第5章 集成学习

机器学习目标是学习出一个稳定的且在各个方面表现都较好的模型，但实际情况往往不这么理想，有时只能得到多个有偏好的学习器（弱学习器，在某些方面表现得比较好），集成学习是将多个弱学习器进行集成组合以期得到一个更好、更全面的强学习器。集成学习本身不是一个单独的机器学习算法，而是通过构建并结合多个机器学习器完成学习任务，通常预测能力比使用单个学习器更优越。

5.1 基本概念

5.1.1 算法起源

集成学习借鉴了“群体智慧”这一思想，也就是常说的“三个臭皮匠，顶个诸葛亮”的道理，即多个才能平庸的人，若能集思广益，也能提出比诸葛亮还周全的计策。1906年，Galton创造了“群体智慧”一词。有一次他参加一个农贸展销会，那里正在举办一场肉牛质量竞猜比赛，牛已经宰好了，内脏掏得一干二净。他用最准确的猜测得分从800名选手中脱颖而出。其做法是收集所有猜测结果并加以分析。他计算了所有猜测得分的均值，惊讶地发现计算结果和实际值极为接近。这种集体竞猜的方法超过所有曾经赢得比赛的选手，甚至跟养牛专业户相比也胜过一筹。选手要取得准确得分，关键在于集成学习。选手独立给出的猜测得分不能受周围他人猜测得分的影响，同时通过一个无差错机制（平均值）强化整个群体的猜测得分。

集成一词的本义是“群策群力”。构建集成分类器，首先通过训练数据产生一组分类器，然后聚合分类器的预测结果，再基于这些结果预测新记录的类别。

5.1.2 基本概念

先定义一些基本的关于学习器的概念。

强学习器（Strong Learner），是相对于弱学习器而言的概念，指的是可以预测相当准确结果的学习算法。

弱学习器（Weak Learner），相对于强学习器而言，通常弱学习器预测的结果只比随机结果稍好一些。

基学习器（Base Learner），集成学习中的个体学习器，经常是弱学习器，但是并非必须为弱学习器，有时可以为强分类器。

基学习算法（Base Learning Algorithm），基学习器所基于的算法，基学习器基于基学习算法生成。

同质基学习器（Homogeneous Base Learner），指使用同样的基学习算法生成的基学习器。

异质基学习器（Heterogeneous Base Learner），指使用不同的基学习算法生成的基学习器。

机器学习算法中的有监督学习指的是利用已经知道类别的样本训练分类器或回归模型，有监督学习算法处理的是分类和回归问题，分类指的是利用一群已经知道类别的样本训练分类器，分类处理的是离散数据；而回归则是利用已知结果且结果为连续数值的样本建立回归模型，处理的是连续数据。现有的分类算法有多种，例如决策树、朴素贝叶斯、支持向量机等。

训练分类器（学习器）的目标是能够从合理数量的训练数据中通过合理的计算量可靠地学习到知识，训练样本的数量和学习所需的计算资源是密切相关的，由于训练数据的随机性，我们只能要求分类器可能学习到一个近似正确的假设，即可能近似正确（Probably Approximate Correct，PAC）学习。PAC学习理论定义了学习算法的强弱。

弱学习算法：识别错误率小于1/2（即准确率仅比随机猜测略高）的学习算法；强学习算法：识别准确率很高并能在多项式时间内完成的学习算法。同时，Valiant和Kearns提出了PAC学习模型中弱学习算法和强学习算法的等价性问题，即任意给定仅比随机猜测略好的弱学习算法，是否可以将其提升为强学习算法，如果可以，那么只找到一个比随机猜测略好的弱学习算法就可以将其提升为强学习算法，而不必寻找很难获得的强学习算法。后来Scphaire证明，强可学习和弱可学习是等价的，弱学习算法可以提升为强学习算法。

对于分类问题而言，给定一个训练样本，求比较粗糙的分类规则比求精确的分类规则要容易得多，由于强可学习和弱可学习是等价的，所以可以通过先找到一些弱分类器，然后寻求将这些弱分类器提升为强分类器的方法来提升分类效果。

