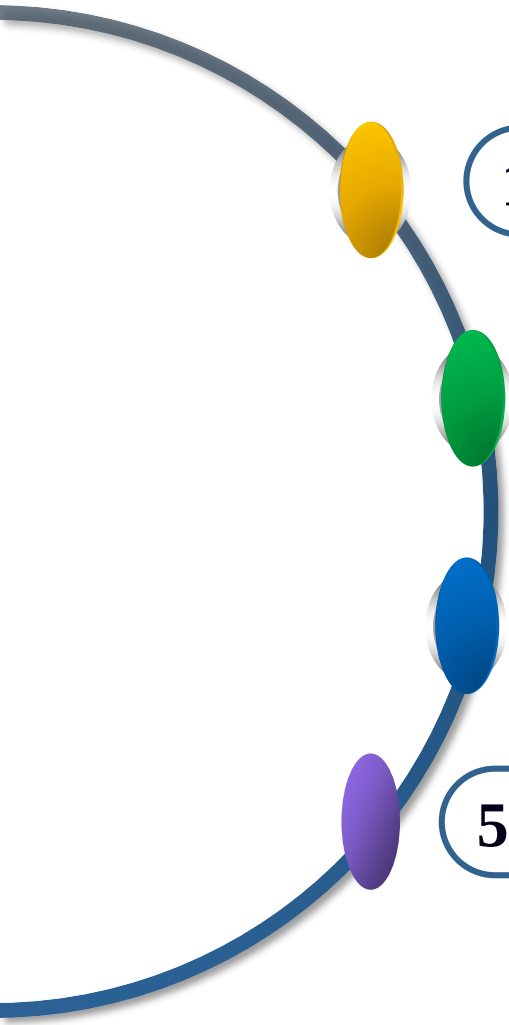
《集成算法》



河北师范大学软件学院
Software College of Hebei Normal University

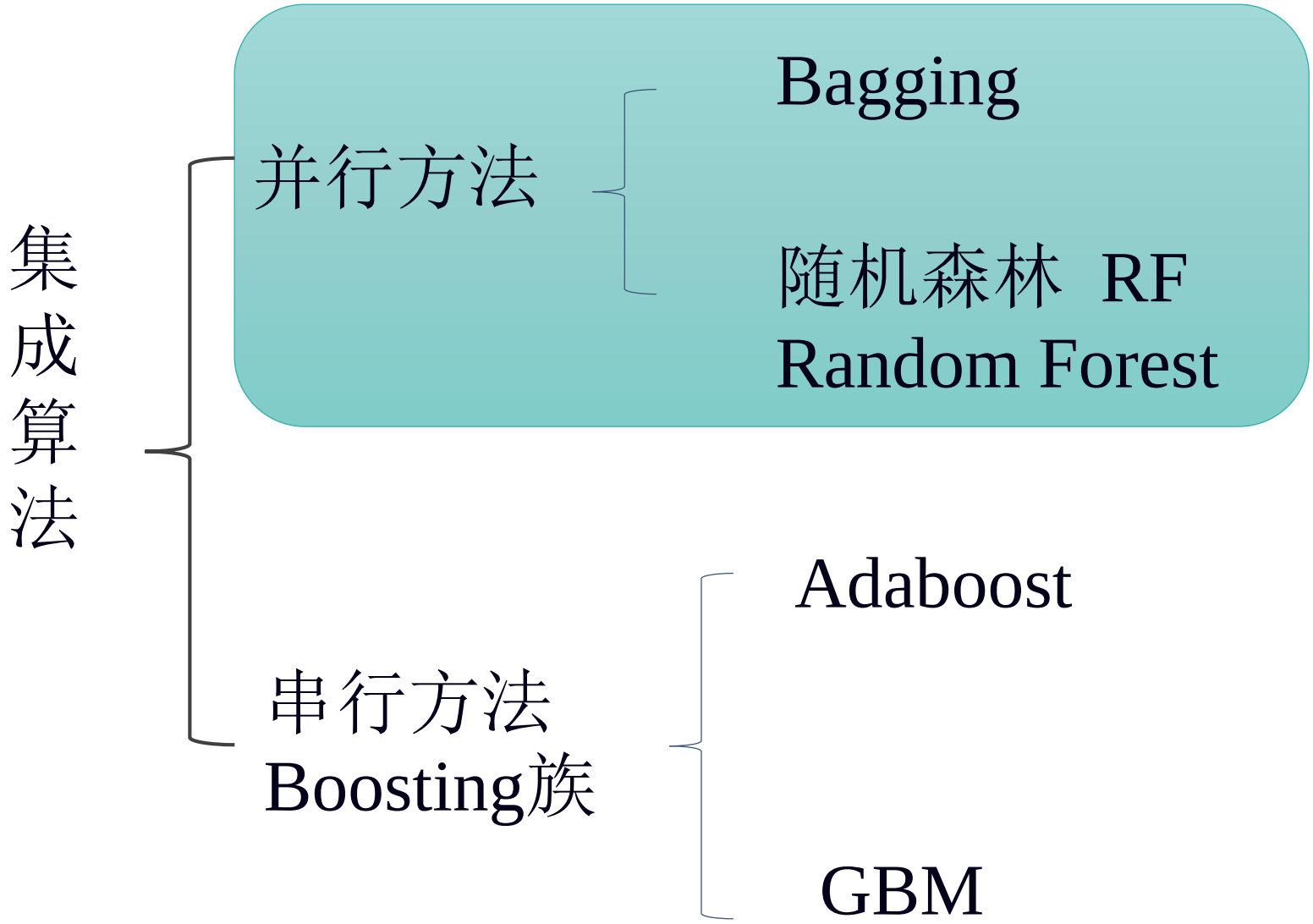
1. 集成算法的基本概念

2. Bagging和随机森林

4. 随机森林的实现

5. 随机森林效果验证

《集成算法之随机森林》


集成算法
并行方法
Bagging
随机森林 RF
Random Forest
串行方法
Boosting族
Adaboost
GBM

1 集成算法

集成思想：对于比较复杂的任务，综合多人的意见来进行决策会比“一家独大”更好。也就是通过适当的方式集成许多“个体模型”所得到的最终模型要不单独的“个体模型”

1. 如何选择、生成“个体模型”

弱学习模型：决策树，决策树桩

2. 如何综合多个模型：

将相同类型但**训练集不同**的弱分类器进行提升

将相同类型但**权重不同**的弱分类器进行提升

将**不同类型**的弱分类器进行提升

1.1集成算法

集成框架的两种模式：

第一种：期望各个分类器之间依赖性不强，可以同时进行生成。这种做法称为**并行方法**，代表为Bagging，适用性更强的拓展便是随机森林。

并行集成方法的重点：如何从总体训练集获得多个不同的子训练集

第二种：弱学习器之间具有强依赖性，只能序列生产，称为**串行方法**。代表方法是Boosting, 包括Adaboost, GBDT, XGBoost和lightGBM

串行集成方法的重点：如何更新弱分类器的权重(弱分类器的整体“话语权”，以及某一样本的识别效果)

2 Bagging和随机森林

