R 机器学习

第 11 讲　随机森林

张敬信

2022 年 10 月 26 日

哈尔滨商业大学

一. 集成学习

集成学习，是通过构建多个基学习器，并按一定策略结合成强学习器来完成学习任务，即所谓“博采众长”，最终效果是优于任何一个原学习器。集成学习可用于分类/回归集成、特征选择集成、异常值检测集成等。

这多个基学习器可以是同质的，比如都用决策树或都用神经网络，以 Bagging 和 Boosting 模式为代表；也可以是异质的，即采用不同的算法，以 Stacking 模式为代表，mlr3pipelines 包提供了方便的实现。

近年来，Kaggle 等数据挖掘竞赛排名靠前的基本都用到了集成学习。

装袋法 (Bagging)

Bagging 采用的是并行机制，即基学习器的训练之间没有前后顺序可以同时进行。

Bagging 是用 “有放回” 抽样（Bootstrap 法）的方式抽取训练集，对于包含 m 个观测的训练集，进行 m 次有放回的随机抽样操作，得到数据副本（有重复）中有接近 36.8% 的样本没有被抽到。

按照同样的方式重复进行，就可以采集到 T 个包含 m 个观测的数据副本，从而训练出 T 个基学习器。

最终对这 T 个基学习器的输出进行结合，分类问题就采用 “多数决”，回归问题就采用 “取平均”。

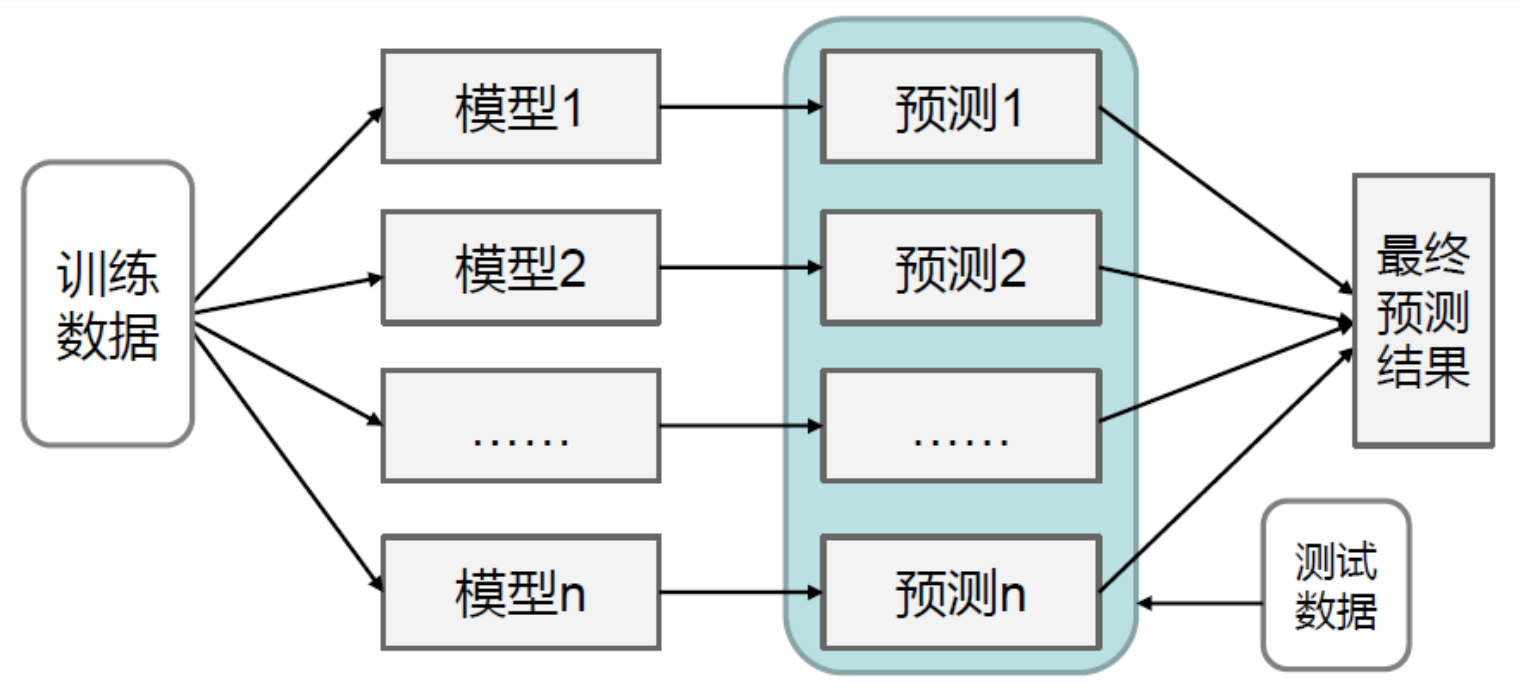


