=2, include_bias=True)
# 该函数用于将原来的数据集转化为1.1.1中(2)式的矩阵
# degree=2表示多项式的阶数为2，include_bias为True代表包含x_0
    =1的项
train_X = pf.fit_transform(X)

# 标准方程法
theta = np.zeros([3, ])
theta = np.linalg.inv((train_X.transpose()) @ train_X) @
    train_X.transpose() @ y
# @符号为python运算中的矩阵点乘符号
```

```python
# 现在我们已经得到了参数，绘制函数，查看拟合效果
x = np.linspace(-3, 3, 100)
hypothesis_function = theta[0] + theta[1] * x + theta[2] * x
    * x
plt.plot(x, hypothesis_function, color='darkorange')

plt.show()

```

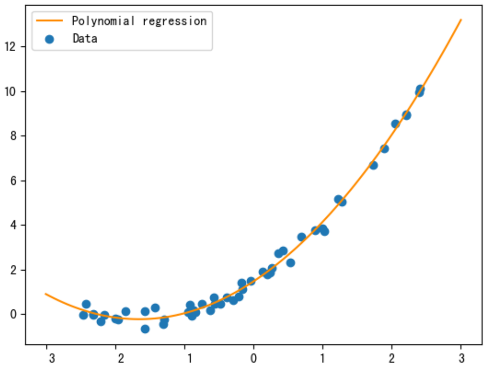

Figure 3: 代码运行结果图像

1.2 径向基回归（Radial-basis Regression）

定义：径向基函数

径向基函数是一个取值仅依赖于到原点距离的实值函数，即 $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$，任一满足 $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$ 的函数都可称作径向函数。

1.2.1 高斯核函数

在这里我们主要介绍最常用的高斯函数，其表达形式如下：

$$\begin{gathered} \hat{y_j} = \sum_{i=1}^{m} \theta_i \varphi_i(x_j) = \boldsymbol{\theta}^T \boldsymbol{\varphi_i}, j = 1, 2, 3, ..., n \\ \varphi_i(x_j) = K_{\lambda_i}(x_j, r_i) = e^{-\frac{(x_j - r_i)^2}{2\lambda_i^2}} \end{gathered} \tag{5}$$

r_i 为均值，λ_i 为方差。矩阵的表达形式为：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} K_{\lambda_1}(x_1, r_i) & K_{\lambda_2}(x_1, r_i) & \dots & K_{\lambda_m}(x_1, r_i) \\ K_{\lambda_1}(x_2, r_i) & K_{\lambda_2}(x_2, r_i) & \dots & K_{\lambda_m}(x_2, r_i) \\ K_{\lambda_1}(x_3, r_i) & K_{\lambda_2}(x_3, r_i) & \dots & K_{\lambda_m}(x_3, r_i) \\ \vdots & \vdots & \ddots & \vdots \\ K_{\lambda_1}(x_n, r_i) & K_{\lambda_2}(x_n, r_i) & \dots & K_{\lambda_m}(x_n, r_i) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix}$$

$$\boldsymbol{y} = \boldsymbol{\varphi\theta} \tag{6}$$

