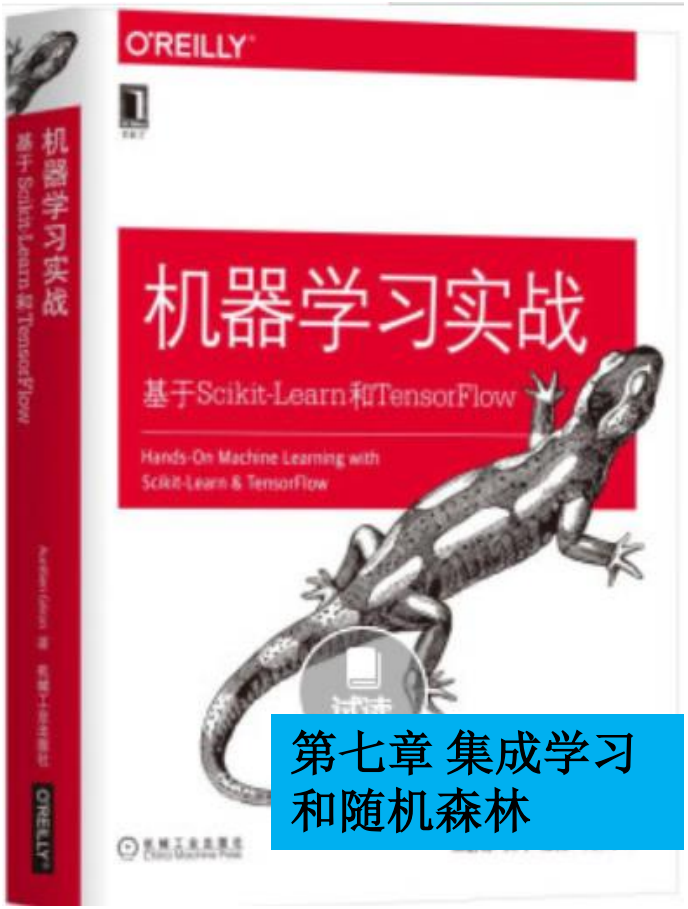
# 模型融合/集成学习

# 参考资料




第七章 集成学习和随机森林

教材代码： https://github.com/ageron/handson-ml

Scikit-Learn: https://sklearn.apachecn.org/docs/0.21.3/12.html

# 本章内容

- 什么是模型融合/集成学习？
  - 模型融合的目标；聚合策略；模型融合方法的分类
- 模型融合/集成学习肯定有效吗？
- 模型融合的常用方案有哪些？
  - Voting
  - Stacking

  基预测器可以为不同种类
  - Bagging
    - 随机森林
  - Boosting
    - AdaBoost
    - Gradient Boosting: GBRT, XGBoost, LightGBM

  基预测器都是同一类
- Kaggle比赛案例展示
  - 帮助快速熟悉支持向量机、决策树、随机森林的代码实现
  - 理解模型融合所带来的效果

## sklearn.ensemble : Ensemble Methods

The `sklearn.ensemble` module includes ensemble-based methods for classification, regression and anomaly detection.

**User guide:** See the Ensemble methods section for further details.

| | |
|---|---|
| `ensemble.AdaBoostClassifier` ([...]) | An AdaBoost classifier. |
| `ensemble.AdaBoostRegressor` ([base_estimator, ...]) | An AdaBoost regressor. |
| `ensemble.BaggingClassifier` ([base_estimator, ...]) | A Bagging classifier. |
| `ensemble.BaggingRegressor` ([base_estimator, ...]) | A Bagging regressor. |
| `ensemble.ExtraTreesClassifier` ([...]) | An extra-trees classifier. |
| `ensemble.ExtraTreesRegressor` ([n_estimators, ...]) | An extra-trees regressor. |
| `ensemble.GradientBoostingClassifier` ([loss, ...]) | Gradient Boosting for classification. |
| `ensemble.GradientBoostingRegressor` ([loss, ...]) | Gradient Boosting for regression. |
| `ensemble.IsolationForest` ([n_estimators, ...]) | Isolation Forest Algorithm |
| `ensemble.RandomForestClassifier` ([...]) | A random forest classifier. |
| `ensemble.RandomForestRegressor` ([...]) | A random forest regressor. |
| `ensemble.RandomTreesEmbedding` ([...]) | An ensemble of totally random trees. |
| `ensemble.VotingClassifier` (estimators[, ...]) | Soft Voting/Majority Rule classifier for unfitted estimators. |
| `ensemble.VotingRegressor` (estimators[, ...]) | Prediction voting regressor for unfitted estimators. |
| `ensemble.HistGradientBoostingRegressor` ([...]) | Histogram-based Gradient Boosting Regression Tree. |
| `ensemble.HistGradientBoostingClassifier` ([...]) | Histogram-based Gradient Boosting Classification Tree. |

https://scikit-learn.org/stable/modules/ensemble.html#ensemble

# 什么是模型融合/集成学习？

- 模型融合/集成学习：
  - 将一组弱/基预测器的预测结果进行融合/集成，以实现一个强预测器，从而获得比单个预测器更好的泛化能力/鲁棒性。
  - 示例：采用多数表决（majority vote）的投票分类器

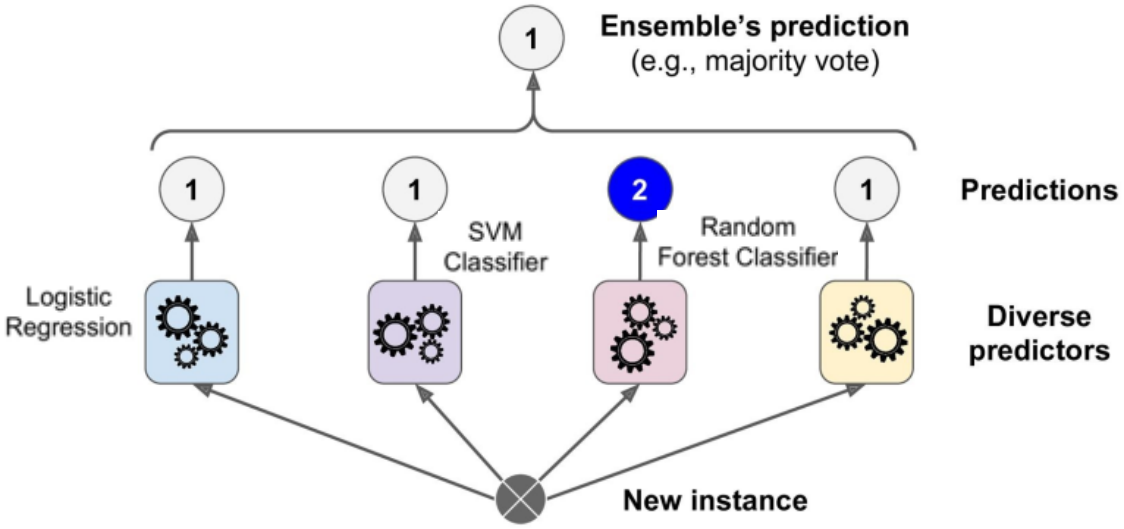



*Figure 7-2. Hard voting classifier predictions*

- 创建和训练在sk-learn中的投票分类器

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.ensemble import VotingClassifier
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.svm import SVC
>>> log_clf = LogisticRegression()
>>> rnd_clf = RandomForestClassifier()
>>> svm_clf = SVC()
>>> voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf', rnd_clf), >>> ('
svc', svm_clf)],voting='hard')
>>> voting_clf.fit(X_train, y_train)
```

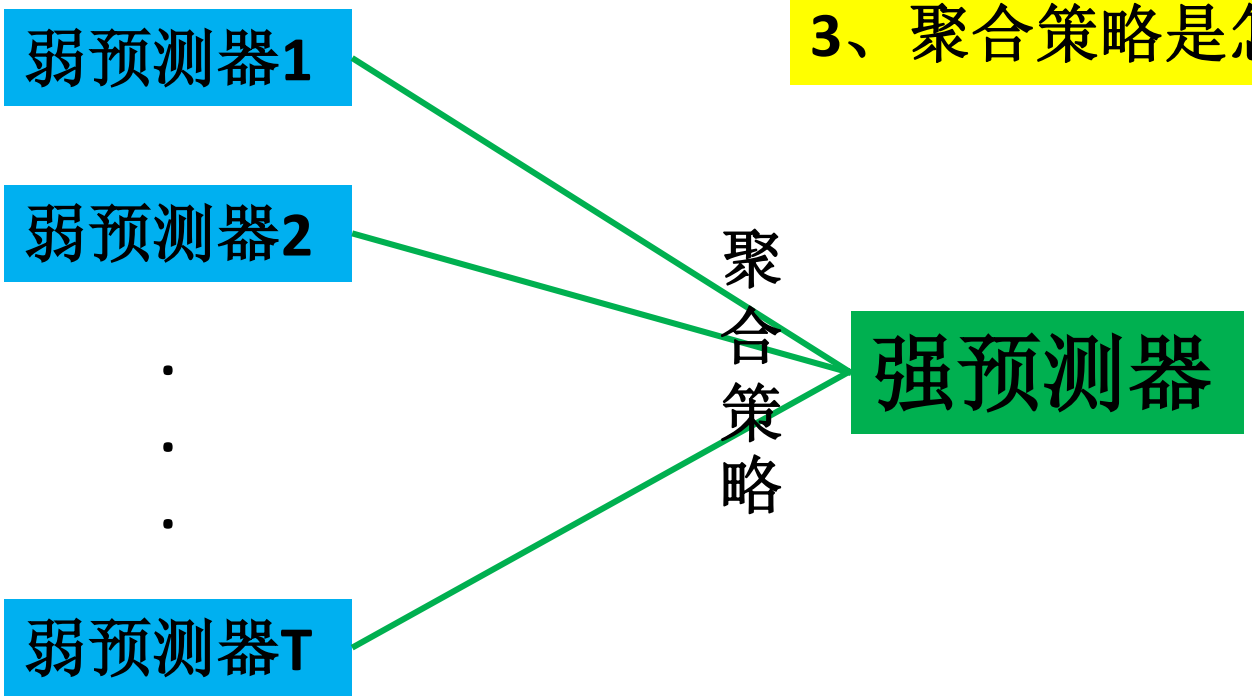
– 展示 在测试集上的准确率

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
>>>     clf.fit(X_train, y_train)
>>>     y_pred = clf.predict(X_test)
>>>     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.896
```

