第 10 章 K 近邻法

第 10-13 章使用“非参数方法”(nonparametric approach)进行监督学习。

最简单的非参数方法就是 **K 近邻法**(K Nearest Neighbors，简记 KNN)，即以最近的K个邻居进行预测，由 Fix and Hodges (1951)提出。

10.1 回归问题的 K 近邻法

首先考虑回归问题。假设响应变量y连续，而$\mathbf{x}$为p维特征向量。根据第 4 章命题 4.1，能使均方误差最小化的函数为条件期望函数$\mathrm{E}(y \mid \mathbf{x})$。

问题是，在实践中应如何估计$\mathrm{E}(y \mid \mathbf{x})$？

如果对于任意给定$\mathbf{x}$，均有很多不同的y观测值，则可对这些y值进行简单算术平均，参见图 10.1。

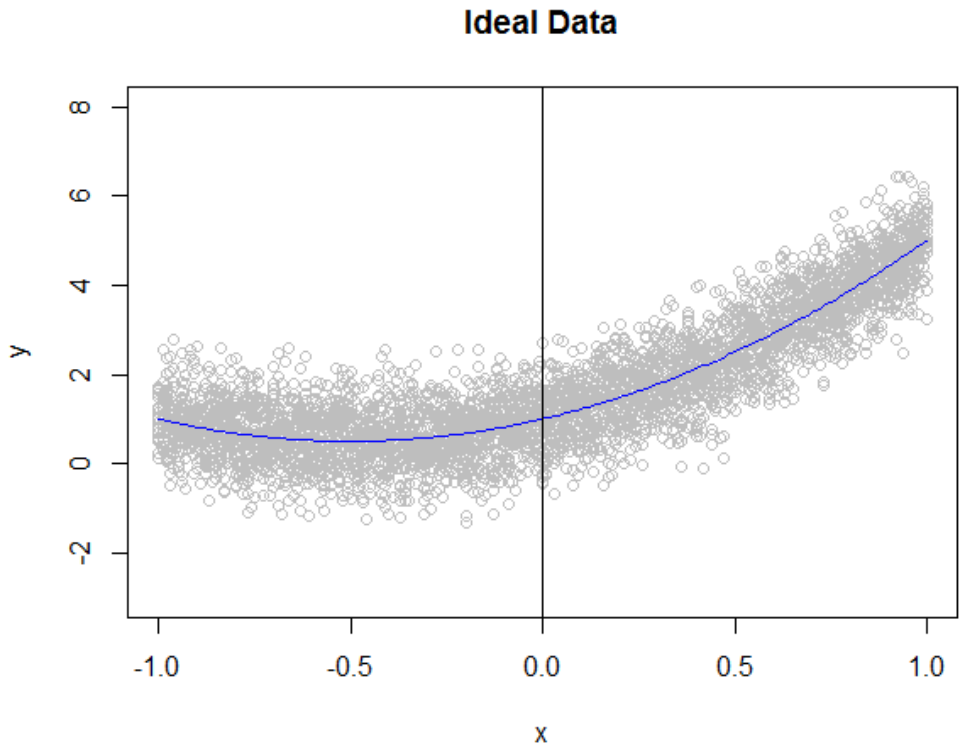

图 10.1 理想的数据

但现实数据大多比较稀疏，比如图 10.2。此时，给定$\mathbf{x}$，可能只有很少的y观测值，甚至连一个y观测值也没有。

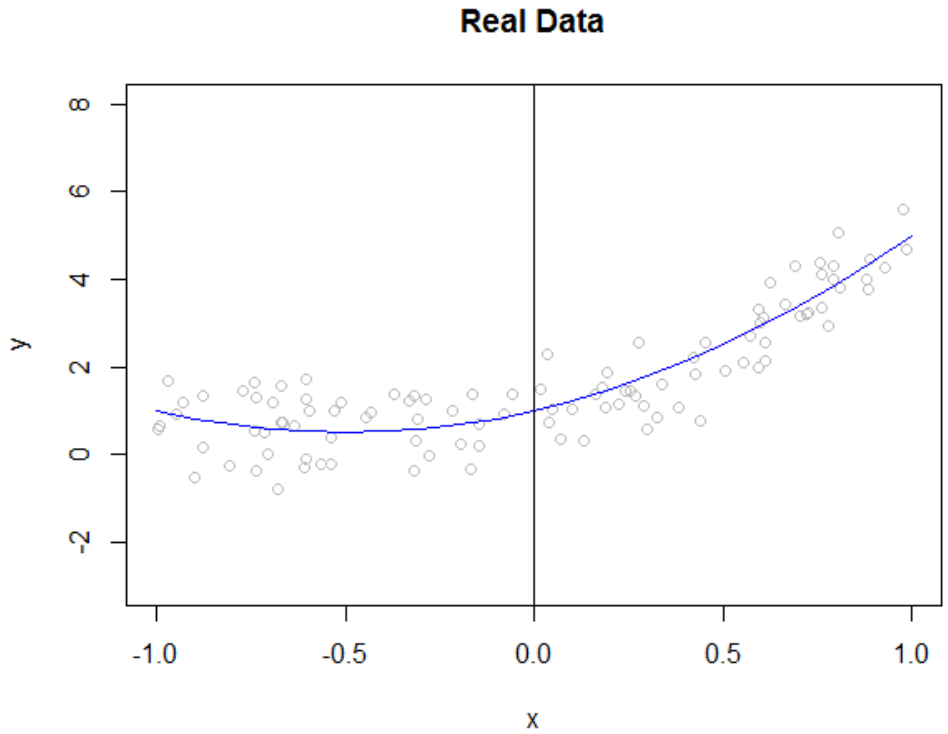

图 10.2 现实的数据

一个解决方法是在特征空间(feature space)中，考虑离$\mathbf{x}$最近的K个邻居。

记$N_K(\mathbf{x})$为最靠近$\mathbf{x}$的K个观测值$\mathbf{x}_i$的集合。

K近邻估计量(K nearest neighbor estimator)以离$\mathbf{x}$最近的K个邻居之y观测值的平均作为预测值：

$$\hat{f}_{KNN}(\mathbf{x}) \equiv \frac{1}{K} \sum_{\mathbf{x}_i \in N_K(\mathbf{x})} y_i \qquad (10.1)$$

如果$K=1$，则为“最近邻法”(nearest neighbor)，即以离$\mathbf{x}$最近邻居的y观测值作为预测值。

为了找到最近的K个邻居，首先需要在特征空间中，定义一个距离函数。通常以欧氏距离(即L_2范数)作为此距离函数，即$\left\|\mathbf{x}_i-\mathbf{x}\right\|_2$。

