8

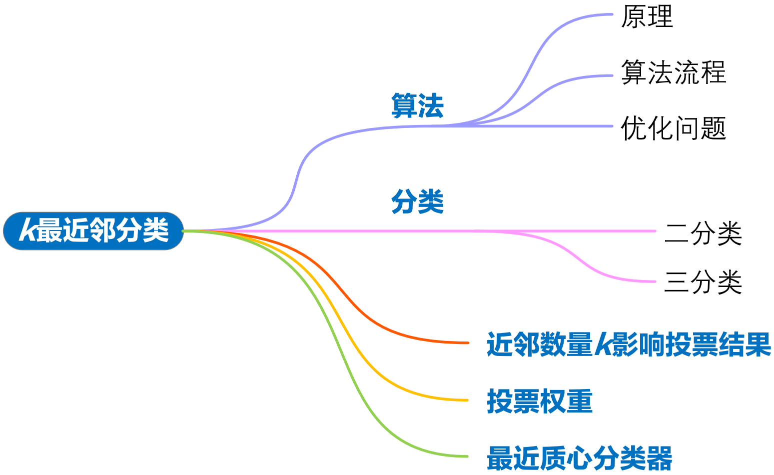
k-nearest neighbors algorithm

k 最近邻分类

小范围投票，少数服从多数

> 如果一台计算机能够欺骗人类，让人类相信它也是人类一员；那么，这台计算机值得被称作智能机器。
>
> ***A computer would deserve to be called intelligent if it could deceive a human into believing that it was human.***
>
> —— 艾伦·图灵 (Alan Turing) | 英国计算机科学家、数学家，人工智能之父 | 1912 ~ 1954

- ◀ `enumerate()` 函数用于将一个可遍历的数据对象，比如列表、元组或字符串等，组合为一个索引序列，同时列出数据和数据下标，一般用在 `for` 循环当中
- ◀ `matplotlib.pyplot.contour()` 绘制等高线线图
- ◀ `matplotlib.pyplot.contourf()` 绘制填充等高线图
- ◀ `matplotlib.pyplot.scatter()` 绘制散点图
- ◀ `numpy.array()` 创建 `array` 数据类型
- ◀ `numpy.c_()` 按列叠加两个矩阵
- ◀ `numpy.diag()` 如果 `A` 为方阵，`numpy.diag(A)` 函数提取对角线元素，以向量形式输入结果；如果 `a` 为向量，`numpy.diag(a)` 函数将向量展开成方阵，方阵对角线元素为 `a` 向量元素
- ◀ `numpy.linalg.inv()` 计算逆矩阵
- ◀ `numpy.linalg.norm()` 计算范数
- ◀ `numpy.linspace()`产生连续均匀向量数值
- ◀ `numpy.meshgrid()` 创建网格化数据
- ◀ `numpy.r_()` 按行叠加两个矩阵
- ◀ `numpy.ravel()` 将矩阵扁平化
- ◀ `sklearn.neighbors.KNeighborsClassifier` 为 k-NN 分类算法函数；函数常用的 mehods 为 `fit(X, y)` 和 `predit(q)`；`fit(X, y)`用来加载样本数据，`predit(q)`用来预测查询点 `q` 的分类
- ◀ `sklearn.neighbors.NearestCentroid` 最近质心分类算法函数


*k*最近邻分类
算法
原理
算法流程
优化问题
分类
二分类
三分类
近邻数量*k*影响投票结果
投票权重
最近质心分类器

8.1 *k* 近邻分类原理：近朱者赤，近墨者黑

***k* 近邻算法** (*k*-nearest neighbors algorithm, *k*-NN) 是最基本监督学习方法之一。这种算法的优点是简单易懂，不需要训练过程，对于非线性分类问题表现良好。

然而，它也存在一些缺点，例如需要大量存储训练集、预测速度较慢、对于高维数据容易出现维数灾难等。此外，在选择 *k* 值时需要进行一定的调参工作，以保证算法的准确性和泛化能力。

⚠ 注意，*k*-NN 中的 *k* 指的是“近邻”的数量。

原理

k-NN 思路很简单——“近朱者赤，近墨者黑”。更准确地说，小范围投票，少数服从多数 (majority rule)，如图 1。

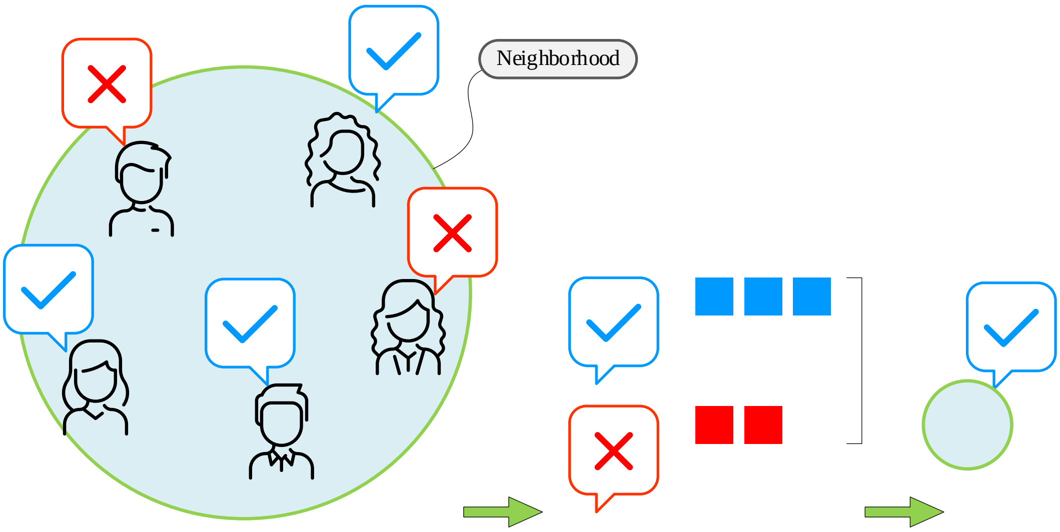


图 1. *k* 近邻分类核心思想——小范围投票，少数服从多数

算法流程

给定样本数据 $\boldsymbol{X}$ ($\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}$)，分别对应已知标签 $\boldsymbol{y}$ ($y^{(1)}, y^{(2)}, \ldots, y^{(n)}$)。**查询点** (query point) $\boldsymbol{q}$ 标签未知，待预测分类。

k-NN 近邻算法流程如下：

- 计算样本数据 $\boldsymbol{X}$ 任意一点 $\boldsymbol{x}$ 和查询点 $\boldsymbol{q}$ 距离；
- 找 $\boldsymbol{X}$ 中距离查询点 $\boldsymbol{q}$ 最近的 *k* 个样本，即 *k* 个“近邻”；
- 根据 *k* 个邻居已知标签，直接投票或加权投票；*k* 个邻居出现数量最多的标签即为查询点 $\boldsymbol{q}$ 预测分类结果。

优化问题

用公式表示，k-NN 算法的优化目标如下，**预测分类** (predicted classification) $\hat{y}$：

$$\hat{y}(\boldsymbol{q}) = \arg\max_{C_k} \sum_{i \in kNN(\boldsymbol{q})} I\left(y^{(i)} = C_k\right) \tag{1}$$

