07 Chapter 集成机器学习

单个模型的性能通常有限，我们可以考虑将多个学习器通过合适的方式组合，得到性能更好的学习器。这种组合多个学习器得到一个性能更好的学习的方式，被称为集成学习，被组合的学习器称为基学习器，融合后的模型被称为强学习器。

本章我们学习 3 种常用的集成学习算法：Bagging、提升（Boosting）和融合（Blending），并讨论这些集成学习算法能提高模型性能的原因。我们还将探讨两种常用的基于决策树的提升算法的高效实现工具包的原理和 API：XGBoost 和 LightGBM。

集成学习将多个学习器组合，以得到性能更好的学习器。被组合的学习器称为基学习器，组合后的模型被称为强学习器。如果基学习器是从同一种学习算法产生，我们称该集成学习是同质的（homogenerous）。如果基学习器是从几种不同学习算法产生，则称集成学习是异质的（heterogenous）。集成学习中，通常基学习器之间的互补性越强，集成效果更好。

7.1 误差的偏差—方差分解

7.1.1 点估计的偏差—方差分解

记根据训练样本集$\mathcal{D}$估计某个参数θ的点估计为：$\hat{\theta} = g(\mathcal{D})$。根据频率学派的观点，真实参数值$\theta$是固定的，但是未知，而$\hat{\theta}$是数据$\mathcal{D}$的函数。由于数据是随机采样的，因此$\hat{\theta}$是个随机变量。

点估计的偏差定义为：

$$bias(\hat{\theta}) = \mathbb{E}\left[\hat{\theta}\right] - f(\theta)。 \tag{7-1}$$

这里期望$\mathbb{E}$作用在所有数据上。为了理解这里的期望，假设我们可以对整个流程重复多次，每次收集训练数据集$\mathcal{D}$，利用训练数据训练得到$\hat{\theta}$。由于每次收集到的样本$\mathcal{D}$稍有不同（每次训练数据集可视为总体数据独立同分布的样本。由于随机性，每次训练样本集会有差异），从而每次得到的参数估计$\hat{\theta}$也稍有不同。这多个不同的估计的均值可视为期望$\mathbb{E}\left[\hat{\theta}\right]$的估计。

如果$bias\left(\hat{\theta}\right) = 0$，则称估计量是无偏的。

点估计的方差记作$Var\left[\hat{\theta}\right]$，它刻画的是从潜在的数据分布中独立地获取样本集时，点估计的变化程度。

例：从均值为μ的贝努利分布中，得到独立同分布样本$x_1, x_2, \ldots, x_N$，

$$\mathbb{E}[x_i] = \mu,\ Var[x_i] = \mu(1-\mu)。$$

样本均值$\bar{x}$可作为参数μ的一个点估计，即：$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$。

因为$\mathbb{E}[\hat{\mu}] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} x_i\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[x_i] = \mu$，所以$\hat{\mu}$为$\mu$的无偏估计。

估计$\hat{\mu}$的方差为：$Var[\hat{\mu}] = Var\left[\frac{1}{N}\sum_{i=1}^{N} x_i\right] = \frac{1}{N^2}\sum_{i=1}^{N} Var[x_i] = \frac{1}{N}\,\mu(1-\mu)$，所以估计的方差随样本数量$N$增加而下降。

估计的方差随着样本数量的增加而下降，这是所有估计的共性。所以可能的话，训练样本数据越多越好。

7.1.2 预测误差的偏差—方差分解

我们希望模型能尽可能准确地描述数据产生的真实规律，这里的准确是指模型测试集上的预测误差小。模型在未知的测试数据上的误差，被称为泛化误差（Generalization Error）。模型的泛化误差有三种来源：随机误差、偏差、方差。

- 随机误差

随机误差ϵ是不可消除的，与数据产生机制有关（如不同精度设备得到的数据随机误差不同），并且与真值y^*相互独立。对数值型响应（如回归任务），一般认为随机误差服从 0 均值的正态分布，记作$\epsilon \sim N(0, \sigma_\epsilon^2)$，其中$N$为正态分布。观测值$y$与真值$y^*$之间的关系为：$y = y^* + \epsilon$，所以$y \sim N(y^*, \sigma_\epsilon^2)$。

- 偏差的平方

假设给定训练数据集$\mathcal{D}$，利用训练数据训练得到模型$f_\mathcal{D}$。根据训练好的模型$f_\mathcal{D}$对测试样本$\boldsymbol{x}$进行预测，得到预测结果$\hat{y}_\mathcal{D} = f_\mathcal{D}(\boldsymbol{x})$。模型预测的**偏差的平方**度量模型预测值$\hat{y}_\mathcal{D}$的期望与真实规律$y^*$之间的差异：

$$bias^2(\hat{y}_\mathcal{D}) = (\mathbb{E}[\hat{y}_\mathcal{D}] - y^*)^2。 \tag{7-2}$$

偏差表示学习算法的期望预测与真实值之间的偏离程度，刻画了学习算法本身的拟合能力。

- 方差

模型预测的方差记为：

$$Var[\hat{y}_\mathcal{D}] = \mathbb{E}[(\hat{y}_\mathcal{D} - \mathbb{E}[\hat{y}_\mathcal{D}])^2]。 \tag{7-3}$$

方差表示由于训练集的变动所导致的预测性能的变化，刻画了数据扰动造成的影响。

当损失函数取平方误差损失（L2 损失）$\mathcal{L}(\hat{y}_\mathcal{D}, y) = (\hat{y}_\mathcal{D} - y)^2$时，记$\overline{y} = \mathbb{E}[\hat{y}_\mathcal{D}]$，则泛化误差为损失函数的期望：

$$\begin{aligned}
Error &= \mathbb{E}[(\hat{y}_\mathcal{D} - y)^2] \\
&= \mathbb{E}[(\hat{y}_\mathcal{D} - (y^* + \epsilon))^2] \\
&= \mathbb{E}[(\hat{y}_\mathcal{D} - y^*)^2] + \mathbb{E}[\epsilon^2] \\
&= \mathbb{E}[(\hat{y}_\mathcal{D} - y^*)^2] + Var[\epsilon] \\
&= \mathbb{E}[(\hat{y}_\mathcal{D} - \overline{y}) + (\overline{y} - y^*)]^2 + Var[\epsilon] \\
&= \mathbb{E}[(\hat{y}_\mathcal{D} - \overline{y})^2] + \mathbb{E}[(\overline{y} - y^*)^2] - 2\mathbb{E}[(\hat{y}_\mathcal{D} - \overline{y})(\overline{y} - y^*))] + Var[\epsilon] \\
&= Var[\hat{y}_\mathcal{D}] + (\overline{y} - y^*)^2 - 2(\overline{y} - y^*)(\mathbb{E}(\hat{y}_\mathcal{D}) - \overline{y}) + Var[\epsilon] \\
&= Var[\hat{y}_\mathcal{D}] + (\overline{y} - y^*)^2 - 2(\overline{y} - y^*)(\overline{y} - \overline{y}) + Var[\epsilon] \\
&= Var[\hat{y}_\mathcal{D}] + (\overline{y} - y^*)^2 + Var[\epsilon],
\end{aligned} \tag{7-4}$$

即泛化误差可以分解为预测的偏差的平方、预测的方差以及数据的噪声。我们称之为泛化误差的**偏差—方差分解**。虽然其他损失函数不能解析证明泛化误差可分解为偏差的平方、方差、和噪声，但大致趋势相同。

偏差—方差分解表明模型性能是由模型的拟合能力、数据的充分性以及学习任务本身的难度共同决定的。

- 偏差：度量模型的期望预测与真实结果之间的偏离程度，刻画模型本身的拟合能力。
- 方差：度量训练集的变动所导致的模型性能的变化，刻画了数据扰动造成的影响。
- 噪声：度量在当前任务上任何模型所能达到的期望泛化误差的下界，刻画了学习问题本身的难度。

例：曲线拟合

数据产生的过程为：$y = \sin(2\pi x) + \varepsilon$，其中输入$x$在$[0,1]$均匀采样 25 个点，$\varepsilon \sim N(0, 0.3^2)$。我们重复$L = 100$次试验，每次用 25 个样本点训练一个 25 阶多项式的岭回归模型，岭回归模型的超参数λ在$[10^{-6}, 10^1]$之间的log空间均匀采样 40 个点，即我们尝试了 40 个不同复杂度的