**结论：集成的投票分类器比其他单独的分类器表现的都要好。**

# 什么是模型融合/集成学习？

三个关键点：
1、弱预测器之间相关性？
2、如何训练各弱预测器？
3、聚合策略是怎样的？



# 模型融合的聚合策略

- 模型融合的聚合策略：对所有弱预测器的预测结果进行聚合所采用的策略。常见的策略有：
  1. 平均法：一般用于回归预测模型中。平均法包括一般的平均和加权平均融合。
     - 投票回归器(Voting Regressor)
     - Boosting系列融合模型
  2. 投票(Voting)法：一般用于分类模型。具体可分为绝对多数投票（得票超过一半），相对多数投票（得票最多），加权投票。
     - bagging模型
     - 投票分类器(Voting Classifier)
  3. 学习法：通过另一个预测器(称为混合器或元学习器)来实现聚合。常见的有Stacking和Blending两种。
     - stacking一般使用交叉验证的方式
     - Blending是建立一个Holdout集

# 模型融合的分类

- 按照弱预测器之间的依赖关系，模型融合方法可分为两类:
  - 弱预测器间不存在强依赖关系、可同时学习的并行化方法
    - 弱预测器只有一种，即所有的弱预测器都是基于同一类模型。代表是Bagging 和”随机森林”。
    - 弱预测器的种类多。比如，投票分类器，投票回归器，Stacking。
  - 弱预测器间存在强依赖关系、必须串行学习的序列化方法，代表是Boosting方法。

注意区分：强依赖 ≠ 强相关性

- 弱预测器之间存在强依赖，是指单个弱预测器的训练存在必需的先后关系；
- 弱预测器之间若存在强相关性，会影响融合后的性能

# 本章内容

- 什么是模型融合/集成学习？
  - 模型融合的目标；聚合策略；模型融合方法的分类
- 模型融合/集成学习肯定有效吗？
- 模型融合的常用方案有哪些？
  - Voting
  - Stacking
  - Bagging
    - 随机森林
  - Boosting
    - AdaBoost
    - Gradient Boosting: GBRT, XGBoost, LightGBM
- Kaggle比赛案例展示
  - 帮助快速熟悉支持向量机、决策树、随机森林的代码实现
  - 理解模型融合所带来的效果

# 模型融合/集成学习一定有效吗？

- 可以通过数学证明：随着**弱预测器数目T**的**增大**，集成后得到的强预测器的**错误率将呈指数级下降**，最终趋向于零。
  - 具体证明见周志华和李航老师的书。
  - 弱预测器数目T过大，可能导致强预测器出现过拟合
- **弱预测器之间的相关性**要尽可能的**小**。
  - 弱预测器之间尽可能相互独立时，集成得到的强预测器的性能最优。
  - 如何降低弱预测器之间的相关性呢？
    - 弱预测器的种类越多
    - 各预测器基于不同的训练子集进行训练：训练样本随机
    - 各预测器基于不同的特征子集进行训练：特征随机

# 本章内容

- 什么是模型融合/集成学习？
  - 模型融合的目标；聚合策略；模型融合方法的分类
- 模型融合/集成学习肯定有效吗？
- 模型融合的常用方案有哪些？
  - Voting
  - Stacking
  - Bagging
    - 随机森林
  - Boosting
    - AdaBoost
    - Gradient Boosting: GBRT, XGBoost, LightGBM
- Kaggle比赛案例展示
  - 帮助快速熟悉支持向量机、决策树、随机森林的代码实现
  - 理解模型融合所带来的效果

# Voting Classifier（投票分类器）

- 分类问题中的模型融合/集成学习
- 结合了多个不同种类的机器学习分类器，并且采用多数表决（majority vote）（硬投票） 或者平均预测概率（软投票）的聚合方式来预测分类标签。
- 可以用于融合/集成一组同样表现良好的模型，以便平衡它们各自的弱点。

# 采用多数表决（majority vote）的投票分类器

- 基于同一个训练集，可以训练出多个不同种类的分类器，包括逻辑回归模型，支持向量机，随机森林等等。

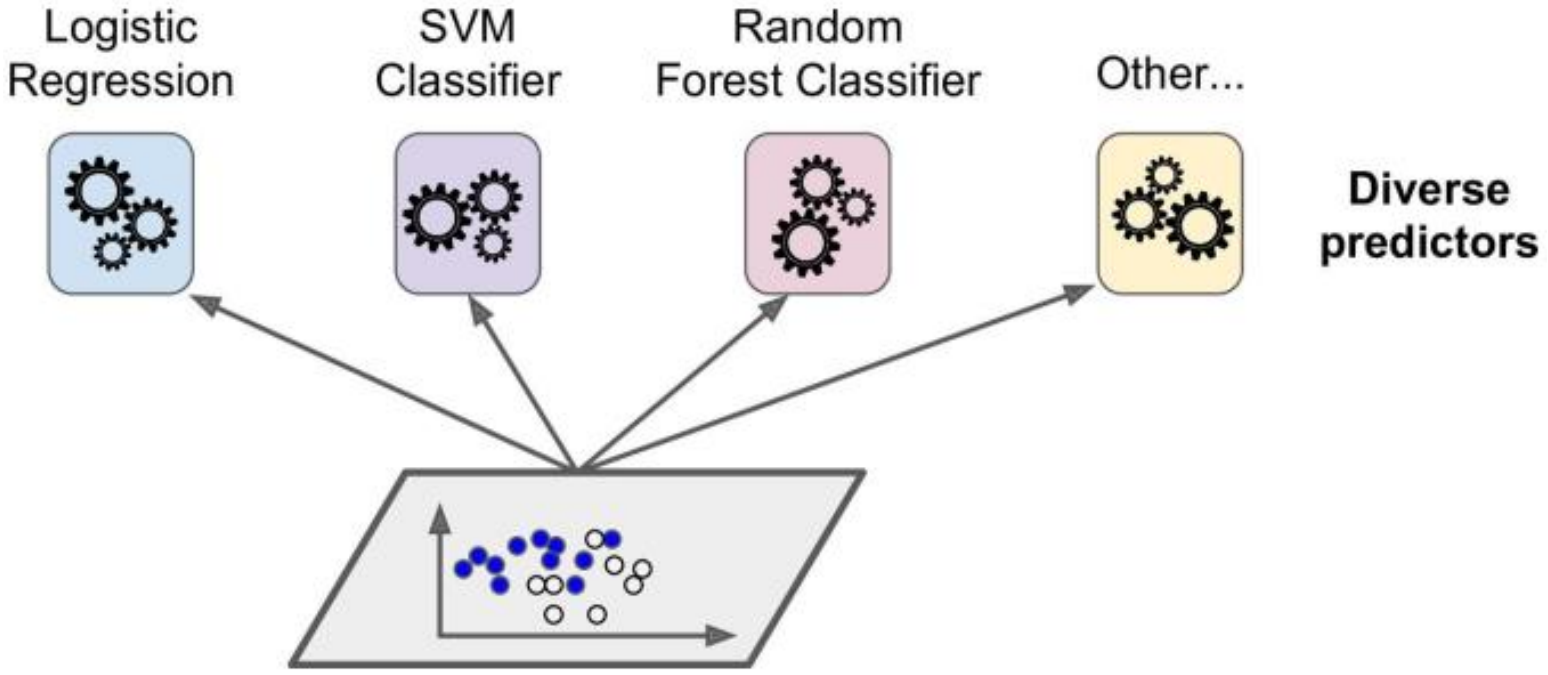



*Figure 7-1. Training diverse classifiers*

# 采用多数表决（majority vote）的投票分类器

- 聚合每个分类器的预测，然后将投票结果最多的结果作为预测的类别。
  - 多数表决(majority vote) / 硬投票(Hard Voting)，是一种聚合策略