其中，kNN($\boldsymbol{q}$)为查询点 $\boldsymbol{q}$ 近邻构成的集合，C_k 为标签为 C_k 的样本数据集合，$k = 1, 2, \ldots, K$。I 为**指示函数** (indicator function)，表示“一人一票”；当 $y^{(i)} = C_k$ 成立时，$I = 1$；否则，$I = 0$。

下面以二分类为例，和大家讲解如何理解 k-NN 算法。

8.2 二分类：非红，即蓝

平面可视化

假设，数据 $\boldsymbol{X}$ 有两个特征，即 $D = 2$；$\boldsymbol{X}$ 两个特征分别为 x_1 和 x_2。也就是说，在 x_1x_2 平面上，$\boldsymbol{X}$ 的第一列数值为横坐标，$\boldsymbol{X}$ 的第二列数值为纵坐标。

$\boldsymbol{y}$ 有两类标签 $K = 2$，即 C_1 和 C_2；红色 ● 表示 C_1，蓝色 ● 表示 C_2。

$\boldsymbol{X}$ 和 $\boldsymbol{y}$ 数据形式及平面可视化如图 2 所示。

显然这是个**二分类** (binary classification, bi-class classification) 问题，查询点 $\boldsymbol{q}$ 的分类可能是 C_1 (红色 ●)，或者 C_2 (蓝色 ●)。

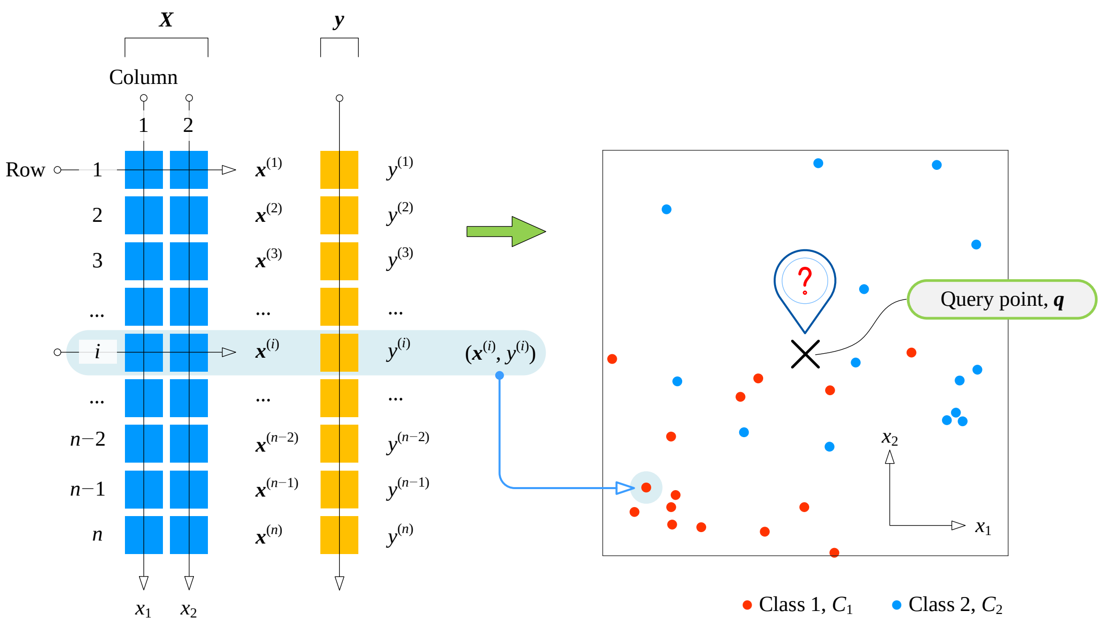


图 2. 两特征 ($D = 2$) 含标签样本数据可视化

四个近邻投票

对于二分类问题，即 $K = 2$，(1) 可以写成：

$$\hat{y}(\boldsymbol{q}) = \max_{C_1, C_2} \left\{ \sum_{i \in kNN(\boldsymbol{q})} I\left(y^{(i)} = C_1\right), \sum_{i \in kNN(\boldsymbol{q})} I\left(y^{(i)} = C_2\right) \right\} \tag{2}$$

在图 3 所示平面上，× 为查询点 $\boldsymbol{q}$，以行向量表达。

如果设定“近邻”数量 $k = 4$，以查询点 $\boldsymbol{q}$ 为圆心圈定的圆形“近邻社区”里有 4 个样本数据点 ($\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$、$\boldsymbol{x}^{(3)}$ 和 $\boldsymbol{x}^{(4)}$)。4 个点中，样本点 $\boldsymbol{x}^{(1)}$ 距离查询点 $\boldsymbol{q}$ 距离 d_1 最近，样本点 $\boldsymbol{x}^{(4)}$ 距离查询点 $\boldsymbol{q}$ 距离 d_4 最远。

显然，查询点 $\boldsymbol{q}$ 近邻社区中四个查询点中，投票为“三比一”——3 个“近邻”标签为 C_1 (红色 ●)，1 个“近邻”标签为 C_2 (蓝色 ●)。也就是：

$$\begin{aligned} \sum_{i \in kNN(\boldsymbol{q})} I\left(y^{(i)} = C_1\right) = 3 \\ \sum_{i \in kNN(\boldsymbol{q})} I\left(y^{(i)} = C_2\right) = 1 \end{aligned} \tag{3}$$

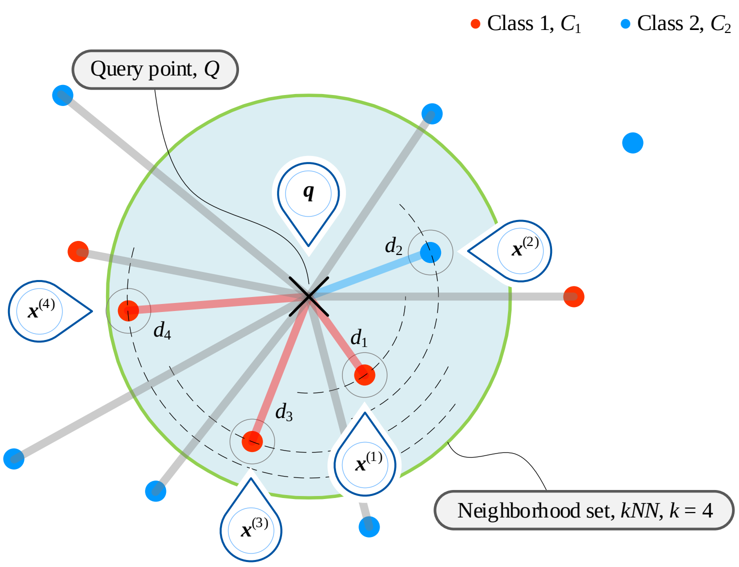


图 3. k 近邻原理

将具体分类标签带入 (2)，可以得到：

$$\hat{y}(\boldsymbol{q}) = \max_{C_1, C_2} \left\{ 3_{(C_1)}, 1_{(C_2)} \right\} = C_1 \tag{4}$$