模型。可以看出，当正则参数很小$\lambda = 10^{-6}$时，不同训练样本集得到的模型变化较大，即方差大，此时偏差几乎为 0；最佳模型为$\lambda = 3\times10^{-4}$，此时偏差的平方+方差最小，模型的预测平方误差和也最小。当正则参数很大$\lambda = 10^{1}$时，方差很小，但此时模型过于平滑，偏差很大。

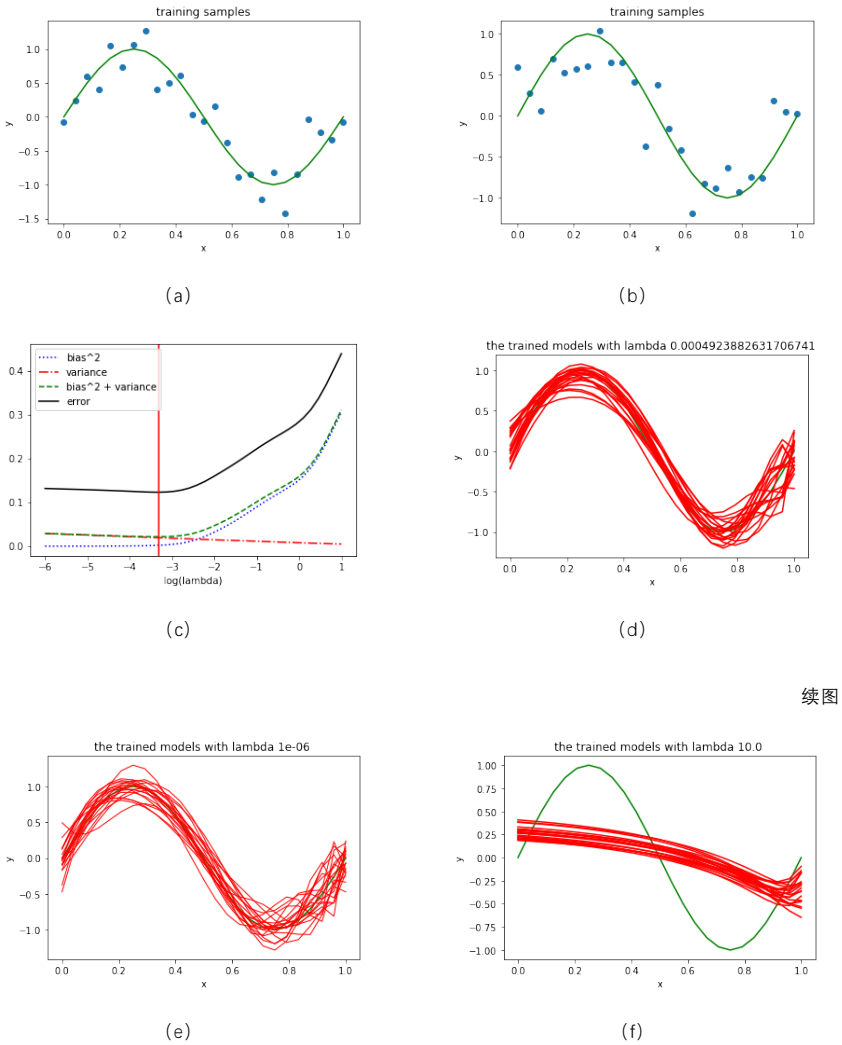


图 7-1　回归模型误差的偏差方差分解示例。(a) 随机采样的训练样本集之一，绿色实线为$y^* = \sin(2\pi x)$；(a) 另一组随机采样的训练样本集；(c) 当模型复杂度变化时，模型的预测误差、偏差的平方、方差，以及偏差的平方+方差；偏差的平方；红色竖线表示最佳超参数的位置；(d) 最佳λ对应的模型（重复了 100 次试验，图中只给出了其中 20 个模型）的预测结果；(e) $\lambda = 10^{-6}$对应模型的预测结果，此时处于过拟合状态，偏差小但方差大；(e) $\lambda = 10$对应模型的预测结果，此时处于欠拟合状态，偏差大但方差小。

7.1.3　降低模型偏差和方差的方法

如果你的学习算法存在着很高的可避免偏差，意味着你可能会尝试以下方法:

- 加大模型规模 (例如神经元/层的数量):这项技术能够使算法更好地拟合训练集，从 而减少偏差。当你发现这样做会增大方差时，通过加入正则化可以抵消方差的增加。
- 根据误差分析结果修改输入特征 :假设误差分析结果鼓励你增加额外的特征，从而帮助 算法消除某个特定类别的误差。(我们会在接下来的章节深入讨论这个话题。)这些新的特征对处理偏差和方差都有所帮助。理论上，添加更多的特征将增大方差;当这种情 况发生

时，你可以加入正则化来抵消方差的增加。

- 减少或者去除正则化 (L2 正则化，L1 正则化，dropout):这将减少可避免偏差，但会增大方差。
- 修改模型架构 (比如神经网络架构)使之更适用于你的问题:这将同时影响偏差和方 差。

如果你的学习算法存在着高方差问题，可以考虑尝试下面的技术:

- 添加更多的训练数据 :这是最简单最可靠的一种处理方差的策略，只要你有大量的数据 和对应的计算能力来处理他们。
- 加入正则化 (L2 正则化，L1 正则化，dropout):这项技术可以降低方差，但却增大了偏差。
- 加入提前终止 (例如根据开发集误差提前终止梯度下降):这项技术可以降低方差但却 增大了偏差。提前终止(Early stopping)有点像正则化理论，一些学者认为它是正则 化技术之一。
- 通过特征选择减少输入特征的数量和种类 :这种技术或许有助于解决方差问题，但也可 能增加偏差。稍微减少特征的数量(比如从 1000 个特征减少到 900 个)也许不会对偏差产生很大的影响，但显著地减少它们(比如从 1000 个特征减少到 100 个，10 倍 地降低)则很有可能产生很大的影响，你也许排除了太多有用的特征。在现代深度学习 研究过程中，当数据充足时，特征选择的比重需要做些调整，现在我们更可能将拥有的 所有特征提供给算法，并让算法根据数据来确定哪些特征可以使用。而当你的训练集很 小的时候，特征选择是非常有用的。
- 减小模型规模 (比如神经元/层的数量):谨 慎使用。 这种技术可以减少方差，同时可 能增加偏差。然而我不推荐这种处理方差的方法，添加正则化通常能更好的提升分类性能。 减少模型规模的好处是降低了计算成本，从而加快了你训练模型的速度。如果加 速模型训练是有用的，那么无论如何都要考虑减少模型的规模。但如果你的目标是减少 方差，且不关心计算成本，那么考虑添加正则化会更好。

下面是两种额外的策略，和解决偏差问题章节所提到的方法重复:

根据误差分析结果修改输入特征 :假设误差分析的结果鼓励你创建额外的特征，从而帮助算法消除某个特定类别的误差。这些新的特征对处理偏差和方差都有所帮助。理论上 ，添加更多的特征将增大方差;当这种情况发生时，加入正则化，这可以消除方差的增 加。 修改模型架构 (比如神经网络架构)使之更适用于你的问题:这项策略将同时影响偏差 和方差。

7.2 Bagging

Bagging 的全称是"**B**ootstrap **agg**rega**ting**"，其中 Bootstrap 是指基学习器的训练样本是对原始训练数据的自助采样（Bootstrap Sampling）得到，aggregating 是指集成学习器的预测结果为多个训练好的基学习器预测结果的平均或投票。

给定包含N个样本的数据集$\mathcal{D}$，自助采样的步骤为：

- 随机取出一个样本放入采样集中，再把该样本放回原始数据集。

- 经过N次上述随机采样操作，得到包含N个样本的采样集。

因此初始训练集中某个样本在采样集中可能出现多次，也有可能不出现。一个样本不在采样集中出现的概率是$\left(1-\frac{1}{N}\right)^N$。$lim_{N\to\infty}\left(1-\frac{1}{N}\right)^N = 0.368$，所以原始训练集中约有 $1-0.368 = 63.2\%$的样本出现在采样集中。

算法 7-1：Bagging

1. for $m = 1, 2, \ldots, M$,

（1）自助采样，得到包含N个训练样本的采样集$\mathcal{D}^{(m)}$,

（2）基于采样集$\mathcal{D}^{(m)}$，训练一个基学习器：f_m。

2. 将M个基学习器进行组合，得到集成模型：

（a）分类任务通常采取简单投票法，取多个基学习器的预测类别的众数。

（b）回归任务通常使用简单平均法，取多个基学习器的预测值的平均：

$$f(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} f_m(\boldsymbol{x})。$$

