第5章　非参数模型

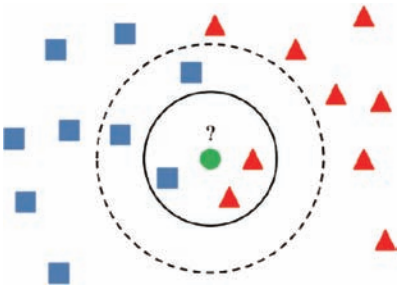
第4章的朴素贝叶斯分类器依据特征的条件独立假设和少量的数学法则就可以简单建立起来，并不需要定义损失函数，即便在处理连续特征取值引入了连续分布，但参数从样本估计而来，而非优化的目标。如果模型本身并不携带需要估计的参数，意味着不需要对分布做出任何假设，而是通过训练数据的统计性质就可以确定模型的结构，这类办法叫作非参数办法。

5.1　K近邻与距离度量

前面曾经提过“特征决定样本”，如果特征相同，那么就是同一个样本，否则我们应该注意数据是否有误，或者有隐变量未被收集到特征中去。一个自然的推论就是，特征的相似性就是样本的相似性。k 近邻(k Nearest Neighbors)算法就是建立在这一假设之上，它面对分类问题时，将与其最相似的 k 个样本所标记的类别做投票或者加权投票，结果作为预测的类别；面对回归的问题时，将与其最相似的 k 个样本的目标值做平均或者加权平均，结果作为预测的目标值。图5.1是KNN算法的一个简单示例，说明 k 的大小对结果的影响。

k 近邻算法原理非常自然，但仍有两个关键性的问题。其一就是该如何比较两个样本的相似度，它与相关度不同，相关度衡量的是随机变量会随着另外一个随机变量如何变化，我们前面提到过的皮尔逊相关系数、spearman相关系数、互信息都是在衡量相关度。机器学习假设了我们采用的样本是独立同分布，样本之间的相关度无从谈起。相似度衡量的是样本的接近程度，我们可以用特征空间的距离来表达接近的程度，越近，则相似度越高。

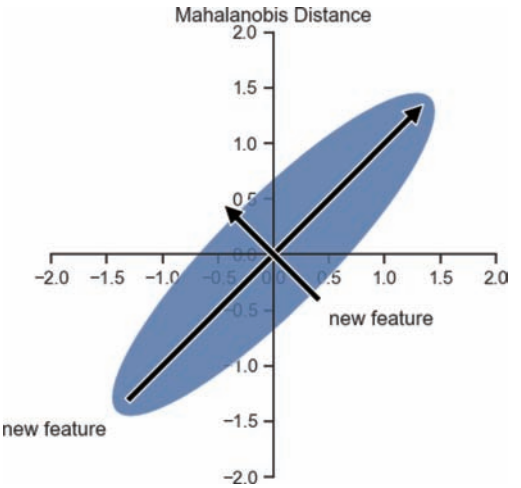
但这并不意味着相似度和相关度毫无联系，特征相关度会影响样本



■图 5.1 圆点表示新样本，三角和矩形点表示两个不同的类别，从图中可以看出，$k=3$ 的时候，新样本会被判定为三角类别，当 $k=5$ 时，新样本会被判定为矩形类别

相似度的计算，比如马哈拉诺比斯距离（Mahalanobis Distance），见定义 5.1，以下称之为马氏距离。马氏距离有两种理解方式，一种是考虑特征相关性会对距离的计算产生影响，协方差矩阵非对角元表示的是特征之间的相关度，距离计算相较于原来会沿着特征相关的方向拉长或者缩小。如果特征之间线性独立，那么协方差矩阵就是对角矩阵。

另一种则是协方差矩阵的引入转换了特征空间，新的特征空间是由原特征之间的线性组合作为轴，新的特征彼此正交，距离计算要先经过特征空间的转换。图 5.2 为原始特征正相关下的马氏距离的“单位圆”，圈上的每一点都在马氏距离度量下等于 1。对应着我们的两条思路：其一，距离计算相较于原来会沿着特征相关的方向拉长，而在相关正交的方向会缩小；其二，原始的特征空间会组合为新的特征，构成马氏距离度量空间的正交基矢。事实上这种思路对应着特征提取，我们会在第 7 章学习到以 PCA 为代表的一系列特征提取方法，新的特征方向是协方差矩阵最大特征值对应的特征向量的方向。



■图 5.2 马氏距离的可能“单位圆”，黑色箭头代表新的特征，由原特征组合而成

定义 5.1（马哈拉诺比斯距离和欧几里得距离） 假设 d 符合独立同分布的数据有 d 个特征，有样本 $s_1=(x_1,x_2,x_3,x_4,\cdots,x_d)$ 和样本 $s_2=(y_1,y_2,y_3,y_4,\cdots,y_d)$，数据的协方

差矩阵为$\boldsymbol{\Sigma}$，则马哈拉诺比斯距离被定义为：

$$d(s_1,s_2)=\sqrt{(s_1-s_2)^{\mathrm{T}}\sum^{-1}(s_1-s_2)}$$

协方差矩阵就成了单位矩阵，就得到了欧几里得距离：

$$d(s_1,s_2)=\sqrt{(s_1-s_2)^{\mathrm{T}}(s_1-s_2)}$$

如果我们对特征做了标准化再进行距离计算，则协方差矩阵并非单位矩阵，则是对角矩阵$\boldsymbol{W}$，对应着标准化的欧几里得距离：

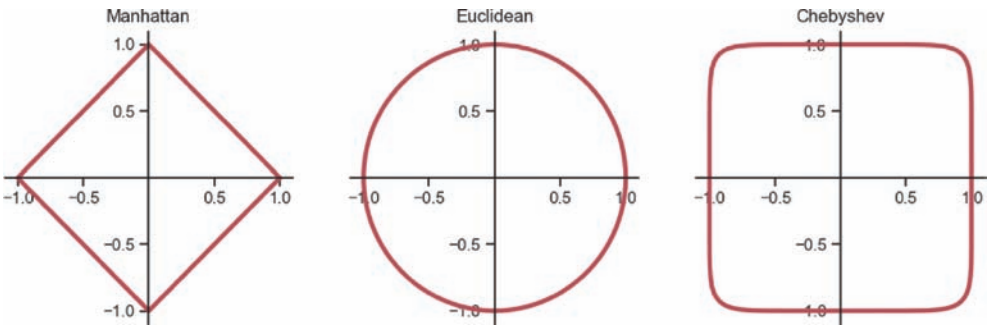
$$d(s_1,s_2)=\sqrt{(s_1-s_2)^{\mathrm{T}}W(s_1-s_2)}$$

