集成学习算法与模型融合

模型集成&融合

概念

- 集成学习属于机器学习的算法模型、模型融合属于数据挖掘多个模型得到结果的融合，但他们本质是一样的都是为了使用多个基本模型来提高泛化能力。
- 为什么多模型能带来更好效果呢？

1. 从统计上来说，由于假设空间很大，可能有多个假设能在训练集达到同等最优性能（但对于测试集表现不同），若使用单个学习器可能因误选使泛化性能不佳，因此多个学习器可减少这一风险。
2. 从计算性能（最优化）来看。学习算法可能陷入局部极小，因此泛化性能不好。多学习器可降低风险
3. 从表示的方面看，某些学习任务的真实假设可能不在当前学习算法考虑的假设空间，多个学习器则扩大来假设空间。
4. 另外Boosting 类算法能够利用之前的基础学习器的结果，给予错误分类样本更大的权重，或者是利用残差学习。降低了模型的偏差，因此效果更好。

模型集成&融合

概念

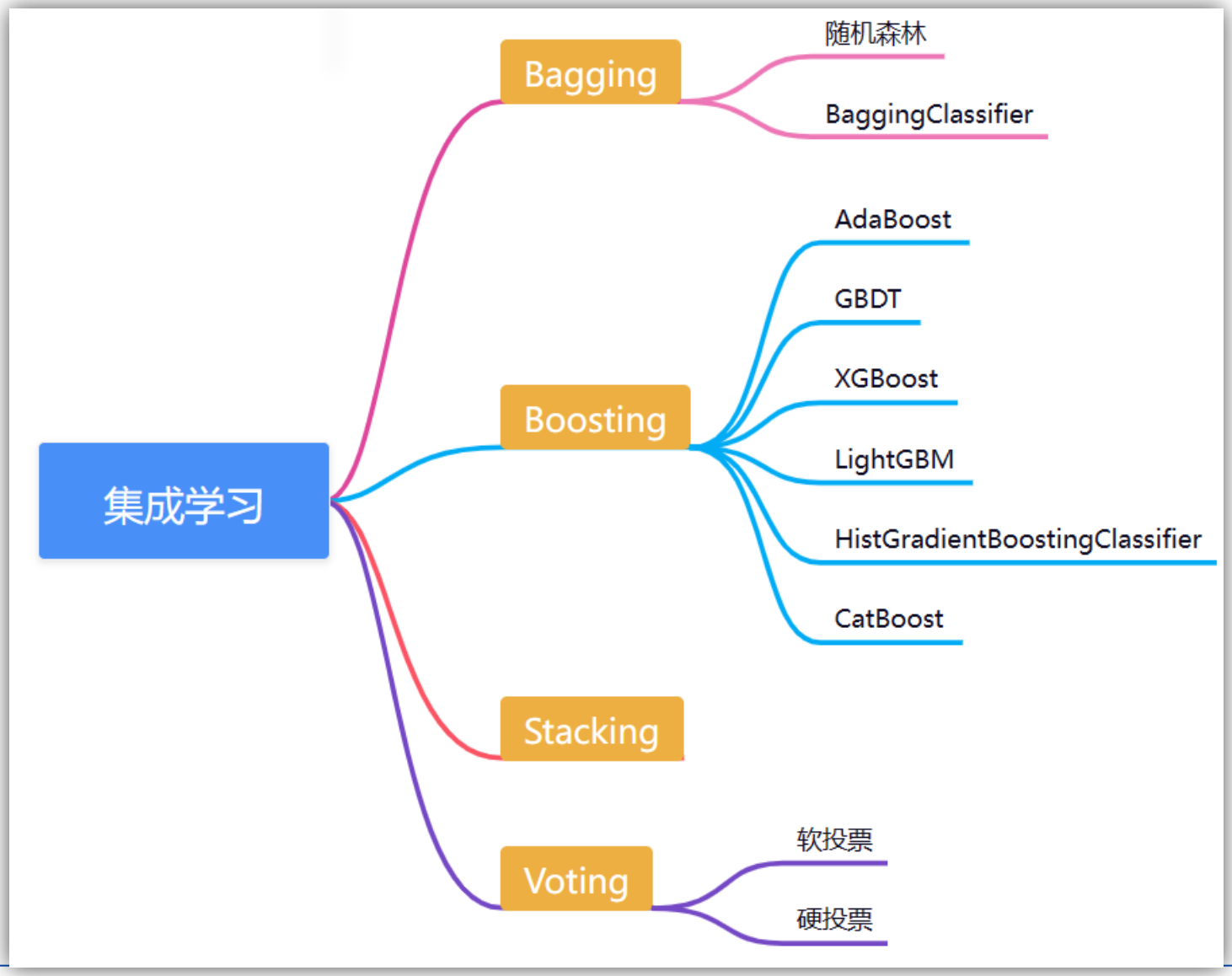
- **集成学习（ensemble learning）是使用一系列学习器进行学习，并使用某种规则把各个学习结果进行整合从而获得比单个学习器更好的学习效果的一种机器学习方法。**
- 通过构建并结合多个学习器来完成学习任务。其一般结构为：先产生一组"个体学习器"（individual learner），再用某种策略将它们结合起来。个体学习器通常由一个现有的学习算法从训练数据产生。
- 集成学习研究的核心，**如何产生并结合"好而不同"的个体学习器**（每个基学习器应尽可能准确，同时尽可能不同）。
- 正所谓三个臭皮匠顶一个诸葛亮，当弱学习器被正确组合时，我们能得到更精确、鲁棒性更好的学习器。

模型集成&融合

常见集成学习方法

- **averaging methods**
- 并行集成方法，其中参与训练的基础学习器并行生成，利用基础学习器之间的独立性降低误差。
- 代表算法：Bagging，Forests
- **boosting methods**
- 串行集成方法，其中参与训练的基础学习器串行训练生成，通过对之前训练中错误标记的样本赋值较高的权重来降低整体预测偏差。
- 代表算法：AdaBoost，Gradient Tree Boosting
- **stacking**
- 模型堆叠，将初级学习器的输出看成次级学习器的输入。

模型集成&融合

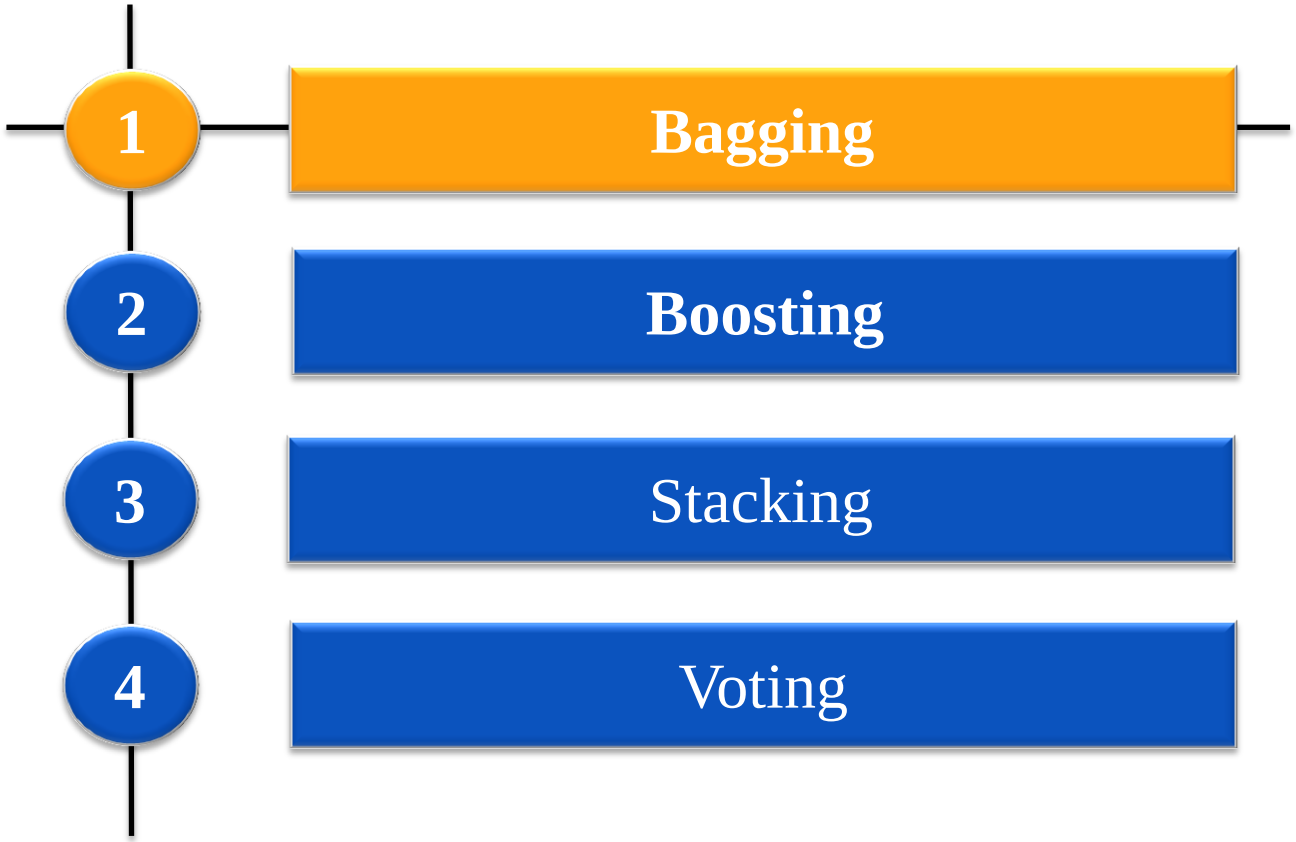

集成学习
Bagging
随机森林
BaggingClassifier
Boosting
AdaBoost
GBDT
XGBoost
LightGBM
HistGradientBoostingClassifier
CatBoost
Stacking
Voting
软投票
硬投票