由于近邻不分远近，投票权相同。图 3 中距离线段线宽代表投票权。少数服从多数，在 $k = 4$ 的条件下，红色 ● “胜出”！因此，查询点 $\boldsymbol{q}$ 的预测分类为 C_1 (红色 ●)。

需要引起注意的是，近邻数量 k 是自定义输入；观察图 3 可以发现，当 k 增大时，查询点 $\boldsymbol{q}$ 的预测分类可能会发生变化。

下一节将会讨论近邻数量 k 如何影响分类预测结果。

使用函数

`sklearn.neighbors.KNeighborsClassifier` 为 Scikit-learn 工具包 *k*-NN 分类算法函数。函数默认的近邻数量 `n_neighbors` 为 5，默认距离度量 `metric` 为**欧氏距离** (Euclidean distance)。这个函数常用的 methods 为 `fit(X, y)` 和 `predit(q)`；`fit(X, y)` 用来拟合样本数据，`predit(q)` 用来预测查询点 $\boldsymbol{q}$ 的分类。

《数据有道》专门总结机器学习中常见距离度量。

8.3 三分类：非红，要么蓝，要么灰

鸢尾花分类问题为三分类问题，即 $K = 3$。图 4 每个圆点 ● 代表一个数据点。其中，● 代表分类为 setosa (C_1, $y = 0$)，● 代表 versicolor (C_2, $y = 1$)，● 代表 virginica (C_3, $y = 2$)。

图 4 所示为利用 `KNeighborsClassifier` 获得的鸢尾花分类结果。输入数据选取鸢尾花数据 2 个特征——花萼长度 x_1，和花萼宽度 x_2。用户输入的近邻数量 `n_neighbors` 为 4。请大家注意，图 4 平面一些位置数据点存在叠合，也就是说一个圆点代表不止一个数据点。

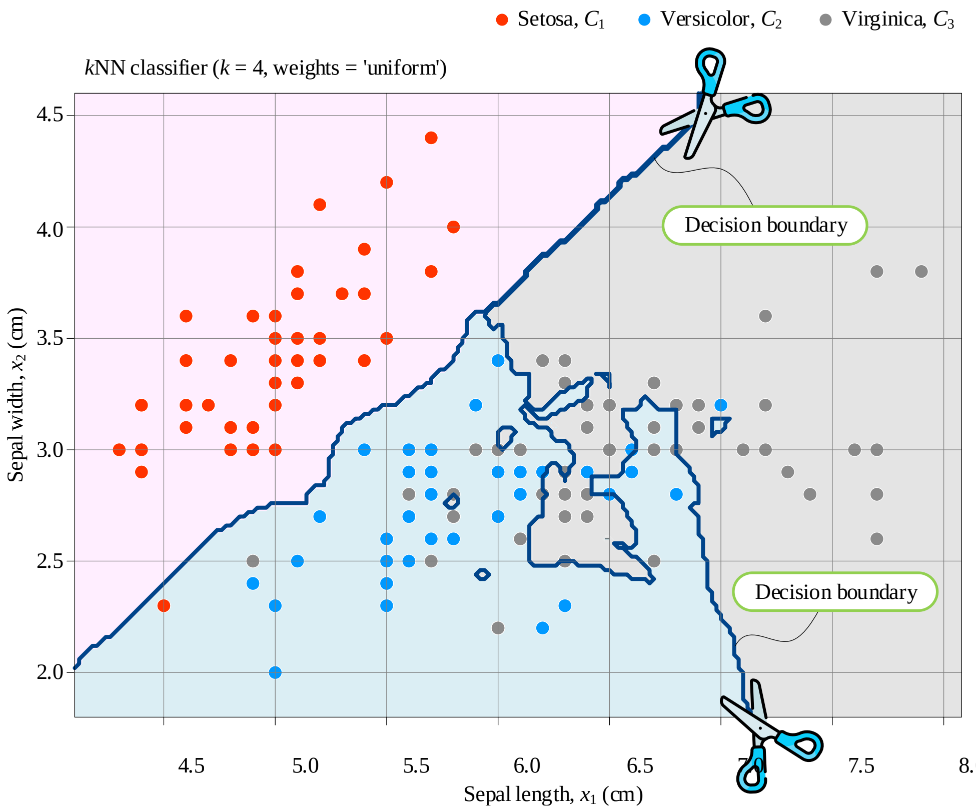


图 4. *k* 近邻分类，$k = 4$，采用 2 个特征 (花萼长度 x_1，和花萼宽度 x_2) 分类三种鸢尾花

注意，欧几里德距离，也称欧氏距离，是最常见的距离度量，本章出现的距离均为欧氏距离。此外，本节利用直接投票 (等权重投票)，而本章第三节将讲解加权投票原理。

决策边界

图 4 中深蓝色曲线为**决策边界** (decision boundary)。如果决策边界是直线、平面或超平面，那么这个分类问题是线性的，分类是线性可分的；否则，分类问题非线性。图 4 所示 *k*-NN 算法决策边界杂乱无章，肯定是非线性，甚至不可能用某个函数来近似。

很多分类算法获得的决策边界都可以通过简单或者复杂函数来描述，比如一次函数、二次函数、二次曲线等等；这类模型也称**参数模型** (parametric model)。与之对应的是，类似 *k*-NN 这样的学习算法得到的决策边界为**非参数模型** (non-parametric model)。

k-NN 基于训练数据，更准确地说是把训练数据以一定的形式存储起来完成学习任务，而不是泛化得到某个解析解进行数据分析或预测。

所谓**泛化能力** (generalization ability) 是指机器学习算法对全新样本的适应能力。适应能力越强，泛化能力越强；否则，泛化能力弱。

举个简单例子解释“泛化能力弱”这一现象；一个学生平时做了很多练习题，每道练习题目都烂熟于心；这个学生虽然刻苦练习，可惜他就题论题，不能举一反三，考试做新题时，分数总是很低。

每当遇到一个新查询点，*k*-NN 分类器分析这个新查询点与早前存储样本数据的关系，并据此把一个预测分类值赋给新查询点。值得注意的是，这些样本数据是以树形结构存储起来，常见的算法是 *kd* 树。

提醒大家注意，学习每一种学习算法时，注意观察决策边界形状特点，并总结规律。

代码 Bk7_Ch08_01.ipynb 可以用来实现本节分类问题，并绘制图 4。下面聊聊其中关键语句。

```
# 近邻数量
a k_neighbors = 4

# kNN分类器
b clf = neighbors.KNeighborsClassifier(k_neighbors)

# 拟合数据
c clf.fit(X, y)

# 查询点
d q = np.c_[xx1.ravel(), xx2.ravel()];

# 预测
e y_predict = clf.predict(q)

# 规整形状
f y_predict = y_predict.reshape(xx1.shape)
```

