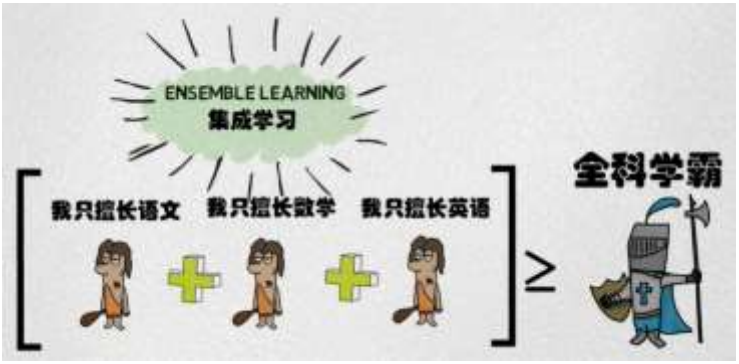
RandomForrest

随机森林原理

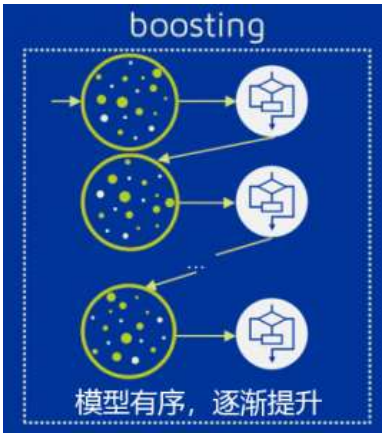
随机森林是集成学习（ensemble learning）中 bagging 的算法

所谓集成学习，可以用这张图来解释，集成学习里面有很多的 weak learner，这些 weak learner 只在某一方面特别强，预测特别准，当他们加起来，合起来的时候就会比单个强



- 集成学习（ensemble learning）是时下非常流行的机器学习算法，它本身不是一个单独的机器学习算法，而是通过在数据上构建多个模型，集成所有模型的建模结果。基本上所有的机器学习领域都可以看到集成学习的身影，在现实中集成学习也有相当大的作用，它可以用来做市场营销模拟的建模，统计客户来源，保留和流失，也可用来预测疾病的风险和病患者的易感性。在现在的各种算法竞赛中，随机森林，梯度提升树（GBDT），Xgboost 等集成算法的身影也随处可见，可见其效果之好，应用之广。
- 集成算法会考虑多个评估器的建模结果，汇总之后得到一个综合的结果，以此来获取比单个模型更好的回归或分类表现。
- 多个模型集成成为的模型叫做集成评估器（ensemble estimator），组成集成评估器的每个模型都叫做基评估器（base estimator）。比如说，随机森林，那么决策树就是 base estimator
- 三类集成算法
    - Bagging

- Boosting



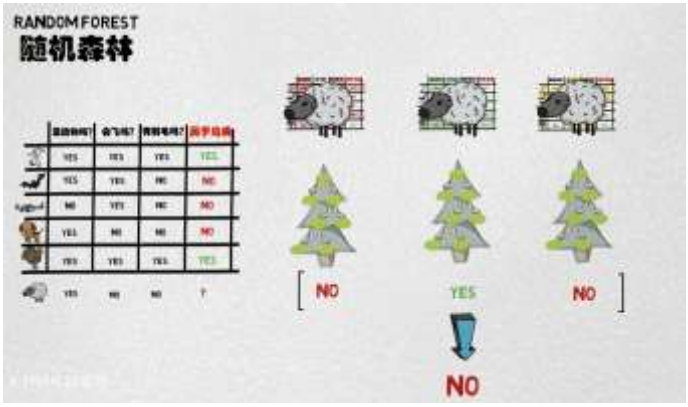
- Stacking

- Bagging VS Boosting
  - 对比

	装袋法 Bagging	提升法 Boosting
评估器	相互独立，同时运行	相互关联，按顺序依次构建，后建的模型会在先建模型预测失败的样本上有更多的权重
抽样数集	有放回抽样	有放回抽样，但会确认数据的权重，每次抽样都会给容易预测失败的样本更多的权重
决定集成的结果	平均或少数服从多数原则	加权平均，在训练集上表现更好的模型会有更大的权重
目标	降低方差，提高模型整体的稳定性	降低偏差，提高模型整体的精确度
单个评估器存在过拟合问题的时候	能够一定程度上解决过拟合问题	可能会加剧过拟合问题
单个评估器的效力比较弱的时候	不是非常有帮助	很可能会提升模型表现
代表算法	随机森林	梯度提升树，Adaboost