图 1: Bagging 法示意图

Bagging 主要通过样本的扰动来增加基学习器之间的多样性，因此，Bagging 的基学习器应为那些对训练集十分敏感的不稳定学习算法，如神经网络与决策树等。

从偏差-方差分解来看，Bagging 法主要关注于降低方差，即通过多次重复训练提高稳定性。

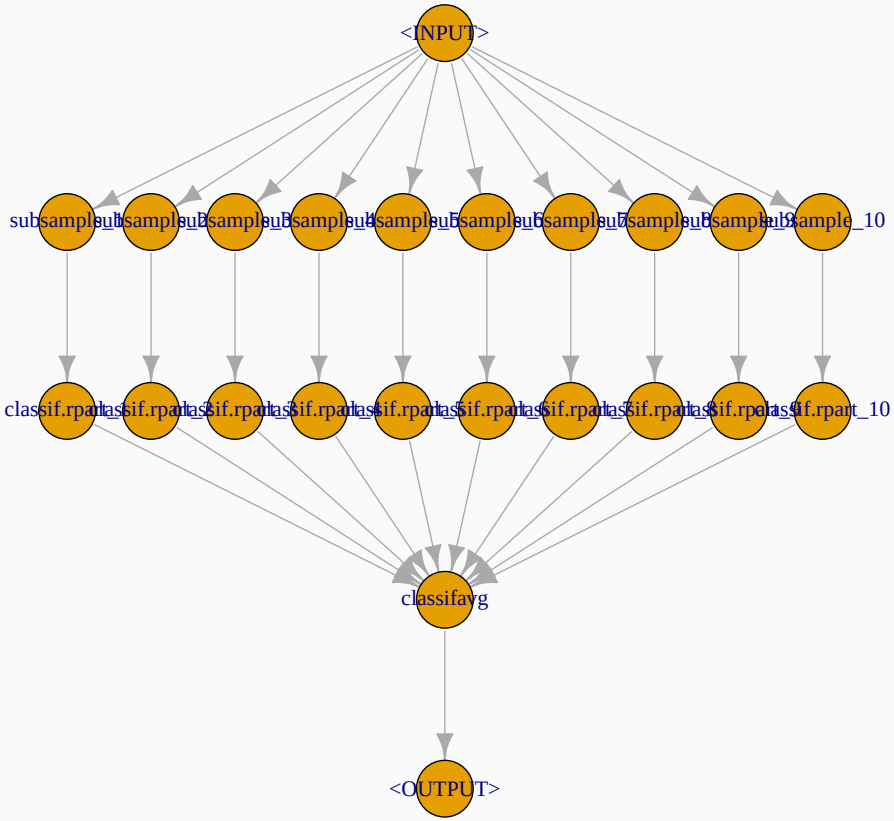
Bagging 法的代表算法是随机森林。

- mlr3pipelines 管道，可以很方便地实现装袋法：

```
library(mlr3verse)
# 单分支：数据子抽样 + 决策树
single_pred = po("subsample", frac = 0.7) %>>%
  po("learner", lrn("classif.rpart"))
# 复制 10 次得到 10 个分支，并装袋
bagging = ppl("greplicate", single_pred, 10L) %>>%
  po("classifavg", innum = 10)
```

- 可视化结构关系：

```
bagging$plot()
```



- 转化为图学习器，即可与普通学习器一样使用：

```
baglrn = as_learner(bagging)
baglrn$train(tsk("iris"))
```

提升法 (Boosting)