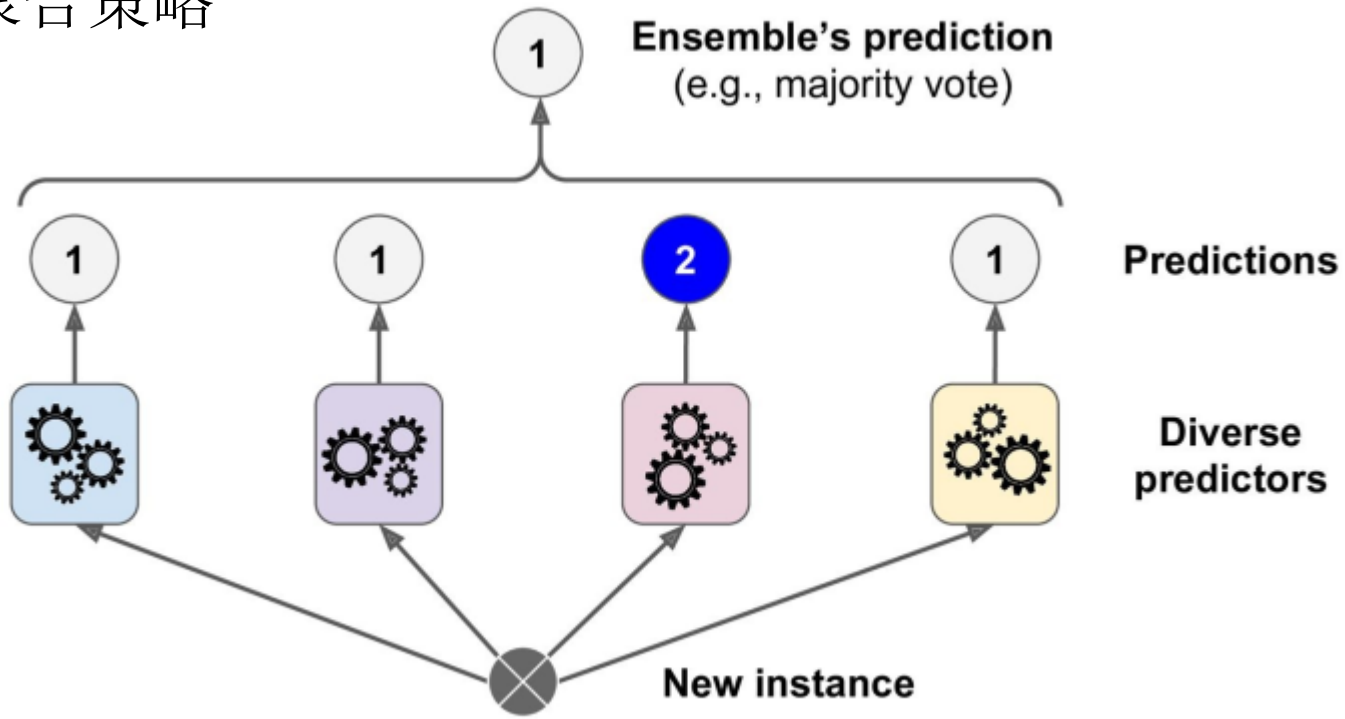



*Figure 7-2. Hard voting classifier predictions*

# 软投票（Soft Voting）

与多数投票（硬投票）相比，软投票将类别标签返回为预测概率之和的 argmax 。

具体的权重可以通过权重参数 `weights` 分配给每个分类器。当提供权重参数 `weights` 时，收集每个分类器的预测分类概率， 乘以分类器权重并取平均值。然后将具有最高平均概率的类别标签确定为最终类别标签。

为了用一个简单的例子来说明这一点，假设我们有 3 个分类器和一个 3 类分类问题，我们给所有分类器赋予相等的权重：w1 = 1,w2 = 1,w3 = 1 。

样本的加权平均概率计算如下：

| 分类器 | 类别 1 | 类别 2 | 类别 3 |
| --- | --- | --- | --- |
| 分类器 1 | w1 * 0.2 | w1 * 0.5 | w1 * 0.3 |
| 分类器 2 | w2 * 0.6 | w2 * 0.3 | w2 * 0.1 |
| 分类器 3 | w3 * 0.3 | w3 * 0.4 | w3 * 0.3 |
| 加权平均的结果 | 0.37 | 0.4 | 0.23 |

这里可以看出，预测的类标签是 2，因为它具有最大的平均概率.

# 堆叠法（Stacking）

- 通过训练一个预测器（称为blender或者meta learner）来实现聚合。
- 如何利用混合器来预测？
  - 逐个遍历每个层来预测一个新的实例

1. 底部三个弱预测器对新实例的预测结果分别为3.1，2.7和2.9；
2. 最终的预测器（叫做blender或者meta learner）将这三个预测结果作为输入，进行最终决策，输出3.0

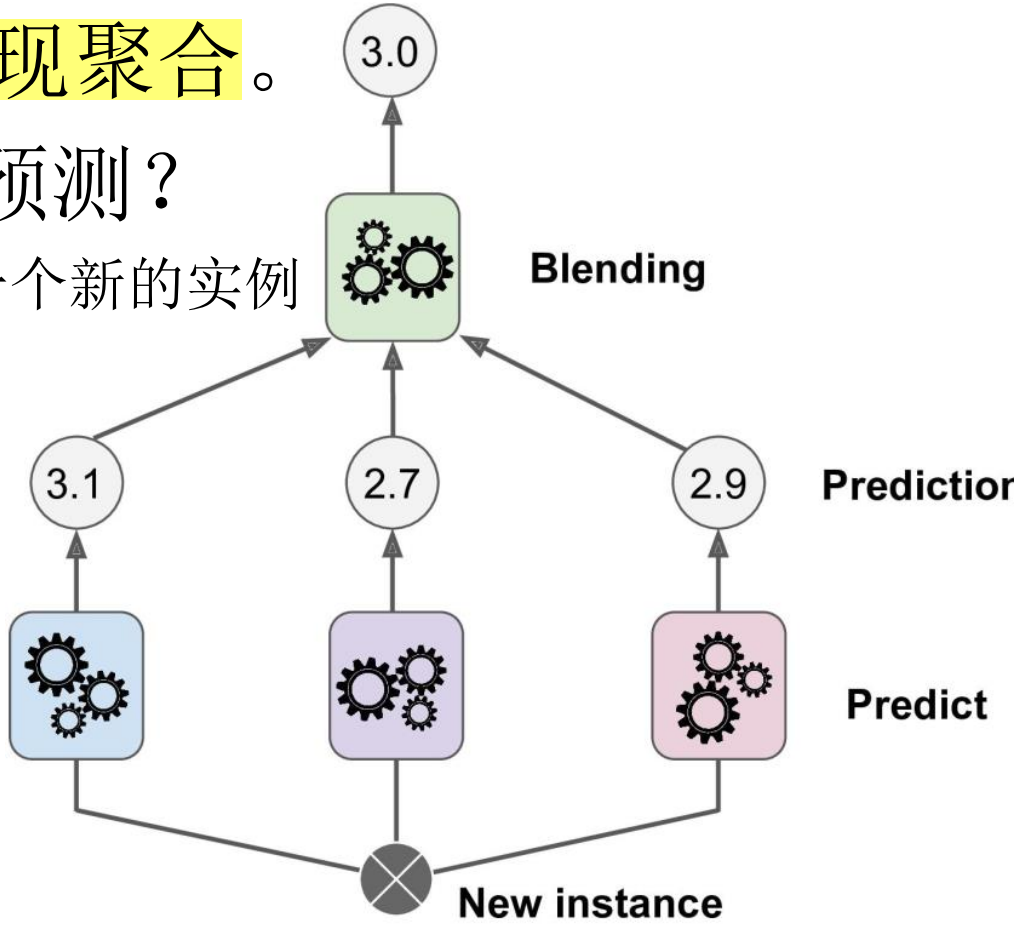



*Figure 7-12. Aggregating predictions using a blending predictor*

- 如何训练混合器？

1. 训练集被分为两个子集：
   - 第一个子集被用作训练第一层的预测器；
   - 第二个子集用于构造混合器的训练集。
2. 在第一个子集上训练第一层的预测器；