模型集成&融合

学习材料

- 集成学习：https://scikit-learn.org/stable/modules/ensemble.html#forest
- 集成学习示例：https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_iris.html#sphx-glr-auto-examples-ensemble-plot-forest-iris-py
- 《机器学习原理与实战》ISBN：9787115563996

目录

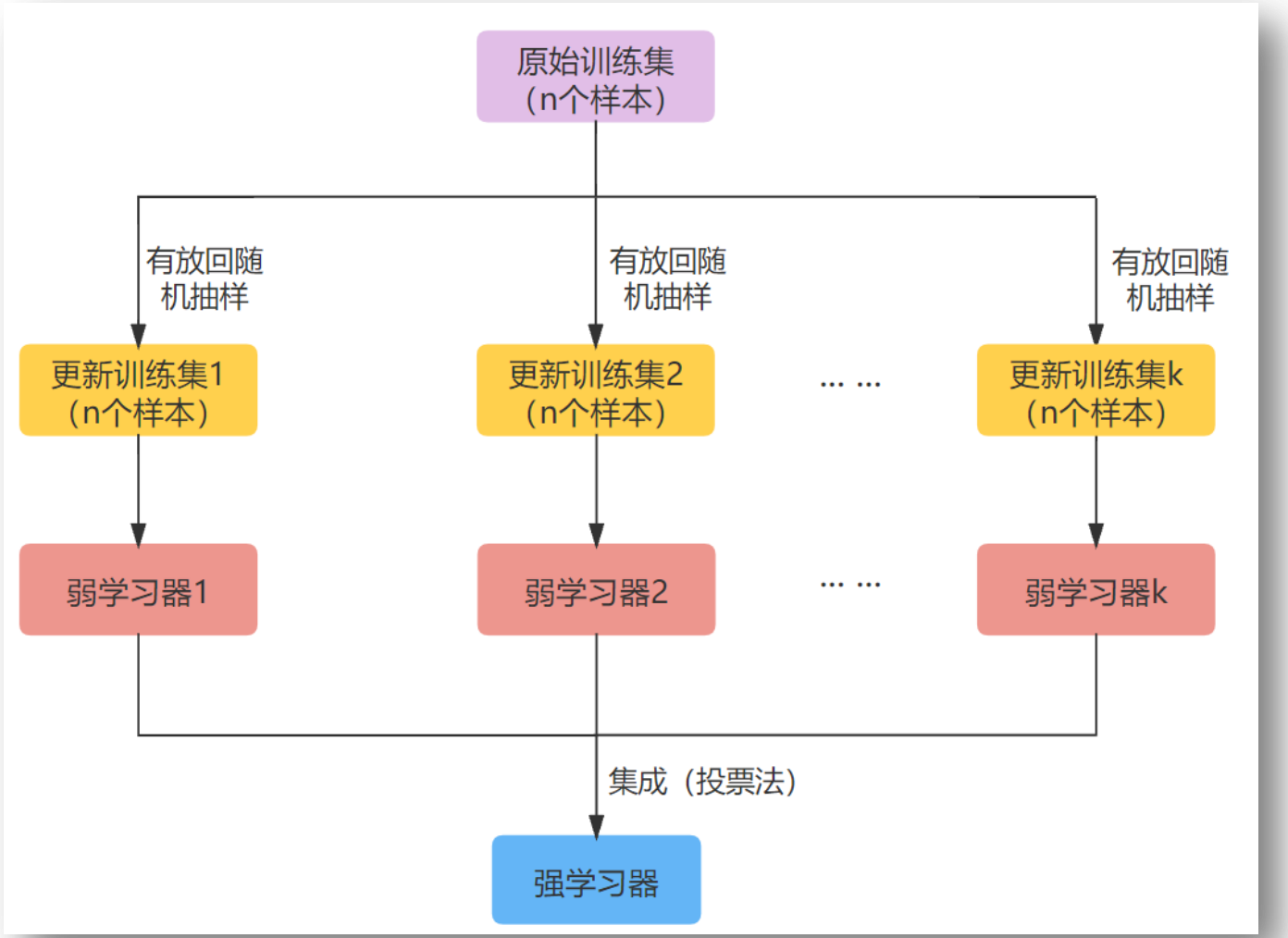


Bagging算法

- Bagging算法(装袋法)是bootstrap aggregating的缩写，它主要对样本训练集合进行随机化抽样，通过反复的抽样训练新的模型，最终在这些模型的基础上取平均。
- Bagging即套袋法，算法过程如下：

1. 从训练样本集中随机可放回抽样（Bootstrapping)N次，得到与训练集相同大小的训练集，重复抽样K次，得到K个训练集 。
2. 每个训练集得到一个最优模型，K个训练集得到K个最优模型。
3. 分类问题：对K个模型采用投票的方式得到分类结果；
4. 回归问题：对K个模型的值求平均得到分类结果。

Bagging算法


原始训练集
（n个样本）
有放回随机抽样
有放回随机抽样
有放回随机抽样
更新训练集1
（n个样本）
更新训练集2
（n个样本）
... ...
更新训练集k
（n个样本）
弱学习器1
弱学习器2
... ...
弱学习器k
集成（投票法）
强学习器

Bagging算法

随机森林

- 随机森林从原始训练样本集N中有放回地重复随机抽取k个样本生成新的训练样本集合，然后根据自助样本集生成k个分类树组成随机森林，新数据的分类结果按分类树投票多少形成的分数而定。
- 其实质是对决策树算法的一种改进，将多个决策树合并在一起，每棵树的建立依赖于一个独立抽取的样品，森林中的每棵树具有相同的分布，分类误差取决于每一棵树的分类能力和它们之间的相关性。