由于每个基学习器的训练样本来自于对原始训练集的随机采样，每个基学习器也会稍有不同，即 Bagging 中基学习器的“多样性”来自于样本扰动。另外每个基学习器的训练集为一个自助采样集，只用到初始训练集中约 63.2%的样本，这部分样本称为包内数据；剩下的约 36.8%的样本称为包外数据，可用作验证集，用于对泛化性能进行估计，这样无需额外留出验证集。

从偏差-方差分解的角度看，Bagging 通过取多个基学习器的平均，集成模型的方差比基学习器的方差小。根据 7.1.1 节中将样本均值作为分布均值的估计，估计的方差样本数增多而减少。这里 1 个基学习器的预测结果可以视为 1 个样本，Bagging 的结果为样本均值，因此样本均值的方差（Bagging 模型）比单个样本（单个基学习器）的方差小，偏差保持不变，从而集成模型总的性能比单个基学习器好。因此 Bagging 对方差大的模型，如非剪枝的决策树、较复杂的神经网络，效果更为明显。

在 Scikit-Learn 中，Bagging 的实现为BaggingClassifier和BaggingRegressor，分别用于分类任务和回归任务。基学习器可为任意学习器，默认为决策树，既支持从原始样本中随机采样，也支持从所有原始特征中随机采样部分特征，构成基学习器的训练样本。在 Bagging 中，通常基学习器的数目越多，效果越好，所以参数基学习器数目不是模型复杂度参数，无需通过验证集来确定。但随着基学习器数目的增多，测试与训练的计算时间也会随之增加。且当基学习器的数量超过一个临界值之后，算法的效果增加变得不再显著。一个典型的曲线如图 7-2 所示，可以看出，基学习器的数目在 10 到 50 之间时，交叉验证得到的模型正确率显著增大，超过 150 后增加缓慢，因此我们可以取 150 个基学习器。

实际应用中，也可以根据特征维数D简单地设置基学习器数目：对分类问题，可设置基学习器数目为$\sqrt{D}$；对回归问题，可设置基学习器数目为$D/3$。

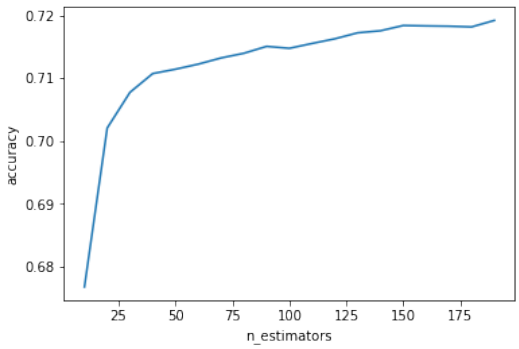


图 7-2　Bagging 中基学习器数目n_estimators与模型性能之间的关系。开始模型性能会随着基学习器数目的增多性能快速提升（1-50），然后性能提升放缓（51-150），后续性能提升不显著（超过 150）。因此基学习器数目取 150 左右比较合适。

7.3 随机森林

7.3.1 随机森林的基本原理

随机森林（Random Forest）是弱学习器为决策树的 Bagging 算法，但更随机。随机森林中每个基学习器的多样性不仅来自样本扰动，还来自属性扰动，即随机采样一部分原始样本的部分属性构成基学习器的训练数据（Scikit-Learn 实现的 Bagging 也支持属性扰动）。这使得集成学习器的泛化性能可以通过基学习器之间差异度的增加而进一步提升。

随机森林的优点：

- 简单、容易实现。
- 训练效率较高，单个决策树只需要考虑所选的属性和样本的子集。
- 多棵决策树可并行训练。
- 在很多现实任务中性能不错，被称作“代表集成学习技术水平的方法”。

随着树的数量的增加，随机森林可以有效缓解过拟合。因为随着树的数量增加，模型的方差会显著降低。但是树的数量增加并不会纠正偏差，因此随机森林的树通常比较复杂，使得模型的偏差不要太大（当然也不能过于复杂，还是可能会过拟合）。

在 Scikit-Learn 中，随机森林支持分类和回归，实现的类分别为 RandomForestClassifier 和 RandomForestRegression。另外 Scikit-Learn 还实现了一种更随机的随机森林，我们称之为极端随机森林，实现的类分别为ExtraTreesClassifier和ExtraTreesRegressor。这种更随机表现在：在分裂时，RandomFores寻找特征最佳阈值；而ExtraTrees随机选取每个候选特征的一些阈值，然后从这些随机选取的阈值中寻找最佳阈值。因此当特征可选划分阈值很多时，ExtraTrees训练更快。在大数据集上，二者的性能差异不大。

7.3.2 案例分析—Otto 商品分类

我们在 Otto 商品分类数据集上采用随机森林模型对商品进行分类。Otto 商品分类数据集的介绍请见 3.7 节。由于特征的单调变换对决策树没有影响，这里我们只采用原始特征+TFIDF

特征。

随机森林和Bagging中，由于学习每个基学习器只用了一部分样本（包内数据），可用其余样本（包外样本）做验证，所以不必再通过交叉验证方式留出验证集，通过设置参数oob_score = True即可用包外数据作为验证集。另外诸如Bagging和随机森林算法，集成模型的结果为多个基本模型的平均，这个平均值很难在 0 和 1 附近，因此需要对其结果输出做概率校准。这可以通过 Scikit-Learn 中的CalibratedClassifierCV类，采用交叉验证的方式得到模型预测值与真实概率之间的映射关系。在 Otto 商品分类任务上，校准后的随机森林的logloss为 0.52046（排名第 1579 位），相比单棵决策树（logloss为 1.07144），性能有了很大提升。

7.4 梯度提升

梯度提升（Gradient Boosting）是另一大类集成学习算法，在众多机器学习任务上取得了优异的成绩。基于决策树的梯度提升算法（Gradient Boosting Decision Tree，GBDT）在工业界广泛应用于点击率预测，搜索排序等任务。GBDT 也是各种数据挖掘竞赛的有利武器。据统计，2015 年 Kaggle 竞赛的优胜解决方案中，有超过一半的解决方案都用了基于 GBDT。

Boosting，译为提升，其基本想法为是否可以将一个弱学习器是修改成为更好的算法。迈克尔·凯恩斯（Michael Kearns）从实践的角度明确提出这个目标：将相对较差的假设转化为非常好的有效算法。弱学习器被定义为一个性能至少略好于随机猜测的学习器。

算法 7-2：提升算法

1．初始化$f_0(\boldsymbol{x})$；

2．for $m = 1{:}M$ do

找一个弱学习器$\phi_m(\boldsymbol{x})$，使得$\phi_m(\mathbf{x})$能改进$f_{m-1}(\boldsymbol{x})$；

更新：$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \phi_m(\boldsymbol{x})$；

3．return $f_M(\boldsymbol{x})$。

梯度提升机（Gradient Boosting Machines，GBM）则是一种实用的实现提升算法的一种方式。GBM 将提升视为一个数值优化问题，通过对目标函数使用类似梯度下降的过程来添加弱学习器。 这类算法也被描述为渐进加法模型（Stage-wised Additive Model），因为每次增加一个新的弱学习器，同时模型中已有的弱学习器被冻结保持不变。这样提升算法可处理任意可微的目标函数，能处理的任务包括两类分类、多类分类、回归，排序学习等。

算法 7-3：梯度提升算法

1．初始化$f_0(\boldsymbol{x})$；

2．for $m = 1{:}M$ do

计算目标函数负梯度：$r_{m,i} = -\frac{\partial J(y_i, f(\boldsymbol{x}_i))}{\partial f}\Big|_{f=f_{m-1}}$；

找一个弱学习器$\phi_m(\boldsymbol{x})$拟合负梯度，使得$\sum_{i=1}^{N}(r_{m,i} - \phi_m(\boldsymbol{x}_i))^2$最小；

更新：$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m\phi_m(\boldsymbol{x})$，其中$\beta_m$为学习率；

3．return $f_M(\boldsymbol{x})$。

例：L2Boosting

当损失函数取 L2 损失$\mathcal{L}(f(\boldsymbol{x}), y) = \frac{1}{2}(y - f(\boldsymbol{x}))^2$时，得到 L2 损失的梯度提升算法