其中，$W_{ii}=\frac{1}{\sigma_i^2}$为第$i$个特征的方差的倒数，表示分布更为集中的特征会在距离计算上有着更大的权重。

除了考虑特征相关度和特征重要性的马氏距离，另外一种常用的距离叫作闵可夫斯基距离(Minkowski Distance)，以下称为闵氏距离，它定义为：

$$\left(\sum^{d}|x_i-y_i|^p\right)^{\frac{1}{p}} \tag{5.1}$$

参数p与向量范数(定义1.4)中的作用一致，当$p=1$就得到了曼哈顿距离(Manhattan Distance)，当$p=2$就得到了欧几里得距离，当$p\to\infty$就得到了切比雪夫距离(Chebyshev Distance)，图5.3描述了不同的距离度量下的“单位圆”，可以看出闵氏距离的参数具体对应着不同形状的划分方式，在固定k的前提下，这些划分直接决定了所包含的具体样本。



■图5.3 从左到右分别为曼哈顿距离、欧几里得距离和切比雪夫距离在二维特征空间的示意

5.2 K近邻与kd数

距离定义问题的解决意味着我们可以使用k近邻算法去做一些简单的任务，我们对每一个新样本计算它与整个训练集数据的距离，然后选出最近的k个，然后将这k个的目标

值或者类别标签用来预测新样本。但对于多特征的大数据集，这一做法由于时间复杂度太高而不切实际，如果我们用于存储数据的结构中本身就包含了距离信息，那么就可以减少计算距离的次数，从而更快地找出最近的 k 个点。

假设数据的特征只有一维，我们可以先对训练数据进行排序，然后采用二分搜索(Binary Search)将新样本定位到两个数据之间，最后按照按顺序来依次找出 k 个最近点。二叉搜索树建立在二分搜索的基础上，见定义 5.2，二叉搜索树的每一个非叶节点都代表着对数轴的一次切分。通过根据数据构建好的二叉搜索树，我们可以快速地查找到距离最近的 k 个点，平衡的二叉搜索树在理论上要比非平衡的搜索速度更快。

定义 5.2(二叉搜索树) *二叉搜索树的任意节点的左子树和右子树均为二叉搜索树，它们均满足左子树上的节点值小于根节点、右子树上的节点值大于根节点。在构造二叉搜索树时就完成了排序，如果是已经排好序的数据，那么构造的二叉搜索树将没有左子树(从小到大排序)或者右子树(从大到小排序)。*

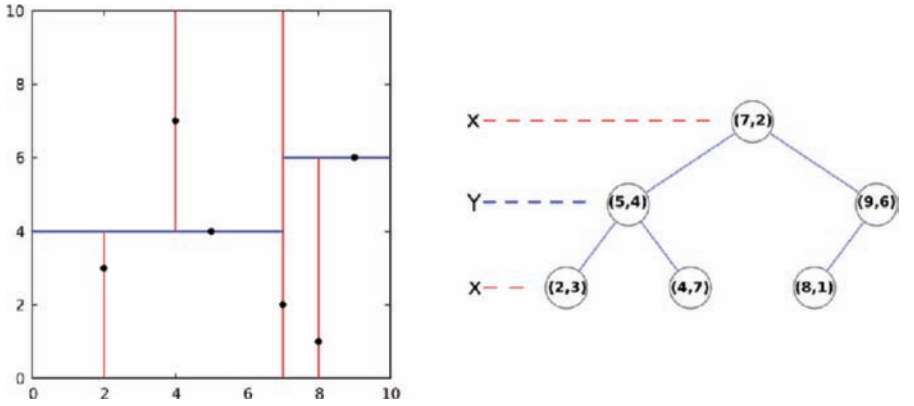
如果数据的特征为二维(x_1 和 x_2)，我们就可以借鉴一维情况的思想，将非叶节点视为对两条轴的轮流切分，按以下流程构建：

(1) 首先我们对 x_1 轴进行切分，切分点如果选择数据在该特征上分布的中位数，那么就会让得到的结构更为平衡，该切分点就作为第一个节点。所有在 x_1 轴上小于该节点的数据进入左子树，否则进入右子树。

(2) 然后，左子树和右子树上的数据分别对 x_2 轴进行切分，继续生成相应的左子树和右子树。

(3) 对所有子树重复上述两步。当某子树只包含一个数据点，则停止切分。

我们将得到的树叫作 kd 树(k-Dimensional Tree)，这里的 k 与 k 近邻含义不同，它指的是数据的维度。如图 5.4 所示，我们首先选取某一特征下的取值 7 作为切分轴，将特征空间分为两部分(大于 7 和小于 7)，生成的两个特征子空间分别再根据另一特征下的取值 4 和 6 作为切分轴，直到每个叶节点上都只有一个数据点。在实际使用中，各个特征下的切分值一般选取该空间包含数据在此特征取值下的中位数，这样可以使树尽可能平衡，保证查询效率。



■图 5.4 左图为特征空间的切分示意，右图为样本空间形成的树结构

5.3 决策树和条件熵

树作为较常见的数据结构，也可以作为一种机器学习算法。比如决策树(Decision Tree)，它模仿了人类的决策过程，如图 5.5 所示，决策中要回答一系列问题，这些问题构成了内部的节点，而我们最后采取的行动构成了叶节点。

■图 5.5 人对于一个问题的决策过程

对应在机器学习的分类问题上，叶节点就表示一个类，内部节点表示某种判断规则。决策树最初步的问题，不是树的形状，而是我们提出“好的问题”作为判断规则，才会使得我们沿着树的路径一直回答问题，直到得出正确的结果。

好的问题就是对我们分类最有用的判断，如果我们能够通过一次判断就可以正确地分类，那么这个问题必然抓住了有价值的信息。虽然往往做不到这一点，但我们至少可以将同一类的样本都尽可能地往树的同一个方向走，而不同类的样本要尽可能地往不同的方向走。从第 4 章的对数损失和信息论节中介绍的信息熵(定义 4.2)可以看出，信息熵有着两个特点：

(1) 不同类的样本集合越是均匀（即不同类样本等概率)，信息熵就越大。反之，信息熵越小，说明某一样本的比例可能越大。换而言之，一个系统的不确定性越大，信息熵就越大。