我们再来看看随机森林

- 随机森林里面包含着很多的决策树，每棵决策树都是 weak learner。众多的决策树就形成了森林
- 所谓随机，就是随机从数据集中采样，以训练模型中的每棵决策树
- 那么最终结果是怎么得来的呢？我们看到有很多决策树，每棵决策树都进行预测，然后统计所有结果，多的一方取胜，最终预测结果就是多的一方
- example



随机森林用了什么方法，来保证集成的效果一定好于单个分类器？

- 随机森林的本质是一种装袋集成算法（bagging），装袋集成算法是对基评估器的预测结果进行平均或用多数表决原则来决定集成评估器的结果。在红酒数据集例子中，我们建立了 25 棵树，对任何一个样本而言，平均或多数表决原则下，当且仅当有 13 棵以上的树判断错误的时候，随机森林才会判断错误。单独一棵决策树对红酒数据集的分类准确率在 0.85 上下浮动

$$e_{random\_forest} = \sum_{i=13}^{25} C_{25}^{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.000369$$

- 其中，i 是判断错误的次数，也是判错的树的数量，ε 是一棵树判断错误的概率，（1-ε）是判断正确的概率，共判对 25-i 次。采用组合，是因为 25 棵树

中，有任意 i 棵都判断错误。

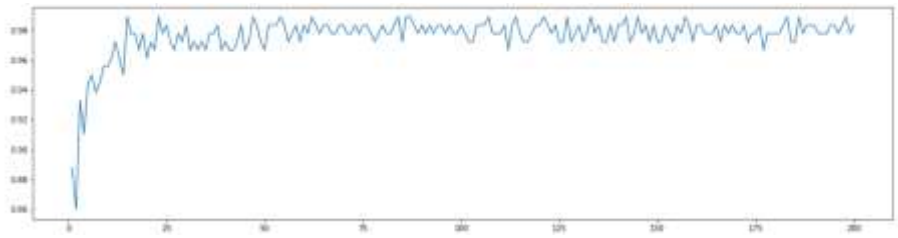
- 可见，判断错误的几率非常小，这让随机森林在红酒数据集上的表现远远好于单棵决策树。
- 那现在就有一个问题了：我们说袋装法服从多数表决原则或对基分类器结果求平均，这即是说，我们默认森林中的每棵树应该是不同的，并且会返回不同的结果。设想一下，如果随机森林里所有的树的判断结果都一致（全判断对或全判断错），那随机森林无论应用何种集成原则来求结果，都应该无法比单棵决策树取得更好的效果才对。但我们使用了一样的类 DecisionTreeClassifier，一样的参数，一样的训练集和测试集，为什么随机森林里的众多树会有不同的判断结果？问到这个问题，很多小伙伴可能就会想到了：sklearn 中的分类树 DecisionTreeClassifier 自带随机性，所以随机森林中的树天生就都是不一样的。我们在讲解分类树时曾提到，决策树从最重要的特征中随机选择出一个特征来进行分枝，因此每次生成的决策树都不一样，这个功能由参数 random_state 控制。

RandomForestClassifier

重要参数

class `sklearn.ensemble.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=None, random_state=None, verbose=0, warm_start=False, class_weight=None)*