代码 1. 用 sklearn.neighbors.KNeighborsClassifier()分类 | Bk7_Ch08_01.ipynb

ⓐ 定义近邻的数量为 4，请大家尝试其他近邻数量。
ⓑ 用 sklearn.neighbors.KNeighborsClassifier()创建 kNN 分类对象。
ⓒ 调用 kNN 分类对象，并拟合数据。
ⓓ 这句话将网格坐标转化为二维数组。
ⓔ 对网格数据进行分类预测。

f 将预测结果规整为和网格数据相同形状，以便于后续可视化。

8.4 近邻数量 k 影响投票结果

近邻数量 k 为用户输入值，而 k 值直接影响查询点分类结果；因此，选取合适 k 值格外重要。本节和大家探讨近邻数量 k 对分类结果影响。

图 5 所示为 k 取四个不同值时，查询点 $\boldsymbol{q}$ 预测分类结果变化情况。如图 5 (a) 所示，当 $k = 4$ 时，查询点 $\boldsymbol{q}$ 近邻中，3 个近邻为 ● (C_1)，1 个近邻为 ● (C_2)；采用等权重投票，查询点 $\boldsymbol{q}$ 预测分类为● (C_1)。

当近邻数量 k 提高到 8 时，近邻社区中，4 个近邻为 ● (C_1)，4 个近邻为 ● (C_2)，如图 5 (b) 所示；等权重投票的话，两个标签各占 50%。因此 $k = 8$ 时，查询点 $\boldsymbol{q}$ 恰好在决策边界上。

如图 5 (c) 所示，当 $k = 12$ 时，查询点 $\boldsymbol{q}$ 近邻中 5 个为 ● (C_1)，7 个为 ● (C_2)；等权重投票条件下，查询点 $\boldsymbol{q}$ 预测标签为 ● (C_2)。当 $k = 16$ 时，如图 5 (d) 所示，查询点 $\boldsymbol{q}$ 预测标签同样为 ● (C_2)。

k-NN 算法选取较小的 k 值虽然能准确捕捉训练数据的分类模式；但是，缺点也很明显，容易受到噪声影响。

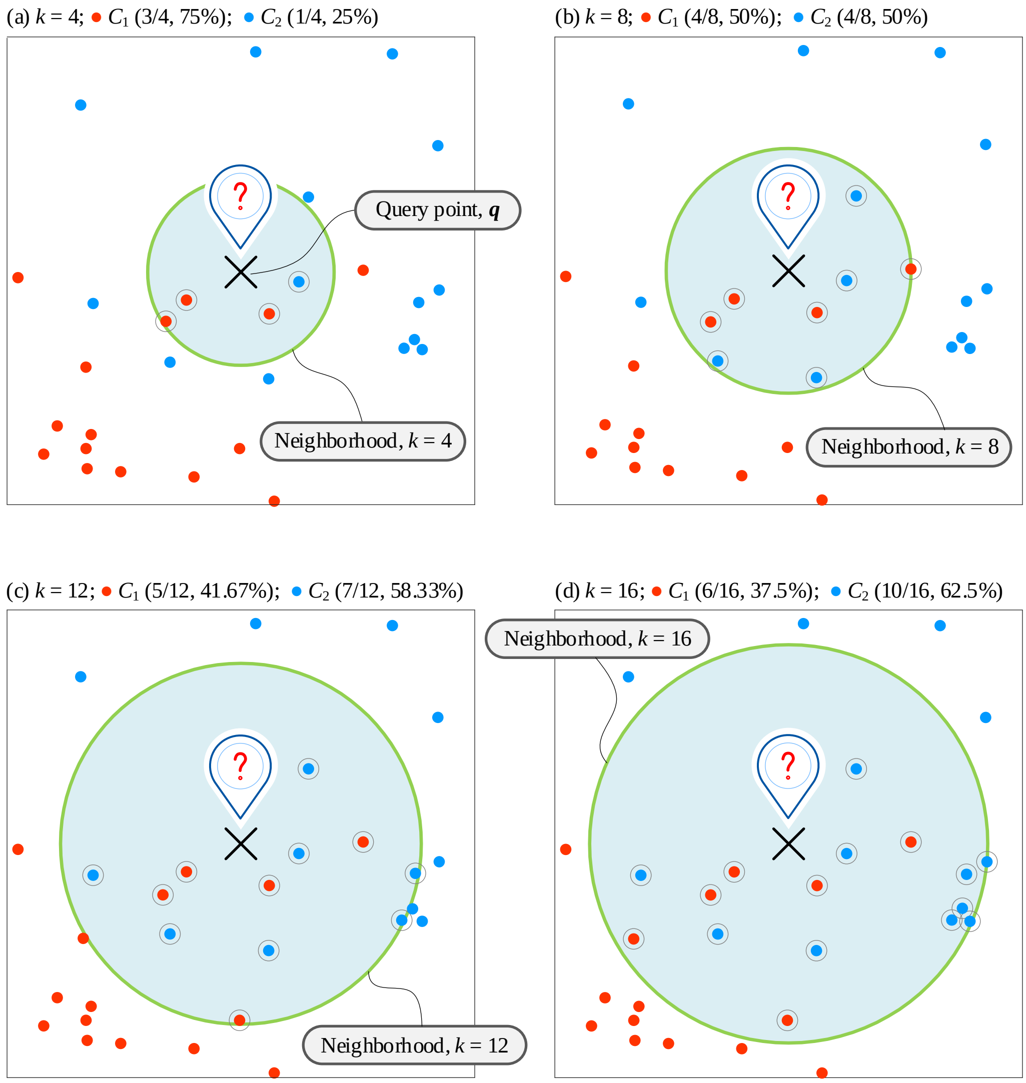


图 5. 近邻数量 k 值影响查询点的分类结果

影响决策边界形状

图 6 所示为 k 选取不同值时对鸢尾花分类影响。观察图 6 四副子图可以发现，当 k 逐步增大时，局部噪音样本对边界的影响逐渐减小，边界形状趋于平滑。

较大的 k 是会抑制噪声的影响，但是使得分类界限不明显。举个极端例子，如果选取 k 值为训练样本数量，即 $k = n$，采用等权重投票，这种情况不管查询 Bk7_Ch08_04 点 $\boldsymbol{q}$ 在任何位置，预测结果仅有一个。这种训练得到的模型过于简化，忽略样本数据中有价值的信息。