集成学习是一个复合模型，由多个基学习器通过组合策略建立一个精确度高的集成分类器。集成学习通常比其基学习器更准确，泛化性能更优越。集成学习按照基学习器（个体学习器）之间是否存在依赖关系可以分为两大类：第一类是个体学习器之间存在强依赖关系，是串行集成分类方法；另一类是个体学习器之间不存在强依赖关系，是并行集成分类方法。在串行集成中，先生成的基学习器会影响后续生成的基学习器，串行集成方法的基本动机是利用基学习器之间的相关性；并行集成分类中，各基学习器互不影响，并行集成方法的基本动机则是利用基学习器之间的独立性，这是因为组合相互独立的基学习器能够显著减小误差。

按照基学习算法的异同，集成的方法可以分为同质集成和异质集成，同质集成是通过一个基学习算法生成同质的基学习器，异质集成的基学习器是不同质的，为了集成后的结果表现最好，异质基学习器需要尽可能准确并且差异性够大。按照个体学习器之间的关系，同质集成学习方法可以分为并行集成的装袋方法Bagging、串行集成的提升方法Boosting和堆叠方法Stacking。

Bagging是一种并行集成学习算法，每个基学习器没有依赖关系，可以并行拟合，Bagging算法是在原始的数据集上采用有放回的随机取样（也叫自助采样）的方式抽取m个子样本，从而利用这m个子样本训练m个基学习器，从而降低模型的方差。装袋方法Bagging的代表算法是随机森林（Random Forest，RF）算法。

Boosting是一种将弱学习器提升为强学习器的串行集成学习算法。该算法先从初始训

练集训练出一个基学习器，再根据基学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多的关注，然后基于调整后的样本分布训练下一个基学习器；如此重复进行，直至基学习器数目达到事先指定的值 T，最终将这 T 个基学习器进行加权组合。Boosting 的代表算法是 AdaBoost、GBDT、XGBoost 等。

堆叠 Stacking 首先使用原始训练数据集学习出若干个基学习器后，使用基学习器的输出作为输入特征，将这几个学习器的预测结果作为新的训练集，来学习一个新的学习器。因此，堆叠 Stacking 是一种通过一个元分类器或者元回归器整合多个分类模型或回归模型的集成学习技术。基础模型通常包含不同的学习算法，因此 Stacking 通常是异质集成的。Stacking 与 Bagging 和 Boosting 主要存在两方面差异：首先，Stacking 通常考虑的是异质学习器（不同的学习算法被组合在一起），而 Bagging 和 Boosting 主要考虑的是同质学习器；其次，Stacking 学习用元模型组合基础模型，而 Bagging 和 Boosting 则根据确定性算法组合基学习器。限于篇幅限制，本书不再讨论堆叠 Stacking 代表的异质集成学习，有兴趣的读者可以自行查阅相关资料。

5.2　Bagging 算法与随机森林

5.2.1　Bagging 算法

Bagging 是 Bootstrap Aggregating 的缩写，是自助结合的意思。“自助”和“结合”是 Bagging 的两个关键，先对数据进行多次自助采样（Bootstrap Sample，也称为可重复采样或有放回采样），再在每个新样本集上训练模型并结合。该算法的基本思想是：从原始的数据集中自助采样抽取同数量的样本，形成 K 个随机的新训练集，然后训练出 K 个不同的学习器，再一起集成。学习器间不存在强依赖关系，集成方式一般为投票。

Bagging 通过抽样创建训练数据集的子样本并在每个子样本上拟合决策树来工作。从训练集进行子抽样组成每个基模型所需要的子训练集，对所有基模型预测的结果进行综合产生最终的预测结果。