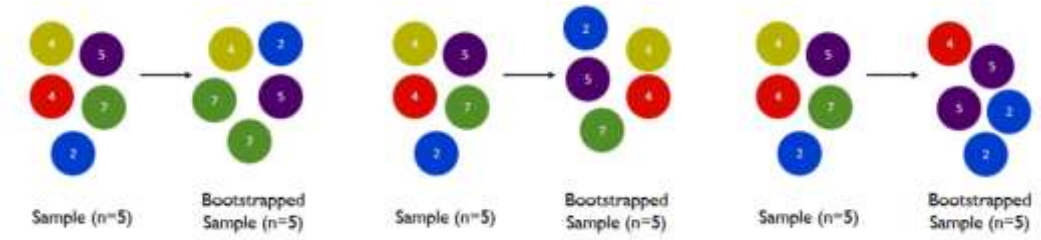
- 大部分参数像 criterion，max_depth 在决策树里有讲，可以自己回去看
- n_estimators
  - 这是森林中树木的数量，即 base estimator 的数量。这个参数对随进森林模型的精确性影响是单调的，n_estimator 越大，模型的效果往往越好。但是相应的，任何模型都有决策边界，n_estimator 达到一定的程度之后，随机森林的精确性往往不再上升，就是开始出现波动。并且，n_estimators 越大，需要的计算量和内存也越大，训练的时间也会越来越长。对于这个参数，我们是渴望在训练难度和模型效果之间取得平衡



- random_state
  - 这个 random state 跟决策树不太一样。在决策树中是保证树是一样的。这里是保证森林是一样的，但是森林里面的决策树还是各不相同的。
  - 我们可以观察到，当 random_state 固定时，随机森林中生成是一组固定的树，但每棵树依然是不一致的，这是用”随机挑选特征进行分枝“的方法得到的随机性。并且我们可以证明，当这种随机性越大的时候，袋装法

的效果一般会越来越好。用袋装法集成时，基分类器应当是相互独立的，是不相同的。但这种做法的局限性是很强的，当我们需要成千上万棵树的时候，数据不一定能够提供成千上万的特征来让我们构筑尽量多尽量不同的树。因此，除了 random_state。我们还需要其他的随机性。

- bootstrap & oob_score
  - bootstrap
    - 要让 base estimators 尽量都不一样，一种很容易理解的方法就是使用不同的训练集来进行训练，而 bagging 正是通过有放回的随机抽样技术来形成不同的训练数据，bootstrap 就是用来控制抽样技术的参数
    - 在一个含有 n 个样本的原始训练集中，我们进行随机采样，每次采样一个样本，并在抽取下一个样本之前将该样本放回原始训练集，也就是说下次采样时这个样本依然可能被采集到，这样采集 n 次，最终得到一个和原始训练集一样大的，n 个样本组成的自助集。由于是随机采样，这样每次的自助集和原始数据集不同，和其他的采样集也是不同的。这样我们就可以自由创造取之不尽用之不竭，并且互不相同的自助集，用这些自助集来训练我们的基分类器，我们的基分类器自然也就各不相同了。这张图就是个例子



    - 然而有放回抽样也会有自己的问题。由于是有放回，一些样本可能在同一个自助集中出现多次，而其他一些却可能被忽略，一般来说，自助集大约平均会包含 63%的原始数据。因为每一个样本被抽到某个自助集中的概率为：

$$1-(1-\frac{1}{n})^n$$

    - 当 n 足够大时，这个概率收敛于 1-(1/e)，约等于 0.632。因此，会有约 37%的训练数据被浪费掉，没有参与建模
  - oob_score
    - 接着上面的 bootstrap，因为会有 37%的训练数据会被浪费掉，这 37%的数据叫做 out of bag data，简称 oob。于是，我们就可以用这 37%的数据来做测试集。我们就可以不用分训练集和测试集了，直接把 oob_score 设为 true，就是用这 37%的数据来做测试集。

```
rfc = RandomForestClassifier(n_estimators=25,oob_score=True)
rfc.fit(X,y)
print(rfc.oob_score_)
```

```
0.9662921348314607
```