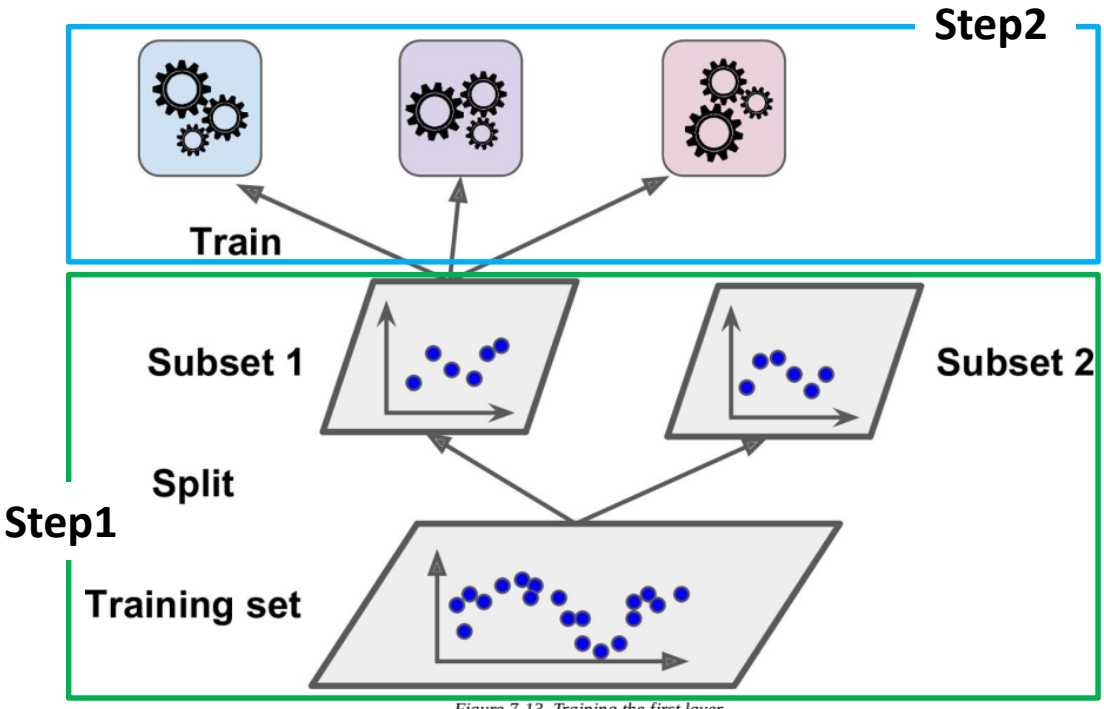



*Figure 7-13. Training the first layer*

- 如何训练混合器？

1. 训练集被分为两个子集：
   - 第一个子集被用作训练第一层的预测器；
   - 第二个子集用于构造混合器的训练集。
2. 在第一个子集上训练第一层的预测器；
3. 用第一层的预测器对第二个子集中的样本进行预测；
4. 将前一步的预测结果作为输入特征，保留第二个子集中的标签，从而创建一个新的训练集；
5. 在该新训练集上训练混合器，即根据第一层预测器的预测值来拟合标签。

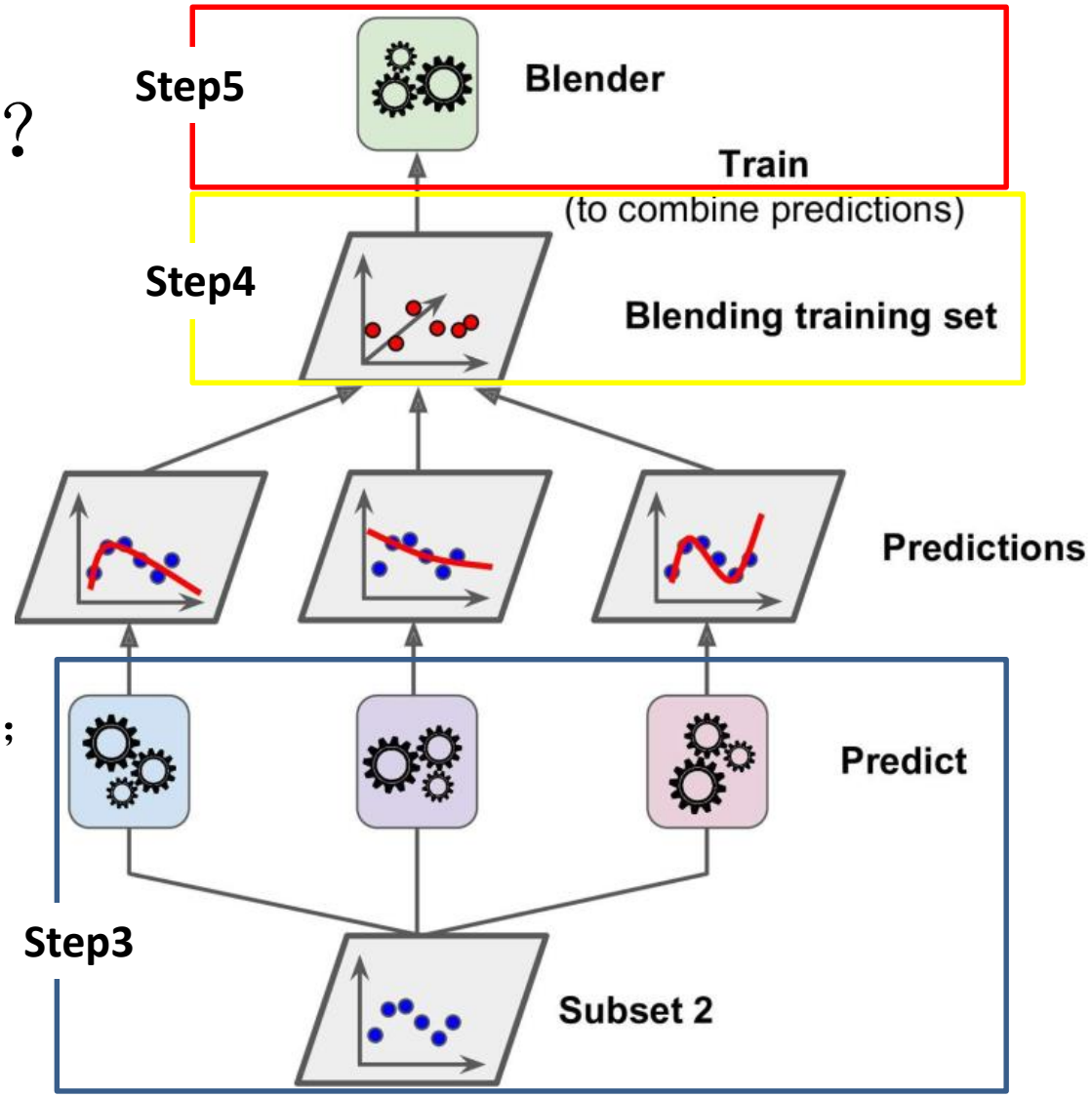



*Figure 7-14. Training the blender*

# 混合器层：包含多个混合器

- 训练多种不同的混合器

- 逐层的训练，直至最末层的混合器训练完毕。
  - 将训练集分成三个子集：
    第一个子集用来训练第一层；
    第二个子集用来创建训练第二层的训练集（使用第一层分类器的预测值作为输入特征，并保留第二个子集的标签）；
    第三个子集被用来创建训练第三层的训练集（使用第二层分类器的预测值作为输入特征，并保留第三个子集的标签）。
- 通过逐个遍历每个层来预测一个新的实例。