图 5.1 为 Bagging 算法原理图。每个分类器都随机从原样本中做有放回的采样，经过多次采样后获得多个平衡的训练子集；然后分别在这些采样后的样本上训练相应的基分类器，再把这些基分类器采用投票方法集成多个基分类器的预测结果。Bagging 算法既可以处理二分类问题，也可以处理多分类问题。Bagging 算法的具体过程如下。

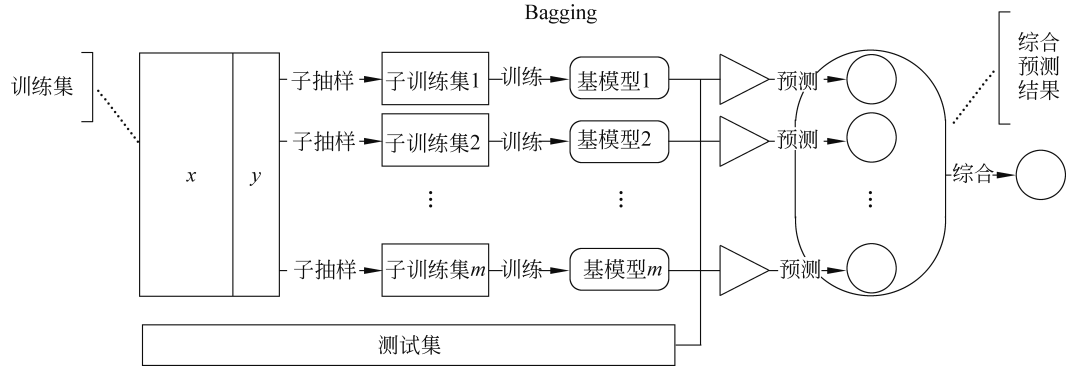

图 5.1　Bagging 算法原理图

从原始样本集中通过随机抽取形成 K 个训练集：每轮抽取 N 个训练样本（有些样本可能被多次抽取，而有些样本可能一次都没有被抽中，这称为有放回的抽样）。K 个训练集共得到 K 个模型。这 K 个训练集是彼此独立的，这个过程也称为 Bootstrap。

算法流程如下。

输入：样本数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$；
　　基学习算法 ε；
　　基学习器数 T。
对于 $t=1,2,\cdots,T$，
步骤 1：自助采样 T 套样本数量为 D 的数据集。
步骤 2：在第 t 套数据集上训练出基学习器 h_t，最后均匀结合 h_t 成 H。
输出：分类问题：用投票方式，$H(x)=\operatorname{sign}\left(\sum_{t=1}^{T}h_t(x)\right)$
　　回归问题：用平均方式，$H(x)=\frac{1}{T}\sum_{t=1}^{T}h_t(x)$

训练时可以根据具体问题采用不同的分类或回归方法，如决策树、神经网络等。每次使用一个训练集通过相同的分类或回归方法得到一个模型，K 个训练集共得到 K 个模型。通常把这些模型称为基模型或者基学习器。在测试阶段，每个测试周期会将全部 K 个训练模型结果组合起来进行预测。

基模型的集成有两种情况。对于分类问题采用投票法，K 个模型采用投票的方式得到分类结果；对于回归问题采用平均法，计算 K 个模型的均值并将其作为最后的结果。

5.2.2 随机森林

随机森林是通过集成学习的思想将多棵决策树集成的一种集成学习（Ensemble Learning）算法，其基本单元是决策树，每棵决策树都是一个分类器。对于一个输入样本，N 棵树会有 N 个分类结果，而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出，是 Bagging 类算法的典型代表。它并行地生成基分类器，并行集成的基本动机是利用基学习器的独立性，通过平均降低方差。

大多数情况下的 Bagging 是基于决策树的，本章默认采用 CART 算法构造决策树。随机森林由多棵不确定性的决策树构建而成，有效解决了单分类器出现的过拟合以及性能差等问题，是最具代表性的集成分类学习算法。由于随机森林算法具有易于理解与实现、准确度高且开销较低等特点，因此其在科技、交通、医学等领域得到了广泛的使用。