(2) 样本都是均匀的两个系统，包含更多类样本的系统，信息熵也会更大。

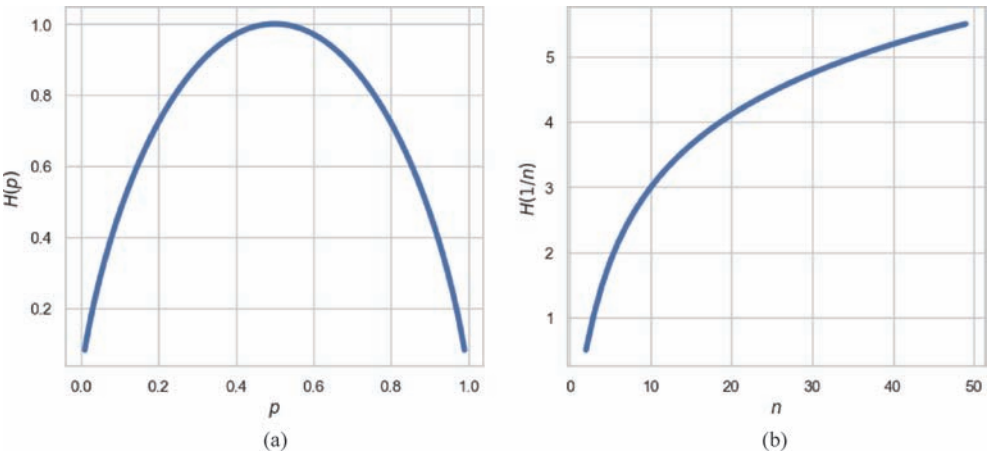
图 5.6 可以对应信息熵的两个主要特点，在伯努利分布下的成功失败概率相等时，信息熵最大，在均匀分布下的可能结果数越多，信息熵越大。

因为特点(1)的存在，我们就可以利用信息熵的变化来选择最适合作为判断规则的特征。熵如果因为某个特征而减小，就意味着类别的不确定性下降了，下降得越多，越说明该特征的分类能力越强。判断后的信息熵可以用条件熵来描述，见定义 5.3。我们用特征 F 表示某个特征，D 表示分类任务，我们需要计算：

$$H(D)-H(D\mid F) \tag{5.2}$$

$$=H(D)-\sum_{F}p(f)H(D\mid F=f) \tag{5.3}$$

这两者的差也被叫作信息增益(information gain)，描述了给定特征 F 后使得分类不确定性减小的程度。



■图 5.6 (a)表示伯努利分布下信息熵随概率 p 的变化,(b)表示均匀分布下信息熵随类别数 n 的变化

定义 5.3(条件熵) 条件概率 $P(Y|X=x)$ 的信息熵是指当 $X=x$ 对变量 Y 的影响,变量 B 对变量 A 的条件熵就是考虑全部的 X 取值,是条件概率的信息熵在 X 概率分布下的期望值:

$$H(Y \mid X)=\sum_{X} p(x) H(Y \mid X=x)$$

可以进一步拆开为:

$$\begin{aligned} H(Y \mid X) &=-\sum_{X} p(x) \sum_{Y} p(y \mid x) \log (y \mid x) \\ &=-\sum_{X} \sum_{Y} p(x, y) \log (y \mid x) \end{aligned}$$

假设数据集中第 k 类样本出现的概率为 P_k,L 是总的类别数,信息熵则是:

$$H(D)=-\sum_{k=1}^{L} P_k \log P_k \tag{5.4}$$

如果我们选取的特征下的某个取值 f 会使得样本集合 D 变成 d,d 就是在某个特征下具有相同取值的样本集合,设定 l 是新的集合中包含的类别数,P_v 是在 d 中每一类所占的比例,此时数据集 d 的信息熵就是 $H(D|F=f)$:

$$H(D \mid F=f)=-\sum_{v=1}^{l} P_v \log_2 P_v \tag{5.5}$$

假设特征 F 有 n 个取值,标记为 $f_1, f_2, \cdots, f_i, \cdots, f_n$,那么我们求取确定取值 f_i 下信息熵的期望值:

$$H(D \mid F)=\sum_{i}^{n} \frac{|d_i|}{|D|} H(D \mid F=f_i) \tag{5.6}$$

这样得到了熵和特征对分类的条件熵之后,就可以对每一个特征进行这样的计算,挑选出信息增益最大的特征,然后将该特征的所有取值作为下一层的节点,如此循环。但是根据

特点(2),特征的取值数 n 越大,信息熵也就会越大,从而造成信息增益率越大。如何理解这样的不合理呢?考虑一个极端的情况,特征取值数太多,造成每个取值下的样本数只有1个,使得式(5.5)中的 $P_v=0$,条件熵为零,虽然此时信息增益最大,但是太过于"精细"以至于泛化能力低下。

对信息增益方法的改进就是惩罚取值数过多的特征,关于特征取值数的信息熵为:

$$H(F)=-\sum_{i}^{n}\frac{|d_i|}{|D|}\log\frac{|d_i|}{|D|} \tag{5.7}$$

当我们的数据集固定好以后,$H(F)$对于特征 F 是确定的。我们用信息增益除去特征取值数的信息熵来实现惩罚效果,这样就得到了信息增益率(gain ratio):

$$\frac{H(D)-H(D\mid F)}{H(F)} \tag{5.8}$$

无论采取怎样的算法,决策树都试图通过有限步骤内的每一步来尽可能地将样本分开,采取的还是贪心策略,对样本直接进行建模,并没有考虑特征和目标值的联合分布,所以是判别式模型的一种。并且在生成树的过程中,比较每个特征的信息增益或者增益率,也是对特征对任务的贡献程度进行比较,从而实现特征重要程度排序。

5.4 决策树的剪枝

我们在上文中说,决策树的"精细"会导致泛化能力低下,与回归曲线精确地绕过每一个样本点、决策边界精确地分开每一个样本点一样,极端复杂的决策树精确地将每一个样本点分配给一个叶节点。其复杂度体现在两个方面:

(1) 树的深度。随着递归调用次数的增加,节点分裂的次数也就会越多,意味着模型的规则变得越来越复杂。

(2) 叶节点包含的样本个数。如果它包含的样本太少,说明决策树为少量的样本创建了规则,使每一个样本被划分正确,很有可能就出现了过拟合。

一般来说树根越深,叶节点也就越多,叶节点包含的样本也就越少。我们主要采用剪枝(prune)的办法来防止过拟合的产生,所谓的剪枝就是将树某些节点去掉来降低树的规模,主要有两种方法来实现:

其一是预剪枝(prepruning)。这里面"预"就是指提前,在树生成之前就对其进行剪枝。其实,这里进行的根本就不是剪枝,而是每次生成枝叶的时候,就对样本做分集测试,判断该节点生成前和生成后的泛化误差。如果在生成节点的过程中,决策树的泛化误差将提升,我们就停止生成该节点,并将该节点变为叶节点,样本数较多的类别作为该叶节点的输出。如果泛化误差降低了,那就生成该节点。

预剪枝与优化过程中的提前终止所起到的作用是一致的,都是通过观察每一步的泛化误差来决定自己的下一步,只是预剪枝是在决定要不要生成节点,而提前终止是在决定要不要继续迭代。缺点也是一样的,都是一种贪心策略,当前如果不能提升,并不代表下一步不

能提升，有些时候，保留当前节点确实会使泛化误差上升，但在保留当前节点的前提下，继续保留下一节点却会使得泛化误差下降，所以我们在使用预剪枝的时候，也可以设置patience，泛化误差连续下降几次才生成该节点，以尽可能地让它多走几步。

其二是后剪枝(postpruning)。与预剪枝相反，后剪枝会在树生成之后对其剪枝，它会按照树的顺序由上到下，将每一个节点替换成叶节点，本质上，就是把原本不输出结果的节点，替换成了输出结果的节点。在做每一次剪枝之前，仍然需要对样本做分集测试，判断该节点删除之后和删除前的泛化误差。如果在删除节点的过程中，决策树的泛化误差将提升，我们就确定删除该节点，如果泛化误差降低了，那就保留该节点。

在后剪枝时，有一个普遍的错误认识，那就是认为预剪枝使用了贪心，而后剪枝没有使用贪心，所以后剪枝的性能往往要比预剪枝好。事实上，后剪枝同样用了贪心，我们在删除节点的过程中，也只是在考虑当前节点的影响，有时候，删除当前节点确实会使泛化误差下降，但保留当前节点，删除另一节点却会使得泛化误差下降得更多，但后剪枝往往会把两个节点全部删除。后剪枝的性能之所以比预剪枝好，是因为它在生成之后才进行剪枝，往往比预剪枝的规模更大，更不容易发生欠拟合。

在实际应用中，我们可以通过限制树的最大深度，叶节点包含的最大样本数，叶节点的最大数量等来限制树的复杂度，虽然将这些作为超参数调试比较方便，但是效果上不如直接剪枝带来的性能更好。

5.5 连续特征取值的处理方法和基尼指数

我们会注意到在树的生成过程中，每个节点分裂出的子节点数目取决于该特征下的取值数，如果特征只有两个取值，那么得到的决策树就是二叉树，并且可以保证分裂的节点中包含的样本数量较多。但如果所有的特征取值数都是连续的，甚至每个样本在特征上的取值均不相同，就无法避免地出现过拟合的情况，剪枝也无从下手。

朴素贝叶斯也面临过相同的问题，它的解决办法是：对连续取值的特征假设一个连续分布。但这样的解决思路会引入参数。在决策树中，我们通常使用二分法(bipartiton)来解决这个问题，它分为三步：

(1) 首先，我们对连续属性值 F 进行排序，形成 $(f_1, f_2, f_3, \cdots, f_n)$。

(2) 确定二分法的划分点 b，将属性值划分为两类，$(f_1, f_2, f_3, \cdots, f_b)$ 和 $(f_{b+1}, f_{b+2}, \cdots, f_n)$，分别表示小于 b 的特征值和大于 b 的特征值。一般地，有无穷个点可以将 F 划分为相同的两类，我们取两个相邻取值的中点作为划分点。

(3) 因为每个划分点都会把特征取值划分为两类，将原本 n 个连续特征值的信息熵转化为大于 b 和小于 b 这两个值的信息熵，式(5.6)就会变为：

$$H(D \mid F) = \frac{|d_1|}{|D|} H(D \mid F > b) + \frac{|d_2|}{|D|} H(D \mid F < b)$$

我们就可以清楚地看到，对于属性的连续取值，所谓的二分法是把原本的很多取值变为

只有两个取值，分别是大于某点的取值，和小于某点的取值。二分法只负责分为两份，而不管是不是均匀。如果我们的连续属性值有 n 个，那么划分点 b 的可能取值就有 $n-1$ 个，我们需要对划分点进行遍历，找到那个使我们信息增益（或者信息增益率）最大的划分点。

同样的道理，我们同样可以尝试三分法、四分法，只是我们进行搜索的代价也会增加。比如考虑三分法，我们需要估计两个划分点（两个划分点不可区分），对于 n 个值，划分点的可能情况就有$\frac{(n)(n-1)}{2}$。

此外，连续特征处理也可以不采用熵来作为判断规则，而是采用基尼指数来实现，见定义 5.4。著名的 CART 算法就是采用了基尼指数，假设特征 F 有 n 个取值，我们遍历每个连续取值，将数据集分为两类，$d_1(F=f_i)$和 $d_2(F\neq f_i)$，此时的基尼指数就为：

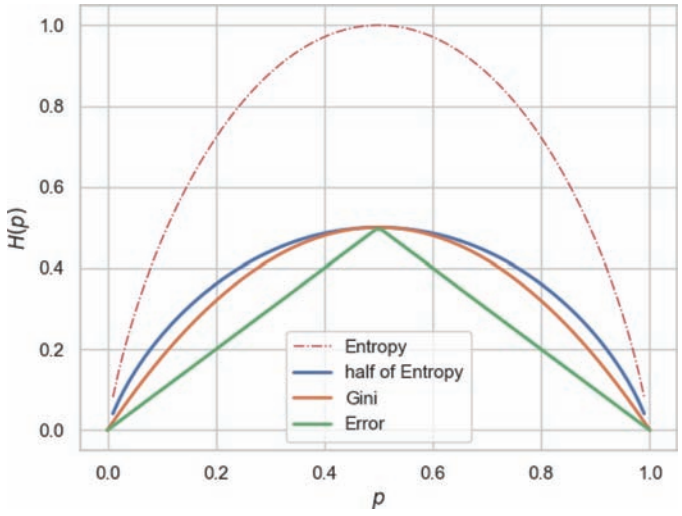
$$Gini(D,F=f_i)=\frac{|d_i|}{|D|}Gini(d_1)+\frac{|d_2|}{|D|}Gini(d_2) \tag{5.9}$$