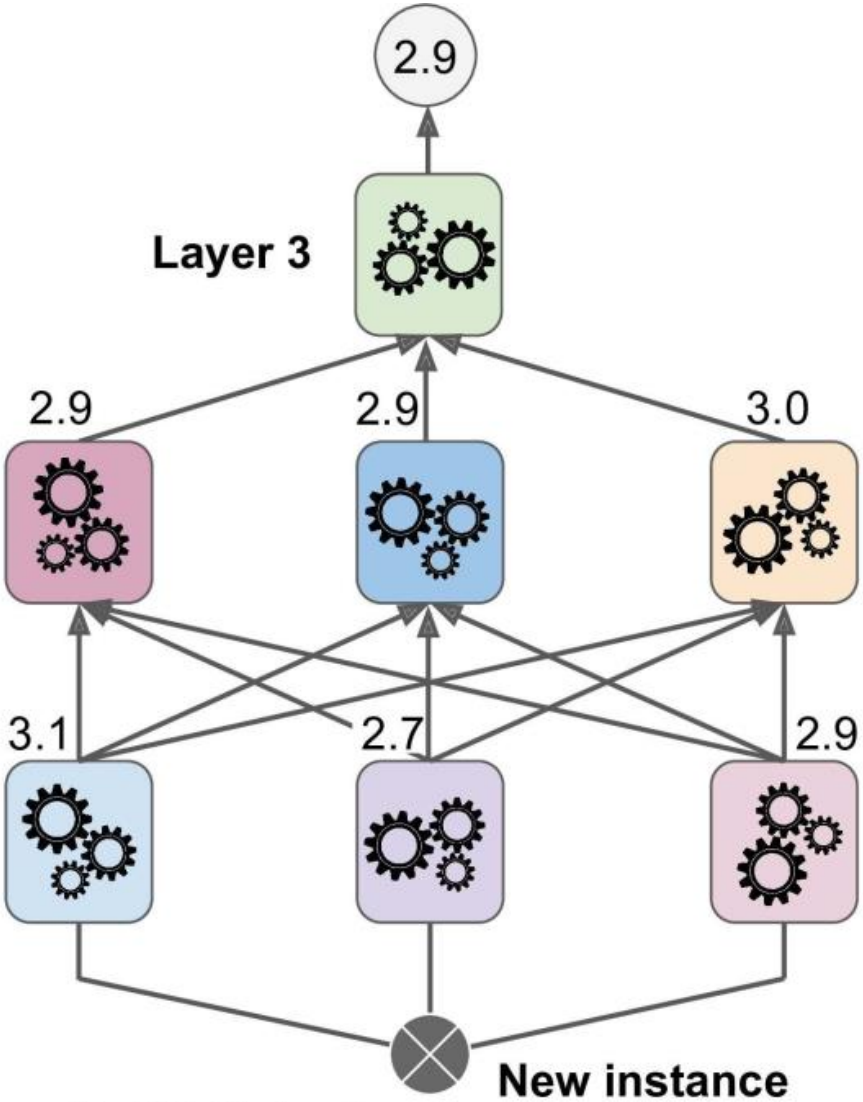


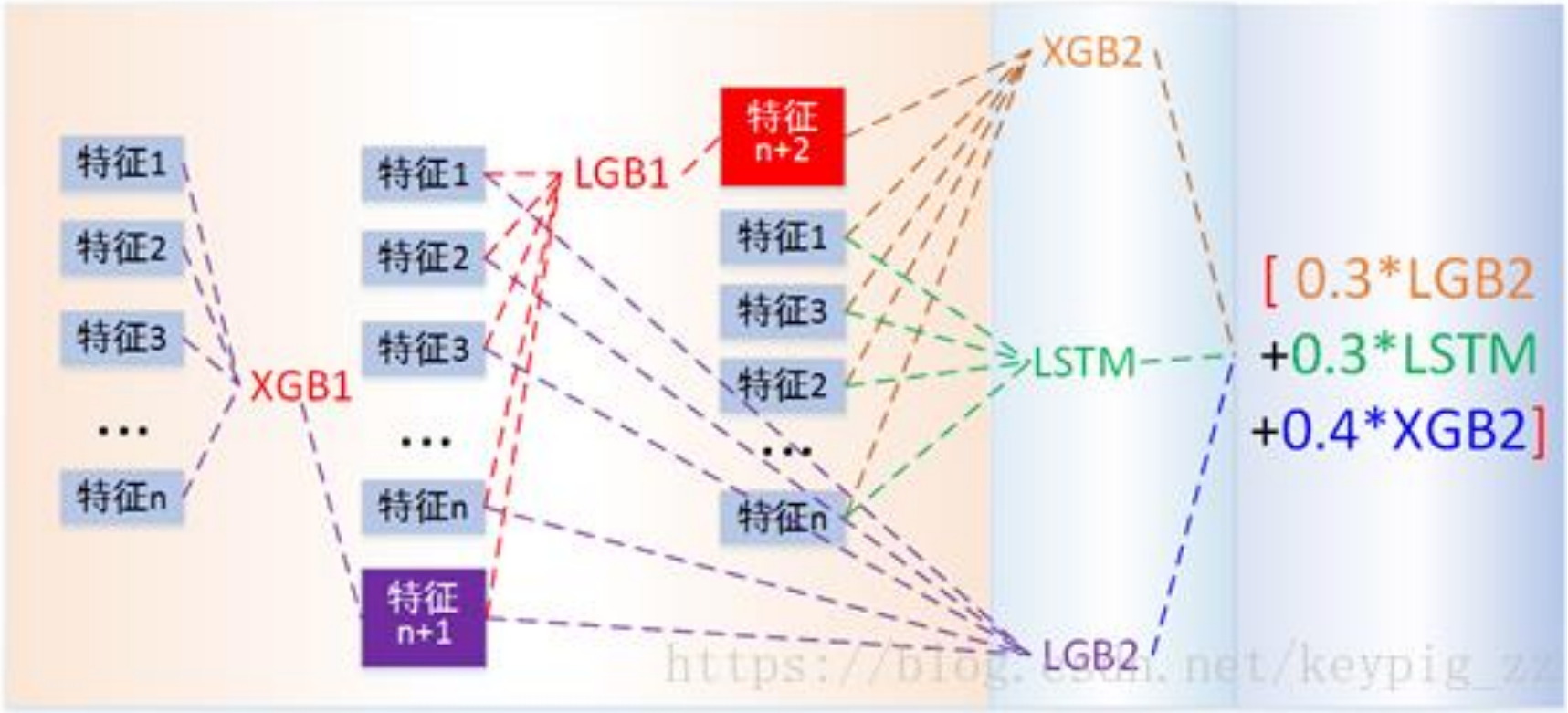

*Figure 7-15. Predictions in a multilayer stacking ensemble*

- **赛题背景：**作为世界第一大清洁能源的太阳能相对煤炭石油等能源来说是可再生、无污染的，只要有太阳就有太阳能，所以太阳能的利用被很多国家列为重点开发项目。但太阳能具有波动性和间歇性的特性，太阳能电站的输出功率受光伏板本体性能、气象条件、运行工况等多种因素影响，具有很强的随机性，由此带来的大规模并网困境严重制约着光伏发电的发展。通过对未来光伏发电功率的短期准确预测并设定调度计划是解决此问题的关键。目前，光伏发电功率预测技术多仅围绕气象条件和历史数据建模，而忽略了光伏板本体性能和实际运行工况对发电效率的影响，因此无法保障短期发电功率预测精度。
- **赛题任务：**在分析光伏发电原理的基础上，论证了辐照度、光伏板工作温度等影响光伏输出功率的因素，通过实时监测的光伏板运行状态参数和气象参数建立预测模型，预估光伏电站瞬时发电量，根据光伏电站DCS系统提供的实际发电量数据进行对比分析，验证模型的实际应用价值。

- **成果：**在比赛中我们使用的核心模型为：XGBoost+LightGBM+LSTM。最终在初赛A榜和B榜分别获得第一名，决赛获得第二名

- **问题设定和分析：**这是一个连续目标变量回归预测问题，很多模型都能有效的解决此类问题。但是，不同的模型原理和所得结果之间是存在差异的。
- **解决思路：**在比赛中我们借鉴了Stacking的思想，融合了LightGBM、XGBoost以及LSTM三个模型。其中前两类可以看作是树模型，LSTM为神经网络模型。这两类模型原理相差较大，产生的结果相关性较低，融合有利于提高预测准确性。
- 原文链接：https://blog.csdn.net/keypig_zz/article/details/82819558

# 预测模型整体结构




- 使用Xgboost_1对特征组合F1进行学习，得到Xgboost_1的预测结果（包括对于训练集和测试集的预测结果），该结果会作为新特征，加入特征组合F2，F3中，分别作为第二层LightGBM_1 和 LightGBM_2的输入特征，LightGBM_1的结果再次作为新特征，加入特征组合F4中，作为第三层Xgboost_2的输入特征，同时第三层包含一个LSTM模型，该模型使用特征组合F5训练，第二层LightGBM_2的结果则与第三层Xgboost_2,LSTM的预测结果进行加权融合作为最终结果。

# Bagging and Pasting

- 每一个弱预测器都使用相同的模型，但是在原始训练集的随机子集或随机特征上去训练它们

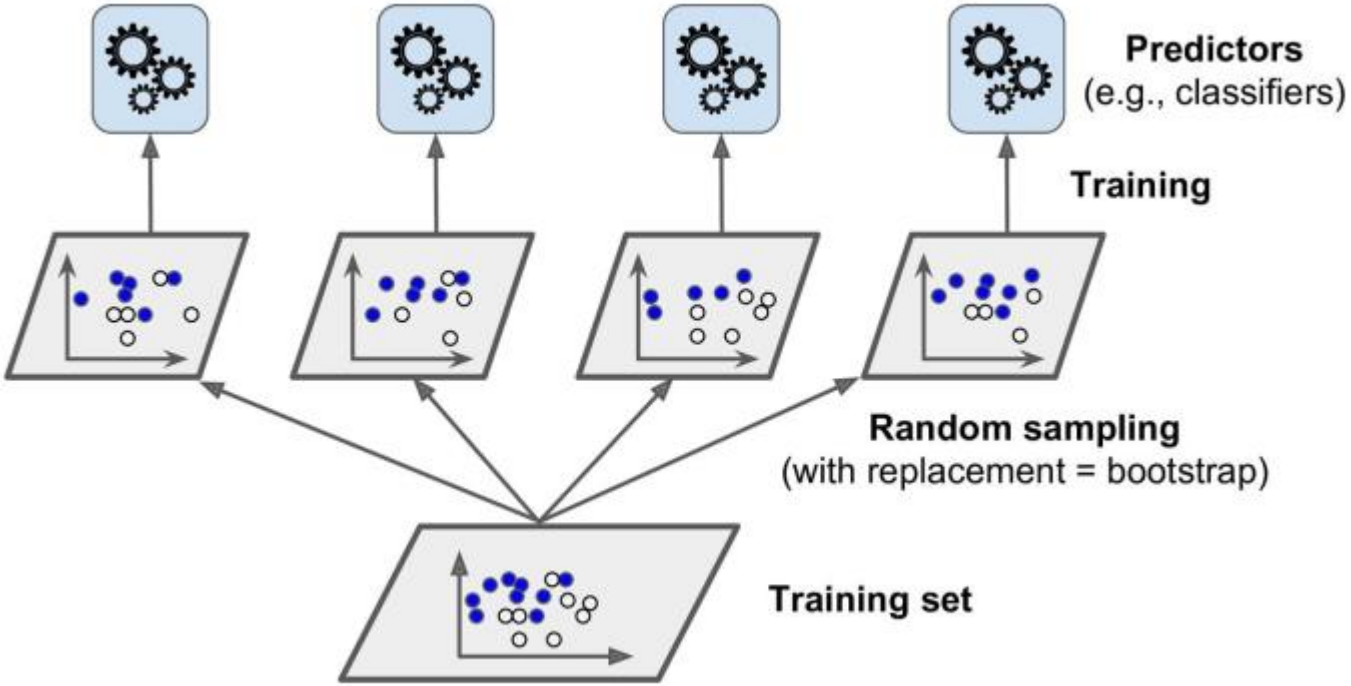



*Figure 7-4. Pasting/bagging training set sampling and training*


- 按照随机抽取训练子集的方法的不同，bagging方法可分为：
  - 如果抽取的数据集的随机子集是**样例的随机子集**，我们叫做**粘贴 (Pasting)** [B1999]。
  - 如果**样例抽取是有放回**的，我们称为 **Bagging** [B1996]。
  - 如果抽取的数据集的随机子集是**特征的随机子集**，我们叫做**随机子空间** (Random Subspaces) [H1998]。
  - 最后，如果基估计器构建在对于**样本和特征**抽取的**子集**之上时，我们叫做**随机补丁** (Random Patches) [LG2012]。