Bagging算法

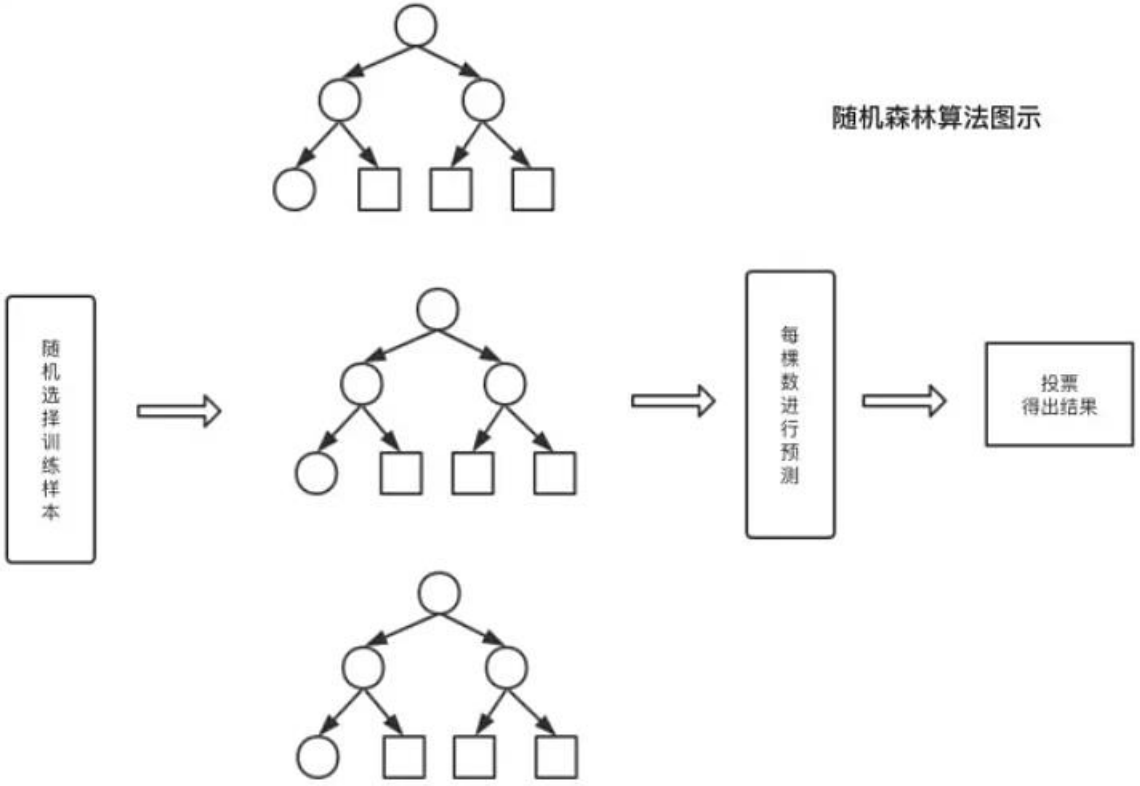
随机森林

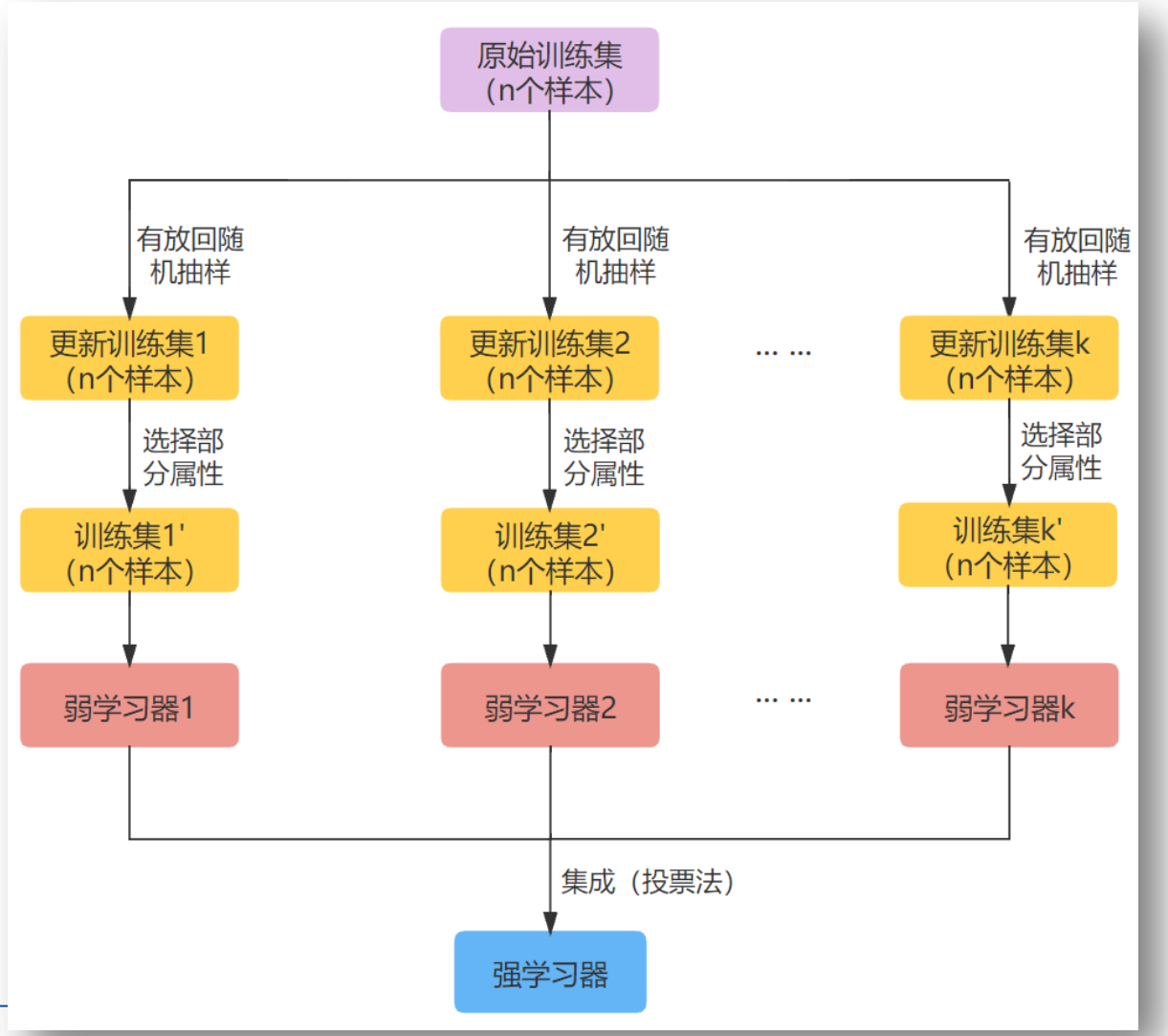
- **随机森林算法过程**
- 从训练数据中选取n个数据作为训练数据输入，一般情况下n是远小于整体的训练数据N的，这样就会造成有一部分数据是无法被取到的，这部分数据称为袋外数据，可以使用袋外数据做误差估计。
- 选取了输入的训练数据的之后，构建决策树，每一个分裂结点从整体的特征集M中选取m个特征构建，一般情况下m远小于M。
- 决策树都采取相同的分裂规则（选取最小的基尼指数进行分裂节点）进行构建，直到该节点的所有训练样例都属于同一类或者达到树的最大深度。
- 重复第2步和第3步多次，每一次输入数据对应一颗决策树，这样就得到了随机森林。输入一个待预测数据，然后多棵决策树同时进行决策，最后采用多数投票的方式进行类别的决策。

Bagging算法

随机森林

- 随机森林算法过程





Bagging算法

随机森林参数

- class sklearn.ensemble.RandomForestClassifier(n_estimators=100,

```
criterion='gini',    # 划分方式：{“gini”, “entropy”, “log_loss”}
max_depth=None,          # 树的最大深度
min_samples_split=2,   # 节点划分至少需要N个样本
min_samples_leaf=1,    # 每个叶节点至少需要N个样本
max_features='sqrt',   # 最多选用多个特征{"sqrt", "log2", None}, int or float
bootstrap=True,          # 是否抽样，否的话使用全部样本
n_jobs=None,             # 并行计算，可以设置任务数
random_state=None,       # 随机种子
,class_weight=None)    # 样本权重，类似{0:10, 1:1},用于样本不均衡
```

Bagging算法

随机森林

- **特征重要性评估**
- 树可用于评估该特征的相对重要性 关于目标变量的可预测性。使用的功能树的顶部有助于最终预测决策输入样本的较大比例。
- 因此，它们贡献的样本可以用作特征相对重要性的估计。在scikit-learn中，特征重要性有助于减少噪声，提高模型的精度。

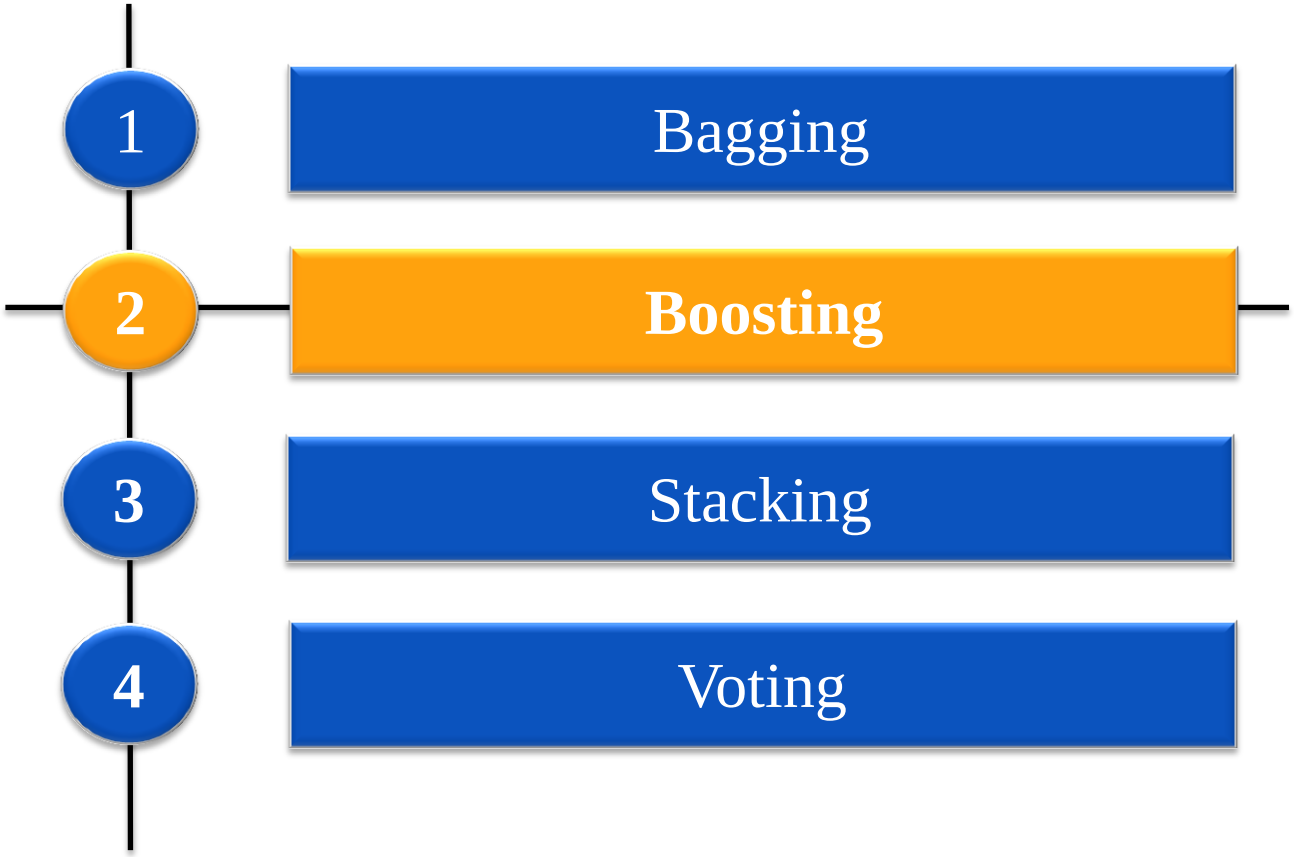
Bagging算法

决策树以外的模型的集合

- Bagging通常有两种类型——决策树的集合(称为随机森林)和决策树以外的模型的集合。两者的工作原理相似，都使用聚合方法生成最终预测，唯一的区别是它们所基于的模型。
- 在sklearn中，有BaggingClassifier类，用于创建除决策树以外的模型。

```
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
clf = BaggingClassifier(base_estimator=SVC(),
                        n_estimators=10, random_state=0)
clf.fit(x1, y1)
pred = clf.predict(x2)
print(classification_report(y2, pred))
```

目录



Boosting算法

Boosting工作机制

- 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关注，然后基于调整后的样本分布来训练下一个基学习器；如此重复进行，直至达到终止条件，最后将得到的个基学习器进行加权结合。
- 从中可知，对提升方法来说，有两个重点：