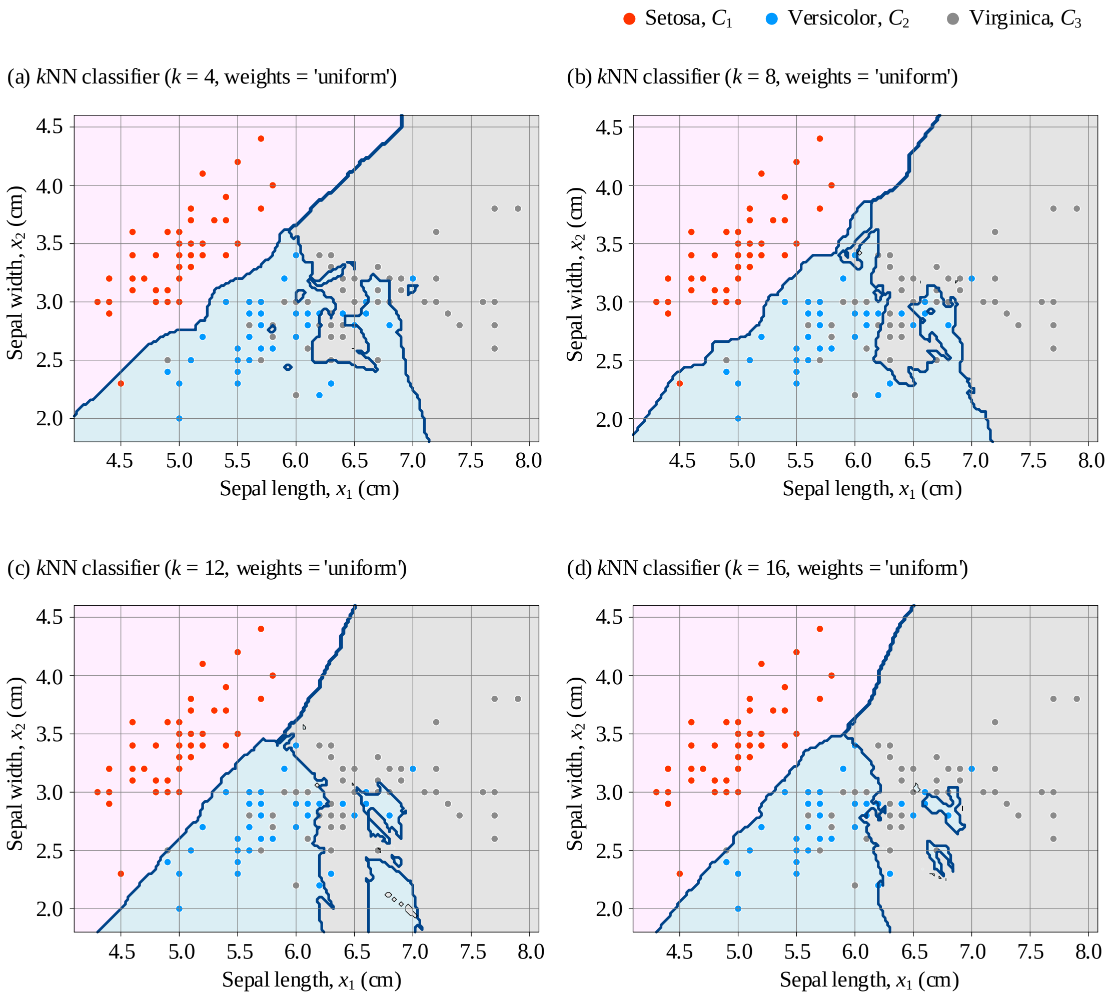


图 6. k-NN，k 选取不同值时对鸢尾花分类影响

图 7 所示为用 Streamlit 搭建的 App 展示 k 对 kNN 聚类结果影响。

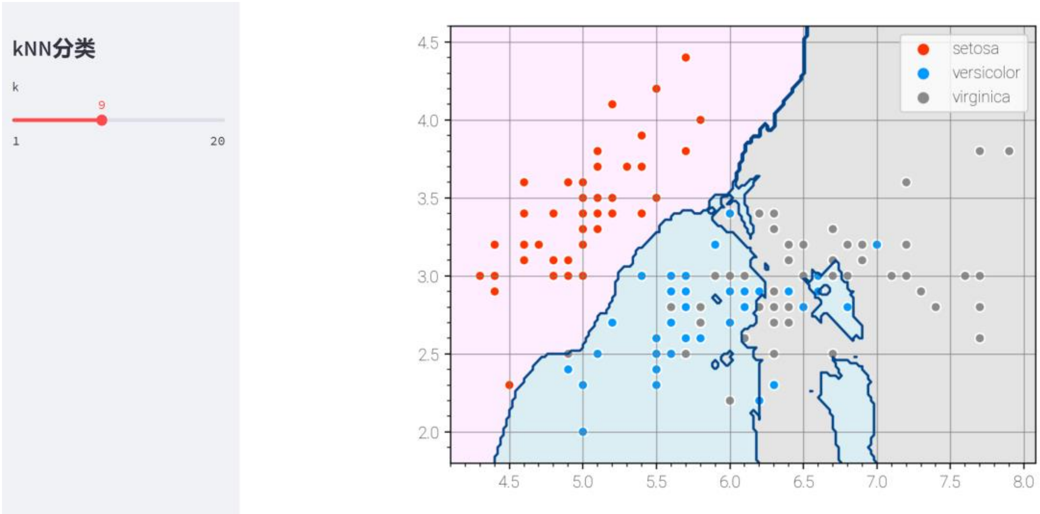


图 7. 展示 k 对 kNN 聚类结果影响的 App，Streamlit 搭建 | Streamlit_Bk7_Ch08_02.py

代码 Streamlit_Bk7_Ch08_02.py 搭建图 7 所示 App，请大家自行学习。

8.5 投票权重：越近，影响力越高

本章前文强调，在"近邻社区"投票时，采用的是"等权重"方式；也就是说，只要在"近邻社区"之内，无论距离远近，一人一票，少数服从多数。

前文 k 近邻分类函数，默认等权重投票，默认值 `weights = 'uniform'`。但是，很多 k 近邻分类问题采用加权投票则更有效。

如图 8 所示，每个近邻的距离线段线宽 w_i 代表各自投票权重。距离查询点越近的近邻，投票权重 w_i 越高；相反，越远的近邻，投票权重 w_i 越低。

对应的优化问题变成：

$$\hat{y}(\boldsymbol{q}) = \arg\max_{C_k} \sum_{i \in kNN(\boldsymbol{q})} w_i \cdot I\left(y^{(i)} = C_k\right) \tag{5}$$

`sklearn.neighbors.KNeighborsClassifier` 函数中，可以设定投票权重与查询点距离成反比，`weights = 'distance'`。

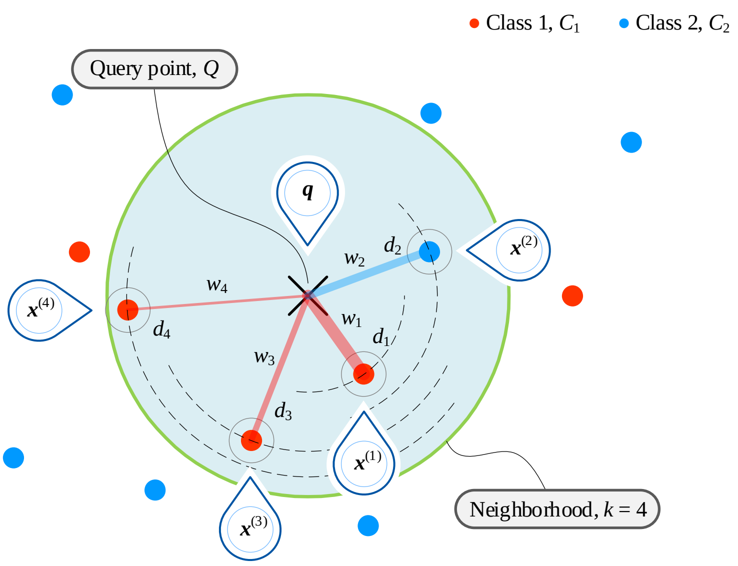


图 8. k 近邻原理，加权投票

此外，近邻投票权 w_i 还可以通过**归一化** (normalization) 处理，如下式：

$$w_i = \frac{\max(d_{NN}) - d_i}{\max(d_{NN}) - \min(d_{NN})} \tag{6}$$

d_{NN} 为所有近邻距离构成的集合，$\max(d_{NN})$ 和 $\min(d_{NN})$ 分别计算得到近邻距离最大和最小值。加权投票权重还可以采用距离平方的倒数，这种权重随着距离增大衰减越快。