L2Boosting。假设目标函数暂时只考虑训练误差：

$$J(\phi,\beta)=\frac{1}{2}\sum_{i=1}^{N}\left(y_i-f(\boldsymbol{x}_i)\right)^2,$$

目标函数的梯度为：

$$\frac{\partial J}{\partial f}=\frac{\partial\frac{1}{2}\sum_{i=1}^{N}\left(y_i-f(\boldsymbol{x}_i)\right)^2}{\partial f}=\sum_{i=1}^{N}\left(f(\boldsymbol{x}_i)-y_i\right)$$

为当前模型的预测残差之和，从而得到 L2Boosting：

1．初始化：$f_0(\boldsymbol{x})=\overline{y}=\frac{1}{N}\sum_{i=1}^{N}y_i$，因为当预测函数取常数时，样本均值$\overline{y}=\frac{1}{N}\sum_{i=1}^{N}y_i$离所有样本的平均距离平方和最小；

2．for $m=1:M$ do

（1）计算目标函数负梯度：

$$r_{m,i}=-\left.\frac{\partial J\left(y_i,f(\boldsymbol{x}_i)\right)}{\partial f}\right|_{f=f_{m-1}}=-\left(f_{m-1}(\boldsymbol{x}_i)-y_i\right)$$

（2）找一个弱学习器$\phi_m(\boldsymbol{x})$，使得$\sum_{i=1}^{N}(r_{m,i}-\phi_m(\boldsymbol{x}_i))^2$最小

（3）更新：$f_m(\boldsymbol{x})=f_{m-1}(\boldsymbol{x})+\beta_m\phi_m(\boldsymbol{x})$，其中$\beta_m$为学习率。

3．return $f_M(\boldsymbol{x})$

由于梯度提升算法中，由于弱学习器要拟合目标函数的负梯度，因此弱学习器训练是一个回归问题（即使原始问题是一个分类任务）。如果弱学习器采用决策树，则我们采用决策树做回归。

读者可自行推导分类任务中损失函数取负 log 似然损失时，对应的梯度提升算法（logitBoost）。有意思的是，最早的 Boosting 算法 AdaBoost 亦可视为损失函数取指数损失$\mathcal{L}(f(\boldsymbol{x}),y)=exp\left(-yf(\boldsymbol{x})\right)$对应的梯度提升算法，其中$y\in\{-1,1\}$，对应两类分类问题。

虽然 Scikit-Learn 中也实现了 GBM，但后来又出现了性能更好、训练更快的 GBM 的实现：XGBoost 和 LightGBM。XGBoost 和 LightGBM 基本原理类似，但 LightGBM 采用了额外的近似技术，使得训练更快。

7.5 XGBoost

XGBoost（e**X**treme **G**radient **Boost**ing）是提升算法的 C++优化实现，快速高效。自 2015 年发布以来，XGBoost 深受大家喜欢，迅速成为各大竞赛任务的神器，在很多任务上取得优异性能。

7.5.1 XGBoost 基本原理

相较于传统的 GBDT，XGBoost 在目标函数中显式地加入了用于控制模型复杂度的正则项$R(f)$，因此目标函数的形式为：

$$J(f)=\sum_{i=1}^{N}\mathcal{L}\left(f(\boldsymbol{x}_i),y_i\right)+R(f)。\qquad(7\text{-}5)$$

当弱学习器为决策树时，正则项可以包含 L1 正则（树的叶子结点数目T）和 L2 正则（叶子结

点的分数w_t的平方和）：

$$R(f) = \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2, \tag{7-5}$$

其中γ、λ分别为 L1 正则和 L2 正则的权重。

传统 GBDT 算法中，采用的是梯度下降，优化时只用到了一阶导数（用决策树拟合目标函数的负梯度）。而 XGBoost 采用二阶泰勒展开近似损失函数，因此 XGBoost 亦被称为牛顿提升法（Newton Boosting）。

根据二阶泰勒展开公式：$\mathcal{L}(f+\Delta f) \approx \mathcal{L}(f) + \mathcal{L}'(f)\Delta f + \frac{1}{2}\mathcal{L}''(f)\Delta f^2$，

在第m步时，令

$$g_{m,i} = \frac{\partial \mathcal{L}(f(\boldsymbol{x}_i), y_i)}{\partial f}\Big|_{f=f_{m-1}}, \quad h_{m,i} = \frac{\partial^2 \mathcal{L}(f(\boldsymbol{x}_i), y_i)}{\partial^2 f}\Big|_{f=f_{m-1}},$$

对损失函数$\mathcal{L}$在f_{m-1}处进行二阶泰勒展开，得到

$$\mathcal{L}(f_{m-1}(\boldsymbol{x}_i) + \phi(\boldsymbol{x}_i), y_i) = \underbrace{\mathcal{L}(f_{m-1}(\boldsymbol{x}_i), y_i)}_{\text{与未知量}\phi(\boldsymbol{x}_i)\text{无关}} + g_{m,i}\phi(\boldsymbol{x}_i) + \frac{1}{2}h_{m,i}\phi(\boldsymbol{x}_i)^2。$$

忽略与未知量$\phi(\boldsymbol{x}_i)$无关的项，得到

$$\mathcal{L}(f_{m-1}(\boldsymbol{x}_i) + \phi(\boldsymbol{x}_i), y_i) = g_{m,i}\phi(\boldsymbol{x}_i) + \frac{1}{2}h_{m,i}\phi(\boldsymbol{x}_i)^2。$$

对 L2 损失，

$$\mathcal{L}(f(\boldsymbol{x}), y) = \frac{1}{2}(y - f(\boldsymbol{x}))^2, \quad \nabla_f \mathcal{L}(f) = f(\boldsymbol{x}) - y, \quad \nabla_f^2 \mathcal{L}(f) = 1,$$

所以，

$$g_{m,i} = f_{m-1}(x_i) - y_i, \quad h_{m,i} = 1。$$

XGBoost 中，弱学习器采用二叉的回归决策树。我们将树拆分成结构部分q和叶子结点分数部分$\boldsymbol{w}$。对输入$\boldsymbol{x}$，弱学习器的预测为$\phi(\boldsymbol{x})$，即$\boldsymbol{x}$所在叶子结点的分数：

$$\phi(\boldsymbol{x}) = w_{q(\boldsymbol{x})}, \quad q: R^D \to \{1, \dots, T\},$$

其中结构函数q将输入$\boldsymbol{x}$映射到叶子的索引号，T为树中叶子结点的数目，D为输入特征的维数。一个例子如图 7-3 所示。树中有 3 个叶子结点，分别编号为 1，2，和 3，小男孩$\boldsymbol{x}_1$在第一个叶子结点，所以$q(\boldsymbol{x}_1) = 1$；老奶奶$\boldsymbol{x}_2$属于第 3 个叶子结点，所以$q(\boldsymbol{x}_2) = 3$。

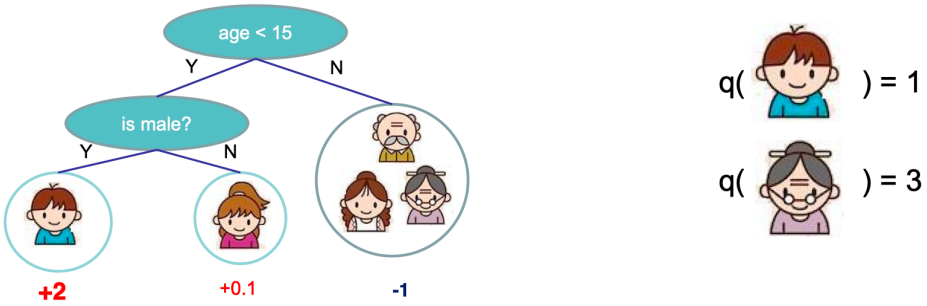


图 7-3　XGBoost 中的弱学习器——回归决策树[8]