随着均值 r_i 和方差 λ_i 的变化，高斯函数的图像也会发生相应的变化，下图为一维特征和二维特征下对应的高斯函数图像：

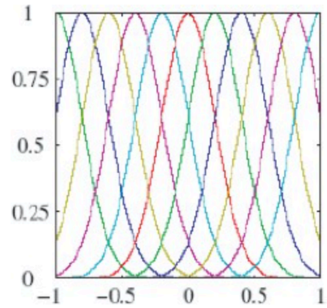

Figure 4: 对应着不同均值的高斯函数图像

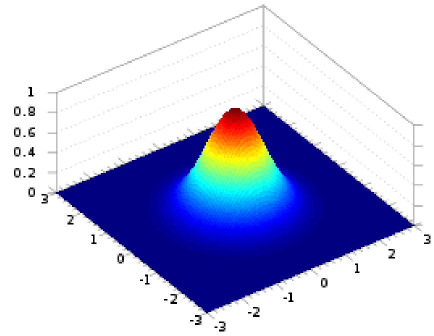

Figure 5: 二维特征、在三维空间内的高斯分布图像

数据的拟合函数曲线是这些均值方差不同的高斯分布函数即 $K_{\lambda_j}(x, r_i)$ 的加权叠加。此外，由于 $K_{\lambda_j}(x, r_i)$ 本身是 e 的指数函数，所以它的值必大于 0，$\varphi^T\varphi \succ 0$，可用标准方程法，得其最优解为：

$$\boldsymbol{\theta}^* = (\boldsymbol{\varphi}^T\boldsymbol{\varphi})^{-1}\boldsymbol{\varphi}^T\boldsymbol{y} \tag{7}$$

1.2.2 例子

从下面给出的样本点分布图可以看出，这应该是由两个高斯函数组成，

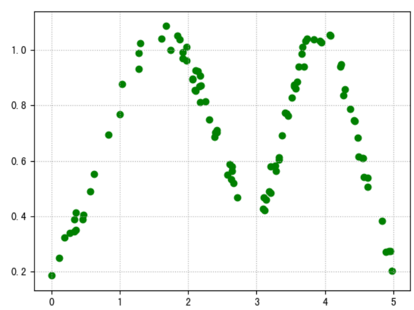

Figure 6: 样本点分布图

为矩阵 φ 选择合适的 $K_{\lambda_j}(x, r_i)$，根据得到的拟合曲线不断调试 r_i，λ_i，计算 $\boldsymbol{\theta}$，得到最终结果：

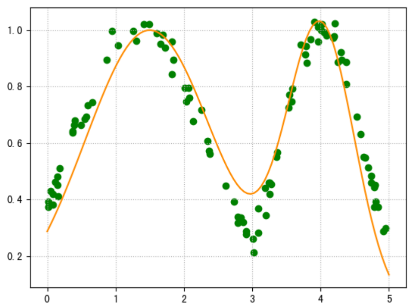

Figure 7: 拟合后的图像

1.2.3 参数的选取

正如学习率之于梯度下降，在 RBF 回归模型中，$K_{\lambda_j}(x, r_i)$ 的 r_i，λ_i 的大小也至关重要，r_i 决定了中心点的位置，而 λ_i 决定了覆盖范围，也可以理解为“胖瘦”，λ_i 越大，图像越“胖”，见下图：

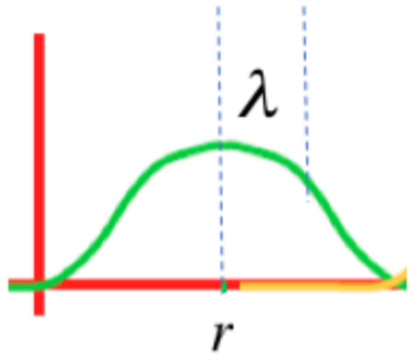

Figure 8: 高斯函数图像

一个好的高斯模型，不仅中心点要分布合理，即圆的圆心要适当，同时还要能够覆盖足够多的样本点，故而圆的半径需要足够宽，这样构建出来的 RBF 模型才有足够好的泛化能力。

- 例：样本点分布在二维特征空间中

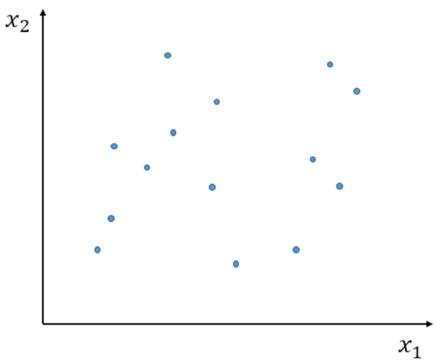

Figure 9: 样本点平面投影图像

如果我们的选取样本点为圆心，且圆的半径太小，那么这样构建出来的 RBF 模型泛化能力很弱（如图 10），如果放在三维空间看，状似“群山耸立”。

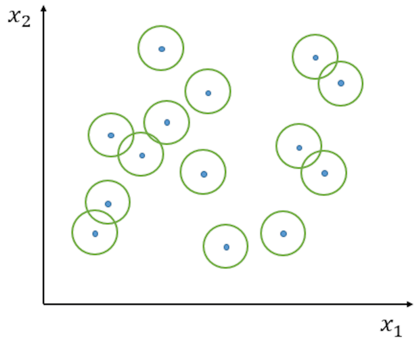

Figure 10: 拟合极差的 RBF 模型平面投影图像

选取适当的点，并设置恰当的半径，调整参数，重新拟合。

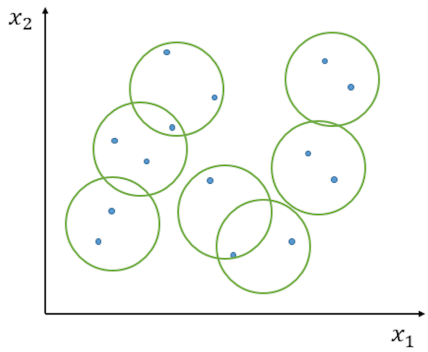

Figure 11: 拟合较好的 RBF 模型平面投影图像

1.3 Sigmoid 函数

Sigmoid 函数得名因其形状像 S 字母。一种常见的 S 函数是逻辑函数：

$$S(t) = \frac{1}{1 + e^{-t}} \tag{8}$$

其标准函数图像为：

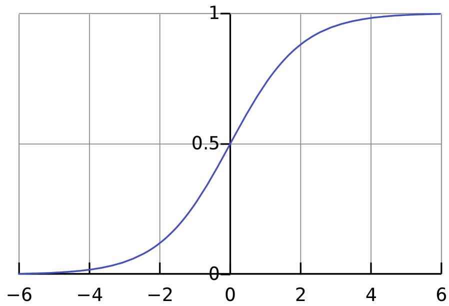

Figure 12: 标准 sigmoid 函数

在后续课程的逻辑回归中，该函数将发挥作用，我们可以用它来解决二分类问题。而在早期神经网络中，它作为激活函数被广泛运用，一方面可以引入了非线性，另一方面，由于其值域在 0-1 之间，亦可用作输出层，表示概率。

