使用集成学习提升机器学习算法性能

> 这篇文章是对 PythonWeekly 推荐的一篇讲集成模型的文章的翻译，原文为 Ensemble Learning to Improve Machine Learning Results，由 Vadim Smolyakov 于 2017 年 8 月 22 日发表在 Medium 上，Vadim Smolyakov 是一名 MIT 的研究生，对数据科学和机器学习充满热情。

集成学习（Ensemble Learning）通过联合几个模型来帮助提高机器学习结果。与单一模型相比，这种方法可以很好地提升模型的预测性能。这也是为什么集成模型在很多著名机器学习比赛中被优先使用的原因，例如 Netflix 比赛，KDD 2009 和 Kaggle。

集成方法是一种将几种机器学习技术组合成一个预测模型的元算法（meta-algorithm），以减小方差（bagging），偏差（boosting），或者改进预测（stacking）。

集成方法可以分为两类：

- 序列集成方法（sequential ensemble methods），基学习器（base learner）顺序生成。序列方法的基本动机是**利用基学习器之间的依赖关系**。算法可以通过提高被分错样本的权重来提高性能。
- 并行集成方法（parallel ensemble methods），基学习器并行生成。并行方法的基本动机是**利用基学习器之间的独立性**，因为可以通过平均来显著降低误差。

大多数集成方法使用一个基学习算法来产生多个同质基学习器（homogeneous base learners），即相同类型的学习器，这就是同质集成（homogeneous ensembles）。

也有一些方法使用的是异质学习器（heterogeneous learner），即不同类型的学习器，这就是异质集成（heterogeneous ensembles）。为了使集成方法能够比任何构成它的单独的方法更准确，基学习器必须尽可能的准确和多样。

Bagging

Bagging 表示的是 **b**ootstrap **agg**regat**ing**。降低一个估计的方差的一个方法就是平均多个估计。例如，我们可以在一个数据集的不同子集上（有放回的随机选取）训练 M 个不同的树然后计算结果的平均值：

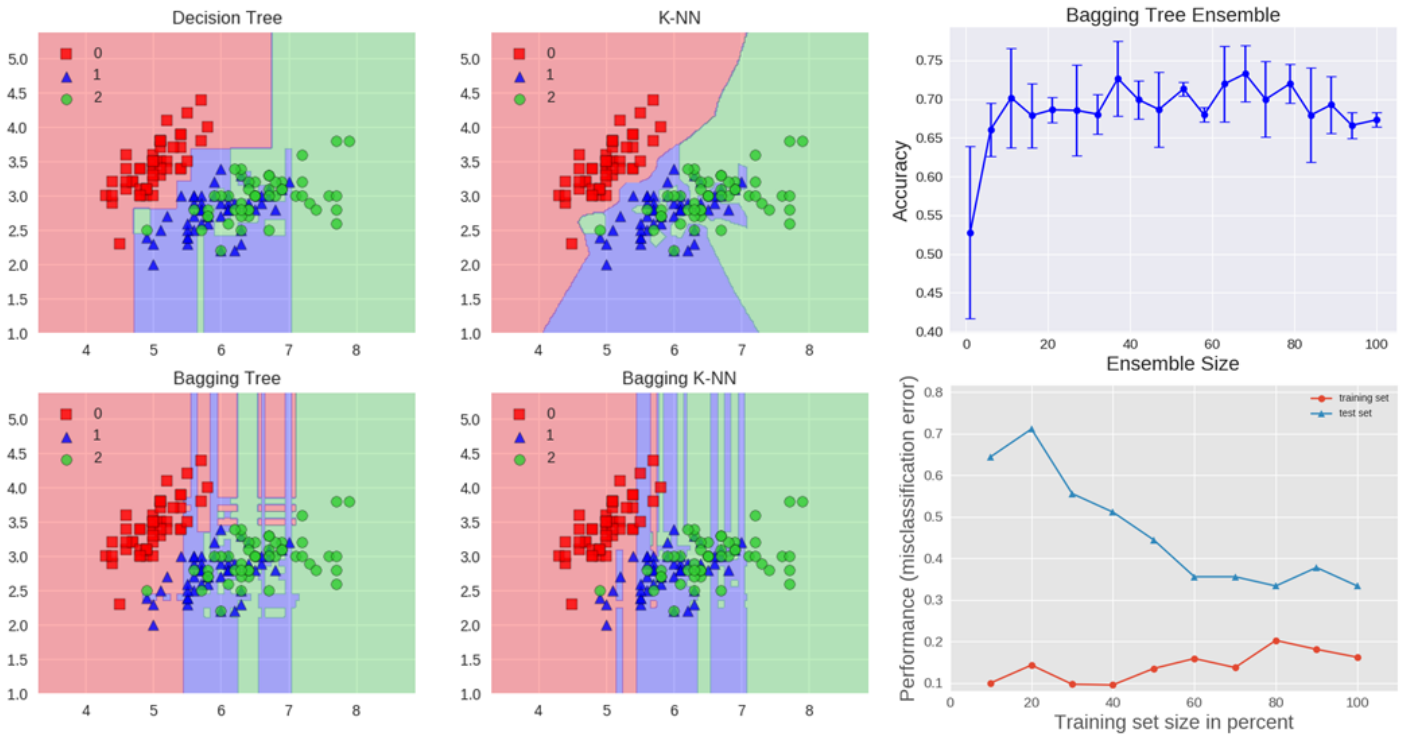
$$f(x) = \frac{1}{M}\Sigma_{m=1}^{M} f_m(x)$$