使用 scikit-learn 的 kNN 分类器时，大家可以自定义加权投票权重函数。

决策边界

图 9 所示为，近邻数量为 k = 50 条件下，`weights = 'distance'`时，k 近邻分类算法获得决策边界。

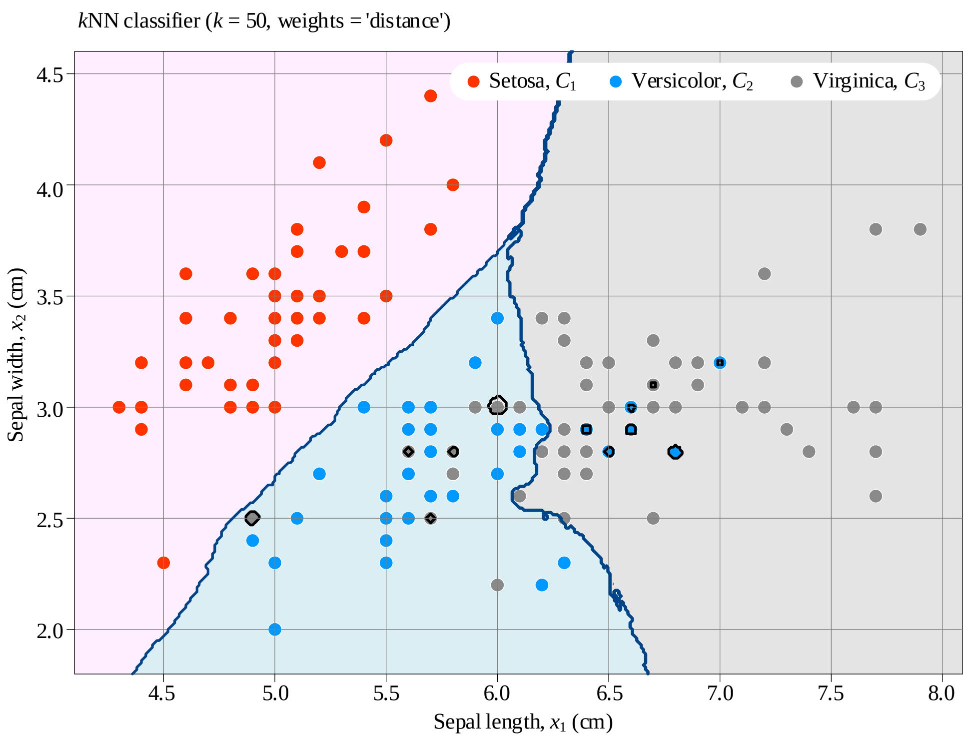


图 9. $k = 50$ 时，鸢尾花分类决策边界，投票权重与查询点距离成反比

8.6 最近质心分类：分类边界为中垂线

最近质心分类器 (Nearest Centroid Classifier, NCC) 思路类似 k-NN。

本章前文讲过，k-NN 以查询点为中心，圈定 k 个近邻，近邻投票。而最近质心分类器，先求解得到不同类别样本数据簇质心位置 $\boldsymbol{\mu}_m$ $(m = 1, 2, \ldots, K)$；查询点 $\boldsymbol{q}$ 距离哪个分类质心越近，其预测分类则被划定为这一类。因此，最近质心分类器不需要设定最近邻数量 k。

《矩阵力量》第 22 章已经讨论过**数据质心** (centroid) 这个概念，它的具体定义如下：

$$\boldsymbol{\mu}_k = \frac{1}{\operatorname{count}(C_k)} \sum_{i \in C_k} \boldsymbol{x}^{(i)} \tag{7}$$

其中，count() 计算某个标签为 C_k 的子集样本数据点的数量。

注意，上式假定 $\boldsymbol{x}^{(i)}$ 和 $\boldsymbol{\mu}_k$ 均为列向量。

分类函数

Python 工具包完成最近质心分类的函数为 `sklearn.neighbors.NearestCentroid`。图 10 所示为通过最近质心分类得到的鸢尾花分类决策边界。图 10 中 $\boldsymbol{\mu}_1$、$\boldsymbol{\mu}_2$ 和 $\boldsymbol{\mu}_3$ 三点分别为 ● setosa (C_1, $y = 0$)、● versicolor (C_2, $y = 1$)和 ● virginica (C_3, $y = 2$) 的质心所在位置。

大家可能已经发现，图 10 中每段决策边界就是两个质心的中垂线！

《矩阵力量》第 19 章讲解过中垂线，请大家回顾。

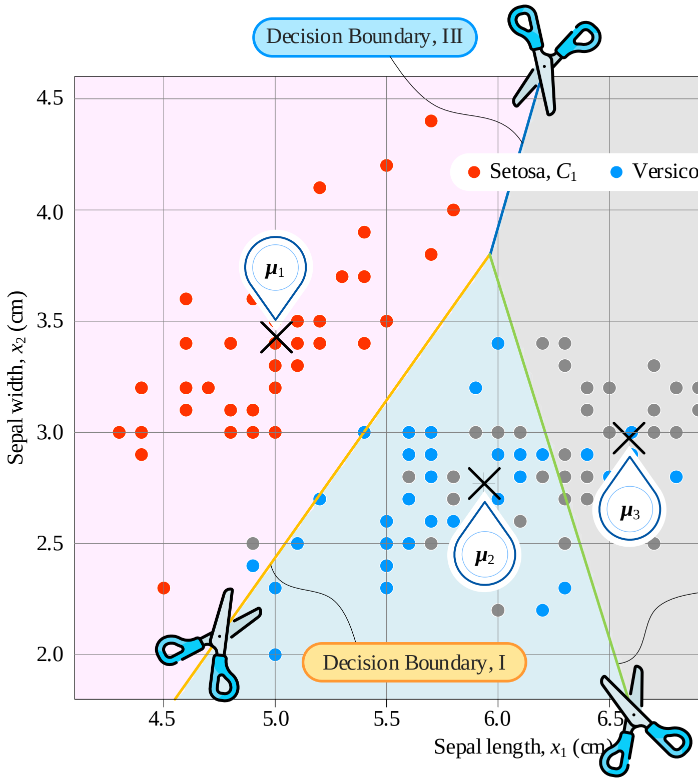


图 10. 鸢尾花分类决策边界，最近质心分类

图解原理

图 11 所示为最近质心分类器边界划分原理图。

平面上，A 和 B 两点中垂线上每一点距离 A 和 B 相等。中垂线垂直于 AB 线段，并经过 AB 线段中点。图 11 中决策边界无非就是，$\boldsymbol{\mu}_1$、$\boldsymbol{\mu}_2$ 和 $\boldsymbol{\mu}_3$ 三个质心点任意两个构造中垂线。