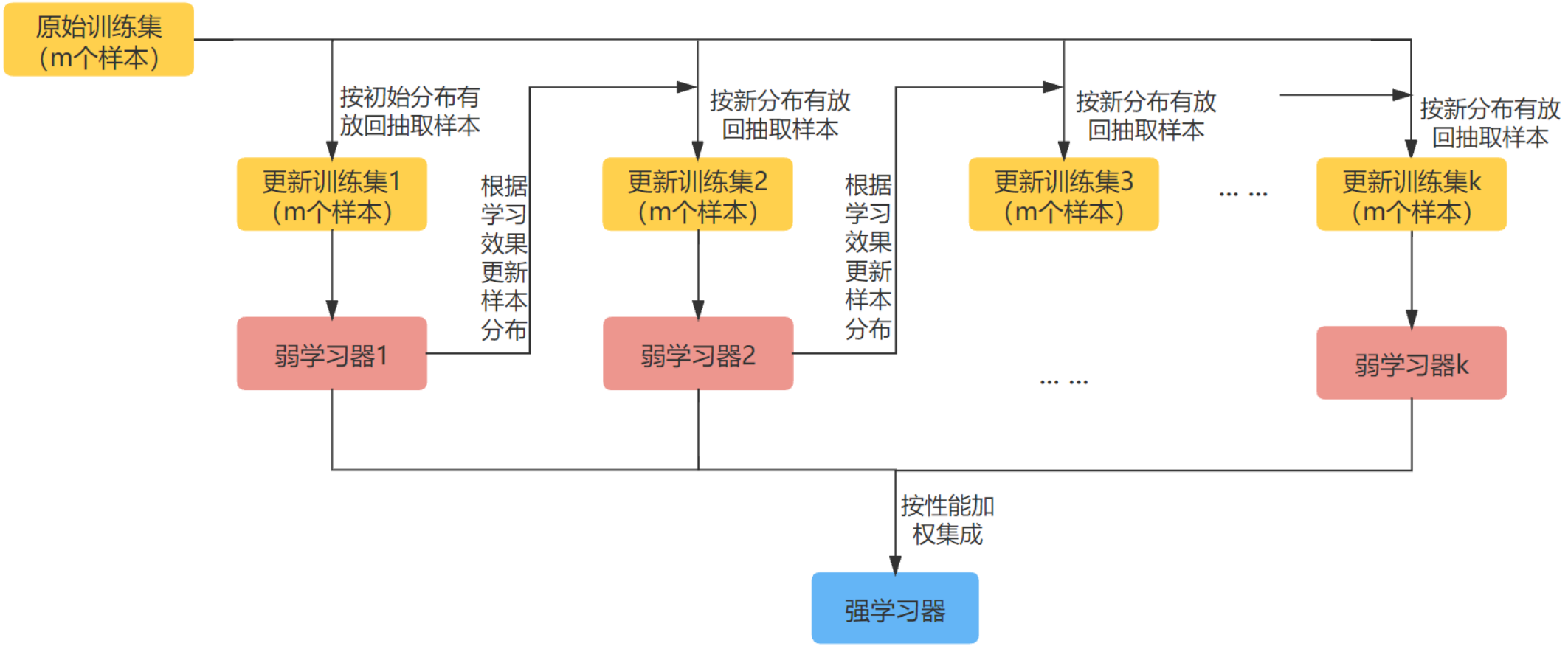
- 每一轮如何改变训练数据的权值和概率分布？

通过提高那些在前一轮被弱学习器分错样例的权值，减小前一轮正确样例的权值，使学习器重点学习分错的样本，提高学习器的性能。

- 通过什么方式来组合弱学习器？

通过加法模型将弱学习器进行线性组合，学习器准确率大，则相应的学习器权值大；反之，则学习器的权值小。即给学习器好的模型一个较大的确信度，提高学习器的性能。

Boosting算法


原始训练集
（m个样本）
按初始分布有
放回抽取样本
更新训练集1
（m个样本）
弱学习器1
根据
学习
效果
更新
样本
分布
按新分布有放
回抽取样本
更新训练集2
（m个样本）
弱学习器2
根据
学习
效果
更新
样本
分布
按新分布有放
回抽取样本
更新训练集3
（m个样本）
… …
… …
按新分布有放
回抽取样本
更新训练集k
（m个样本）
弱学习器k
按性能加
权集成
强学习器

Boosting算法

AdaBoost算法

针对上述问题，AdaBoost的做法是：

- 提高那些被前一轮弱分类器错误分类样本的权值，使其在后续分类中受到更大关注
- 关于弱分类器的组合，采用加权多数表决的方法。即加大分类误差率小的弱分类器的权值。
- Adaboost算法可以简述为三个步骤：

1. 初始化训练数据的权值分布D1。假设有N个训练样本数据，则每一个训练样本最开始时，有相同的权值：w1=1/N。
2. 训练弱分类器Hi。具体训练过程中是：如果某个训练样本点，被弱分类器Hi准确地分类，那么在构造下一个训练集中，它对应的权值要减小；相反，它的权值就应该增大。权值更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去，直到达到迭代次数或者损失函数小于某一阈值。。
3. 将各个训练得到的弱分类器组合成一个强分类器。误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

Boosting算法

AdaBoost算法

- 由于Boosting方法更关注误分类样本，所以最终模型比单个模型有着更高的准确率，但是可能有过拟合的风险。

假设训练数据集为其中有 $T = \{(X1, Y1), (X2, Y2), (X3, Y3), (X4, Y4), (X5, Y5)\}$

其中有 $Y_i = \{-1, 1\}$

- **算法过程**

1. 初始化训练数据的分布；训练数据的权重平均分布 $D = \{W_{11}, W_{12}, W_{13}, W_{14}, W_{15}\}$ 其中 $W_{1i} = \frac{1}{N}$
2. 选择基本分类器；这里选择最简单的线性分类器 $y = aX + b$ 分类器选定之后，最小化分类误差可以求得参数。
3. 计算分类器的系数和更新数据权重；误差率也可以求出来为 e_1 .同时可以求出这个分类器的系数。基本的Adaboost给出的系数计算公式为 $\alpha_m = \frac{1}{2}\log\frac{1-e_m}{e_m}$ 然后更新训练权重分布为 $D_{m+1} = \{w_{m+1,1}, \ldots, w_{m+1,i}, \ldots, w_{m+1,N}\}$ 其中 $w_{m+1,i} = \frac{w_{m,i}}{Z_m}\exp(-\alpha_m y_i G_m(x_i))$ 规范因子为

$$Z_m = \sum i = 1^N w_{m,i} \exp(-\alpha_m y_i G_m(x_i))$$

4. 得出分类器的组合 $f(x) = \sum_{i=1}^{N} exp(-\alpha_m y_i G_m(x_i))$

Boosting算法

AdaBoost算法

- class sklearn.ensemble.AdaBoostClassifier(

```
    estimator=None,        # 基学习器，默认DecisionTreeClassifier，也可自行指定
    n_estimators=50,       # 多少个基学习器
    learning_rate=1.0,     # 学习率，通过learning_rate来减少每个类别的贡献
    algorithm='SAMME.R',   # {'SAMME','SAMME.R'}, SAMME.R用作提升算法。基学习去必须支持计算类概率
SAMME离散提升算法。SAMME.R通常用较少的迭代就可获得较低的测试误差
    random_state=None,     # 随机种子
    )
```

Boosting算法

- 梯度提升机（Gradient Boosting Machine，GBM）是一种Boosting的方法，其提高模型精度的方法与传统Boosting对正确、错误样本进行加权不同，该模型通过在残差减少的梯度（Gradient）方向上建立一个新的模型，从而降低新模型的残差（Residual）。
- 即每个新模型的建立是为了使得之前模型的残差往梯度方向减少。

Boosting算法

- **GBDT算法（梯度提升树）**
- 主要思想为通过每轮产生一颗CART树，对上一轮的CART树的残差进行学习，最终通过对弱分类器进行加权求和,得到最终的分类器，被认为是统计学习中性能最好的方法之一。
- 提升方法实际采用**加法模型**（即基函数的线性组合）与**前向分步算法**。以决策树为基函数的提升方法称为提升树。提升树模型可以表示为决策树的加法模型：

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$$

其中，$T(x;\Theta_m)$ 表示决策树；Θ_m 为决策树的参数；M 为树的个数

Boosting算法

- **GBDT算法（梯度提升树）**
- 主要思想为通过每轮产生一颗CART树，对上一轮的CART树的残差进行学习，最终通过对弱分类器进行加权求和,得到最终的分类器，被认为是统计学习中性能最好的方法之一。
- 提升方法实际采用**加法模型**（即基函数的线性组合）与**前向分步算法**。以决策树为基函数的提升方法称为提升树。提升树模型可以表示为决策树的加法模型：

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$$

其中，$T(x;\Theta_m)$ 表示决策树；Θ_m 为决策树的参数；M 为树的个数

Boosting算法

- **GBDT算法（梯度提升树）**
- class sklearn.ensemble.GradientBoostingClassifier(

```
loss='log_loss',    # 损失{'log_loss', 'deviance', 'exponential'}
learning_rate=0.1,    # 学习率
n_estimators=100,     # 学习器个数
subsample=1.0,        # 参与训练的样本比例
tol=0.0001,           # 迭代停止的误差
criterion='friedman_mse',  # 拆分标准，{'friedman_mse',  'squared_error'} "friedman_mse"表示改进分数为 弗里德曼，"squared_error"表示均方误差。默认值 "friedman_mse"通常是最好的，因为它可以提供更好的 在某些情况下是近似值。

min_samples_split=2, min_samples_leaf=1, max_depth=3, random_state=None,
max_features=None,max_leaf_nodes=None)
```