- 当然，这也不是绝对的，当 n 和 n_estimators 都不够大的时候，很可能就没有数据掉落在袋外，自然也就无法使用 oob 数据来测试模型了。

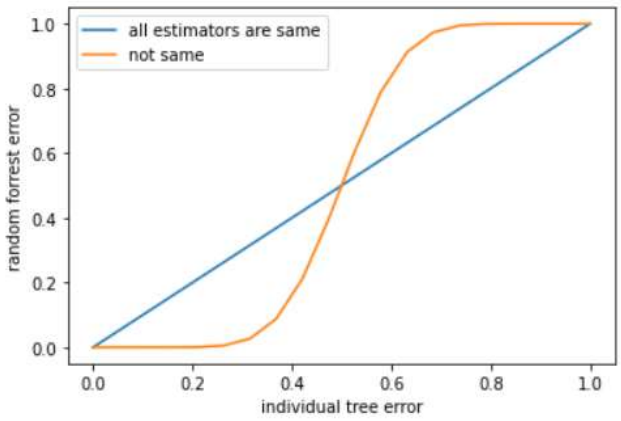
- n_jobs

重要属性和接口

- 随机森林的接口与决策树完全一致，因此依然有四个常用接口：apply, fit, predict 和 score。除此之外，还需要注意随机森林的 predict_proba 接口，这个接口返回每个测试样本对应的被分到每一类标签的概率，标签有几个分类就返回几个概率。如果是二分类问题，则 predict_proba 返回的数值大于 0.5 的，被分为 1，小于 0.5 的，被分为 0。传统的随机森林是利用袋装法中的规则，平均或少数服从多数来决定集成的结果，而 sklearn 中的随机森林是平均每个样本对应的 predict_proba 返回的概率，得到一个平均概率，从而决定测试样本的分类。

Bagging 的另一个条件

- 之前我们说过，在使用袋装法时要求基评估器要尽量独立。其实，袋装法还有另一个必要条件：基分类器的判断准确率至少要超过随机分类器，即时说，基分类器的判断准确率至少要超过 50%。之前我们已经展示过随机森林的准确率公式，基于这个公式，我们画出了基分类器的误差率 ε 和随机森林的误差率之间的图像。



- 可以从图像上看出，当基分类器的误差率小于 0.5，即准确率大于 0.5 时，集成的效果是比基分类器要好的。相反，当基分类器的误差率大于 0.5，袋装的集成算法就失效了。所以在使用随机森林之前，一定要检查，用来组成随机森林的分类树们是否都有至少 50%的预测正确率。

RandomForestRegressor

重要参数，属性与接口

class `sklearn.ensemble.RandomForestRegressor` *(n_estimators='warn', 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=None, random_state=None, verbose=0, warm_start=False)*

- 跟 decision tree 的参数意思差不多。

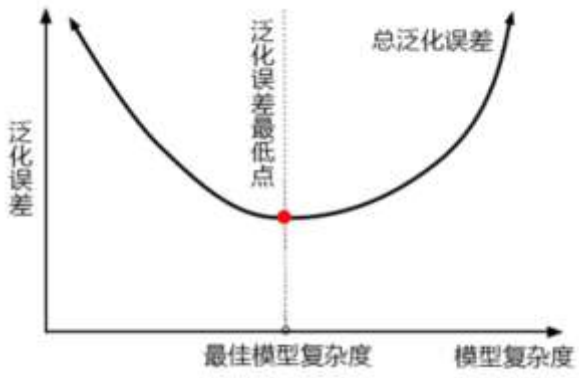
机器学习中调参的基本思想

通过画学习曲线，或者网格搜索，我们能够探索到调参边缘（代价可能是训练一次模型要跑三天三夜），但是在现实中，高手调参恐怕还是多依赖于经验，而这些经验，我们可以学习他们对模型评估指标的理解和调参的思路。

- 非常正确的调参思路和方法
- 对模型评估指标的理解
- 对数据的感觉和经验
- 用洪荒之力去不断地尝试

那我们首先来讲讲正确的调参思路。模型调参，第一步是要找准目标：我们要做什么？一般来说，这个目标是提升某个模型评估指标，比如对于随机森林来说，我们想要提升的是模型在未知数据上的准确率（由 score 或 oob_score 来衡量）。找准了这个目标，我们就需要思考：模型在未知数据上的准确率受什么因素影响？在机器学习中，我们用来衡量模型在未知数据上的准确率的指标，叫做泛化误差（Genelization error）。

- 泛化误差
  - 当模型在未知数据（测试集或者袋外数据）上表现糟糕时，我们说模型的泛化程度不够，泛化误差大，模型的效果不好。泛化误差受到模型的结构（复杂度）影响。看下面这张图，它准确地描绘了泛化误差与模型复杂度的关系，当模型太复杂，模型就会过拟合，泛化能力就不够，所以泛化误差大。当模型太简单，模型就会欠拟合，拟合能力就不够，所以误差也会大。只有当模型的复杂度刚刚好的才能够达到泛化误差最小的目标。