随机森林是 Bagging 的扩展，与 Bagging 一样，随机森林在训练数据集上有放回地随机抽样来拟合决策树。随机森林在随机抽取样本数据的同时，也对样本属性进行随机抽取，进一步体现了随机森林分类模型的灵活性，避免出现过拟合的问题。与 Bagging 不同，随机森林还对每个数据集的特征（列）进行采样。

随机森林的构造基本过程：假设用于建模的训练数据集中含有 N 个观测样本、M 个自变量和 1 个因变量，首先利用 Bootstrap 抽样法从原始训练集中有放回地抽取出 $n(n<N)$ 个观测样本用于构建单棵决策树；然后从 M 个自变量中随机选择 $m(m<M)$ 个字段用于 CART 决策树结点的字段选择；最后根据 Gini 指数生成一棵未经剪枝的 CART 树。最终

通过多轮的抽样，生成 k 个数据集，进而组装成含有 k 棵树的随机森林。

随机森林的随机性体现在两方面：每棵树的训练样本是随机的；树中每个结点的分裂字段是随机选择的。两个随机性的引入，使得随机森林不容易陷入过拟合，随机选择样本和 Bagging 相同，采用的是 Bootstrap 自助采样法；随机选择特征是指在每个结点以及在分裂过程中都是随机选择特征的。这种随机性导致随机森林的偏差会稍微增加（相比于单棵不随机树），但是随机森林的“平均”特性，会使得它的方差减小，而且方差的减小补偿了偏差的增大，因此总体而言是更好的模型。

随机森林是集成学习思想下的产物，它将许多棵决策树整合成森林，决策树之间没有任何关联，它们合起来用来预测最终结果。随机森林的构建步骤如下。

步骤 1：设样本数量为 N，有放回地随机选择 n 个样本（$n<N$）来构造决策树。

步骤 2：设该样本的属性数量为 M，随机选择 m 个属性（$m<M$）作为构造决策树的特征。

步骤 3：将 m 个属性采用 ID3、C4.5 或 CART 算法进行最优分裂构造决策树。

步骤 4：重复步骤 1～步骤 3，构造大量决策树，形成随机森林。

这样，随机森林会通过所有决策树对待预测的输入数据进行训练，然后取加权平均或多数投票得出最终的预测结果。图 5.2 为随机森林构建流程图，对于每个结点：随机选择部分数据子集，求得分割最优化的变量（及其值）。

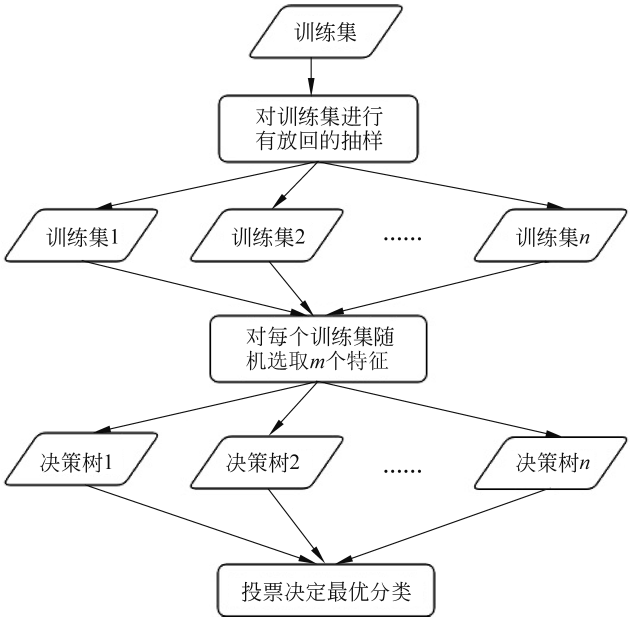

图 5.2　随机森林构建流程图

随机森林采用了随机选择样本数量和样本属性构建大量决策树的方法来避免过拟合，同时可以有效处理数据量大且维度高的样本集，并且不用降维，无须做特征选择，在建模过程中训练速度较快且可以并发化训练模型。