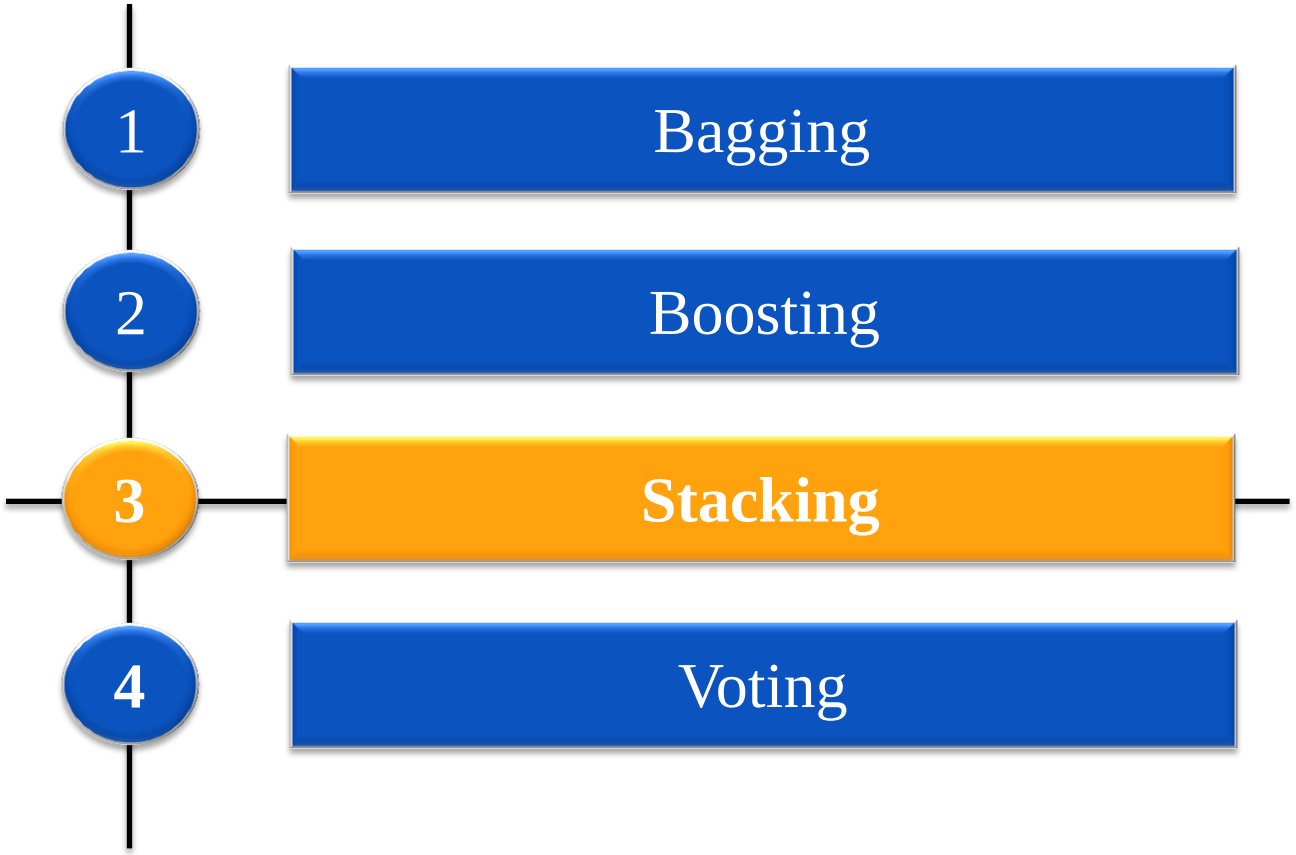
Boosting算法

【GBDT和AdaBoost区别】

1. GBDT弱分类器为回归树
2. GBDT不涉及样本权重的更新
3. Loss一般采用均方差或自定义损失函数

- 更多算法介绍：https://blog.csdn.net/qq602683200/article/details/109198813

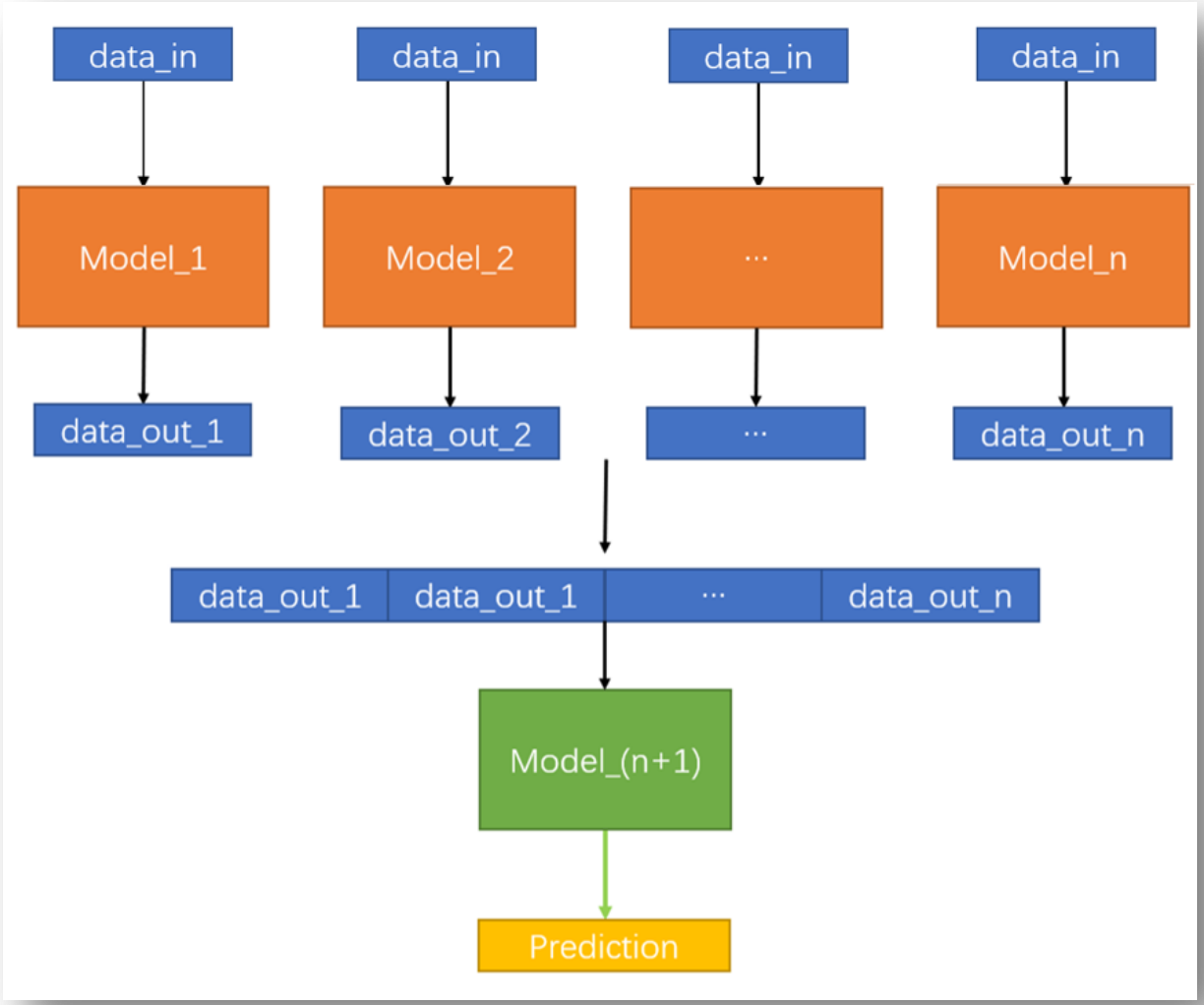
目录



Stacking算法

Stacking简介

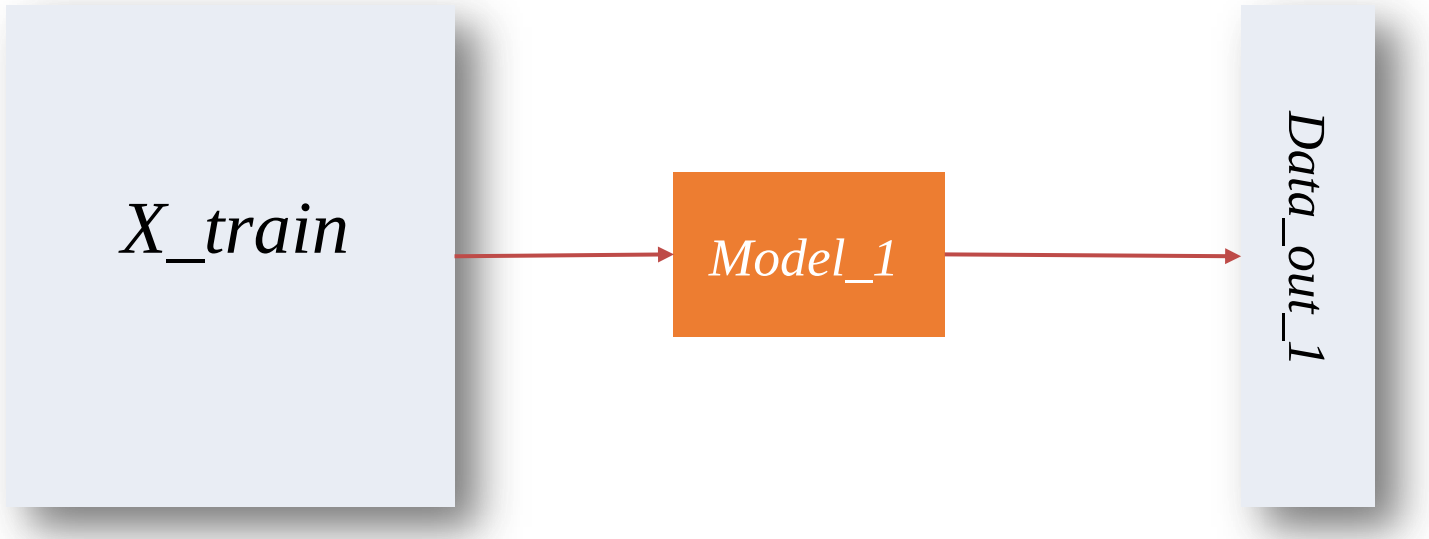
- Stacking方法的思想是将一系列初级学习器的训练结果当作是特征，结合起来输入到一个次级学习器中，最终的输出结果由次级学习器产生。
- Stacking将各初级学习器的优点进行融合从而进一步提升模型性能。
- 在叠加过程中，将数据分为训练集和测试集两部分。训练集会被进一步划分为k-fold。基础模型在k-1部分进行训练，在k^{th}部分进行预测。这个过程被反复迭代，直到每一折都被预测出来。然后将基本模型拟合到整个数据集，并计算性能。