- 那模型的复杂度与我们的参数有什么关系呢？对树模型来说，树越茂盛，深度越深，枝叶越多，模型就越复杂。所以树模型是天生位于图的右上角的模型，随机森林是以树模型为基础，所以随机森林也是天生复杂度高的模型。随机森林的参数，都是向着一个目标去：减少模型的复杂度，把模型往图像的左边移动，防止过拟合。当然了，调参没有绝对，也有天生处于图像左边的随机森林，所以调参之前，我们要先判断，模型现在究竟处于图像的哪一边。

泛化误差的背后其实是“偏差-方差困境”

- 模型太复杂或者太简单，都会让泛化误差高，我们追求的是位于中间的平衡点
- 模型太复杂就会过拟合，模型太简单就会欠拟合
- 对树模型和树的集成模型来说，树的深度越深，枝叶越多，模型越复杂
- 树模型和树的集成模型的目标，都是减少模型复杂度，把模型往图像的左边移动
- NYU 课程讲过，自己回去复习

那具体每个参数，都如何影响我们的复杂度和模型呢？我们一直以来调参，都是在学习曲线上轮流找最优值，盼望能够将准确率修正到一个比较高的水平。然而我们现在了解了随机森林的调参方向：降低复杂度，我们就可以将那些对复杂度影响巨大的参数挑选出来，研究他们的单调性，然后专注调整那些能最大限度让复杂度降低的参数。对于那些不单调的参数，或者反而会让复杂度升高的参数，我们就视情况使用，大多时候甚至可以退避。基于经验，我对各个参数对模型的影响程度做了一个排序。在我们调参的时候，大家可以参考这个顺序。

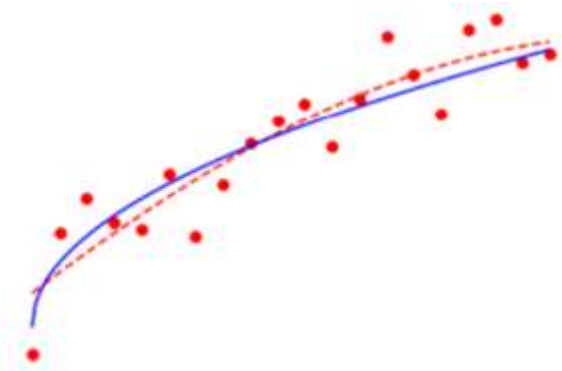
参数	对模型在未知数据上的评估性能的影响	影响程度
n_estimators	提升至平稳，n_estimators↑，不影响单个模型的复杂度	☆☆☆☆
max_depth	有增有减，默认最大深度，即最高复杂度，向复杂度降低的方向调参 max_depth↓，模型更简单，且向图像的左边移动	☆☆☆
min_samples_leaf	有增有减，默认最小限制1，即最高复杂度，向复杂度降低的方向调参 min_samples_leaf↑，模型更简单，且向图像的左边移动	☆☆
min_samples_split	有增有减，默认最小限制2，即最高复杂度，向复杂度降低的方向调参 min_samples_split↑，模型更简单，且向图像的左边移动	☆☆
max_features	有增有减，默认auto，是特征总数的开平方，位于中间复杂度，既可以向复杂度升高的方向，也可以向复杂度降低的方向调参 max_features↓，模型更简单，图像左移 max_features↑，模型更复杂，图像右移 *max_features是唯一的，既能够让模型更简单，也能够让模型更复杂的参数，所以在调整这个参数的时候，需要考虑我们调参的方向*	☆
criterion	有增有减，一般使用gini	看具体情况