算法实例

表 5.1 是二维数据表，表示 8 位男生的属性以及女孩对应的见面决策。

表 5.1　8位男生的属性以及女孩对应的见面决策 1

男生编号	长　相	性　格	工资/元	类　别
1号	帅	好	60万	见
2号	一般	坏	45万	不见
3号	丑	好	80万	见
4号	帅	坏	15万	见
5号	一般	坏	5万	不见
6号	一般	坏	2万	不见
7号	丑	好	35万	见
8号	丑	好	25万	见

根据数据的二维结构，生成随机数据的方法有如下两种。

（1）在样本上随机（行采样）：采用有放回采样，在采样得到的样本集合中可能有重复的样本。通常从含有 N 个原始样本集中有放回地采样出一个新的含有 n 个样本的样本集。随机选行，例如得到如表5.2和表5.3所示的结果。

表 5.2　随机行采样结果 1

男生编号	长　相	性　格	工资/元	类　别
1号	帅	好	60万	见
3号	丑	好	80万	见
6号	一般	坏	2万	不见
8号	丑	好	25万	见

表 5.3　随机行采样结果 2

男生编号	长　相	性　格	工资/元	类　别
1号	帅	好	60万	见
3号	丑	好	80万	见
6号	一般	坏	2万	不见
8号	丑	好	25万	见

（2）在特征上随机（列采样）：因为树可以在特征集上分裂，从 m 个特征中随机选择 j 个，通常 $j=\sqrt{m}$ 或 $j=\log_2 m$，并且规定这个分裂点只能在这 j 个特征上进行分裂。随机选列，例如得到如表5.4和表5.5所示的随机选列结果。

表 5.4　随机列采样结果 1

男生编号	长　相	工资/元	类　别
1号	帅	60万	见
2号	一般	45万	不见

续表

男生编号	长　相	工资/元	类　别
3号	丑	80万	见
4号	帅	15万	见
5号	一般	5万	不见
6号	一般	2万	不见
7号	丑	35万	见
8号	丑	25万	见

表5.5 随机列采样结果2

男生编号	长　相	性　格	类　别
1号	帅	好	见
2号	一般	坏	不见
3号	丑	好	见
4号	帅	坏	见
5号	一般	坏	不见
6号	一般	坏	不见
7号	丑	好	见
8号	丑	好	见

因此，随机森林有两种常见的形式：第一种是只做行采样；第二种是既做行采样，又做列采样。第二种随机森林中的树更加随机，因此在实践中用得也更多。

随机选出数据和特征之后，使用完全分裂的方式建立决策树。一般决策树算法中都有一个重要的步骤，即剪枝，但是在随机森林中不需要此步骤。由于之前介绍的两种随机采样过程保证了随机性，因此只要森林中的树足够多，就算不剪枝，也不容易出现过拟合。在生成森林之后，对于一个新的输入样本，森林中的每棵决策树会分别对其进行判断。

分类树：对于每个样本在不同树中得到的类别，找出得票最多的类别作为最终类别。

回归树：对于每个样本在不同树中得到的数值，求它们的平均值并将其作为最终数值。

由于在数据和特征上都注入了随机的成分，因此，可大致认为随机森林中的每棵决策树之间是相互独立的。

包外估计：对样本数量为 n 的初始数据集自助采样，如果采样集的样本数量也为 n，那么没有被选到的样本大概占 $(1-1/n)^n$，当 n 很大时，则有下列的极限公式：

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n=\lim_{n\to\infty}\frac{1}{\left(1+\frac{1}{-n}\right)^{-n}}=\frac{1}{\mathrm{e}}\approx 0.368 \tag{5.1}$$

因此，每做这样一次自助采样，初始数据集中只有63.2%的数据被选中当作训练数据，剩下36.8%没被选中的数据可以自动作为验证数据。这些验证数据可以对随机森林的泛化能力做包外(Out-Of-Bag，OOB)估计。