使用 KNN 法的一个前提是，所有特征变量均为数值型；否则，将无法计算欧氏距离。

为避免某些变量对于距离函数的影响太大，一般建议先将所有变量**标准化**(standardization)，即减去该变量的均值，再除以其标准差，使得所有变量的均值为 0 而标准差为 1。

10.2 如何选择 K

在进行 KNN 估计时，一个重要选择为如何确定K。本质上，这依然有关“偏差与方差的权衡”(bias and variance trade-off)。

在一个极端，如果$K=1$(最近邻法)，则偏差较小(仅使用最近邻的信息)，但方差较大(未将近邻的y观测值进行平均)。所估计的回归函数很不规则，导致“过拟合”(overfit)，使得模型的泛化能力(generalization)较差。

在另一极端，如果K很大，则偏差较大(用到更远邻居的信息)，而方差较小(用更多观测值进行平均)。如果$K=n$，则使用样本均值进行所有预测，导致偏差很大，出现“欠拟合”(underfit)。如果K太大，回归函数将过于光滑，未能充分捕捉数据中的信号，也使得算法的泛化能力下降。

最优的邻居数K应在偏差与方差之间保持较好的权衡。

在实践中，一般可用交叉验证(Cross-validation)选择最优的K。

比如，对于$K=1,\cdots,50$，分别计算相应的交叉验证误差(CV Error)，然后选择使交叉验证误差最小的K值。

下面使用一个关于摩托车撞击实验的数据 mcycle，来演示 KNN 回归。

mcycle 是非参数统计的一个经典数据集。

首先，导入所需模块：

```
In [1]: import numpy as np
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: import seaborn as sns
   ...: from sklearn.neighbors import
            KNeighborsRegressor
```

其次，载入数据文件 mcycle.csv，并考察其形状：

```
In [2]: mcycle = pd.read_csv('mcycle.csv')
   ...: mcycle.shape
Out[2]: (133, 2)
```