如图 11 所示，为了确定查询点 $\boldsymbol{q}$ 的预测分类，计算 $\boldsymbol{q}$ 到 $\boldsymbol{\mu}_1$、$\boldsymbol{\mu}_2$ 和 $\boldsymbol{\mu}_3$ 三个质心点距离度量。比较 AQ、BQ 和 CQ 三段距离长度，发现 CQ 最短，因此查询点 $\boldsymbol{q}$ 预测分类为 ● virginica (C_3)。

图 11 有专门的名字——**沃罗诺伊图** (Voronoi diagram)。本书将会在 K 均值聚类一章介绍。

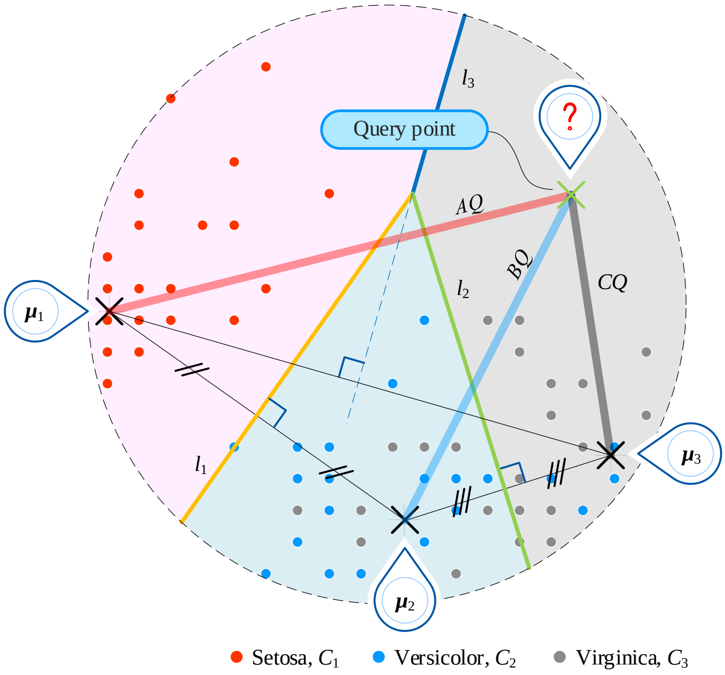


图 11. 最近质心分类决策边界原理

收缩阈值

sklearn.neighbors.NearestCentroid 函数还提供**收缩阈值** (shrink threshold)，获得**最近收缩质心** (nearest shrunken centroid)。说的通俗一点，根据收缩阈值大小，每个类别数据质心向样本数据总体质心 $\boldsymbol{\mu}_X$ 靠拢。图 12 展示的是随着收缩阈值不断增大，分类数据质心不断向 $\boldsymbol{\mu}_X$ 靠拢，分类边界不断变化的过程。

NearestCentroid 函数定义收缩阈值如何工作。对此感兴趣的话，大家可以自行打开 NearestCentroid 函数，查找 if self.shrink_threshold:对应的一段。

代码 Bk7_Ch08_03.ipynb 绘制图 12 所示四幅图像。

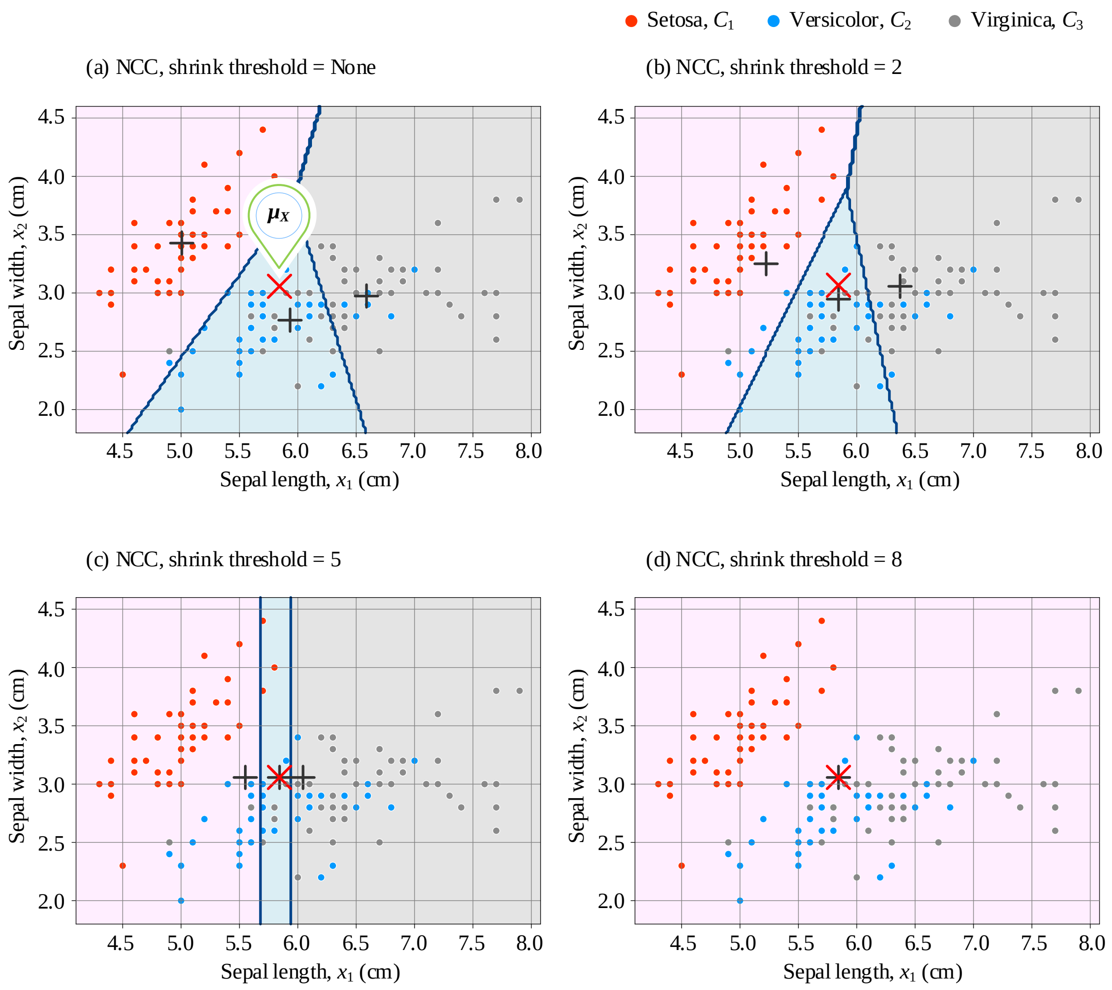


图 12. 收缩阈值增大对决策边界影响

8.7 *k*-NN 回归：非参数回归

本章前文的 *k*-NN 分类算法针对离散标签，比如 C_1 (红色 ●) 和 C_2 (蓝色 ●)。当输出值 y 为连续数据时，监督学习便是回归问题。本节讲解如何利用 *k*-NN 求解回归问题。