特征选择：特征选择(Feature Selection)的目的是选择需要的特征，将冗余的、不相关的特征忽略。线性模型的特征选择根据其重要性选择特征。

但是，对于非线性模型，特征选择就没有这么简单了。随机森林虽是非线性模型，但是其特有的机制可以让其很容易做到初步的特征选择。其核心思想是："如果特征 j 是重要特征，那么加入一些随机噪声后模型性能会下降。"直接在原有数据上加入正态分布的噪声好吗？不好，因为这样做会改变原有数据的分布。常用的做法是：把所有数据在特征 j 上的值重新随机排列，此做法被称为置换检验。这样可以保证随机打乱的数据分布和原有数据接近一致。表 5.6 展示了在"性格"特征上随机排列后的数据，随机排列将"好坏好坏坏坏好好"排成"坏坏好坏好坏坏好"。对表 5.1 的数据进行置换之后的结果如表 5.6 所示。

表 5.6　8 位男生的属性以及女孩对应的见面决策 2

男生编号	长　相	性　格	工资/元	类　别
1 号	帅	坏	60 万	见
2 号	一般	坏	45 万	不见
3 号	丑	好	80 万	见
4 号	帅	坏	15 万	见
5 号	一般	好	5 万	不见
6 号	一般	坏	2 万	不见
7 号	丑	坏	35 万	见
8 号	丑	好	25 万	见

在"性格"一列进行置换。置换检验后，某个特征 j 的重要性可被看成森林"在原有数据中的性能"和"在某特征 j 数据置换之后的性能"的差距：

$$\text{重要性 } j = |\text{性能}(\overbrace{D}^{\text{原有数据}}) - \text{性能}(\overbrace{D^P}^{\text{原有数据}})|$$

但是这样做太耗时，还要重新训练一遍随机森林。利用随机森林的 OOB 数据，可以节省时间：

$$\text{重要性 } j = |\text{性能}(\overbrace{D}^{\text{原有数据}}) - \text{性能}(\overbrace{D^P}^{\text{原有数据}})| \approx |\text{误差}(\overbrace{D_{\text{OOB}}}^{\text{原有OOB数据}}) - \text{误差}(\overbrace{D^P_{\text{OOB}}}^{\text{置换后的OOB数据}})|$$

这样就不用重新训练模型了。训练好森林后，将每棵树 h_t 对应的 OOB 数据，在特征 j 上随机打乱(注意，现在随机打乱的是 OOB 数据，而不是全部数据)，分别计算打乱前和打乱后的误差，最后在森林层面再求 T 个误差差值的平均值，公式如下。

$$\text{重要性 } j = \frac{1}{T}\sum_{t=1}^{T} |\text{误差}(\overbrace{D_{\text{OOB}(t)}}^{\text{第}t\text{棵树原有的OOB数据}}) - \text{误差}(\overbrace{D^P_{\text{OOB}(t)}}^{\text{第}t\text{棵树置换后的OOB数据}})|$$

若给特征 j 随机加入噪声，则 OOB 数据的误差率会大幅提高，而重要性 j 也会大幅提高，因此该特征比较重要而被选择。

算法应用

sklearn 的子模块 ensemble 提供了产生随机森林的"类"RandomForestClassifier，该"类"的语法和参数含义如下。

```
RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
```

```
                    min_samples_split=2, min_samples_leaf=1,
                    min_weight_fraction_leaf=0.0, max_features=`auto`,
                    max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, bootstrap=True,
                    oob_score=False, n_jobs=1, random_state=None,
                    verbose=0, warm_start=False, class_weight=None)
RandomForestRegressor(n_estimators=10, criterion='mse', max_depth=None,
                    min_samples_split=2,min_samples_leaf=1,
                    min_weight_fraction_leaf=0.0, max_features=`auto`,
                    max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, bootstrap=True,
                    oob_score=False, n_jobs=1, random_state=None,
                    verbose=0, warm_start=False)
```