Bagging是于1996年Breiman提出的，它的思想根源是数理统计学中非常重要的Bootstrap理论。

Bootstrap:“自举”（“自采样”），通过模拟的方法来逼近样本的概率分布。

子样本之与样本，
可以类比于样本之于总体

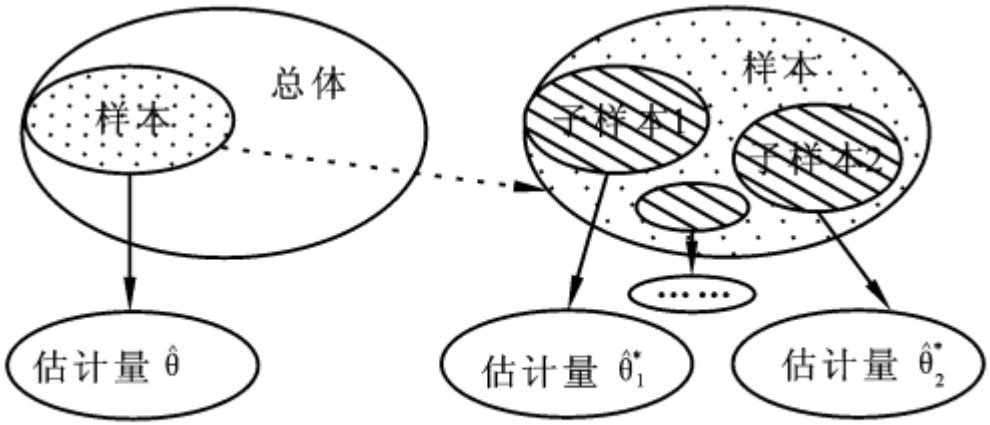


图 1　重抽样方法的思想示意图

2.1 Bootstrap方法

如何统计池塘的鱼的数量

1. 承包鱼塘，不让别人捞鱼(规定总体分布不变)。
2. 自己捞鱼，捞100条，都打上标签
3. 把鱼放回鱼塘，休息一晚(使之混入整个鱼群，确保之后抽样随机)
4. 开始捞鱼，每次捞100条，数一下，自己昨天标记的鱼有多少条，占比多少(一次重采样取分布)。
5. 重复3，4步骤n次，然后对分布结果进行(求均值，方差)

2.1 Bootstrap方法

1. 假设，第一次捕鱼100条，发现里面有标记的鱼12条，记下为12%，放回去；
2. 再捕鱼100条(假设不放回的捕100条鱼不会改变总体分布)，发现标记的为9条，记下9%，放回去；
3. 重复重复好多次之后，你会发现，每次捕鱼平均在10条左右有标记，所以，我们可以大致推测出鱼塘有1000条左右。

现在的要求：重采样只能进行一次(也就是现在的训练样本集只有一个)，如何能够估计总体数量；并且使预测结果尽可能准确

2.1 Bootstrap方法

Bootstrap就针对这种情况提出了一个解决方案，通过不断地“自采样”来模拟真实分布成的数据集。具体而言，Bootstrap的做法是：

假设样本集 $X=[x_1,x_2,\cdots,x_N]^T$

从 X 中随机抽取一个样本(即 $x_1,\cdots,x_N$ 的概率相同)