结果显示，数据框 mcycle 包含 133 个观测值与 2 个变量。考察前 5 个观测值：

```
In [3]: mcycle.head()
Out[3]:
   times  accel
0    2.4    0.0
1    2.6   -1.3
2    3.2   -2.7
3    3.6    0.0
4    4.0   -2.7
```

结果显示，数据框 mcycle 的两个变量分别为 times(时间，以毫秒计)与 accel(加速度)。

更直观地，使用 seaborn 的 scatterplot()函数画散点图，结果参见图 10.3：

```
In [4]: sns.scatterplot(x='times', y='accel',
                        data=mcycle)
   ...: plt.title('Simulated Motorcycle Accident')
```

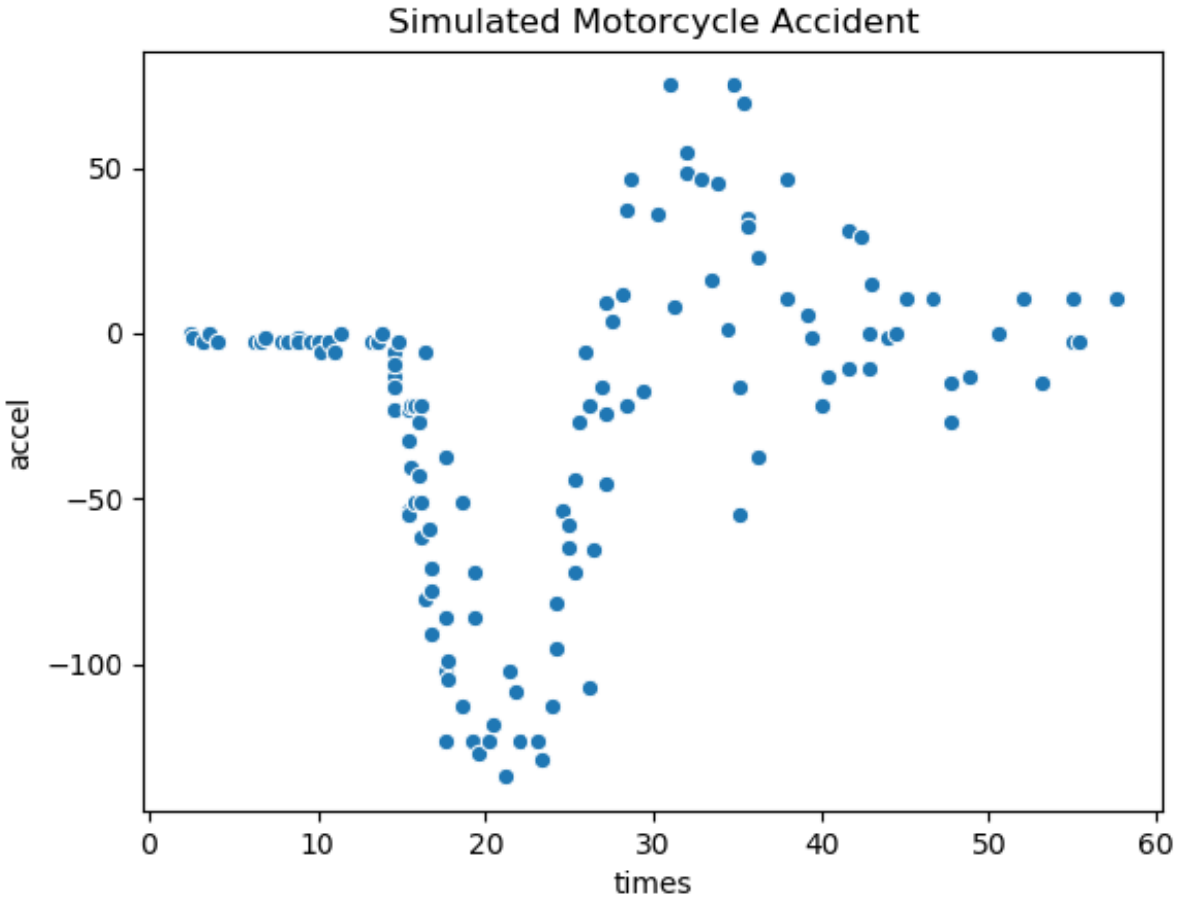

图 10.3 数据框 mcycle 的散点图

图 10.3 显示，加速度(accel)是关于时间(times)的一个高度非线性的函数。因此，统计学常使用 mcycle 数据演示非参数回归。

下面，使用 sklearn 模块的 KNeighborsRegressor 类进行 KNN 回归。先取出特征变量：

```
In [5]: X = np.array(mcycle.times).reshape(-1, 1)
```