bagging 使用自助抽样法（bootstrapping）来为每个基学习器获得一个数据集的子集。对于如何聚合多个基学习器的结果，bagging 在分类任务中使用投票，而在回归任务重使用平均。

我们可以通过在 Iris 数据集上执行分类任务来学习 bagging。我们选择两种基学习器：决策树（decision tree）和 kNN 分类器。下图显示了基学习器在 Iris 上学习到的决策边界和他们 bagging 集

成之后学习到的决策边界。

- 决策树准确率：0.63（+/- 0.02）
- kNN 准确率：0.70（+/- 0.02）
- bagging 树准确率：0.64（+/- 0.01）
- bagging kNN准确率：0.59（+/- 0.07）



Bagging

决策树的边界与轴平行，而 $k = 1$ 时的 kNN 算法与数据点紧密贴合。该集成方法使用了 10 个基学习器，训练子集由原训练数据和特征的 80% 构成。

决策树集成相对于 kNN 集成达到了较高的准确率。kNN 对训练样本的扰动不敏感，因此也被称为稳定学习器（stable learner）。

> 稳定学习器的集成不太有利，因为这样的集成并不会提升泛化性能。

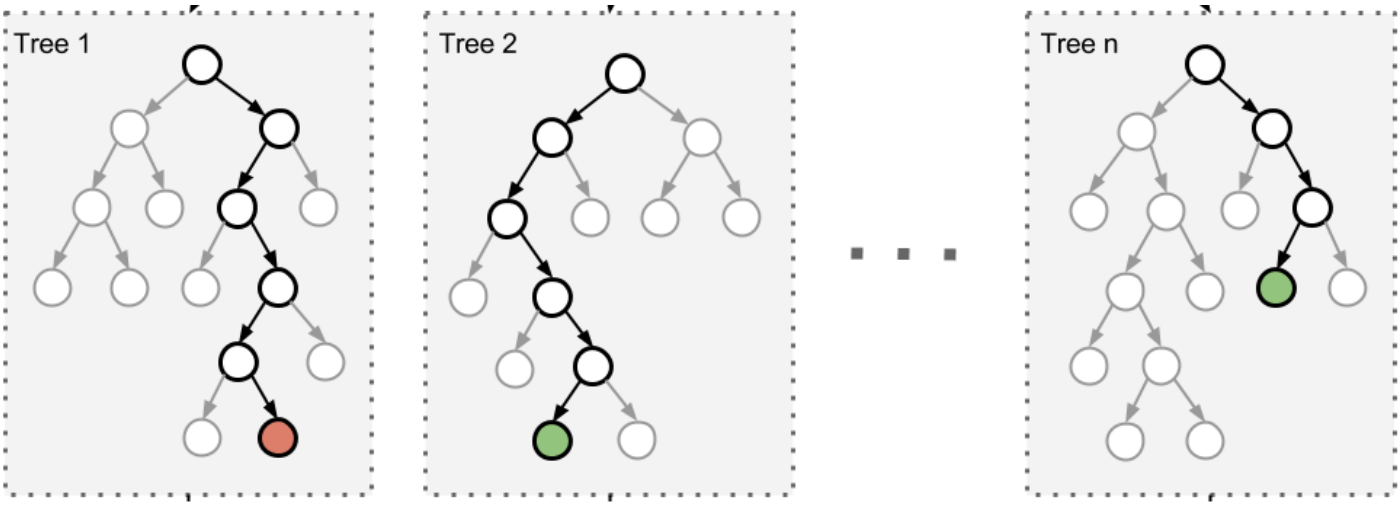
图一也显示了集成大小是如何提高测试准确率的。基于交叉验证的结果，我们可以看到在大约 10 个基学习器前准确率一直在增加，随后趋于平缓，在 0.7 左右上下波动。因此，再增加超过 10 个基学习器不仅没有得到准确率的提升，反而增加了计算复杂度。