假设决策树的结构已知，且令每个叶子结点t上的样本集合为$I_t = \{i | q(\boldsymbol{x}_i) = t\}$，综合损失函数的二阶泰勒展开和正则项，得到目标函数为：

$$
\begin{aligned}
J(\boldsymbol{w}) &= \sum_{i=1}^{N} \mathcal{L}\left(f_{m-1}(\boldsymbol{x}_i) + \phi(\boldsymbol{x}_i), y_i\right) + R(f) \\
&\approx \sum_{i=1}^{N} \left(g_{m,i}\phi(\boldsymbol{x}_i) + \frac{1}{2}h_{m,i}\phi(\boldsymbol{x}_i)^2\right) + \gamma T + \frac{1}{2}\lambda\sum_{t=1}^{T} w_t^2 \\
&= \sum_{i=1}^{N} \left(g_{m,i}w_{q(\boldsymbol{x})} + \frac{1}{2}h_{m,i}w_{q(\boldsymbol{x})}^2\right) + \gamma T + \frac{1}{2}\lambda\sum_{t=1}^{T} w_t^2 \\
&= \sum_{t=1}^{T} \left(\underbrace{\sum_{i\in I_t} g_{m,i}}_{G_t} w_t + \frac{1}{2}\underbrace{\sum_{i\in I_t} h_{m,i}}_{H_t} w_t^2 \right) + \gamma T + \frac{1}{2}\lambda\sum_{t=1}^{T} w_t^2 \\
&= \sum_{t=1}^{T} \left(G_t w_t + \frac{1}{2}H_t w_t^2\right) + \gamma T + \frac{1}{2}\lambda\sum_{t=1}^{T} w_t^2 \qquad (7\text{-}6) \\
&= \sum_{t=1}^{T} \left(G_t w_t + \frac{1}{2}H_t w_t^2 + \frac{1}{2}\lambda w_t^2\right) + \gamma T \\
&= \sum_{t=1}^{T} \left(G_t w_t + \frac{1}{2}(H_t + \lambda) w_t^2\right) + \gamma T。
\end{aligned}
$$

令$\frac{\partial J(\boldsymbol{w})}{\partial w_t} = G_t + (H_t + \lambda)w_t = 0$，得到最佳的模型参数

$$w_t = -\frac{G_t}{H_t + \lambda}。$$

最佳w_t对应的目标函数的值为：

$$J(\boldsymbol{w}) = -\frac{1}{2}\sum_{t=1}^{T}\frac{G_t^2}{H_t + \lambda} + \gamma T。$$

上面我们讨论了已知决策树结构，如何计算每个叶子结点的分数。接下来我们来讨论树结构确定。

最佳的树树结构是使得目标函数J最小的树结构。找到最佳的树结构是一个 NP 困难问题，因此我们通常采用一些启发式规则进行贪心建树。XGBoost 的决策树为分类和回归树（Classification And Regression Tree，CART）。

对树的每个叶子结点，尝试增加一个分裂。根据某个特征的某个阈值，对当前结点中样本集合$\mathcal{D}$中的每个样本，若该样本的特征值小于等于阈值，将该样本划分到的左叶子结点；否则将其划分到右叶子结点。令$\mathcal{D}_L$和$\mathcal{D}_R$分别表示分裂后左右叶子结点的样本集合，定义

$$G_L=\sum_{i\in\mathcal{D}_L} g_{m,i},\quad G_R=\sum_{i\in\mathcal{D}_R} g_{m,i},$$
$$H_L=\sum_{i\in\mathcal{D}_L} h_{m,i},\quad H_R=\sum_{i\in\mathcal{D}_R} h_{m,i},\tag{7-7}$$

则增加该分裂后目标函数的变化为该分裂带来的增益：

$$\text{Gain}=\frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{G_L^2+G_R^2}{H_L+H_R\lambda}\right)+\gamma。\tag{7-8}$$

如果Gain是正的，并且值越大，表示分裂后的目标函数的值越小于分裂前的目标函数值，表示越值得分裂。如果Gain是负的，表明分裂后目标函数反而变大了，则该分裂得不偿失。

穷举搜索所有可能特征、所有可能分裂点，从中选择最佳分裂的精确贪心搜索分裂点的建树过程如算法 7-4 所示。

算法 7-4：穷举精确搜索分裂点

输入：当前结点的样本集合$\mathcal{D}$

特征维度D

输出：最优分裂

1．初始化增益和梯度统计信息：

Gain $\leftarrow 0$

$G=\sum_{i\in I} g_i, H=\sum_{i\in I} h_i$ #分裂前

2．对所有特征都按照特征的数值进行预排序

3．对每个特征$j=1\ to\ D$，执行如下操作：

(1)初始化左子节点的梯度统计信息：　$G_L\leftarrow 0, H_L\leftarrow 0$

(2)对当前结点包含的所有样本，根据特征j的数值排序，遍历排序后取第i个样本的特征值x_{ik}为阈值（for i in $sorted(\mathcal{D}, by\ x_{ij})$），计算左右叶子结点的梯度统计信息和分裂增益，并记录迄今为止增益最大的分裂：

$G_L=G_L+g_i, H_L=H_L+h_i$　#分裂后左叶子结点

$G_R=G-G_L, H_R=H-H_L$　#分裂后右叶子结点

Gain $\leftarrow \max(\text{Gain},\ \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{G_L^2+G_R^2}{H_L+H_R+\lambda}\right)+\gamma)$

4．按最大增益(Gain)对应的分裂特征和阈值进行分裂

7.5.2 XGBoost 优化

XGBoost 在实现细节上做了很多工作，以优化参数学习和迭代速度，包括特征压缩、缺失值处理、样本采样和特征采样、基于直方图的阈值分裂、和特征选择并行等。

- 对缺失值的处理。对于有缺失特征值的样本，XGBoost 自动学习出缺失值走左侧分支还是右侧分支。
- 并行。XGBoost 虽然不能在树粒度并行，但支持特征粒度上的并行。决策树训练最耗时的一个步骤是对特征值进行排序(确定每个特征的最佳分裂阈值需要)。XGBoost 在训练之前，预先对数据进行排序，然后保存为块（block）结构，后面的迭代中重复地使

用这个结构，大大减小计算量。这个块结构也使得并行成为了可能，在进行结点分裂时，需要计算每个特征各阈值对应的分裂的增益，不同特征对应的增益计算可以开多线程进行。这是 XGBoost 比一般 GBDT 快的一个重要原因。

- 根据特征直方图近似寻找最佳分裂点。树结点在进行分裂时，穷举法计算每个特征的每个候选分裂点（每对样本特征值的均值）对应的增益。当数据无法一次载入内存或者在分布式情况下，精确算法效率就会变得很低，所以 XGBoost 还提出了一种直方图近似算法，用于高效地生成候选的分裂点。LightGBM 则完全放弃了穷举搜索，采用直方图方式寻找最佳特征分裂点。

7.5.3 XGBoost 使用指南

XGBoost 的参数较多，下面只介绍常用的参数和 API。更多参数和 API、以及 GPU 和分布式模式等介绍请读者仔细阅读其官方文档：https://xgboost.readthedocs.io/en/latest/。

XGBoost 的参数大致分为三类：

1．通用参数：

- booster：弱学习器类型。可取值gbtree、gbliner、dart，分别表示弱学习器采用决策树、线性模型、带丢弃（Dropout）的决策树，默认值为gbtree。
- nthread：线程数，推荐设置为 CPU 的核数。由于现代的 CPU 都支持超线程，如 4 核 8 线程。此时nthread应设置为 4 而不是 8。默认值为当前系统可以获得的最大线程数。

2．booster参数：控制弱学习器的参数，这里我们只介绍弱学习器为决策树的情况。这部分参数通常需要通过交叉验证等方式针对具体任务进行超参数调优。另外为了兼容，有些参数有多个可选的名字/别名（如learning_rate和eta表示是同一个参数），大家根据自己习惯选用一套表示就好。

- 学习率learning_rate和弱学习器数目n_estimators：通常学习率越小，n_estimators越大。在实际应用中，通常我们固定learning_rate，再采用交叉验证的方式得到最佳的n_estimators。因为 GBDT 中弱学习器的增加是串行（要得m个弱学习器的模型，肯定已有$m-1$个弱学习器的模型），所以我们可以对连续的n_estimators验证其性能，而一般交叉验证中我们通常只能对有限的几个超参数进行验证。在 XGBoost 和 LightGBM 内嵌的交叉验证就是以这种方式实现。learning_rate取值范围为[0,1]，默认值为 0.3。
- 树的最大深度max_depth：建议[3-10]，值越大模型越复杂，默认值为 6。注意因为随机森林不能改变模型的偏差，所以单棵决策树需要足够复杂，max_depth值较大。而 GBDT 中，单棵决策树用来拟合当前目标函数的负梯度，所以决策树不用太复杂，max_depth值无需设得过大。
- 叶子结点最小样本权重和min_child_weight：默认值为 1。这里权重和是指叶子结点中所有样本的 Hessian 和（$H_t=\sum_{i\in I_t}h_i$）。当损失函数取 L2 损失时，$h_i=1$，此时H_t为叶子结点的样本数。min_child_weight值越大，模型越保守（简单）。
- 控制是否预剪枝的参数gamma：如果分裂一个叶子结点带来的损失减少少于gamma，则不再分裂该叶子结点。默认值为 0，值越大，模型越保守。
- 训练决策树的样本（行）采样比例subsample：取值范围为（0,1）。如果设置为 0.6，则意味着从整个样本集合中随机抽取出 60%的样本训练树模型。默认值为 1.0，表示训练每棵树用到所有样本。Subsample值越小，训练单棵决策树的样本越少，训练速度越快，但训练决策树的样本数过小会使得决策树训练不充分。同时不同树的训练样本不同还可以增加决策树之间的多样性。
- 训练决策树的特征（列）采样的比例colsample_bytree：取值范围为（0,1）。如果设置为 0.6，则意味着从所有特征中随机抽取出 60%的特征训练树模型。默认值为 1.0，表示训练每棵树用到所有特征。