- 训练了一个500个决策树分类器的集成，每一个都是在数据集上有放回采样100个训练实例下进行训练。

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
        DecisionTreeClassifier(), n_estimators=500,
        max_samples=100, bootstrap=True, n_jobs=-1
    )
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

- 集成前后的效果对比：

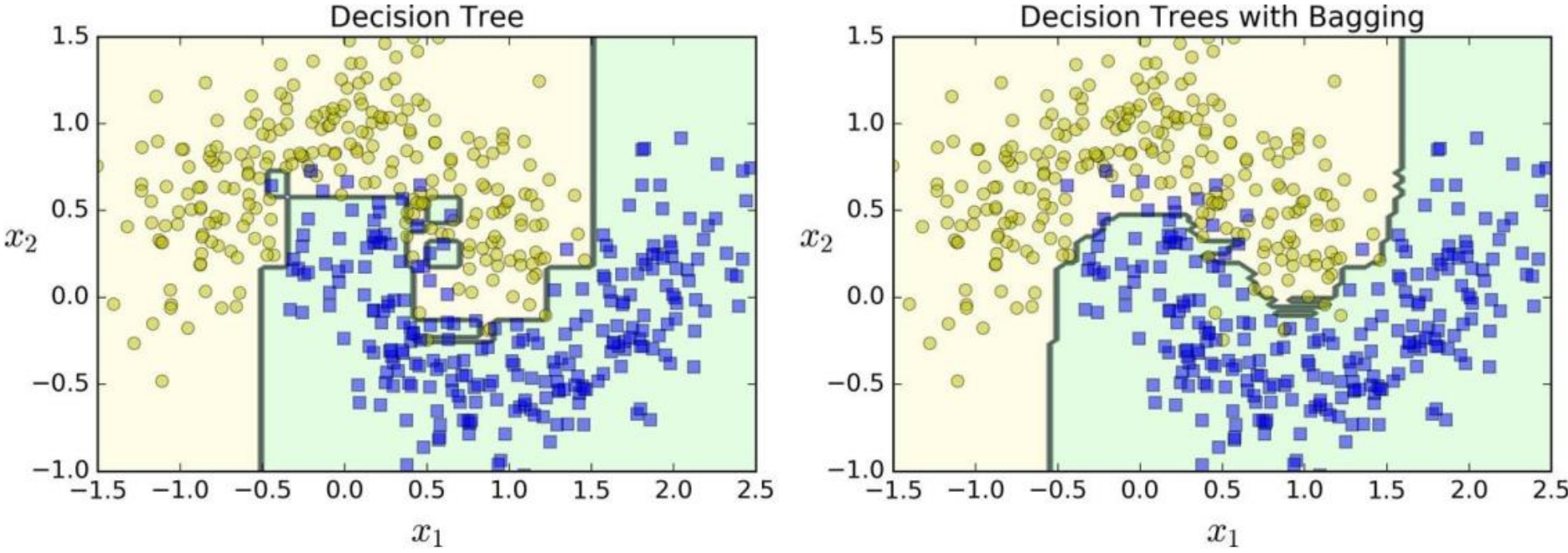



*Figure 7-5. A single Decision Tree versus a bagging ensemble of 500 trees*

对比了单一决策树的决策边界和Bagging集成500个树的决策边界，两者都在moons数据集上训练。正如你所看到的，集成的分类比起单一决策树的分类产生情况更好：集成有一个可比较的偏差但是有一个较小的方差（它在训练集上的错误数目大致相同，但决策边界较不规则）。

# 随机森林(Random Forest)

- 随机森林：是决策树的一种集成，通常是通过bagging方法（有时是pasting方法）进行训练。
- 包含了500个决策树的随机森林的定义和训练

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

≈

```
bag_clf = BaggingClassifier(
        DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),
        n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1
    )
```

# 随机森林(Random Forest)

- 特点：
  1. 随机选择样本（放回抽样）；
  2. 随机选择特征属性；
  3. 基预测器为决策树；
  4. 随机森林投票（平均）　因此防止过拟合能力更强，降低方差。

# 随机森林 VS. Bagging方法

- 随机森林： 是对bagging算法的改进。
  - 改进1：基本预测器限定为决策树
  - 改进2：除了bagging的在样本上加上扰动，同时在属性上也加上扰动，即是在决策树学习的过程中引入了随机属性选择，对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含k个属性的子集，然后再从这个子集中选择一个最优属性用于划分。

# 利用随机森林来预测特征的重要性

- 单一决策树，重要的特征会出现在更靠近根部的位置，而不重要的特征会经常出现在靠近叶子的位置。
- 可以通过计算一个特征在森林的全部树中出现的平均深度来预测特征的重要性。
  - 利用Sklearn中的feature_importances_变量来查看特征的重要度

- 在iris数据集上训练一个Random Forest Classifier 模型，然后输出了每个特征的重要性。

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
>>>     print(name, score)
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

**结论：最重要的特征是花瓣长度（44%）和宽度（42%），而萼片长度和宽度相对比较是不重要的（分别为11%和2%）**

# Boosting方法（提升法）

- Boosting方法的总体思路：逐步增加同一类别的弱预测器，使得每一个弱预测器都要尝试修正前一个弱预测器的预测结果。按照修正预测结果的方式的不同，可以分为：
  - AdaBoost方法
    - 更新样本的权重，对拟合效果不好的训练实例多加关注
  - 提升树(boosting tree)系列算法
    - 让当前弱预测器针对前一个弱预测器预测的残差进行拟合

# Boosting VS. Bagging

- 相同点：
  - 基预测器的种类单一
- 不同点：
  - Bagging：
    - 基预测器可并行训练
    - 主要在优化variance（即模型的鲁棒性）
    - bagging 方法可以减小过拟合，所以通常在强分类器和复杂模型上使用时表现的很好（例如，完全生长的决策树，fully developed decision trees）
  - boosting 方法
    - 基预测器串行训练
    - boosting主要在优化bias（即模型的精确性）
    - 在弱模型上表现更好（例如，浅层决策树，shallow decision trees）。

# AdaBoost算法

- AdaBoosting算法：对之前拟合效果不好的训练实例多加关注，来实现对前序弱预测器预测结果的修正。
  - 如何训练？依次训练弱预测器，计算当前弱预测器的加权值，不断更新样本权重。
  1. 首先从训练集用初始权重训练出一个弱预测器1
  2. 根据弱预测器1的学习误差率表现，计算弱预测器1的权重，并更新训练样本的权重，对弱预测器1误差率高的训练样本增加其权重，使得这些训练误差率高的样本在下一个弱预测器2中得到更多的重视。
  3. 基于调整权重后的训练集来训练弱预测器2；如此重复进行，直至弱预测器数量达到事先指定的数目T，
  4. 最终将这T个弱预测器通过聚合策略进行整合，得到最终的强学习器。
- 如何预测？
  - 强学习器的预测结果 = 各个弱学习器预测结果的加权和。