Stacking算法

单个初级学习器训练过程

- 模型输入为训练集样本自变量，输出为目标变量或目标变量的概率分布
- 采用K折交叉验证方式进行训练



Stacking算法

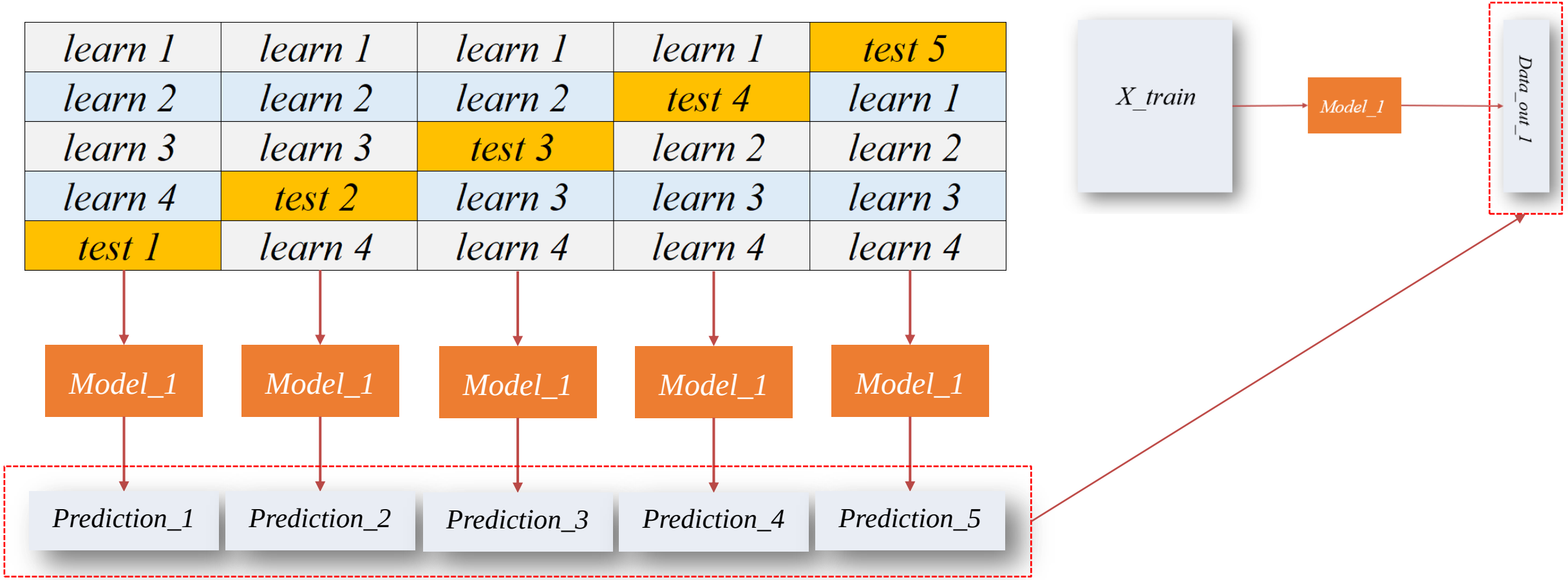
单个初级学习器训练过程

- 将训练集随机划分成k个大小相同的子集
- 依次将每个子集作为测试集，其余$k-1$个子集作为训练集，进行模型训练及预测

Fold 1	*learn 1*	*learn 1*	*learn 1*	*learn 1*	*test 5*
Fold 2	*learn 2*	*learn 2*	*learn 2*	*test 4*	*learn 1*
Fold 3	*learn 3*	*learn 3*	*test 3*	*learn 2*	*learn 2*
Fold 4	*learn 4*	*test 2*	*learn 3*	*learn 3*	*learn 3*
Fold 5	*test 1*	*learn 4*	*learn 4*	*learn 4*	*learn 4*