其中，“np.array(mcycle.times)”将 mcycle.times 变为 Numpy 的一维数组(向量)，再通过 reshape(-1, 1)方法，变为133×1的矩阵(“-1”表示不指定行数)；这是因为 sklearn 一般不接受一维数组作为数据矩阵。

然后，取出响应变量：

```
In [6]: y = mcycle.accel
```

我们将使用 for 循环，分别以$K=1,10,25,50$进行 KNN 回归，然后把所得结果在2×2的画布上展示。可输入如下命令：

```
In [6]: fig, ax = plt.subplots(2, 2, figsize=(9, 6),
                               sharex=True, sharey=True)
   ...: fig.subplots_adjust(hspace=0.5, wspace=0.5)
   ...: for i,k in zip([1, 2, 3, 4], [1, 10, 25, 50]):
   ...:     model = KNeighborsRegressor(n_neighbors=k)
   ...:     model.fit(X, y)
```

```
...:     pred =
        model.predict(np.arange(60).reshape(-1, 1))
...:     plt.subplot(2, 2, i)
...:     sns.scatterplot(x='times', y='accel', s=20,
        data=mcycle,facecolor='none', edgecolor='k')
...:     plt.plot(np.arange(60), pred, 'b')
...:     plt.text(0, 55, f'K = {k}')
...: plt.tight_layout()
```

其中，zip()为 Python 内建函数，“zip([1, 2, 3, 4], [1, 10, 25, 50])”返回一个可迭代的 zip 对象，将列表[1, 2, 3, 4]与[1, 10, 25, 50]以“拉链”方式构成一个元组的序列(参见第 2 章)。

将此 zip 对象变为列表，进行考察：

```
In [8]: list(zip([1, 2, 3, 4], [1, 10, 25, 50]))
Out[8]: [(1, 1), (2, 10), (3, 25), (4, 50)]
```

因此，在上述 for 循环中，依次将循环变量(i, k)赋值为(1, 1)，(2, 10)，(3, 25)与(4, 50)。

其中，“i”用于指定在画布上画第 i 个子图(subplot)，而 k 指定 KNN 回归的 k 个邻居。

在循环体中，命令“model = KNeighborsRegressor(n_neighbors=k)”生成 KNeighborsRegressor 类的一个实例 model，并指定参数为 k 个邻居；