n_estimators：用于指定随机森林所包含的决策树个数。

criterion：用于指定每棵决策树结点的分割字段所使用的度量标准，用于分类的随机森林，默认的 criterion 值为'gini'；用于回归的随机森林，默认的 criterion 值为'mse'。

max_depth：用于指定每棵决策树的最大深度，默认不限制树的生长深度。

min_samples_split：用于指定每棵决策树根结点或中间结点能够继续分割的最小样本量，默认为 2。

min_samples_leaf：用于指定每棵决策树叶子结点的最小样本量，默认为 1。

min_weight_fraction_leaf：用于指定每棵决策树叶子结点最小的样本权重，默认为 None，表示不考虑叶子结点的样本权值。

max_features：用于指定每棵决策树包含的最多分割字段数，默认为 None，表示分割时使用所有的字段。

max_leaf_nodes：用于指定每棵决策树最大的叶子结点个数，默认为 None，表示对叶子结点个数不做任何限制。

min_impurity_decrease：用于指定每棵决策树的结点是否继续分割的最小不纯度值，默认为 0。

bootstrap：bool 类型参数，是否对原始数据集进行 Bootstrap 抽样，用于子树的构建，默认为 True。

oob_score：bool 类型参数，是否使用包外样本计算泛化误差，默认为 False。包外样本是指每次 Bootstrap 抽样时没有被抽中的样本。

n_jobs：用于指定计算随机森林算法的 CPU 个数，默认为 1。

random_state：用于指定随机数生成器的种子，默认为 None，表示使用默认的随机数生成器。

verbose：用于指定随机森林计算过程中是否输出日志信息，默认为 0，表示不输出。

warm_start：bool 类型参数，是否基于上一次的训练结果进行本次的运算，默认为 False。

class_weight：用于指定因变量中类别之间的权重，默认为 None，表示每个类别的权重都相等。

本节利用随机森林对 Titanic 乘客数据集进行乘客存活预测，该数据集一共包含 891 个观测样本和 12 个变量。其中变量 Survived 为因变量，1 表示存活，0 表示未存活。具体

Python 实现代码如下。

1. 导入数据并对数据集进行拆分

```
#导入第三方包
from sklearn import model_selection
import pandas as pd
#读入数据
Titanic = pd.read_csv(r'C:\Users\Administrator\Desktop\Titanic.csv')
#取出所有自变量名称
predictors = Titanic.columns[1:]
#将数据集拆分为训练集和测试集,且测试集的比例为 25%
X_train,X_test, y_train, y_test = model_selection.train_test_split(Titanic
[predictors], Titanic.Survived, test_size = 0.25, random_state = 1234)
```

2. 构建随机森林模型

```
#导入第三方包
from sklearn.ensemble import RandomForestClassifier
#构建随机森林
RF_class = RandomForestClassifier(n_estimators=200, random_state=1234)
#随机森林的拟合
RF_class.fit(X_train, y_train)
#模型在测试集上的预测
RFclass_pred = RF_class.predict(X_test)
#模型的准确率
print('模型在测试集的预测准确率:\n',metrics.accuracy_score(y_test, RFclass_
pred))
```

输出结果如下。

模型在测试集的预测准确率：

0.852017937219731

利用随机森林对数据进行分类，预测准确率超过 85%。

为了进一步验证模型在测试集上的预测效果，需要绘制 ROC 曲线，结果如图 5.3 所示，代码如下。

```
#计算绘图数据
y_score = RF_class.predict_proba(X_test)[:,1]
fpr,tpr,threshold = metrics.roc_curve(y_test, y_score)
roc_auc = metrics.auc(fpr,tpr)
#绘图
plt.stackplot(fpr, tpr, color='steelblue', alpha = 0.5, edgecolor = 'black')
plt.plot(fpr, tpr, color='black', lw = 1)
plt.plot([0,1],[0,1], color = 'red', linestyle = '--')
plt.text(0.5,0.3,'ROC curve (area = %0.2f)' % roc_auc)
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.show()
```