偏差 VS 方差

- 一个集成模型(f)在未知数据集(D)上的泛化误差 E(f;D)，由方差(var)，偏差(bais)和噪声(ε)共同决定。

$$E(f;D) = bias^2(x) + var(x) + \varepsilon^2$$

- 关键概念
  - 观察下面的图像，每个点就是集成算法中的一个基评估器产生的预测值。红色虚线代表着这些预测值的均值，而蓝色的线代表着数据本来的面貌。



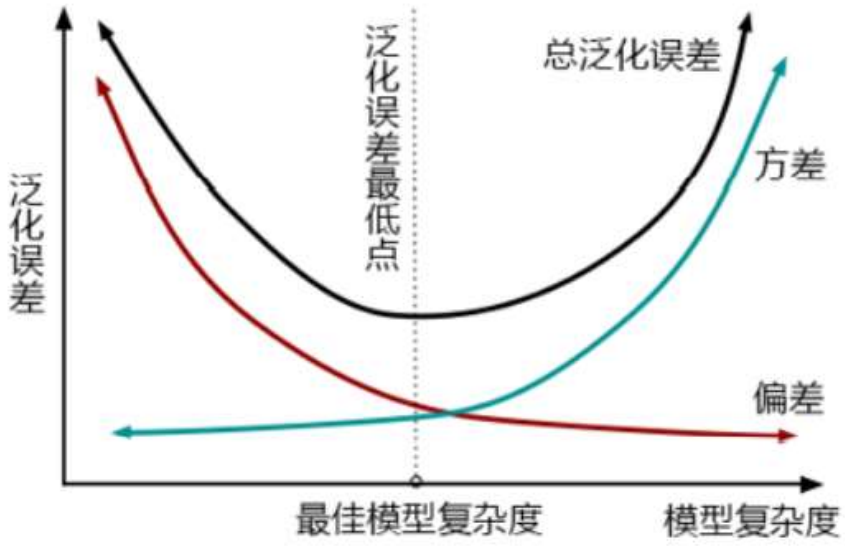
  - 偏差
    - 模型的预测值与真实值之间的差异，即每一个红点到蓝线的距离。在集成算法中，每个基评估器都会有自己的偏差，集成评估器的偏差是所有基评估器偏差的均值。模型越精确，偏差越低。
  - 方差

- 反映的是模型每一次输出结果与模型预测值的平均水平之间的误差，即每一个红点到红色虚线的距离，衡量模型的稳定性。模型越稳定，方差越低。

- 其中偏差衡量模型是否预测得准确，偏差越小，模型越“准”；而方差衡量模型每次预测的结果是否接近，即是说方差越小，模型越“稳”；噪声是机器学习无法干涉的部分，为了让世界美好一点，我们就不去研究了。一个好的模型，要对大多数未知数据都预测得”准“又”稳“。即是说，当偏差和方差都很低的时候，模型的泛化误差就小，在未知数据上的准确率就高。

泛化误差与模型复杂度

- 关系图



- 从图上可以看出，模型复杂度大的时候，方差高，偏差低。偏差低，就是要求模型要预测得“准”。模型就会更努力去学习更多信息，会具体于训练数据，这会导致，模型在一部分数据上表现很好，在另一部分数据上表现却很糟糕。模型泛化性差，在不同数据上表现不稳定，所以方差就大。而要尽量学习训练集，模型的建立必然更多细节，复杂程度必然上升。所以，复杂度高，方差高，总泛化误差高。
- 相对的，复杂度低的时候，方差低，偏差高。方差低，要求模型预测得“稳”，泛化性更强，那对于模型来说，它就不需要对数据进行一个太深的学习，只需要建立一个比较简单，判定比较宽泛的模型就可以了。结果就是，模型无法在某一类或者某一组数据上达成很高的准确度，所以偏差就会大。所以，复杂度低，偏差高，总泛化误差高。

- Summary
  - 总结

	偏差大	偏差小
方差大	模型不适合这个数据 换模型	过拟合 模型很复杂 对某些数据集预测很准确 对某些数据集预测很糟糕
方差小	欠拟合 模型相对简单 预测很稳定 但对所有的数据预测都不太准确	泛化误差小，我们的目标

- 我们调参的目标是，达到方差和偏差的完美平衡！虽然方差和偏差不能同时达到最小值，但他们组成的泛化误差却可以有一个最低点，而我们就是要寻找这个最低点。对复杂度大的模型，要降低方差，对相对简单的模型，要降低偏差。随机森林的基评估器都拥有较低的偏差和较高的方差，因为决策树本身是预测比较”准“，比较容易过拟合的模型，装袋法本身也要求基分类器的准确率必须要有 50%以上。所以以随机森林为代表的装袋法的训练过程旨在降低方差，即降低模型复杂度，所以随机森林参数的默认设定都是假设模型本身在泛化误差最低点的右边。