- 训练决策树的每个分裂/层的特征（列）采样比例colsample_bylevel：取值范围为(0,1]，默认值为 1.0。
- L2 正则参数lambda：默认为 1，该值越大则模型越简单。
- L1 正则参数alpha：默认为 0，该值越大则模型越简单。
- 正负样本的权重scale_pos_weight：常用于类别不平衡的分类问题。默认为 1，可设置为：负样本数量/正样本数量。

3．学习目标参数：与学习目标有关的参数。

- 任务类型objective：XGBoost 可用于回归、两类分类、多类分类、排序任务，默认为reg: linear，表示线性回归模型。当任务为多类分类问题时，还需设置参数num_class（类别个数）。
- 验证集的评估指标eval_metric：与任务类型有关的模型性能评价指标 。

XGBoost 支持穷举或近似搜索特征的分裂阈值，支持 GPU 和分布式计算，具体详见官方文档。

XGBoost 使用 key-value 字典的方式存储参数，一个典型的参数设置为：

```
params = {
   'booster': 'gbtree',
   'objective': 'multi:softmax',
   'num_class': 3,
   'nthread': 4,
   'learning_rate': 0.1,
   'n_estimators': 200,
   'max_depth': 6,
   'min_child_weight': 1,
   'gamma': 0.1,
   'lambda': 2,
   'subsample': 0.7,
   'colsample_bytree': 0.7
}
```

为了使得训练速度更快，可以先设置learning_rate为一个较大的值（如 0.1），此时最佳的n_estimators较小。在此参数情况下对其他超参数进行调优，最后再调小learning_rate，找最佳的n_estimators。

当模型出现过拟合时，有两类参数可以缓解：

- 第一类参数：直接控制模型的复杂度。包括max_depth，min_child_weight，gamma等参数。
- 第二类参数：增加随机性，从而使得模型在训练时对噪音不敏感。包括subsample，colsample_bytree等。

XGBoost 的常用的 Python API 有两种类型：XGBoost 原生接口和 Scikit-Learn 封装接口。虽然只用一种接口也可以完成任务，但二者搭配使用效率更高。

1．XGBoost 原生接口

在 XGBoost 原生接口中，我们完成一个学习任务的步骤为：

（1）装载数据：XGBoost 的数据存储在DMatrix对象中。XGBoost 支持直接从 libsvm 文本格式的文件、Numpy 的二维数组、Scipy 的稀疏数组和 XGBoost 二进制文件加载数据。

（2）训练模型：Booster 是 XGBoost 的模型，包含了训练、预测、评估等任务的底层实现。但 Booster 对象没有模型训练方法，模型训练可以通过多次调用xgboost. Booster. update()方法或者直接调用xgboost. train()方法或xgboost. cv()方法。在train()方法中，需设置模型超参数、训练数据集、校验集和评价指标、及早停止参数（early_stoppingrounds）等。cv()的功能基本与train()类似，只是校验集通过交叉验证方式得到。如果模型训练时设置了early_stoppingrounds参数，则模型会一直增加弱学习器，直到验证集上的评估指标连续early_stoppingrounds次不再上升为止。如果early_stoppingrounds存在，则模型Booster会生成三个属性，best_score、best_iteration和best_ntree_limit。注意train()会返回最后一次迭代的模型，而不是最佳模型。cv()方法返回历史评价，最后一个元素为最佳的迭代次数。XGBoost 允许在每一轮迭代（每增加一个弱学习器）中使用交叉验证，因此可以方便地获得最优迭代次数。需要注意的是如果需要进行超参数调优，需要自己写代码管理超参数空间的搜索和不同超参数对应的性能。如果采用下面的 Scikit-Learn 接口，可以更方面的结合 Scikit-Learn 中的GridSearchCV进行超参数调优，但寻找最优迭代次数时，如果采用GridSearchCV，只能评估有限个迭代次数。因此实际应用中，我们对迭代次数参数采用 cv()方法进行超参数调优，其他超参数调优可用GridSearchCV。具体请见下一节案例分析代码。

（3）用训练好的模型进行预测：调用Booster的predict()方法。

（4）模型解释和可视化：plot_importance()方法可以给出模型中每个特征的重要性、plot_tree()方法可以将模型中指定的树可视化。

2．Scikit-Learn 封装接口

XGBoost 给出了针对 Scikit-Learn 接口的 API，这样 XGBoost 的调用方式同 Scikit-Learn 中其他学习器（如 Logistic 回归）类似，并且方便采用GridSearchCV进行超参数调优。XGBoost 中分类器和回归器分别为XGBClassifier和XGBRegressor。

7.5.4 XGBoost 案例分析——Otto 商品分类

我们在 Otto 商品分类数据集上采用 XGBoost 对商品进行分类。Otto 商品分类数据集的介绍请见 3.7 节，由于 XGBoost 超参数较多，训练速度较慢，这里我们只使用了原始特征。

在案例中，我们对影响 XGBoost 性能的几个主要超参数进行了调优。由于超参数很多，无法对这些超参数一起进行调优，我们采用类似坐标轴下降方式，一种可选的超参数调优步骤为：

1．设置较小的学习率（learning_rate = 0.1），采用默认超参数，采用xgboost. cv()方法寻找最佳的树的数目n_estimators ；

2．采用GridSearchCV，对参数max_depth和min_child_weight进行超参数调优；

3．采用GridSearchCV，对随机采样参数colsample_bytree和subsample进行超参数调优；

4．采用GridSearchCV，对正则参数lambdal1（reg_alpha）,lambdal2（reg_lambda）进行超参数调优；

5．调小学习率（learning_rate = 0.01），再次xgboost.cv()方法寻找最佳的树的数目n_estimators。

经过上述超参数调优后，XGBoost 在测试集上的 logloss 为 0.44729（排名第 636 位），比采用原始特征+TFIDF 特征的随机森林（0.52046）性能又有了很大提升。

7.6 LightGBM

LightGBM（**Light G**radient **B**oosting **M**achine）是 GBDT 模型的另一个进化版本，由微软提供。LightGBM 原理上和 XGBoost 类似，但训练速度更快、内存消耗更低、支持并行化学习、可处理大规模。LightGBM 的主要特点包括：

- 基于直方图的决策树构造算法
- 直方图做差加速
- 带深度限制的叶子优先的叶子生长策略
- 直接支持离散型特征
- 缓存命中率优化
- 基于直方图的稀疏特征优化
- 多线程优化

7.6.1 基于直方图的决策树构造算法

XGBoost 默认使用预排序算法，能够准确找到分裂点，但是在空间和时间开销大。LightGBM 使用直方图算法，基本思想将连续特征离散成K个离散值，并构造桶（bins）数目为K的直方图，统计落入每个箱子中的样本数。在构造决策树选择特征及特征分裂点时，遍历K个离散值，遍历寻找最优分裂点。直方图算法具体如算法 7-5 所述。