2 模型选择

到目前为止，我们模型的目的都是为了在获得的数据 X 上进行预测并获得尽量小的误差。然而，一个好的预测不应该只考虑过去已知的数据，还应该考虑未来将要获得的数据。这是模型一般化的要求，即通过已知的数据来进行学习，使得预测系数在未来获得的数

据上仍然具有好的预测效果。因此，一个机器学习模型在实际使用时通常会先在训练数据集 $\mathcal{D}_{\text{train}}$ 上进行训练，并在测试数据集 $\mathcal{D}_{\text{test}}$ 上测试模型的一般性。在调整我们的模型以获得一般性的过程中，根据模型在 $\mathcal{D}_{\text{train}}$ 和 $\mathcal{D}_{\text{test}}$ 上误差的不同，我们的模型可能会进入两种状态：欠拟合（underfitting）与过拟合（overfitting）。

2.1 欠拟合与过拟合

定义：欠拟合

当我们的模型过于简单而无法达到需要的效果时，模型在 $\mathcal{D}_{\text{train}}$ 和 $\mathcal{D}_{\text{test}}$ 上的预测误差都会很大。这时我们称模型欠拟合。

定义：过拟合

当我们的模型过于复杂而学到了一些“噪声”时，尽管模型在 $\mathcal{D}_{\text{train}}$ 上预测误差较小，但在 $\mathcal{D}_{\text{test}}$ 上的预测误差会很大。这时的模型缺少一般性，我们称模型过拟合。

图13展示了出现欠拟合与过拟合的情况。假设我们有一个变量 x 和对应的标签 y。当我们假设 $y = \theta_0 + \theta_1 x$ 时（图13左图），只考虑了 x 的一阶，因此模型的拟合效果不好，处于欠拟合的状态。而当我们考虑更高阶的情况时（图13右图），模型会变的复杂。这时的模型虽然能给够完美拟合每一个样本，但拟合出的曲线显然不具有一般性。因此我们需要谨慎地设置模型来避免过拟合。

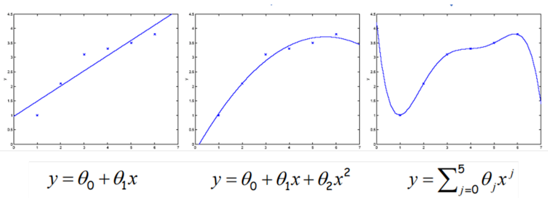

Figure 13: 欠拟合与过拟合。

3 训练-测试法

前面提到，对于一个模型，我们通常在训练集上训练，再在测试集上测试。这种模型学习方法称为训练-测试法。

定义：训练-测试法

将已知数据集划分为训练集与测试集，仅使用训练集训练设计的模型，用训练好的模型尝试预测测试集每个样本的标签。通常地，我们计算出均方误差 MSE，用于评估该模型的准确性，并且可以与其他模型比较。

3.1 优点

- 原理简单，容易实现。
- 比较多个模型时非常直观，只需要选择 MSE 最低的模型使用即可。

3.2 缺点

- 测试集的数据被完全浪费掉了，训练模型时根本没有利用。
- 如果数据集不够大，或者选取测试集样本时随机性不够高，会使测试结果不够客观准确，也即该方法的偏差（Variance）较大。

4 交叉验证

传统的训练-测试评估法中，需要取出相当一部分数据用于测试，势必造成数据的浪费。同时，在许多情况下数据集中数据量较少，去除测试集后训练效果不佳，因此我们希望能有一种充分利用所有数据的模型评估方法。

定义：K 折交叉验证

交叉验证是指，数据集中的每个数据既可用作训练，也可用作测试，根据测试的总体情况来评估模型的方法。K 折则表示将整个数据集随机等分为 K 份，交叉验证时轮流使用每份数据作为测试集，进行 K 次训练-测试。

在训练-测试法中，我们通常采用均方误差（MSE）作为模型的评价标准。同样地，在交叉验证中也常用模型在测试集上的 MSE 作为依据。

交叉验证中 K 的选择较为关键。较大的 K 有助于减少数据浪费，但同时增加了训练和测试的耗时；较低的 K 节约时间，但可能会导致学习数据量不够。