然后用命令“`model.fit(X, y)`”进行 KNN 回归的估计。

命令“`pred = model.predict(np.arange(60).reshape(-1, 1))`”以$[0,60)$区间中的整数进行预测。

接着，画原始样本的散点图，以及 KNN 回归的拟合图。

其中，`sns.scatterplot()`的参数“`facecolor='none'`”表示无填充颜色，即空心圆形；

参数“`edgecolor='k'`”将空心圆形的边界设为黑色。画图结果参见图 10.4。

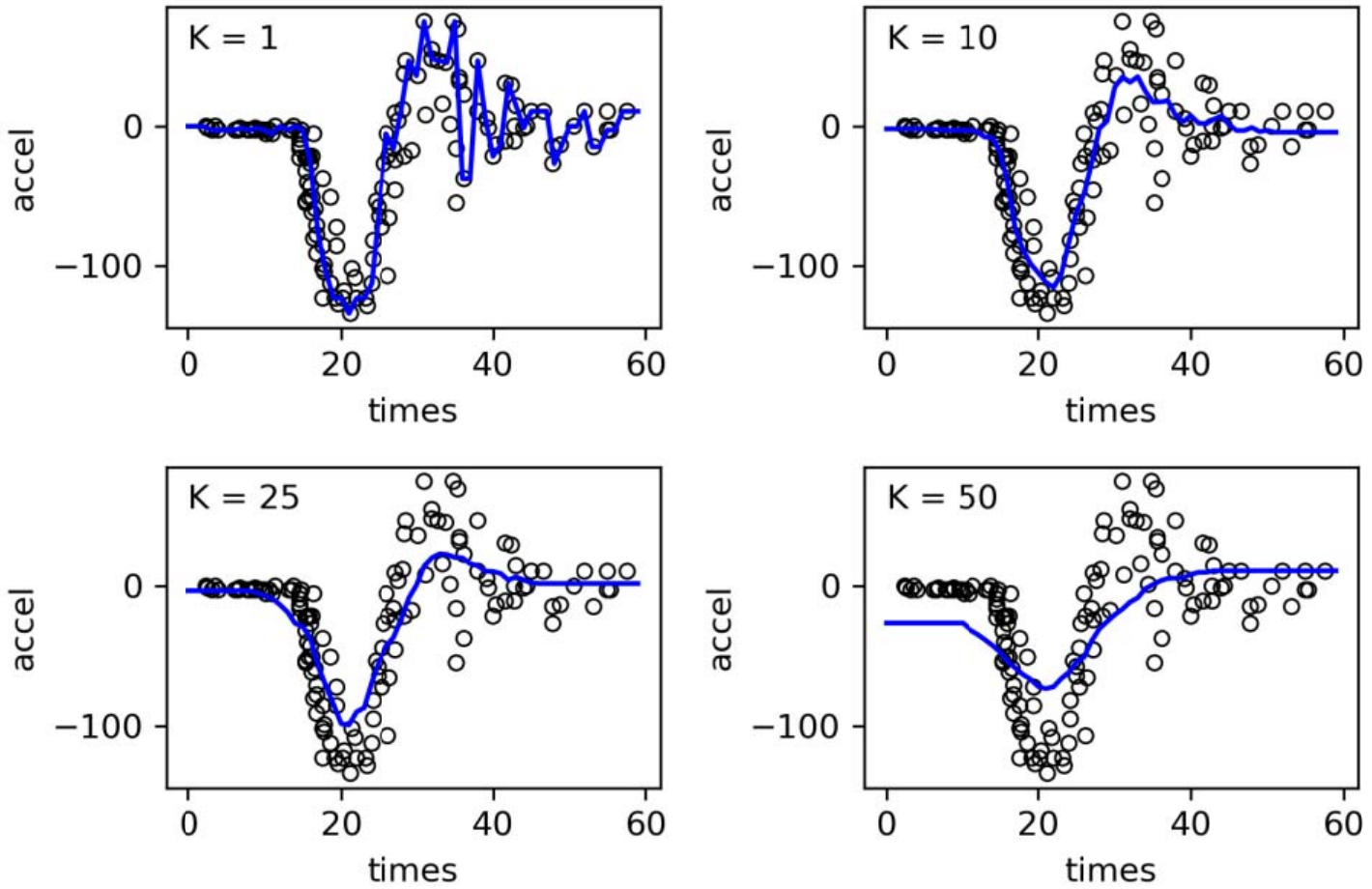

图 10.4 KNN 回归的拟合效果

在图 10.4 的左上角，为$K=1$(最近邻法)的拟合效果。回归函数非常不光滑，呈锯齿状跳跃。此时，训练误差很小，但显然存在过拟合(overfit)，模型的泛化能力较差。

在图 10.4 的右上角，为$K=10$的拟合效果。回归函数虽然仍不太光滑，但较好地抓住了数据的特征，拟合程度比较合适(good fit)。

在图 10.4 的左下角，为$K=25$的拟合效果。此时，回归函数更为光滑，但已不能充分捕捉数据的趋势特征，存在欠拟合(underfit)。

在图 10.4 的右下角，为$K=50$的拟合效果。回归函数已不能捕捉数据的主要特征，存在严重的欠拟合。

10.3 分类问题的 K 近邻法

对于分类问题，同样可使用 KNN 法。

特征变量$\mathbf{x}$依然为数值型，而响应变量y则为离散型，仅取有限的几个值，比如$y \in \{1, 2, \cdots, J\}$，共分为$J$类。