定义 5.4（基尼指数） 如果从数据集中随机挑选两个样本，类别标记不一致的概率越大，意味着类别的不确定性越高，所以我们可以用不一致的概率来作为判断规则，基尼指数就是基于此定义，假设数据集中第 k 类样本出现的概率为 P_k，L 是总的类别数，基尼指数就为：

$$Gini(D,F=f_i)=\frac{|d_1|}{|D|}Gini(d_1)+\frac{|d_2|}{|D|}Gini(d_2)$$

基尼指数越小，代表着类别不一致的概率越小，不确定性越小。

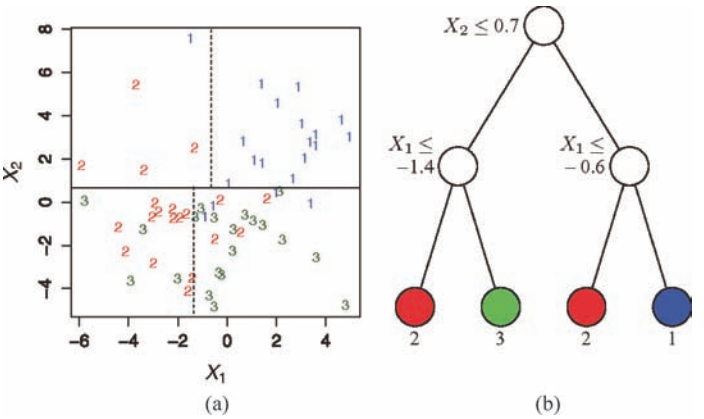
从图 5.7 可以看出，错误率的曲线在半熵和基尼指数的下面，说明我们在使用决策树来降低熵或者基尼指数的同时，也在降低着错误率。我们计算每个特征的每个取值下的基尼指数，然后选取能够使基尼指数最小的特征和相应的取值，每次的划分规则就是“是否等于该取值”，所以 CART 算法的分类任务得到的决策树均为二叉树。



■图 5.7 考虑二分类，从上到下的曲线分别为熵、熵的一半、基尼指数和错误率

5.6 回归树

决策树用于回归任务的主要思路是将连续的空间离散化，目标值和特征值作为一个整体被切分，如图 5.8 的图(a)，我们先依据某个特征来进行切分，该切分点尽可能地将差距较大目标值的样本分开，比如挑选了 X_2 作为特征，$X_2=0.7$ 作为最佳切分点，将差距较大的 1 和 3 分开，然后在子节点上继续进行这一操作。



■图 5.8 (a)表示回归树在特征空间的切分方式，(b)表示相应的树结构

真正的问题是我们如何选择该特征，以及该特征上的最佳切分点。我们来考虑最简单的单变量回归数据，我们的样本只有一个特征：x，和一个目标值：y。m 个样本的集合就可以被表示为$(x_1,y_1),(x_2,y_2),(x_3,y_3)\cdots(x_m,y_m)$。采取与 5.5 节一样的做法，我们采用二分法，划分点就有 $n-1$ 种，对于其中的可能划分点 b，会把我们的目标值和特征值所组成的数据同时划分为 d_1、d_2，我们在此基础上定义平方损失：

$$\sum_{d_1}(y_i-\bar{y}_{d_1})^2+\sum_{d_2}(y_i-\bar{y}_{d_2})^2 \tag{5.10}$$

y_i 是样本的目标值，$\bar{y}_{D_1}$ 是 D_1 所包含样本的目标值的平均值。整个损失，其实就是切分后各个样本集合损失之和。所以我们对每一个划分点做如上的计算，挑出损失最小对应的划分点，作为我们的最佳切分点。

目前我们挑选的只是固定特征下的最佳切分点。如果将特征拓展到更多，我们对每一个特征都做出如上的操作，然后将拥有最小损失的特征作为最佳特征。换而言之，我们用每个特征最佳切分点对应的损失作为该特征的损失，最佳特征和其对应的最佳划分点是同时得出的。

我们每生成一个节点，就是对特征空间的一块区域进行了划分，如此往复，直到所有的切分方式差异较小就可以停止递归。极端的情况是如果无终止地划分下去，每一小块区域都只有一个样本时，决策树叶节点也只包含了一个样本，就对应着我们上文中的过拟合。

5.7 使用 scikit-learn

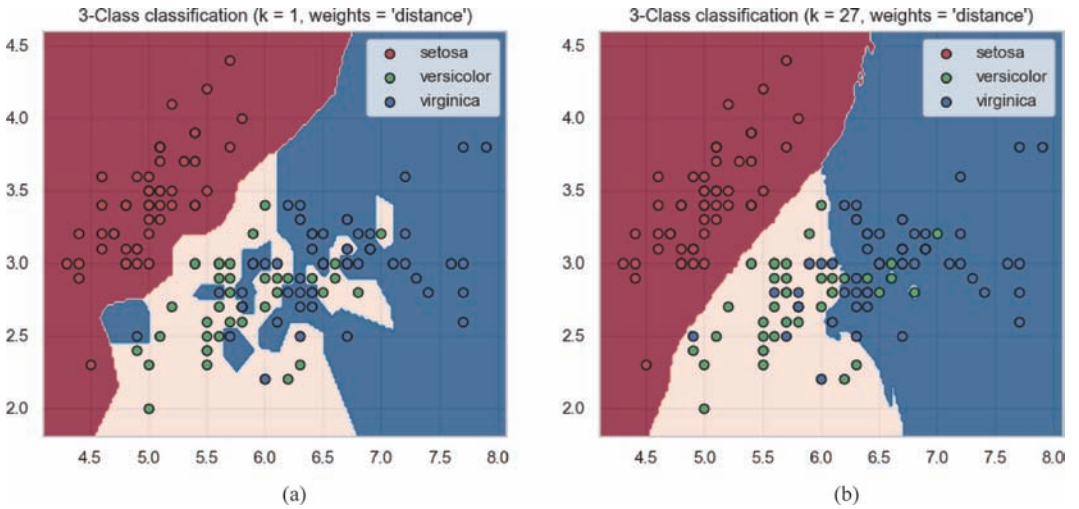
KNN 算法所包含的三个要点分别为 k 的大小、判断的规则和距离函数，我们分别对这三个要素做具体的研究，主要通过二维特征空间的结果可视化来定性地探究其影响。