我们也可以看到 bagging 树集成的学习曲线。注意到训练数据的平均误差为 0.3 和 测试数据的 U 型误差曲线。训练和测试误差差距最小时发生在 Training set size in percent 为 80% 左右。

> 一种常用的集成算法是随机森林。

在随机森林算法中，每个树都是基于从原训练数据集中有放回抽样（即自助抽样法）得到的子集训练的。另外，也对特征进行自助抽样，而不是使用全部特征。

最终随机森林的偏差可能会轻微增大，但是由于平均了几个不相关的树的结果，降低了方差，导致最终模型的整体性能更好。（*译者注：个人觉得类似于时间序列分析中的移动平均*）



在极限随机树（extremely randomized tree）算法中，随机性更近了一步：分裂阈值是随机选取的。与寻找最具有判别性的阈值不同，极限随机树为每个候选特征选取一个阈值，并将这些阈值的最佳值作为最终的分割阈值。这通常会降低方差，偏差会稍微增大。

Boosting

Boosting 指的是能够将弱学习器转化为强学习器的一系列算法。Boosting 的主要原理是给一系列的弱学习器赋予权重，这些弱学习器各自的性能仅仅比随机猜测高一些，例如小的决策树。先前被分错的样本会被给予更多权重。

在进行分类任务时，使用一个加权投票来决定最终的预测，而在回归任务时使用加权和。Boosting 和其他方法例如 bagging 的主要区别是基学习器使用加权版本的数据集来顺序生成。

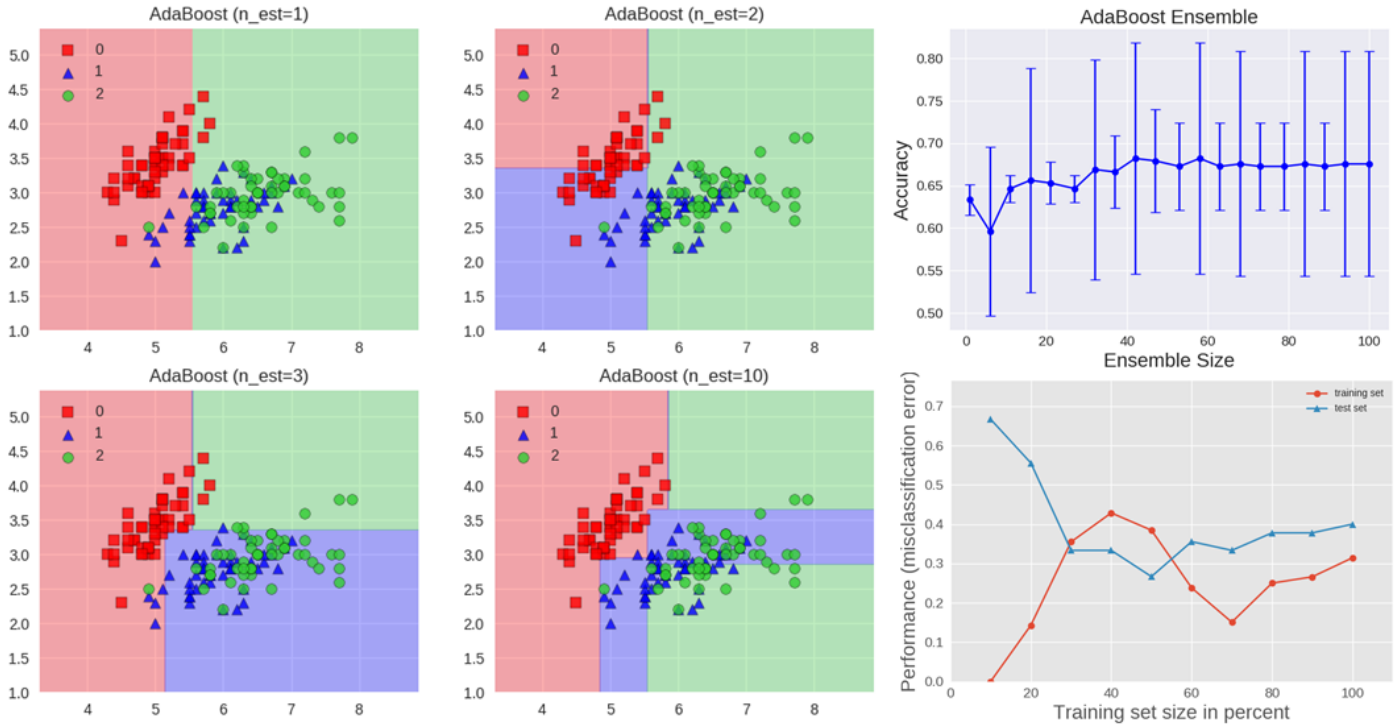
下面的算法是 AdaBoost（**ada**ptive **boost**ing），这是目前使用最广泛的 boosting 算法：