4.1 交叉验证在线性回归上的应用

记第 k 次训练得到的参数为 θ_k，T_k 为将数据集 K 等分后的第 k 组数据（在第 k 次训练后作为测试集），则此次测试的 MSE 为

$$J_k = \frac{1}{|T_k|} \sum_{(x_i, y_i) \in T_k} (y_i - \theta_k^T x_i)^2, \tag{9}$$

其中 $|T_k|$ 为 T_k 中样本的数量，即 $\frac{n}{K}$。交叉验证得到的误差为

$$J_{CV} = \frac{1}{K} \sum_{k=1}^{K} J_k. \tag{10}$$

4.2 选择合适的 K

下图列出了一些常用的 K 取值及其比较，在实际问题中需要根据情况选择合适的 K，从而得到理想的结果。

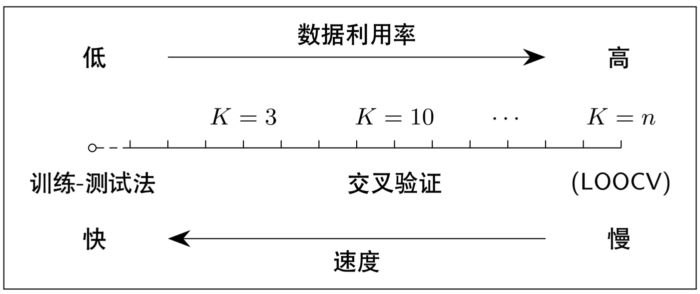

Figure 14: K 对交叉验证性能的影响

5 基于交叉验证的模型选择

当我们使用同样的 K 对两个模型进行交叉验证后，就可以对比它们的误差 J_{CV} 来判断模型的性能。通常来说，我们可以同时计算出模型在对应训练集上的 MSE（称为训练 MSE）供模型选择参考。下图是一个例子，对 3 个不同模型进行交叉验证后画出误差柱状图。

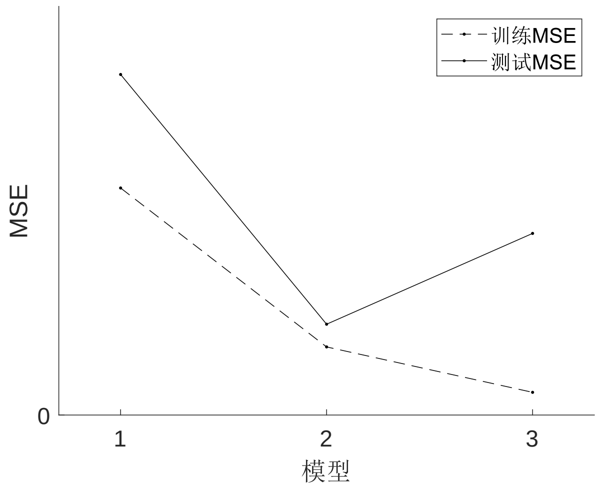

Figure 15: 交叉验证选择模型的一个例子

通过上图也可以清楚看出，欠拟合即指在训练集和测试集上 MSE 都较高（如模型 1），过拟合则指训练 MSE 极低、测试 MSE 较高（如模型 3）。取舍之后，我们认为模型 3 表现较好，可以进行进一步训练或使用等后续步骤。

6 交叉验证的一个实现

本节我们用 Python 语言简单实现线性回归的交叉验证过程。

```python
import numpy as np

# 设定维度和样本数，与数据集一致
p = 10
n = 100

# 设定交叉验证折数K
K = 10

# 准备数据集文件"train.csv"，每行形式为(x1, x2, ..., xp, y)
dataset = read('train.csv') # 读取数据集存入ndarray对象，该read函数需要另外实现
x_train = dataset[...,:-1]  # 待回归样本
y_train = dataset[...,-1] # 对应的标签值

### 交叉验证过程 ###
# train, test函数需要另外实现

Jcv = 0.0           # 交叉验证MSE

for i in range(K):
  x_train_k = np.concatenate([
          x_train[:i*n/K],
          x_train[(i+1)*n/K:]
        ]) # 不用真的将数据集切分，利用索引切片即可
  y_train_k = np.concatenate([
          y_train[:i*n/K],
          y_train[(i+1)*n/K:]
        ])
  x_test_k = x_train[i*n/K : (i+1)*n/K]
  y_test_k = y_train[i*n/K : (i+1)*n/K]

  W = train(x_train_k, y_train_k)   # 训练得到参数
  J_k = test(W, x_test_k, y_test_k) # 测试得到MSE

  Jcv += J_k

Jcv /= K
print('交叉验证MSE为', Jcv)
```