首先我们在经典的 Iris 数据中使用 KNN 算法，分别采用小的 k 值和大的 k 值来对数据集中的每个样本进行预测。虽然这里我们并没有进行分集，但直接利用数据集也可以看出来这三要素的影响。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors, datasets
import seaborn as sns
from sklearn.neighbors import DistanceMetric

iris = datasets.load_iris()
X = iris.data[:,:2]
y = iris.target

def surface_plot(model,X,y):
    def make_meshgrid(x, y, h=.02):
        x_min, x_max = x.min() - 0.2, x.max() + 0.2
        y_min, y_max = y.min() - 0.2, y.max() + 0.2
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),\
            np.arange(y_min, y_max, h))
        return(xx, yy)
    model.fit(X,y)
    xx,yy=make_meshgrid(X[:,0],X[:,1])
    Z=model.predict(np.c_[xx.ravel(),yy.ravel()])
    Z=Z.reshape(xx.shape)
    plt.contourf(xx,yy,Z,cmap=plt.cm.RdBu,alpha=.8)
    for c, i, target_name in zip("rgb", [0,1,2],iris.target_names):
        plt.scatter(X[y == i,0], X[y == i,1], c = c, label = target_name,\
            edgecolors = 'k')
weight = 'distance'

sns.set(style = 'whitegrid')
for j,k in enumerate([1,27]):
    clf = neighbors.KNeighborsClassifier(k, weights = weight)
    plt.subplot(1,2,j+1)
    surface_plot(clf,X,y)
    plt.title("3-Class classification (k = %i, weights = '%s')"%
```

```
            (k,weight))
        plt.legend()
    plt.show()
```

从图 5.9 中可以看出，大的 k 值会尽可能地平均多的结果，因为样本的判别都要利用更大的样本集合，这样就相当于对大量的样本做了平均化，决策边界自然也就会变得更加光滑。较小的 k 值则不会产生这样的效果，尤其等于 1 的时候，表示我们只通过与其距离最近的点来判断该点的类别，就无法区分具有相似特征的不同类别。



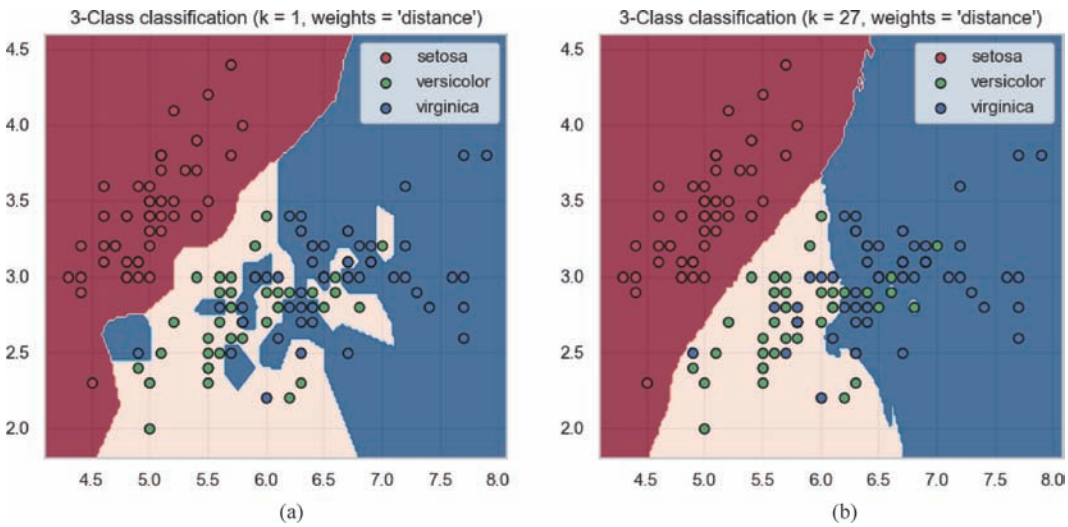
■图 5.9　使用加权平均投票和欧几里得距离的 KNN 算法，(a)为 $k=1$ 的预测结果，(b)为 $k=27$ 的结果

接下来，我们改变不同的距离度量，来探究其对 KNN 算法结果的影响，具体对比欧几里得距离和切比雪夫距离，在上述基础上添加如下代码：

```
plt.figure()
for j,dist in enumerate(['euclidean', 'chebyshev']):
    clf = neighbors.KNeighborsClassifier(algorithm = 'auto', n_neighbors = 7,
        weights = weight, metric = dist)
    plt.subplot(1, 2, j + 1)
    surface_plot(clf, X, y)
    plt.title("3 - Class classification (k =  % i, distance =  '% s')" % (7, dist))
    plt.legend()
plt.show()
```

如图 5.10 所示，采用不同的距离对结果的影响较小，微小的差异主要集中在 versicolor 和 virginica 的分界面，这可以说明欧几里得距离和切比雪夫距离在这里具有相当的一致性，可以部分说明特征有着统一的量纲。

然后我们在该数据上使用决策树算法，分别将判断准则设为信息熵和基尼指数，在上述



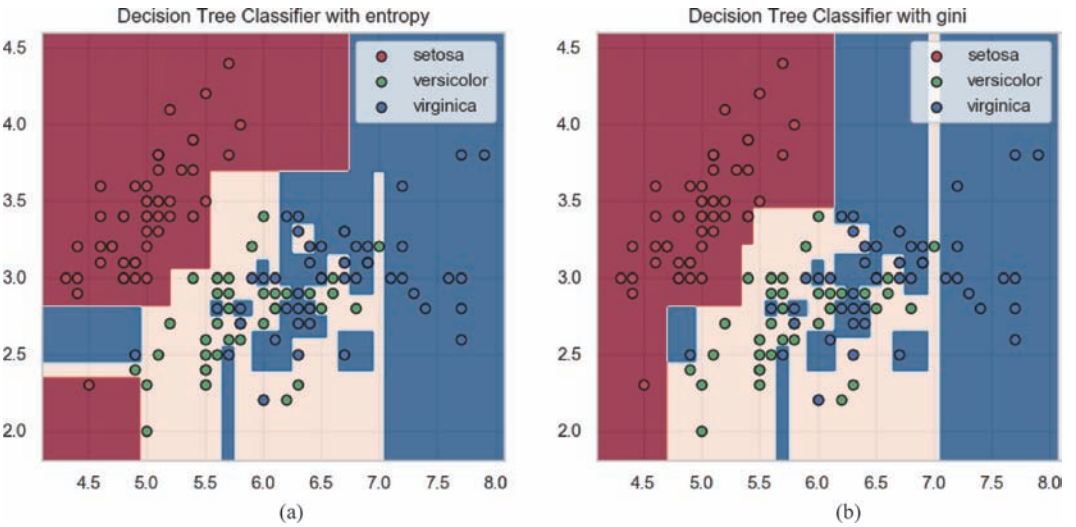
■图 5.10 $k=7$ 时的 KNN 算法，(a)为欧几里得距离的结果，(b)为切比雪夫距离的结果

代码基础上添加以下代码：