> 1. 初始化样本权重 w_n 为 $1/N$
> 2. for $m = 1$ to M
> 3. 通过最小化加权误差函数 J_m 训练一个分类器 $y_m(x)$，此处 $J_m = \Sigma_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
> 4. 计算 $\epsilon_m = \dfrac{\Sigma_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]}{\Sigma_{n=1}^{N} w_n^{(m)}}$
> 5. 计算 $\alpha_m = \log(\dfrac{1-\epsilon_m}{\epsilon_m})$
> 6. 更新样本权重：$w_n^{m+1} = w_n^m e^{\alpha_m 1[y_m(x_n) \neq t_n]}$
> 7. end for
> 8. 使用最终的模型进行预测：$Y_M(x) = \text{sign}\left(\Sigma_{m=1} M \alpha_m y_m(x)\right)$

其中 N 为样本数，M 为集成大小，即基学习器的数量。我们可以看到第一个基分类器 $y_1(x)$ 训练时所使用的权重均为 $1/N$。在接下来的 boosting 循环中，那些被分错的样本的权重会被增大，而那些

被分对的样本的权重会减小。

ϵ 表示及基分类器的加权错误率。因此，权重系数 α 会给予准确率更高的分类器更大的权重。



AdaBoost

如上图所示，每一个基学习器由一个深度为 1 的决策树组成，因此当 n_est=1 时模型只是基于一个特征阈值将样本空间分为两部分。上图也显示了集成大小是如何提高测试准确率的以及训练样本和测试样本的学习曲线。

梯度树提升（**Gradient Tree Boosting**）是 boosting 在任意可微分损失函数的一种推广，既可用于回归也可用于分类，同样是顺序生成基学习器。

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

每一步基于当前模型 $F_{m-1}(x)$ 来计算损失 L，并最小化 L 来训练决策树 $h_m(x)$：

$$F_m(x) = F_{m-1}(x) + \text{argmin}_h \Sigma_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$$