算法 7-5：直方图算法构造决策树（按叶子生长方式）

输入：所有训练数据$\boldsymbol{X}$，当前模型$T_{c-1}(\boldsymbol{X})$
　　所有训练样本的一阶梯度$\boldsymbol{g}$和海森（二阶梯度）值$\boldsymbol{h}$
for all leaf p in $T_{c-1}(\mathcal{D})$ #遍历所有叶子结点
　　for all f in $\boldsymbol{X}.Features$ #遍历所有每个特征
　　H = new Histgorm() #生成新的直方图
　　for i in $p.num\_of\_rows$ #遍历该叶子结点的所有样本
　　$H[f.bins[i]].g \mathrel{+}= g_i$ #每个箱子的一阶梯度和
　　$H[f.bins[i]].h \mathrel{+}= h_i$ #每个箱子的二阶梯度和
　　end
　　Gain $\leftarrow 0$ #初始化
　　for i in $len(H)$ #在所有直方图的箱子中寻找最优分裂点
　　$G_L \mathrel{+}= H[i].g, H_L \mathrel{+}= H[i].h$
　　$G_R = G_p - G_L, H_R = H_p - H_L$
　　Gain $\leftarrow \max(\text{Gain}, \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G_L^2+G_R^2}{H_L+H_R+\lambda}\right) + \gamma)$

```
                    end
                end
            end
        输出：Gain对应的分裂特征和阈值，分裂叶子结点，输出新的模型T_c(X)
```

具体实现时，LightGBM 支持真实海森值和常数海森值（常数值取 1，等于 L2 损失函数对应的海森值，此时海森值之和为样本数）。直方图算法的特征分裂点只能在K个候选中找，找到的并不是精确的分裂点，因此直方图算法是牺牲了一定的分裂准确性，以换取训练速度以及节省内存空间消耗。但在不同的数据集上的结果表明，离散化的分裂点对最终的精度影响并不大，甚至有时候还会更好一点。可能的原因包括：决策树本来就是弱模型，分裂点是不是精确并不是太重要；另外较粗的分裂点也有正则化的效果，可以有效防止过拟合；即使单棵树的训练误差比精确分裂点的算法稍大，但在梯度提升的框架下没有太大的影响。

7.6.2 直方图加速：基于梯度的单边采样

在上述直方图算法中，在构造每个特征的直方图时，需要遍历所有叶子结点的所有样本，即时间复杂度为$O(\#\text{data} \times \#\text{feature})$，其中#feature为特征维度，为#data样本数。如果能降低样本数，训练的时间会大大减少。因此 LightGBM 提出了基于梯度的单边采样（Gradient-based One-Side Sampling，GOSS）算法，减少参与计算的样本数目。

要减少样本数，一个直接的想法是抛弃那些不太重要的样本，而梯度恰好是一个很好衡量样本重要性的指标。如果一个样本的梯度很小，说明该样本的训练误差很小，或者说该样本已经能被模型很好表示。但是直接抛弃梯度很小的样本会改变样训练集的分布，可能会使得模型准确率下降。

GOSS 根据样本的梯度的绝对值进行采样，减少实际访问样本数，步骤如下：

1．根据梯度的绝对值将样本进行降序排序；

2．保留所有的梯度较大的样本：选择前$a\times100$的样本，这些样本称为$\mathcal{A}$；

3．随机采样梯度小的样本：剩下$(1-a)\times100\%$的数据中，随机抽取$b\times100\%$的数据，这些样本称为$\mathcal{B}$；

4．为了抵消对数据分布的影响，在计算增益时，对小梯度数扩大常量倍：对$\mathcal{B}$中样本的梯度$(1-a)/b$倍。

算法 7-6：基于梯度的单边采样（GOSS）算法

```
输入：所有训练数据𝒟
      迭代次数M（弱学习器的数目）
      大梯度的采样比例a
      小梯度采样比例b
      损失函数ℒ
      弱学习器L
models ← 0        #模型初始化
fact ← (1−a)/b    #小梯度样本的梯度缩放因子
topN ← a×len(𝒟)   #大梯度的样本数目
```

```
    randN ← b×len(𝒟)   #随机选择的小梯度的样本数目
    for i = 1 to M
        preds ← models.predict(𝒟)   #当前模型的预测值
        g ← ℒ(𝒟, preds)   #当前损失
        sorted ← GetSortedIndices(abs(g))  #根据梯度的绝对值降序排序
        topSet ← sorted[1:topN]  #取前topN个样本（梯度绝对值最大）
        randSet ← RandomPick(sorted[topN:len(𝒟)], randN)  #在剩下的样本集中随机取randN个样本（梯度小）
        usedSet ← topSet + randSet
        w[randSet] ×= fact   #修改小梯度样本的权重
        newModel ← L(I[usedSet], −g[usedSet], w[usedSet])   #在新样本集上学习新的弱学习器。
        models.append(newModel)
```

7.6.3 直方图加速：互斥特征捆绑

另一种加速直方图构建的方式是减少特征数目。高维数据通常是非常稀疏的，而且很多特征是互斥的（两个或多个特征列不会同时为 0），LightGBM 对这类数据采用互斥特征捆绑（Exclusive Feature Bundling, EFB）策略，将这些互斥特征捆绑成一束（bundle）。通过这种方式，降低特征维度，从而构建直方图的时间复杂度也从$O(\#\text{data} \times \#\text{feature})$变为$O(\#\text{data} \times \#\text{bundle})$。

要完成这个任务，需解决两个问题：

1．哪些特征可以捆绑在一起，组成一束；

2．如何构建特征束，从而实现特征降维。

将特征分组为少量互斥特征束是 NP 困难的。LightGBM 将这个问题转化为图着色问题，每个特征为图$\mathcal{G}$的一个顶点，两个特征的总冲突值为边的权重，图着色问题对相邻的顶点涂上不同的颜色（不互斥的顶点放在不同的特征束），同时总的颜色最少越好（总的特征束越少越好）。如果算法允许小的冲突，可以得到更小的特征束数量，计算效率会更高。证明发现随机污染一小部分特征值，最多影响训练精度$O([(1-\gamma)n]-2/3)$，其中γ是束中最大的冲突比率。通过选取合适的γ，可以在效率和精度之间寻找平衡。

基于上述讨论，特征捆绑算法如算法 7-7 所示。首先对计算结点的度，根据度对特征进行排序。然后对每个特征，看是否加入到一个已有特征束还是新建一个特征束。该过程在训练之前只处理一次，其时间复杂度为$O(\#\text{feature}^2)$。当在特征数目特别多时，直接根据特征的非零值个数排序，即用特征非零值数目近似图结点的度。

算法 7-7：贪心特征捆绑

```
输入：特征F，
      最大冲突计数K
构造图𝒢模型
searchOrder ← 𝒢.sortByDegree()   #根据图中每个特征的度对特质进行排序
bundles ← {}，bundlesConflict ← {}   #初始化
```

```
    for i in searchOrder #按顺序遍历特征
        needNew ← True #初始化，默认需要增加一个新的特征束
        for j = 1 to len(bundles) #对每个已有的束
            cnt ← Conflict(bundles[j], F[i]) #计算特征与已有束的冲突程度
            if cnt + bundlesConflict[i] ≤ K then
                bundles[j].add(F[i]) #将特征加入到束j
                needNew ← False
                break #不再加入其他束
            end
        end
        if needNew then #不能加入到已有的束
            Add F[i] 到一个新的束bundle,
            将bundle加入到bundles
        end
    end
    输出：bundles
```

确定每个特征束后的特征后，我们对每个特征束构造一个直方图用于后续操作。为了在特征束的直方图能区分不同特征，我们用直方图中不同的桶来表示不同的特征。想象将类别型特征独热编码编程成多个特征（编码后的特征互斥）、然后再将这些特征合并成一个直方图表示，直方图的每个桶表示一个特征取值对应的样本数目。当然在 LightGBM 中对类别型特征无需经过上述过程，而是采用 7.6.4 节中的方式直接用直方图表示。特征捆绑的直方图构建类似，只是每个特征可以用多个桶，这样不用特征用直方图中不同的桶的区间表示，后续特征的桶加上偏置量（前面特征已占用的桶的数目）。例如，特征 A 的桶值为[0,10)，特征 B 的桶值为[0,20)，若将两个特征合并，合并后桶的数目为 30，特征 A 的桶的值仍是[0,10)，而特征B的特征桶的值加上偏置量 10，其取值区间将变为[10,30]。

7.6.4 支持离散型特征

在很多应用如 CTR 预估等任务中，有大量的离散型特征，如商品 ID、用户地址等，但大多数机器学习工具都无法直接支持离散型特征，一般需要把类别型特征通过独热编码编码成多维稀疏特征，降低了空间和时间效率。LightGBM 优化了对类别特征型的支持，可以直接输入类别特征，并在决策树算法上增加了类别特征的决策规则。