```
import graphviz
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier as DTC

sns.set(style = 'whitegrid')
for j,k in enumerate(['entropy','gini']):
    clf = DTC(criterion = k)
    plt.subplot(1,2,j + 1)
    surface_plot(clf,X,y)
    plt.title("Decision Tree Classifier with %s" %k)
    plt.legend()
plt.show()
```

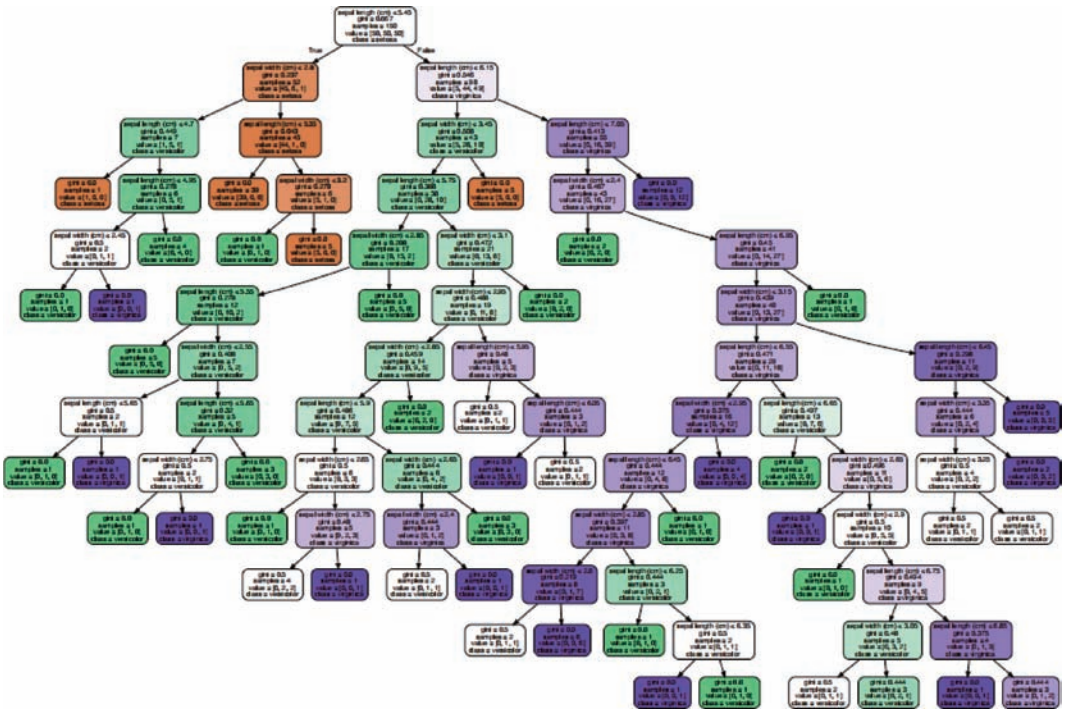
如图 5.11 所示，无论采取什么样的划分准则，决策树的决策边界总是平行于特征轴，这是因为决策树的每一个节点都对应着一次划分，这样的划分总是递归地在子空间进行。对于决策树这类解释性比较强的算法来说，更重要的是我们要获得决策图，从而可以直观地看到整个决策流程，使用 graphviz 工具，并在上述基础上添加如下代码，就可以得到决策图文件：

```
import graphviz

dot_data = tree.export_graphviz(clf, out_file = None,
                    feature_names = iris.feature_names[:2],
                class_names = iris.target_names,
                filled = True, rounded = True,
                    special_characters = True)
graph = graphviz.Source(dot_data)
graph.render('Decision Tree')
```



■图 5.11　未做任何限制的决策树的结果，(a)为使用信息熵，(b)为使用 Gini 指数的结果

如图 5.12 所示，注意到决策图是根据 *Gini* 指数来作为划分准则，所以最后的决策树是二叉树。同时也可以发现这颗决策树规模较大，叶节点包含的样本特别少，对于少量的样本单独创建的一些规则，可能是过拟合的表现。



■图 5.12　上述代码所生成的决策树结构

我们接下来对树的深度做出限制，并采取 10 折的交叉验证来分别观察训练准确率和测试准确率随树深度的变化，在上述代码基础上添加如下代码：

```
from sklearn.model_selection import cross_validate

test_acc = []
train_acc = []
depths = range(1,20)
for d in depths:
    clf  = DTC(criterion = 'gini',max_depth = d)
    clf_dict = cross_validate(clf,X,y,cv = 10,scoring = 'accuracy')
    test_acc.append(clf_dict['test_score'].mean())
    train_acc.append(clf_dict['train_score'].mean())
sns.set(style = 'white')
plt.plot(depths,train_acc,'b-',label = 'Train Accuracy')
plt.plot(depths,test_acc,'r-',label = 'Test Accuracy')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