将该样本拷贝放入数据集 X_j

将该样本放回 X 中

以上三个步骤重复N 次，从而使得X_j中有N 个样本

以上四个步骤重复M 次，生成 $X_1,X_2,\cdots,X_M$

简单的说就是有放回的抽样。

2.1 Bagging方法与Bootstrap

Bootstrap是一个有放回的随机抽样过程，所以原始数据中的样本可能重复出现，也有可能不出现在其中一个样本集中。事实上，在N次采样中，始终不被采到的概率为

$$\lim_{n\to\infty}(1-\frac{1}{N})^N \to \frac{1}{e} \approx 0.368$$

在统计意义上可以认为，X_j中含义有X中63.2%的样本。

每个子样本集中未出现的样本称为袋外数据

这种模拟方法的本质和经验分布函数对真实分布函数的模拟几乎是一致的。

2.2 bootstrap的实现

通过有放回的抽样获得子样本集：sub_data函数

输入：具有N个样本的样本集 data_set

输出：子样本集 sub_samples

当sub_samples中样本少于N时：

从data_set中**随机选择一个样本**

将该样本放入子样本集sub_samples中

返回 sub_samples

关键：如何随机抽取一个样本：

生成一个[0 ,N)的随机整数index, (random.randrange函数)

选取样本集中第index个样本

2.2 Bootstrap的实现

Bootstrap有放回的抽样，其中一个子样本集

```
from random import randrange
def sub_data(data_set):
        N=len(data_set)
        sub_samples=list()
        for i in range(N):
                index=randrange(N)
                sub_samples.append(data_set[index])
        return sub_samples
```

2.3 Bagging方法

Bagging方法：全称Bootstrap Aggregating

用Bootstrap生成出M个数据集

用这M个数据集训练出M个弱分类器

最终模型即为这M个弱分类器的简单组合

所谓简单组合就是：

对于分类问题使用投票表决的方法

对于回归问题使用简单的取平均

2.3 Bagging算法实现

from CART import *

调用CART.py 决策树创建bagging函数

输入：样本集data_set，树个数n_trees=5

输出：多棵树组成的列表trees

初始化一棵CART树 tree=CARTClassifier()

当树的数量小于n_trees时：

利用bootstrap自动生成样本集sub_samples

使用数据集sub_samples训练模型tree

将tree放入trees中

返回 trees

2.3 Bagging算法预测

使用trees和一个测试样本的特征，预测该样本的类别

bagging_predict；**用每棵树对sample进行预测，并将结果放在名为b_p的列表中，b_p中出现次数最多的类别即为bagging方法预测的类别**

输入：bagging所得trees和样本sample

输出：该sample的预测类别

创建预测类别b_p=list()

对trees中的每一个树：

使用CART中的predict函数预测在该树中sample的类别，并将预测到的类别添加到b_p中

返回b_p中出现次数最多的类别

3 随机森林

随机森林：特殊的Bagging算法

1. 随机森林算法不仅对样本集进行Bootstrap采样
2. 每次对node进行划分时(每个决策节点)，都从d个特征中随机挑选k个，然后依信息增益从这个k个特征中选出最好的特征并确定划分标准
3. 每棵树都不进行后剪枝
4. 最终随机森林的预测结果即为所有树模型预测结果的的简单组合(分类：投票决定)

如果k=d, 那随机森林和Bagging算法相同

如果k=1, 那就是完全随机

一般k=d、k=log2(d)或者k=sqrt(d)

3.1 随机森林的算法实现

修改CART的决策点生成函数：

get_split函数：(CART决策节点)
输入：data ，n_features
具有两层循环

从特征索引列表中随机抽取n_features个

外层是对特征索引进行循环
内层是对该特征的二分标准进行循环
对样本集进行划分，并求出基尼系数
通过比较得到最优
二分标准的选取有三种常用的方法，此处采用最简单的方法将特征空间的每个值作为二分标准

3.1 随机森林的算法实现

修改CART的决策点生成函数：

CART函数的初始化中添加n_features默认为None

决策点生成函数添加n_features

如果初始化为None，那么建树函数中跟据样本的特征长度添加n_features

n_features 接收int 'log2' 或者 'sqrt'

修改bagging函数变成randomForest函数：

Bagging函数添加n_features参数：

初始化CARTClassifier()类时添加n_features参数

或者调用build_tree()函数时添加n_features参数

3.1 随机森林的算法实现

修改CART的决策点生成函数：

CART函数的初始化中添加n_features默认为None

决策点生成函数添加n_features

如果初始化为None，那么建树函数中跟据样本的特征长度添加n_features

n_features 接收int 'log2' 或者 'sqrt'

修改bagging函数变成randomForest函数：

Bagging函数添加n_features参数：

初始化CARTClassifier()类时添加n_features参数

或者调用build_tree()函数时添加n_features参数