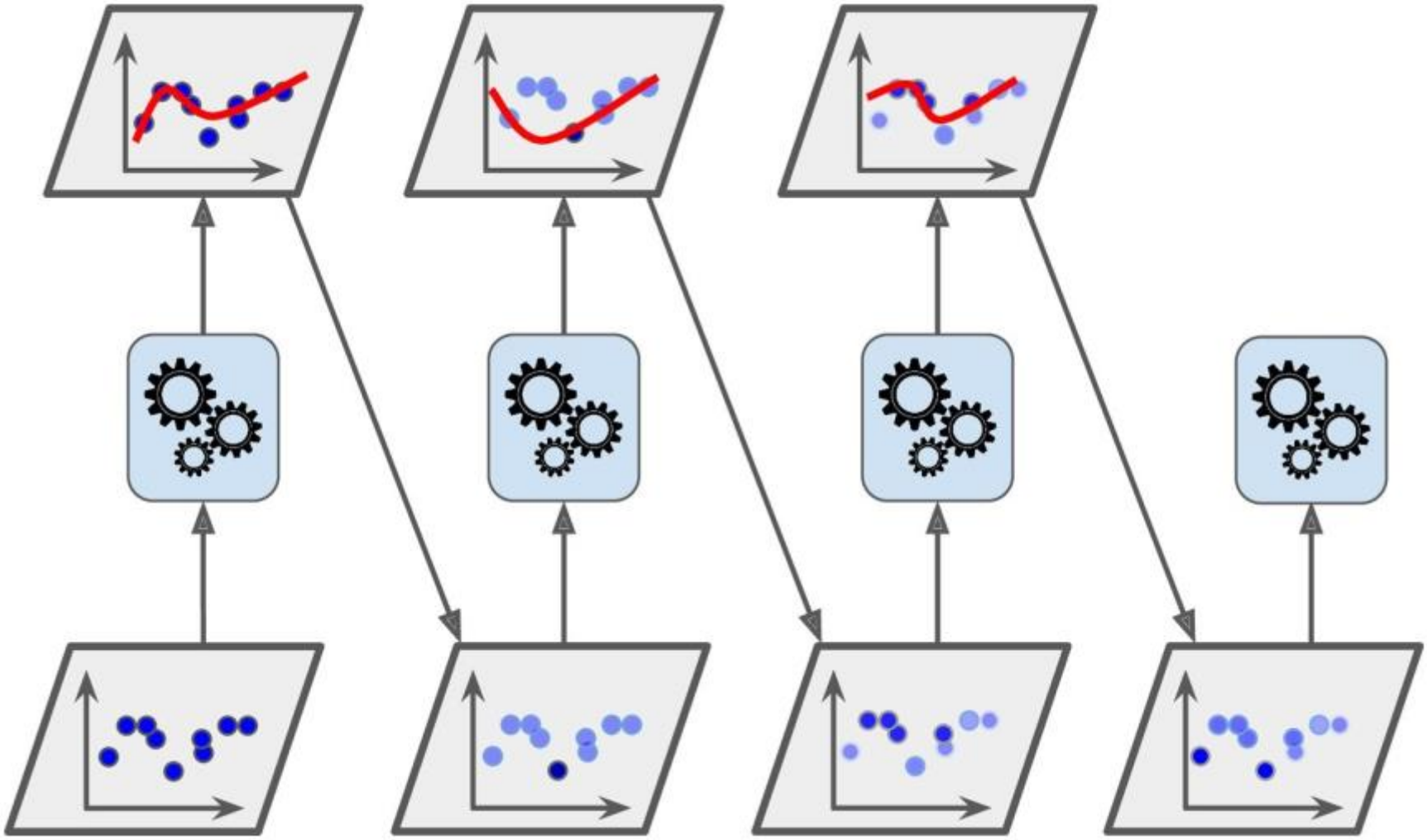


*Figure 7-7. AdaBoost sequential training with instance weight updates*

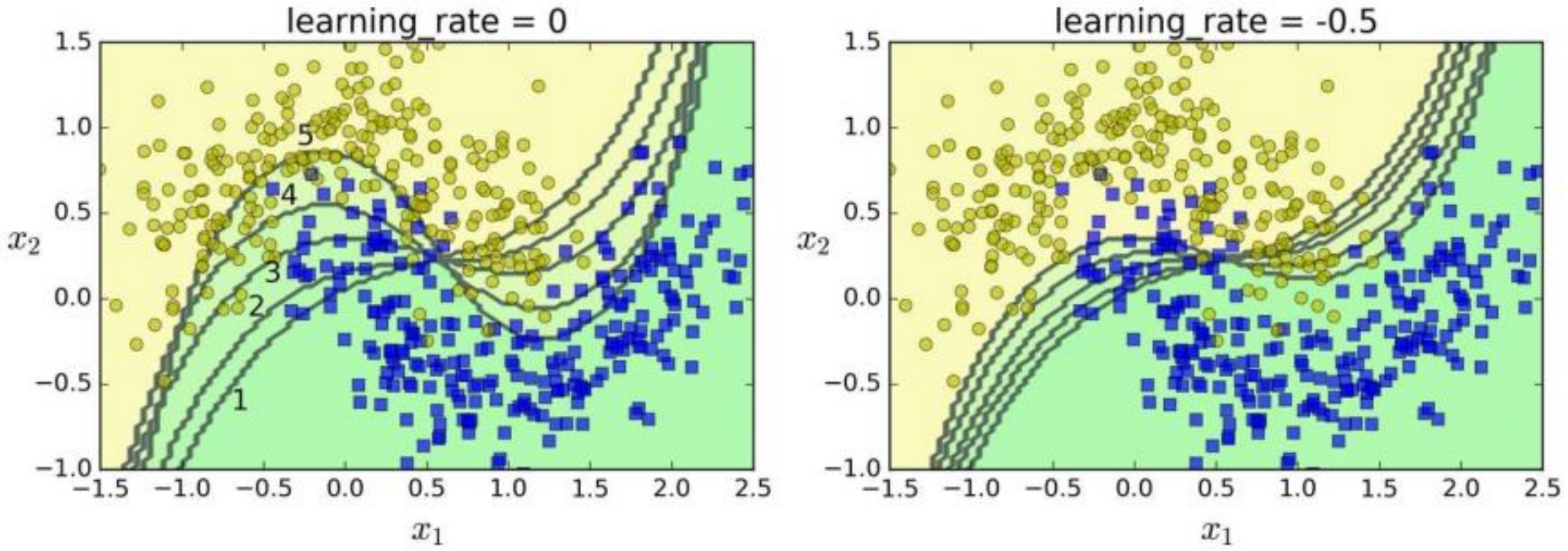
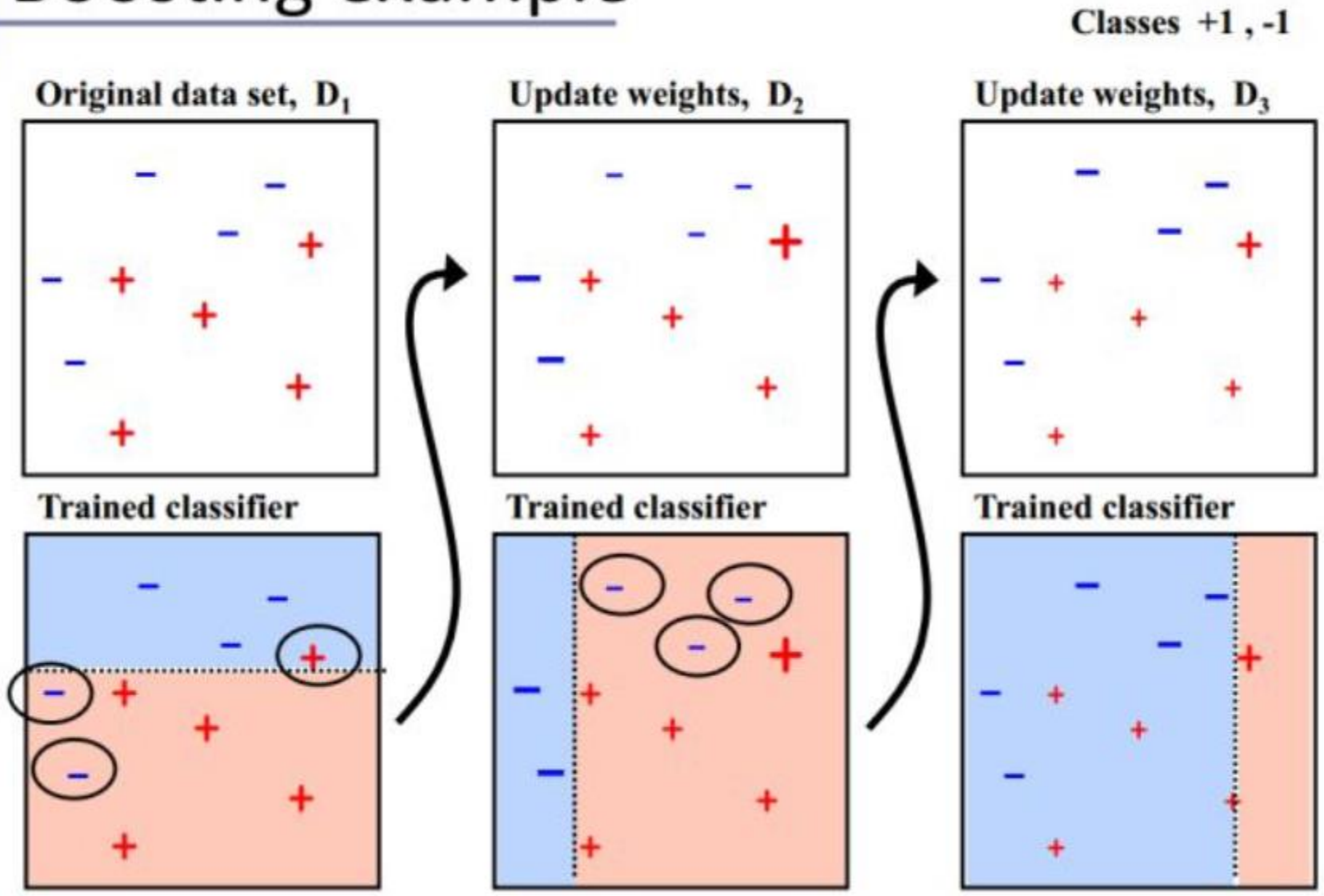