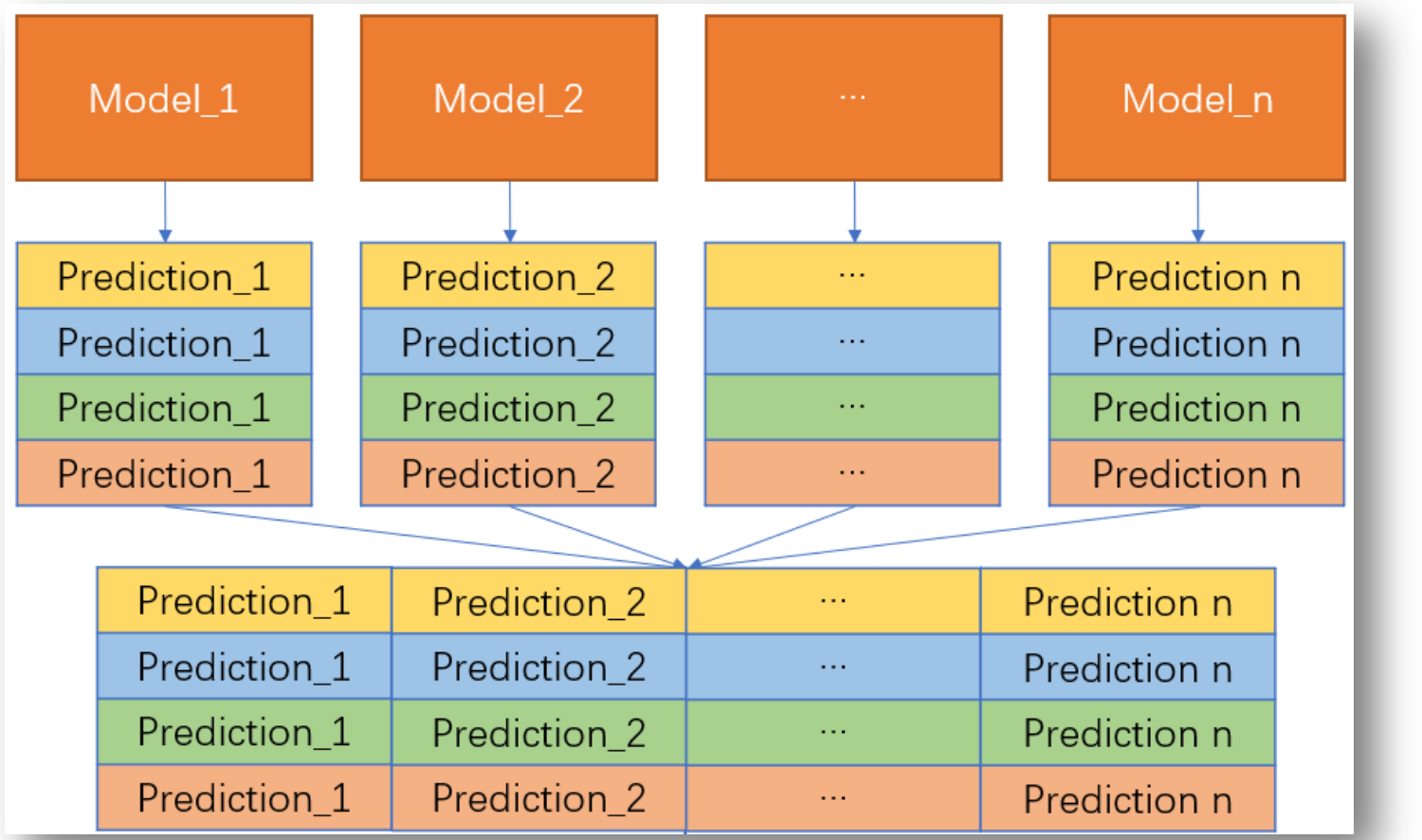
Stacking算法

单个初级学习器训练过程



Stacking算法

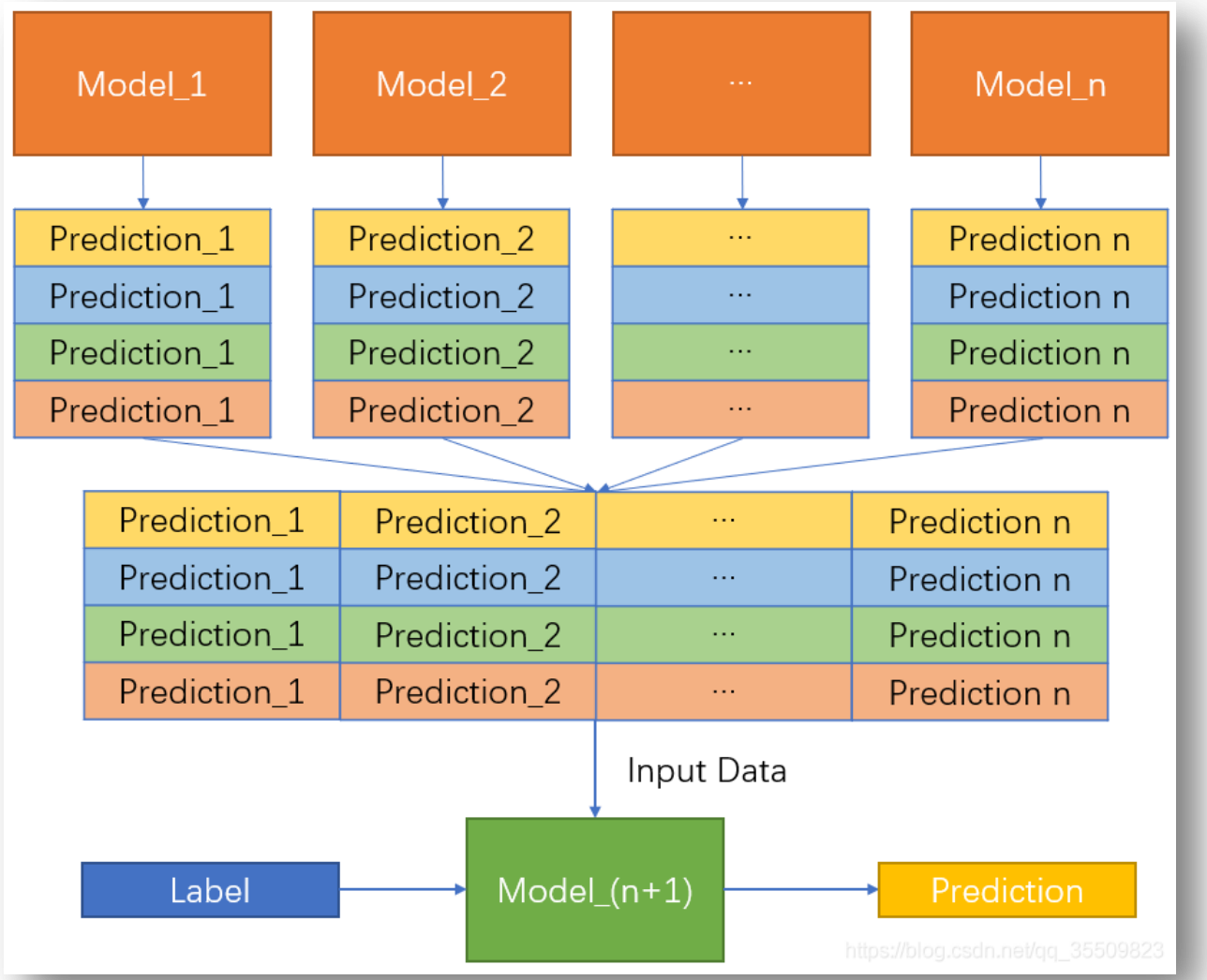
组合初级学习器输出



Stacking算法

次级学习器训练

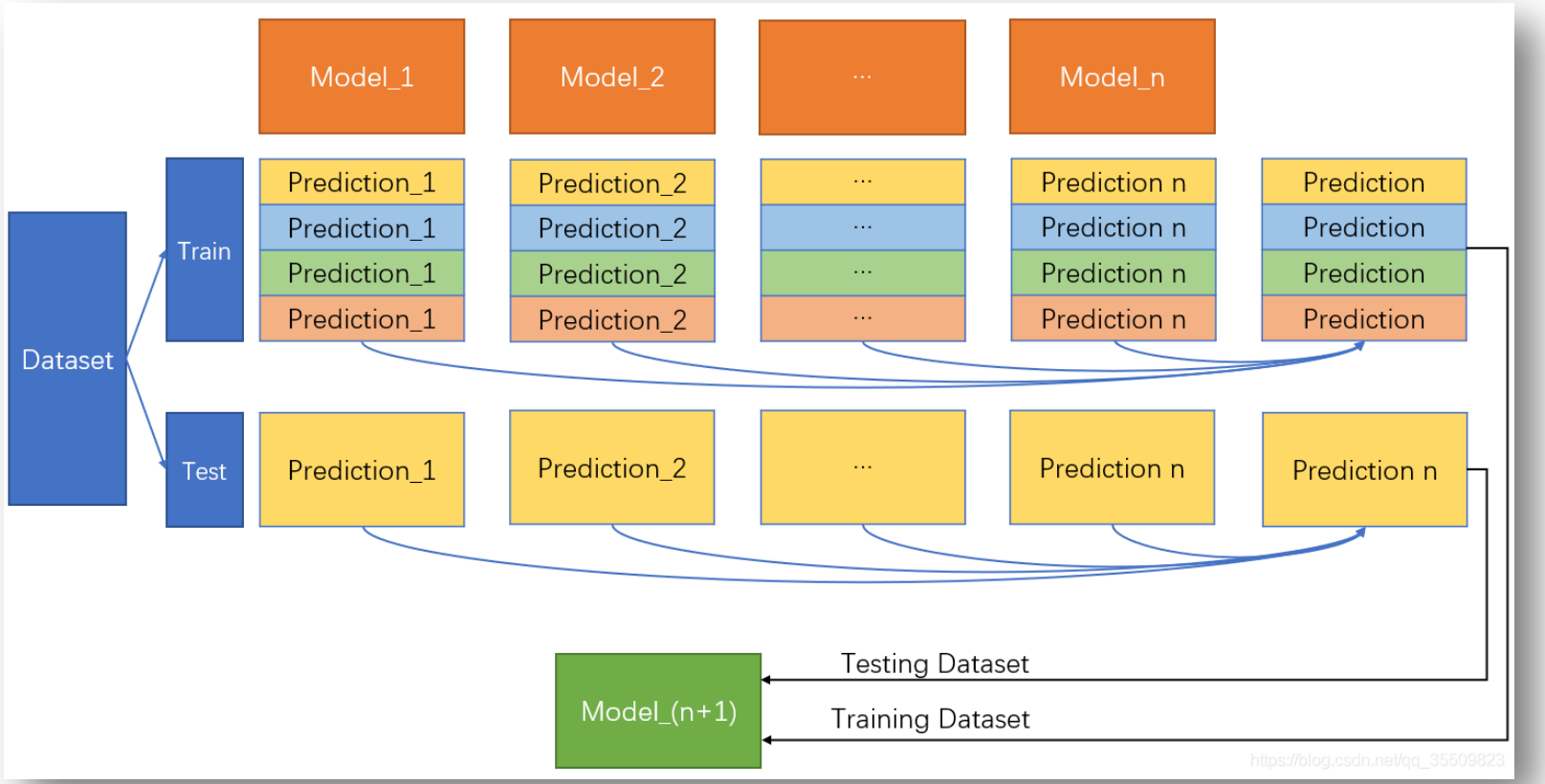
- 输入：元学习器的输出
- 目标输出：原始样本标签



Stacking算法

次级学习器训练

在最开始，将整体数据分为测试集和训练集，即可完成Stacking模型的训练和评估



Stacking算法

Stacking核心代码

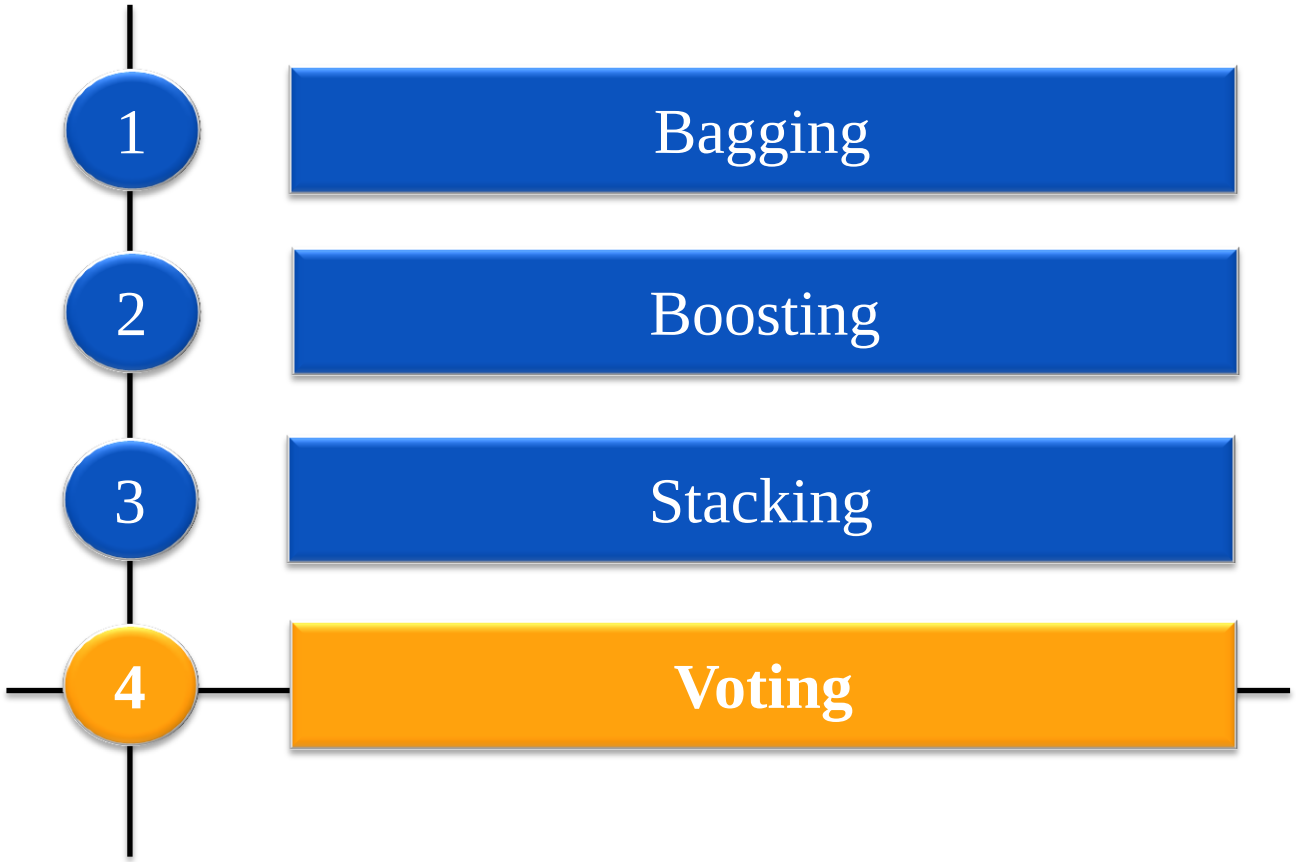
```
estimators = [('ridge', RidgeCV()),                    # 元学习器1
              ('lasso', LassoCV(random_state=42)),  # 元学习器2
              ('knr', KNeighborsRegressor(n_neighbors=20))] # 元学习器3


final_estimator = GradientBoostingRegressor(         # 次级学习器
                        n_estimators=25, subsample=0.5, min_samples_leaf=25,
                        max_features=1,random_state=42)


reg = StackingRegressor(            # Stacking操作
                        estimators=estimators,
                        final_estimator=final_estimator)
```

Stacking算法

优点

1. 是目前提升机器学习效果最好的方法之一
2. 有可能将集成的知识迁移到到简单的分类器上
3. 自动化的大型集成策略可以通过添加正则项有效的对抗过拟合，而且并不需要太多的调参和特征选择。

目录



Voting算法

➢ 投票和平均 Voting and Average

对于**分类任务**来说，可以使用**投票**的方法：

- 简单投票法： $H(\mathbf{x}) = c_{\underset{x}{\operatorname{argmin}}} \sum_{i=1}^{T} h_i^j(\mathbf{x})$
  - 即各个分类器输出其预测的类别，取最高票对应的类别作为结果。若有多个类别都是最高票，那么随机选取一个。