独热编码是处理类别型特征的一个通用方法，然而在树模型中，这可能并不一定是一个好的方法，尤其当类别型特征中类别个数很多的情况下。主要问题包括：

- 可能无法在这个类别特征上进行切分（即浪费了这个特征）。使独热编码的话，意味着在每一个决策节点上只能使用“1 vs 其他”（例如是狗和不是狗）的分裂方式。当类别值很多时，每个类别上的数据可能会比较少，这时候切分会产生不平衡，这意味着切分增益也会很小。
- 影响决策树的学习。当特征取值较多时，可能会将数据分裂到很多零碎小空间，从而有些小空间的样本数不多，统计信息不准确，使得决策树学习困难。

在 LightGBM 中，通常只有当类别型特征取值很少，才采用独热编码方式。更多其他采用了“多对多”的分裂方式，实现类别型特征的最优分裂。基本思想是将每个特征取值作为一个箱子，建立直方图，并去掉那些样本数少的箱子，然后根据每个箱子中的平均梯度$\frac{G}{H+\lambda}$进行排序。对排序好的直方图，按类似连续特征的方式寻找最佳分裂点。

7.6.5 带深度限制的按叶子生长的叶子生长策略

LightGBM 抛弃了大多数 GBDT 工具使用的按层生长（level-wise）的决策树生长策略，而使用了带有深度限制的按叶子生长（leaf-wise）算法。按层生长过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度。但按层生长不加区分的对待同一层的叶子，带来了一些没必要的开销，因为有些叶子的分裂增益较低，没必要进行搜索和分裂。

按叶子生长策略每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。因此同按层生长相比，在分裂次数相同的情况下，按叶子生长策略可以降低更多的误差，得到更好的精度。但按叶子生长策略可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在按叶子生长策之上增加一个最大深度的限制，在保证高效率的同时防止过拟合。

7.6.6 LightGBM——Otto 商品分类

LightGBM 的使用和 XGBoost 基本类似，一些不同点包括：

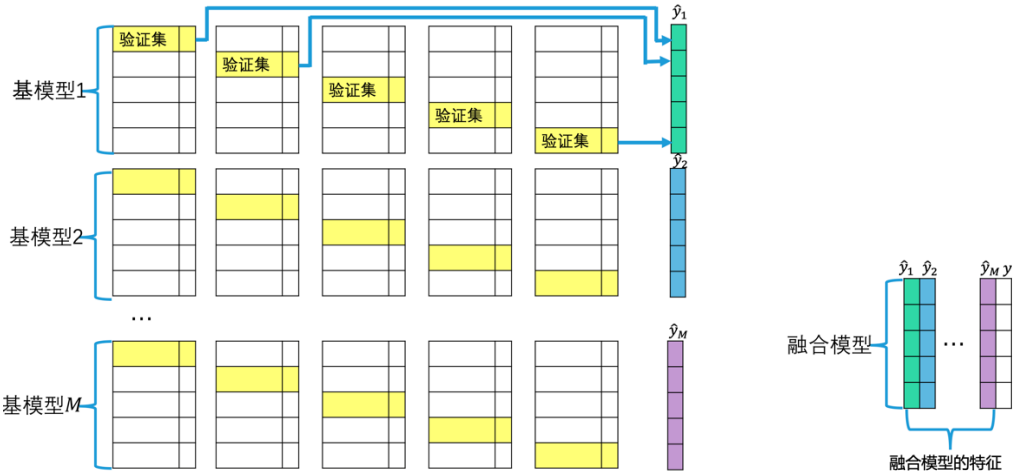
1 .LightGBM 采用按叶子生长方式构造决策树，因此超参数叶子结点数目num_leaves比数的最大深度max_depth更重要。通常我们将num_leaves可在 50-90 之间搜索最佳，max_depth设置成满足$\text{num_leaves} < 2^{\text{max_depth}}$。

2 . LightGBM 采用直方图方式，可以设置每个特征直方图的桶的数目max_bin，默认值为 255。

由于 LightGBM 训练速度比 XGBoost 更快，我们可以考虑更多特征，以获得更好的性能。在 Otto 商品分类任务上，我们采用了原始特征和 TFIDF 变换后的特征（特征重要性表明 TFIDF 变换后的特征更重要）。案例中boosting_type设置为‘gbdt’，在测试集上的性能为 0.44366。如果希望速度更快，可考虑设置boosting_type为‘goss’。

7.7 融合

基于融合的集成学习可以将不同的基学习器组合起来。将训练数据分为两部分，其中较多的部分用于训练基学习器，较少的部分用于训练融合模型。如果训练数据不足够多，可采用交叉验证方式将训练数据分成K份，其中每$K-1$用于训练基学习器，剩下的 1 份数据用于产生集成模型的训练集，训练融合模型。模型融合流程如图 7-4 所示。



（a）采用交叉验证方式训练基模型，得到融合模型的输入特征$\hat{y}_{i,j}, i = 1,2 \ldots, N, j = 1,2, \ldots, M$　　（b）训练融合模型

图 7-4　采用交叉验证方式的模型融合示意

融合集成算法的工作流程如下：

- 1. 训练基学习器。若采用K交叉验证方式，对每一种基模，每次用$K-1$份数据训练该基模型，然后对剩下的 1 份数据做预测，预测结果为融合模型对应样本的特征之一。所以若有M种基模型，融合模型的输入特征为M维。由于每折交叉验证的训练数据不同，每种基模型会有K个不同的模型。这些模型对每个测试样本进行预测，会得到K个预测值，这K个值的平均作为融合模型的该测试样本的输入特征。
- 2. 训练融合模型。融合模型通常采用简单的线性回归或 Logistic 回归。

7.8 小结

Bagging 和融合集成学习中，各基学习的地位相同，可以并行训练各个基学习器。通常基学习器的多样性越大，集成模型的性能更好。这种多样性一般通过学习过程中引入随机性实现。如果基模型的类型相同，可通过对样本、输入特征、输出表示、算法参数进行随机扰动。通常对决策树和神经网络等“不稳定基学习器”（训练样本稍加变化就会导致学习器有显著的变动），采用训练样本扰动（如随机采样）；对线性学习器、支持向量机、朴素贝叶斯、 近邻学习器等"稳定基学习器"（对数据样本的扰动不敏感），采用输入特征扰动等其他机制。输出扰动对训练样本的输出表示稍作变动，而算法参数扰动是指基学习器的超参数设置成不同值或随机设置，从而产生差别较大的基学习器。当然不同的多样性增强机制可以同时使用。如随机森林同时是用了数据样本扰动和输入属性扰动。

提升算法中不同基学习器不能并行训练，当然也可以看成是每个基学习器的训练样本的输出不同（目标函数的当前梯度），看成是一种特殊的输出扰动。

7.9 练习

1. 下列关于 Bagging 和提升算法的描述中，哪些是正确的？

（A） 在 Bagging 中，每个弱学习器都是独立的；

（B） Bagging 是通过对弱学习器的结果进行综合来提升能力的方法；

（C） 在提升算法中，每个弱学习器是相互独立的；

（D） 提升算法是通过对弱学习器的结果进行综合来提升能力的方法。

2. 在随机森林里，你生成了几百颗树，然后对这些树的结果进行综合，下面关于随机森林中每颗树的说法哪些是正确的？

（A） 每棵树是通过数据集的子集和特征的子集构建的；

（B） 每棵树是通过所有的特征构建的；

（C） 每棵树是通过所有数据的子集构建的；

（D） 每棵树是通过所有的数据构建的。

3. XGBoost 中，下面关于超参数max_depth的的说法，哪些是正确的?

（A） 对于相同的验证准确率，越低越好；

（B） 对于相同的验证准确率，越高越好；

（C） max_depth 增加可能会导致过拟合；

（D） max_depth 增加可能会导致欠拟合。

4. 如果随机森林模型现在处在欠拟合状态，下列哪个操作可以提升其性能？

（A） 增大叶子结点的最小样本数；

（B） 增大树的最大深度；

（C） 增大中间结点分裂的最小样本数。

5. 推导两类分类任务中交叉熵损失（不考虑正则项）的梯度提升算法。

6. 推导两类分类任务中指数损失（不考虑正则项）模型的梯度提升算法。

7. 请用随机森林和 LightGBM 对 3.9 节的第 10 题的数据进行建模，并比较模型与 Logistic 回归模型和 SVM 的性能。

8. 请用随机森林和 LightGBM 对 2.9 节的第 6 题的数据进行建模，并比较这些模型与线性回归模型和 SVR 的性能。