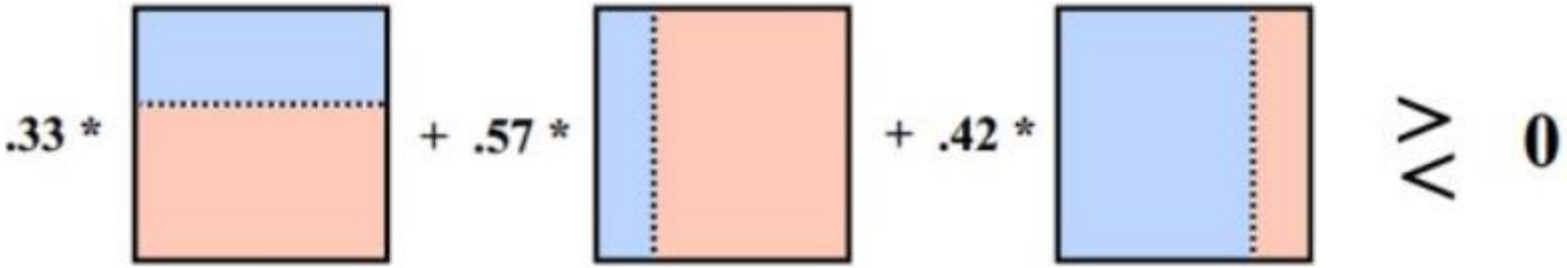

*Figure 7-8. Decision boundaries of consecutive predictors*

Learning_rate参数用来控制每个弱预测器器对 最终的强预测器的预测结果的贡献程度

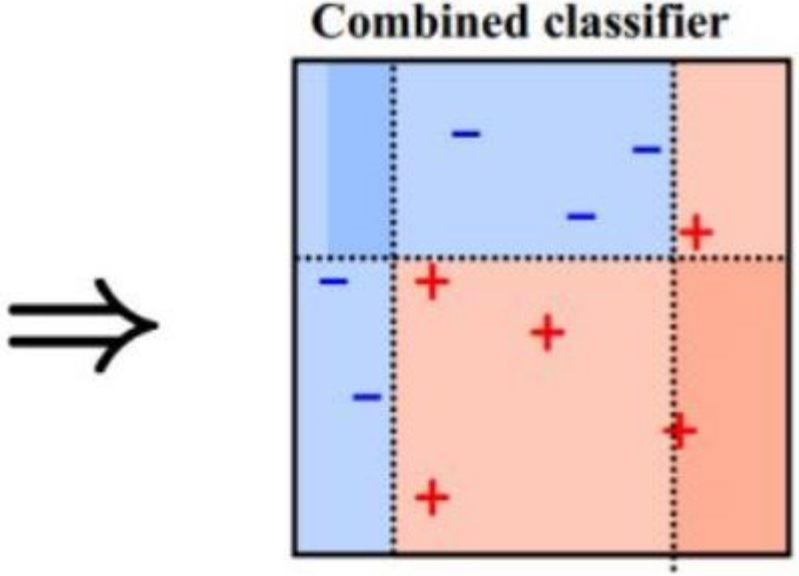
# Boosting example



**Weight each classifier and combine them:**




**Combined classifier**



**1-node decision trees**
**"decision stumps"**
***very simple classifiers***

# 梯度提升(Gradient Boosting)

- 让当前弱预测器针对前一个弱预测器预测的残差进行拟合，来实现对前序弱预测器预测结果的修正。

- 比如，梯度提升回归树(GBRT，Gradient Tree Boosting 或者Gradient Boosted Regression Trees）
  - 示例：使用决策树当做基/弱分类器，集成三个决策树来解决回归问题

# GBRT（梯度提升回归树）

**训练**

- 用DecisionTreeRegressor去拟合训练集

```
>>>from sklearn.tree import DecisionTreeRegressor
>>>tree_reg1 = DecisionTreeRegressor(max_depth=2)
>>>tree_reg1.fit(X, y)
```

- 针对第一个弱预测器的残差，训练第二个DecisionTreeRegressor：

```
>>>y2 = y - tree_reg1.predict(X)
>>>tree_reg2 = DecisionTreeRegressor(max_depth=2)
>>>tree_reg2.fit(X, y2)
```

- 针对第二个弱预测器的残差，训练第三个DecisionTreeRegressor：

```
>>>y3 = y2 - tree_reg1.predict(X)
>>>tree_reg3 = DecisionTreeRegressor(max_depth=2)
>>>tree_reg3.fit(X, y3)
```

**预测**

- 利用强预测器来做预测：将所有弱预测器的预测结果相加

```
>>>y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

# 扩展：XGBoost (eXtreme Gradient Boosting)

- XGBoost：是集成学习Boosting家族的成员，是在GBDT的基础上对boosting算法进行的改进。
  - GBDT是用模型在数据上的负梯度作为残差的近似值，从而拟合残差。XGBoost也是拟合的在数据上的残差。
- XGBoost改进点：
  - 损失函数的改变：（导数和正则项）
    - 用泰勒展开式对模型损失残差的近似；同时XGBoost对模型的损失函数进行的改进，并加入了模型复杂度的正则项
  - 工具的优化：（趋势值和并行）
    - shrinkage（缩减），相当于学习速率（XGBoost中学习速率为eta）
    - 列抽样
    - 对缺失值的处理
    - XGBoost工具支持并行

# XGBoost 相关网页资源

- **xgboost 算法原理：**
  https://blog.csdn.net/a1b2c3d4123456/article/details/52849091
- **xgboost原理：**
  https://blog.csdn.net/a819825294/article/details/51206410

- **Scikit中的特征选择，XGboost进行回归预测，模型优化的实战**
  https://blog.csdn.net/sinat_35512245/article/details/79668363
- **XGboost数据比赛实战之调参篇(完整流程)：**
  https://blog.csdn.net/sinat_35512245/article/details/79700029

- **带你走入Kaggle 竞赛top20%的分析方法：**
  **https://blog.csdn.net/FRBeVrQbN4L/article/details/78849087**
- **TOP5%Kaggler：如何在 Kaggle 首战中进入前 10% | 干货：**
  **https://www.leiphone.com/news/201703/kCMQyffeP0qUgD9a.html**

# 扩展：**LightGBM (**Light Gradient Boosting Machine)

- LightGBM：是微软出的新的boosting框架，基本原理与XGBoost一样，只是在框架上做了一优化（重点在模型的训练速度的优化） **LightGBM**中文文档：http://lightgbm.apachecn.org/cn/latest/index.html
- 与XGBoost对比：
  1. XGBoost采用的是level-wise的分裂策略，而LightGBM采用了leaf-wise的策略
  2. LightGBM使用了基于histogram的决策树算法，这一点不同与XGBoost中的exact 算法（tree_method 可以使用 hist参数），histogram算法在内存和计算代价上都有不小优势。

     （1）内存上优势。

     （2）计算上的优势。
  3. 直方图做差加速一个子节点的直方图可以通过父节点的直方图减去兄弟节点的直方图得到，从而加速计算。
  4. LightGBM支持直接输入categorical 的feature在对离散特征分裂时，每个取值都当作一个桶，分裂时的增益算的是”是否属于某个category“的gain。类似于one-hot编码。
  5. 多线程优化

# 扩展：Stacking相关介绍

- **kaggle比赛集成指南** https://blog.csdn.net/dukuku5038/article/details/84195750
- Guide to Model Stacking (i.e. Meta Ensembling) : https://www.gormanalysis.com/blog/guide-to-model-stacking-i-e-meta-ensembling/
- **A Kaggler's Guide to Model Stacking in Practice: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/**
- **Blending in Python: https://github.com/emanuele/kaggle_pbr**
- **数据比赛大杀器----模型融合(stacking&blending): https://blog.csdn.net/u014356002/article/details/54376138**

# 名人访谈录

- Kaggle大师访谈：我的ML竞赛之旅 https://mp.weixin.qq.com/s/iIjmRi06s_WEFzrUefTvHg
- **Kaggle 大神 Eureka 的高手进阶之路 https://www.leiphone.com/news/201803/QtcJFW9OoDI8CMWA.html**

# 竞赛相关资源

- 数据挖掘竞赛，算法刷题网址汇总 https://blog.csdn.net/qq_32146369/article/details/57406618
- Competition——AI：国内外人工智能比赛平台以及竞赛类型、竞赛题目、举行时间等之详细攻略(最全+ing) https://blog.csdn.net/qq_41185868/article/details/83758440
- （干货）各大AI竞赛 Top 解决方案开源汇总+大牛经验（Kaggle，Ali，Tencent、JD、KDD Cup...） https://blog.csdn.net/dukuku5038/article/details/84195750