给定$\mathbf{x}$，在进行 KNN 估计时，首先确定离$\mathbf{x}$最近的K个邻居之集合$N_K(\mathbf{x})$。

然后，采取**多数票规则**(majority vote rule)，即以K个近邻中最常见的类别作为预测。

如果K近邻中最常见的类别有两个或多个并列(ties)，则可随机选一个最常见类别作为预测结果。

对于分类问题进行 KNN 估计时，如何选择K同样重要。

在一个极端，如果$K=1$(最近邻法)，则偏差较小，但方差较大。最近邻法($K=1$)虽可使得训练误差为 0(完美解释训练数据)，但决策边界通常很不规则，导致过拟合，使得模型的泛化能力较差。

在另一极端，如果K很大，则偏差较大，而方差较小。在极端情况下，如果$K=n$，则使用样本中最常见的类别进行所有预测。如果K太大，则决策边界将过于光滑，无法充分捕捉数据中的信号，使得算法的泛化能力下降，导致欠拟合。

下面以 iris 数据为例，直观地展示不同K值，对于分类问题的决策边界之影响。首先，导入所需模块：

```
In [1]: import matplotlib.pyplot as plt
   ...: from sklearn.datasets import load_iris
   ...: from sklearn.neighbors import
            KNeighborsClassifier
   ...: from mlxtend.plotting import
            plot_decision_regions
```

其中，KNeighborsClassifier 为 sklearn 模块用于进行 KNN 分类的“类”(class)；而 plot_decision_regions 为 mlxtend 模块画决策边界的函数。

其次，载入 iris 数据：

```
In [2]: X, y = load_iris(return_X_y=True)
   ...: X2 = X[:, 2:4]
```

其中，为便于展示决策边界，仅使用 iris 数据的后两个特征变量，即 petal_length 与 petal_width，构成数据矩阵 X2。

使用 for 循环，考察$K=1,10,25,50$，对于 KNN 决策边界的不同影响，并以2×2的画布展示。可输入如下命令，结果参见图 10.5(画此图较费时)：

```
In [3]: fig, ax = plt.subplots(2, 2, figsize=(9, 6),
                               sharex=True, sharey=True)
   ...: fig.subplots_adjust(hspace=0.1, wspace=0.1)
   ...: for i, k in zip([1, 2, 3, 4], [1, 10, 25, 50]):
   ...:     model =
                    KNeighborsClassifier(n_neighbors=k)
   ...:     model.fit(X2, y)
   ...:     plt.subplot(2, 2, i)
   ...:     plot_decision_regions(X2, y, model)
   ...:     plt.xlabel('petal_length')
   ...:     plt.ylabel('petal_width')
   ...:     plt.text(0.3, 3, f'K = {k}')
   ...: plt.tight_layout()
```