- 加权投票法： $H(\mathbf{x}) = c_{\underset{x}{\operatorname{argmin}}} \sum_{i=1}^{T} \alpha_i \cdot h_i^j(\mathbf{x})$
  - 和上面的简单投票法类似，不过多了权重αi，这样可以区分分类器的重要程度，通常 $\alpha_i \geq 0;\ \sum_{i=1}^{T} \alpha_i = 1$

对于**回归任务**来说，采用的为**平均法(Average)**：

- 简单平均： $H(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^{T} h_i(\mathbf{x})$
- 加权平均： $H(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^{T} \alpha_i \cdot h_i(\mathbf{x});\ \ \alpha_i \geq 0;\ \ \sum_{i=1}^{T} \alpha_i = 1$

Voting算法

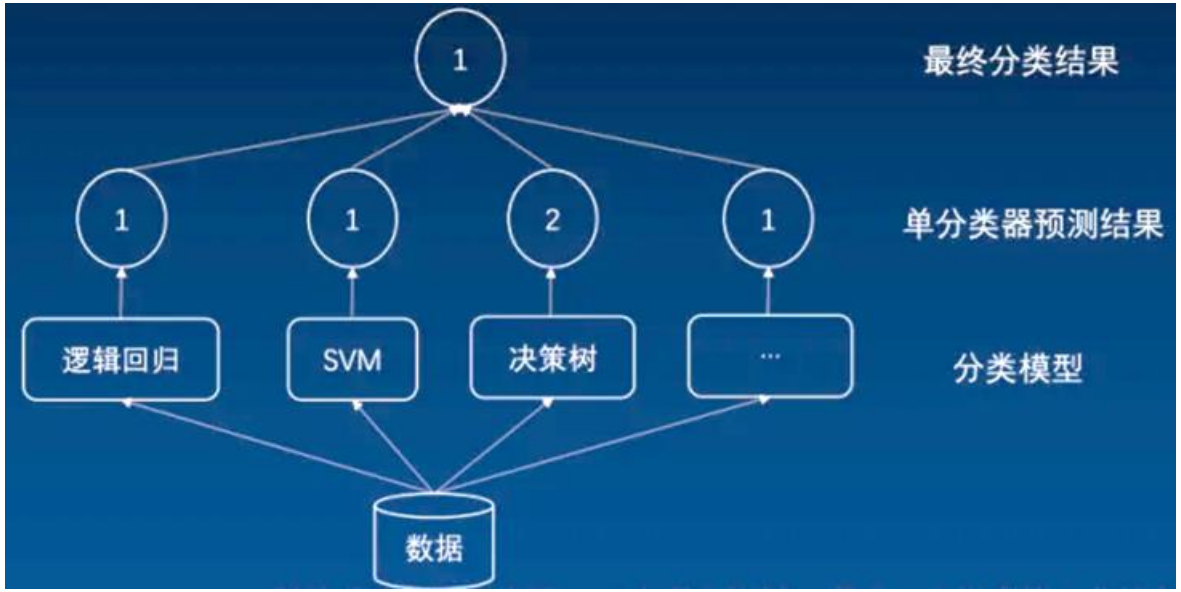
- **投票法（voting）**
- **基本思想：**多个模型对样本进行分类，以"投票"的形式，投票最多者为最终的分类。
- 假设对于一个二分类问题，有3个基础模型，现在我们可以在这些基学习器的基础上得到一个投票的分类器，把票数最多的类作为我们要预测的类别。
- **绝对多数投票法：**最终结果必须在投票中占一半以上。
- **相对多数投票法：**最终结果在投票中票数最多。
- **加权投票法：**按模型权重计算最后的结果。

Voting算法

- **硬投票：**对多个模型直接进行投票，不区分模型结果的相对重要度，最终投票数最多的类为最终被预测的

1： 3个

2： 1个

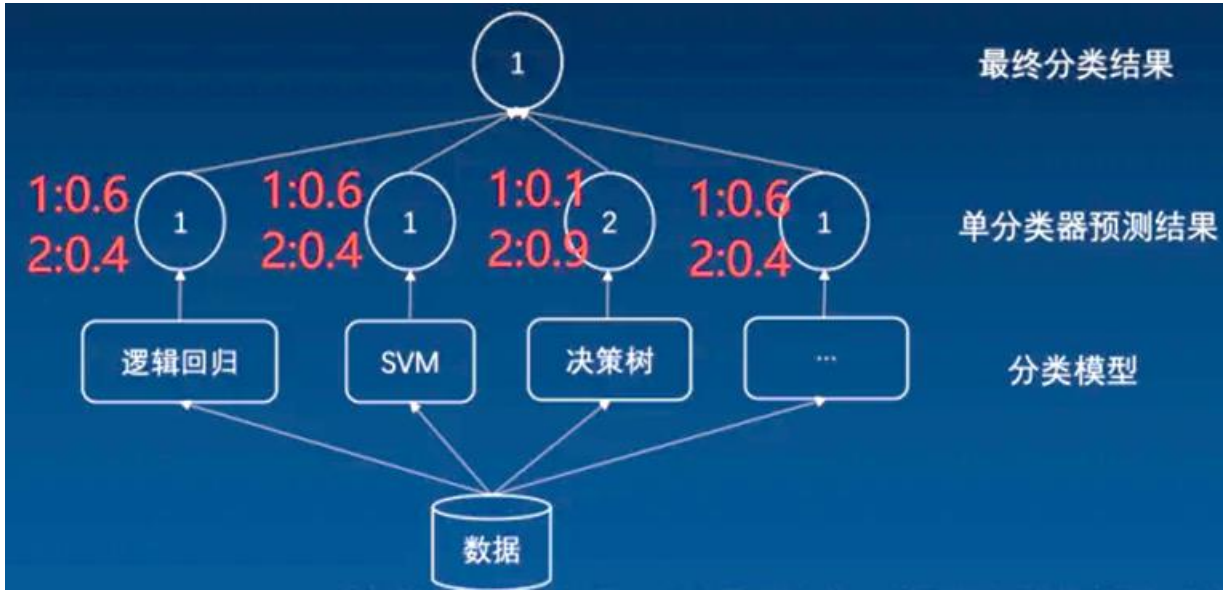


Voting算法

- **软投票：**增加了设置权重的功能，可以为不同模型设置不同权重，进而区别模型不同的重要度。

1： (0.6*3+0.6*2+0.1*1+0.6*1)/4

2： (0.4*3+0.4*2+0.9*1+0.4)*1/4



Voting算法

- Voting实现
- class sklearn.ensemble.VotingClassifier(

```
estimators,      # 学习器列表
voting='hard',   # 硬/软投票{'hard', 'soft'}
weights=None,      # 模型权重数组，软投票时可以指定
n_jobs=None,       # 并行任务
verbose=False)     # 是否打印过程
```

Thank you!