使用该算法进行回归和分类任务时不同在于损失函数。

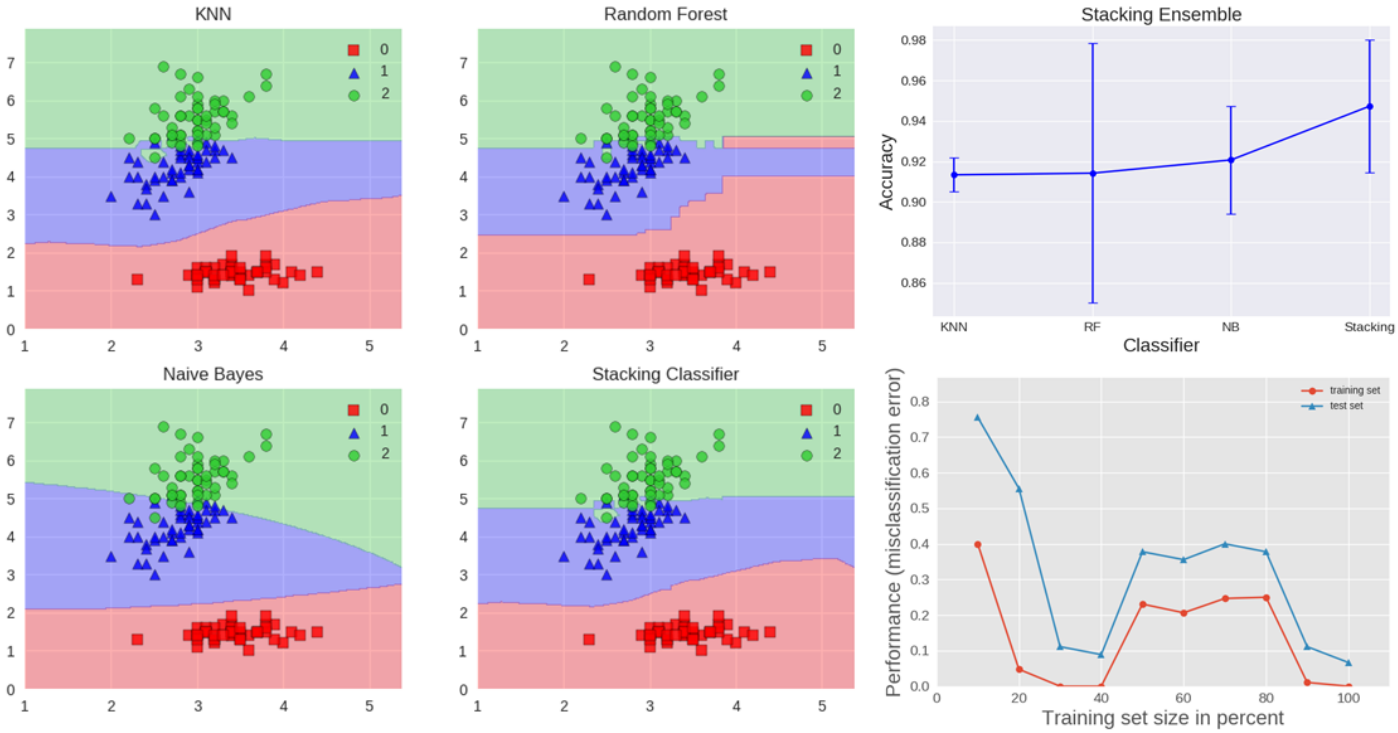
Stacking

Stacking 是一种通过元分类器（meta-classifier）或者元回归器（meta-regressor）来综合几个分类模型和回归模型的集成学习技术。基模型（base level model）基于整个数据集进行训练，然后元模型（meta-model）将基模型的输出作为特征来进行训练。（*译者注：个人觉得和 stacking-autoencoder 一个思想*）

基模型通常由不同的学习算法组成，因此 stacking 集成通常是异构的。下面的算法总结了 stacking ：

> 1. 输入：训练样本 $D = \{x_i, y_i\}_{i=1}^{m}$
> 2. 输出：集成分类器 H
> 3. *Step 1：学习基分类器*
> 4. for $t = 1$ to T
> 5. 基于 D 训练 h_t
> 6. end for
> 7. *Step 2：构建新数据*
> 8. for $i = 1$ to m
> 9. $D_h = \{x_i^{'}, y_i\}$，其中 $x_i^{'} = \{h_1(x_i), \ldots, h_T(x_i)\}$
> 10. end for
> 11. *Step 3：学习元分类器*
> 12. 基于 D_h 训练 H
> 13. 返回 H

- KNN 准确率：0.91（+/- 0.01）
- 随机森林 准确率：0.91（+/- 0.06）
- 朴素贝叶斯 准确率：0.92（+/- 0.03）
- Stacking 准确率：0.95（+/- 0.03）