如图 5.13 所示，随着树深度的增加，模型的拟合能力不断增加，泛化性能却是先增加后下降，表示增加到一定程度发生了过拟合。

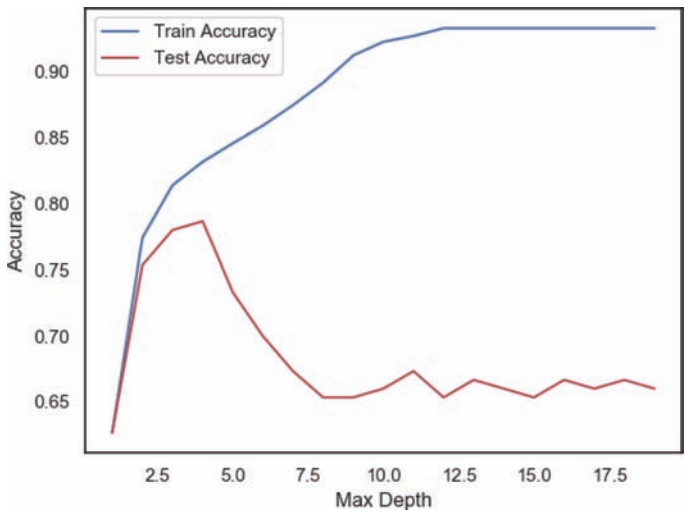
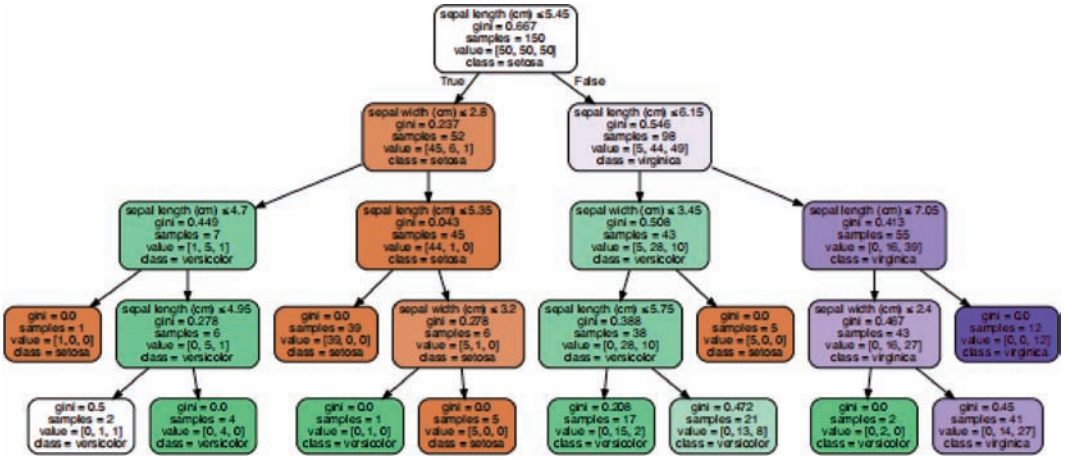


图 5.13 决策树的训练准确率和测试准确率随着树深度的变化

并且从图 5.13 中可以看出，当树的深度为 4 的时候，模型的测试误差最小。所以我们限制模型的深度为 4，延续上述的步骤就可以将对应的模型可视化，如图 5.14 所示，可以看到性能更好的树规模也小了不少。

以上的实践均是面对分类任务，接下来我们尝试使用 k 近邻和回归树来进行回归任



■图 5.14　决策树在限制深度为 4 的时候训练完成后的结构

务。我们构建一个包含噪声的正弦函数，分别利用深度为 2 和 5 的决策树以及 k 为 1 和 10 的 k 近邻学习器来完成学习，代码如下：

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
import seaborn as sns

def DT(depths, X, y):
        plt.scatter(X, y, c = 'k', s = 10, label = 'data')
        for depth in depths:
        DT_reg = DecisionTreeRegressor(max_depth = depth)
        DT_reg.fit(X, y)
        y_pre = DT_reg.predict(X)
        plt.plot(X, y_pre, label = 'depth = %s' % depth)
        plt.xlabel('X')
        plt.ylabel('y')
        plt.title('Decision Tree for regression')
        plt.legend()

def KNN(neighbors, X ,y):
    plt.scatter(X, y, c = 'k', s = 10, label = 'data')
    for neighbor in neighbors:
        KNN_reg = KNeighborsRegressor(neighbor)
        KNN_reg.fit(X, y)
        y_pre = KNN_reg.predict(X)
        plt.plot(X, y_pre, label = 'neighbor = %s' % neighbor)
        plt.xlabel('X')
        plt.ylabel('y')
```

```
        plt.title('KNN for regression')
        plt.legend()

X = np.linspace(0, 10, 80)[:, np.newaxis]
y = np.sin(X) + np.random.rand(80,1)

sns.set(style = 'white')
plt.subplot(1, 2, 1)
DT([2, 5], X, y)
plt.subplot(1, 2, 2)
KNN([1,10], X, y)

plt.show()
```

从图 5.15 中可以看出，对于回归树来说，树的深度其实就是递归的次数，递归的次数越少，表示对特征空间划分的次数越少，所以深度为 5 的回归树更加复杂，但它们均与特征轴平行；对于 k 近邻来说，k 的大小决定了局部的相关性，k 越大，表示与其相关的局部范围越大，与分类任务表示类似，大的 k 值会将整体平均化，所以图中 $k=10$ 的曲线看起来像是 $k=1$ 的曲线的平均值。

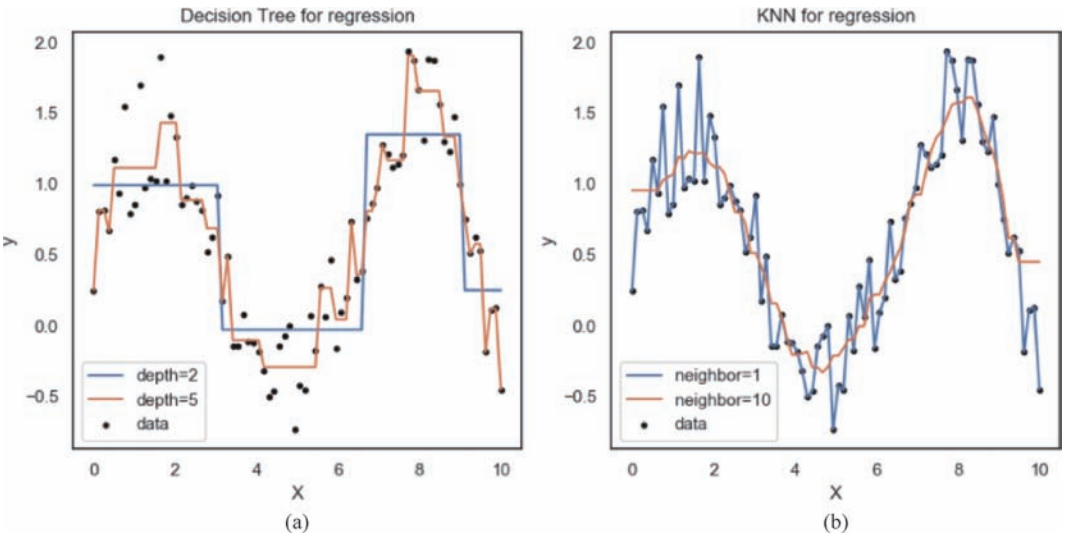


图 5.15 (a)为回归树在深度为 2 和 5 时的表现，(b)为 k 近邻在 k 为 1 和 10 时的表现