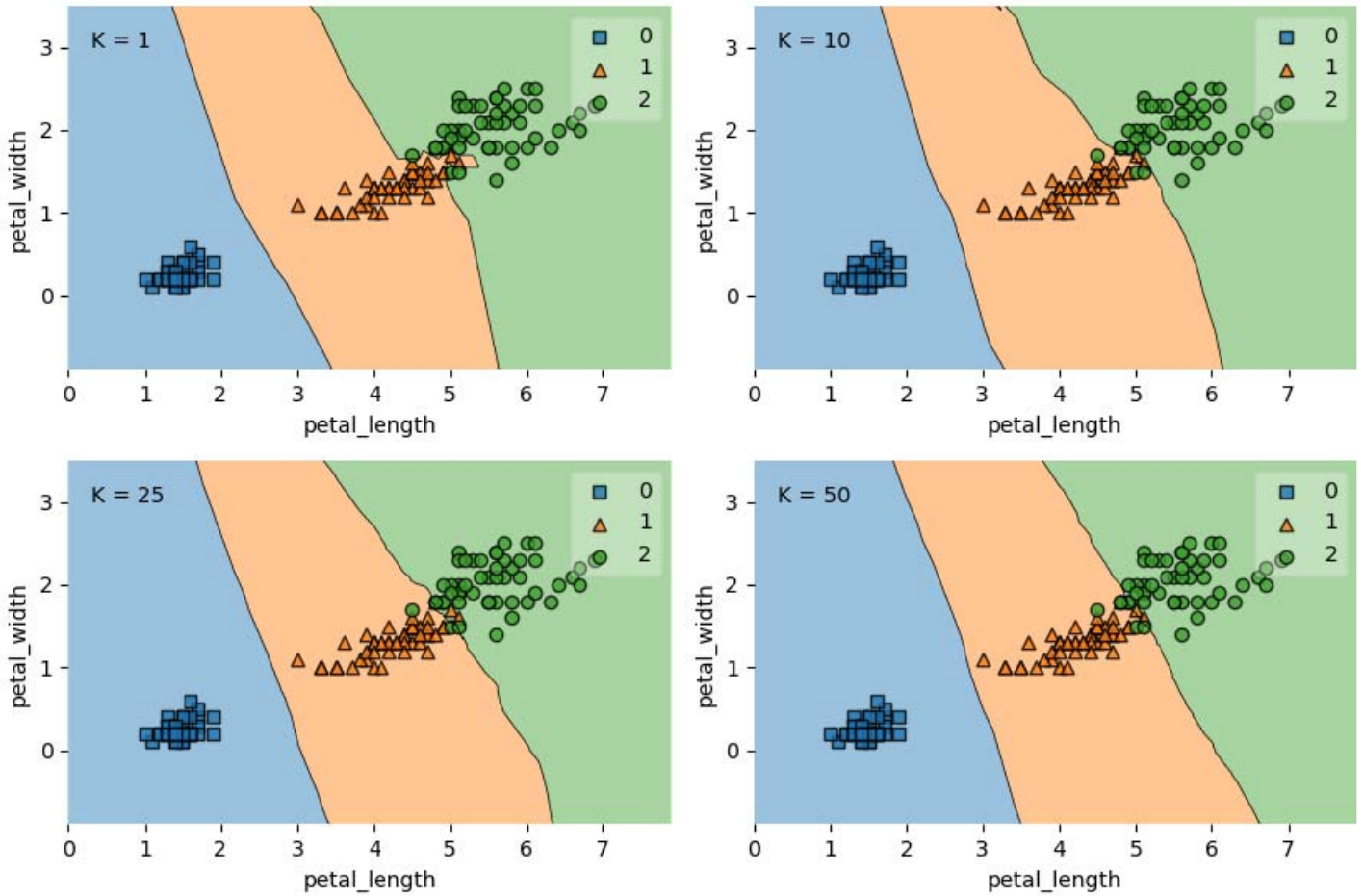

图 10.5 KNN 分类的决策边界

从图 10.5 可见，随着K的增大，KNN 决策边界变得越来越光滑，可能导致“欠拟合”(underfit)。

由于 `iris` 数据过于简单，即使$K=1$，也不容易看到“过拟合”(overfit)现象。

10.4 *K* 近邻法的优缺点

KNN 的优点之一是算法简单易懂。

作为非参数估计，KNN 不依赖于具体的函数形式，故较为稳健。

但 KNN 所估计的回归函数或决策边界一般较不规则。

这是因为，当$\mathbf{x}$在特征空间变动时，其K个邻居之集合$N_K(\mathbf{x})$可能发生不连续的变化，即加入新邻居而去掉旧邻居。

因此，如果真实的回归函数或决策边界也较不规则，则 KNN 效果较好。

KNN 是一种“懒惰学习”(lazy learning)。

KNN 算法平时不学习，属于“死记硬背型”(memory-based)，并没有估计真正意义上的模型。

KNN 直到预测时才去找邻居，导致预测较慢，不适用于“在线学习”(online learning)，即需要实时预测的场景。

另外，在高维空间中可能很难找到邻居，会遇到所谓“维度灾难”(curse of dimensionality)，此时 KNN 算法的效果可能较差。

KNN 算法一般要求$n \gg p$，即样本容量n须远大于特征向量的维度p。

KNN 对于“噪音变量”(noise variable)也不稳健。如果特征向量$\mathbf{x}_i$包含对y_i毫无作用的噪音变量，KNN 在计算观测值之间的欧氏距离时，也依然不加区别地对待这些变量，导致估计效率下降。

10.5 *K* 近邻法的 Python 案例

使用威斯康辛乳腺癌(Wisconsin breast cancer)的数据演示KNN分类算法。

此数据来自威斯康辛大学麦迪逊分校医院，包含 569 位病人的观测值，以及与乳腺癌诊断有关的 32 个变量(有关肿瘤细胞核的各种特征)。

响应变量 `diagnosis` 取值为 `0`(恶性肿瘤)或 `1`(良性肿瘤)。

* 详见教材，以及配套 Python 程序（现场演示）。