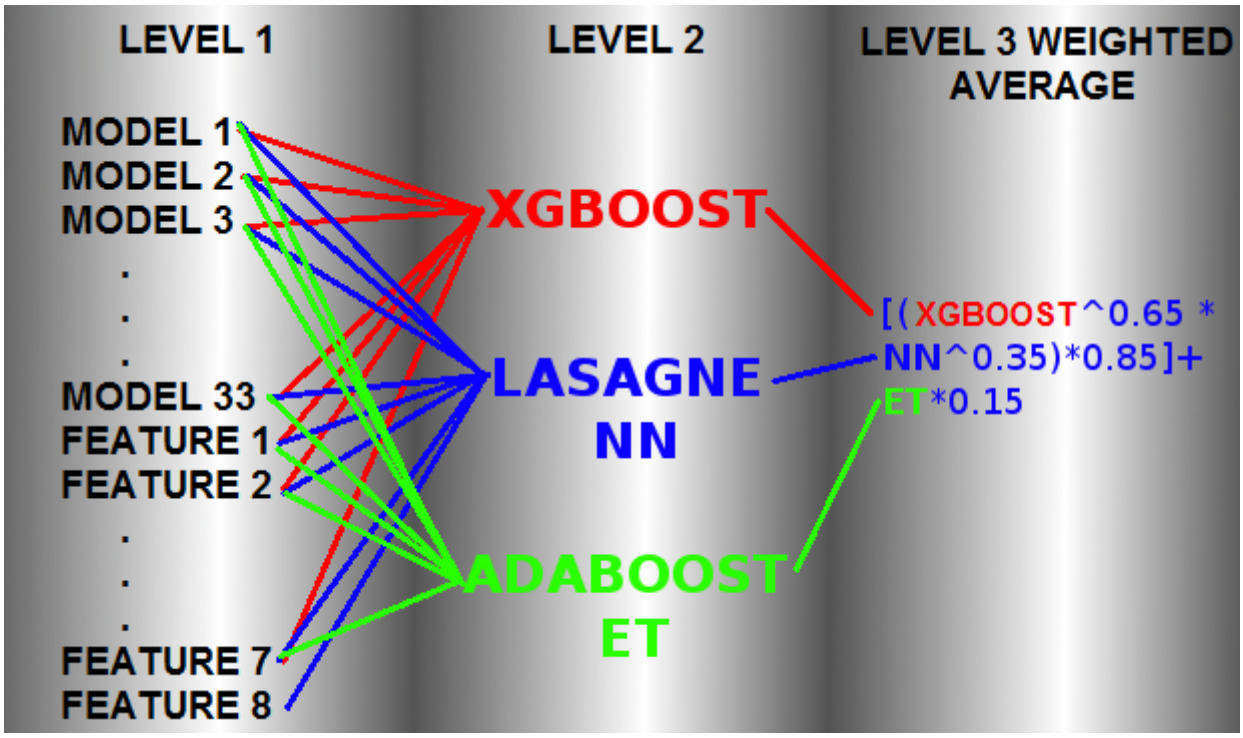


Stacking

如上图所示，该 stacking 集成由 KNN、随机森林和朴素贝叶斯这几个基分类器构成，其预测输出再由逻辑斯底回归综合构成一个元分类器。我们可以看到 stacking 实现了决策边界的混合。上图还显示了 stacking 的准确率要高于单独的分类器，且基于学习曲线，模型没有过拟合的迹象。

Stacking 在 Kaggle 比赛中经常被用到。例如，在 Otto Group Product Classification 比赛中第一名就使用了超过 30 个模型，这些模型的输出又被作为特征来训练得到 3 个元分类器：XGBoost、神经网络和 AdaBoost。可以在 这里 查看细节。

译者注：为了直观，我把获胜者所使用的模型结构图放在下面。



Code

生成本文所用图片的代码在这个 jupyter notebook 上。

Conclusion

除了本文研究的方法外，集成学习也通常用于深度学习来训练多样化和高准确率的分类器。可以通过不同的网络结构、超参数设置和训练技巧来达到多样性。

集成方法在比赛数据集中获得创纪录的性能方面非常成功，也是 Kaggle 数据科学比赛中优胜者常用的算法。

Recommended reading

- Zhi-Hua Zhou, “Ensemble Methods: Foundations and Algorithms”, CRC Press, 2012（ *译者注：原文这里没有链接，我自己找了找资源加上了资源地址，如若下载不下来请在 这里 下载* ）
- L. Kuncheva, “Combining Pattern Classifiers: Methods and Algorithms”, Wiley, 2004（ *译者注：原文这里没有链接，我自己找了找资源加上了资源地址，如若下载不下来请在 这里 下载* ）
- Kaggle Ensembling Guide
- Scikit Learn Ensemble Guide

- S. Rachka, MLxtend library
- Kaggle Winning Ensemble