对分类问题，一个查询点的标签预测是由它附近 k 个近邻中占多数的标签决定；同样，某个查询点的回归值，也是由其附近 k 个近邻的输出值决定。

采用等权重条件下，查询点 $\boldsymbol{q}$ 回归值 $\hat{y}$ 可以通过下式计算获得：

$$\hat{y}(\boldsymbol{q}) = \frac{1}{k} \sum_{i \in kNN(\boldsymbol{q})} y^{(i)} \tag{8}$$

其中，kNN($\boldsymbol{q}$) 为查询点 $\boldsymbol{q}$ 的 k 个近邻构成的集合。

举个例子

如图 13 所示，当 $k = 3$ 时，查询点 Q 附近三个近邻 $\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$ 和 $\boldsymbol{x}^{(3)}$ 标记为蓝色 ●。这三个点对应的连续输出值分别为 $y^{(1)}$、$y^{(2)}$ 和 $y^{(3)}$。根据 (8) 计算 $y^{(1)}$、$y^{(2)}$ 和 $y^{(3)}$ 平均值，得到查询点回归预测值 $\hat{y}$：

$$\hat{y}(\boldsymbol{q}) = \frac{1}{3}\left(y^{(1)} + y^{(2)} + y^{(3)}\right) = \frac{1}{3}(5+3+4) = 4 \tag{9}$$

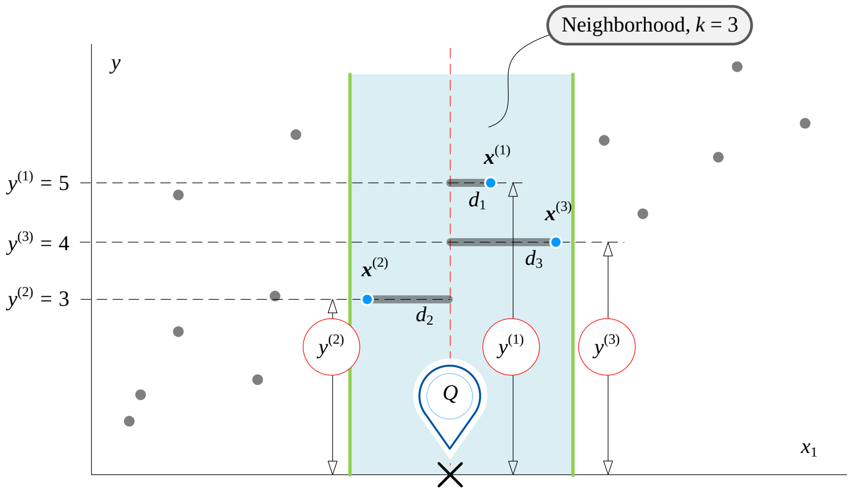


图 13. *k*-NN 回归算法原理

函数

`sklearn.neighbors.KNeighborsRegressor` 函数完成 *k*-NN 回归问题求解。默认等权重投票，`weights = 'uniform'`。

如果 *k*-NN 回归中考虑近邻投票权重，查询点 $\boldsymbol{q}$ 回归值 $\hat{y}$ 可以通过下式计算获得：

$$\hat{y}(\boldsymbol{q}) = \frac{1}{\sum_{i \in kNN(\boldsymbol{q})} w_i} \sum_{i \in kNN(\boldsymbol{q})} w_i y^{(i)} \tag{10}$$

类似 *k*-NN 分类，`weights = 'distance'`设置样本数据权重与到查询点距离成反比。

图 14 所示为利用 *k*-NN 回归得到的不同种类鸢尾花花萼长度 x_1 和花萼宽度 x_2 回归关系。花萼宽度 x_2 相当于 (10) 中 y。图 14 (a) 采用等权重投票，图 14 (b) 中投票权重与查询点距离成反比。

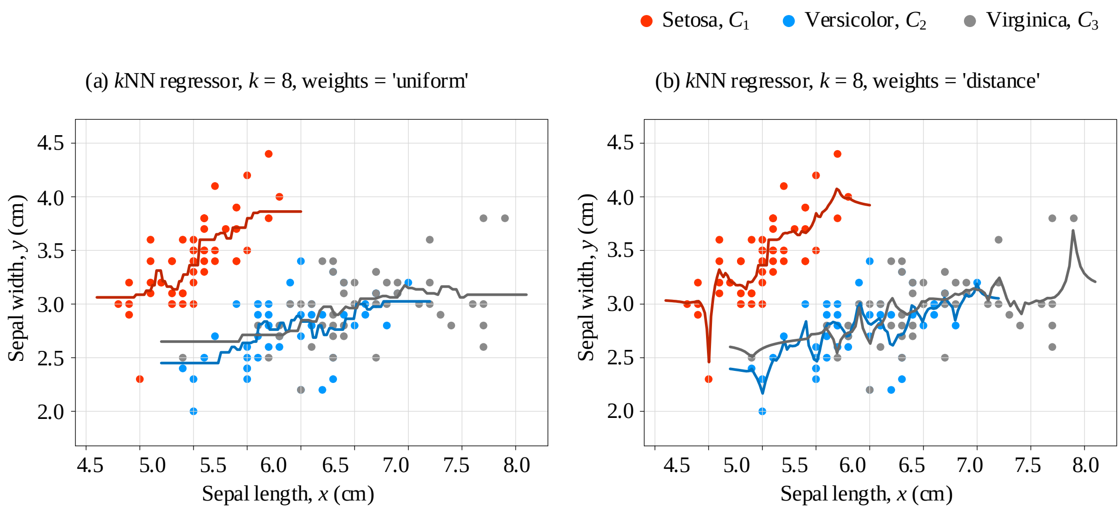


图 14. *k*-NN 回归，不同种类鸢尾花花萼长度和花萼宽度回归关系

代码 Bk7_Ch08_04.ipynb 完成 *k*-NN 回归，并绘制图 14 两幅图像。

本章探讨最简单的监督学习方法之一——最近邻 *k*-NN。最近邻方法可以用于分类问题，也可以用于回归问题。本书后文将介绍如何用最近邻 *k*-NN 完成回归任务。使用 *k*-NN 算法时，要注意近邻 *k* 值选择、距离度量，以及是否采用加权投票。

此外，最近质心分类 NCC 可以看做 *k*-NN 的简化版本，NCC 利用某一类成员质心表示该类别数据，不需要用户提供近邻数量 *k* 值，决策边界为中垂线。

最近邻这一思路是很多其他机器学习算法的基础，比如 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)、**流形学习** (manifold learning) 和**谱聚类** (spectral clustering) 也是基于最近邻思想。

本章给出的例子中距离度量均为欧氏距离；而实际应用中，距离度量种类繁多，需要大家理解不同距离度量的具体定义以及优缺点。