Boosting 采用的是串行机制，即基学习器的训练存在依赖关系，按次序一一进行训练（实现上可以做到并行)。

基本思想：基模型的训练集按照某种策略每次都进行一定的转化，对所有基模型预测的结果进行线性合成产生最终的预测结果。

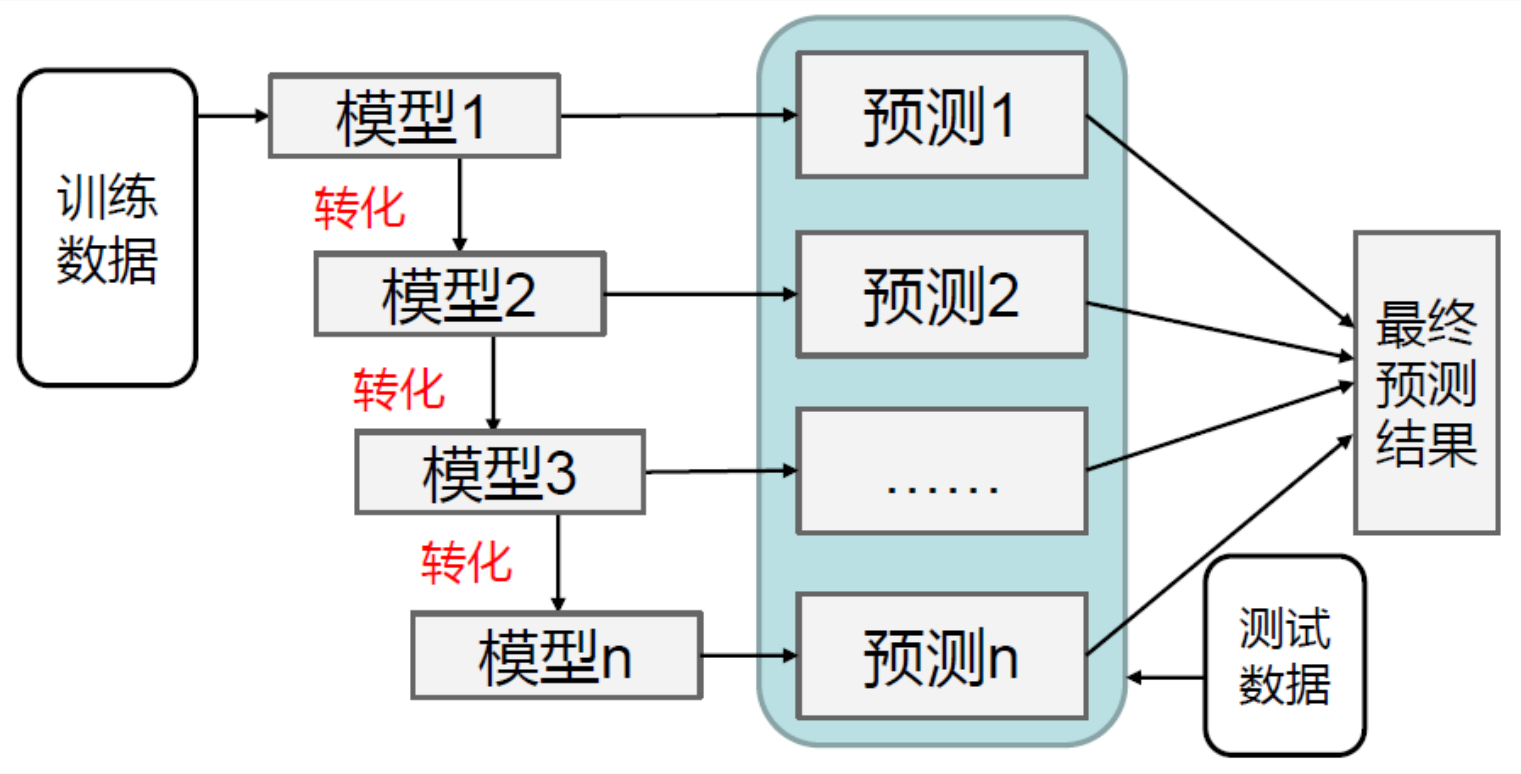


图 2: Boosting 法示意图

从偏差-方差分解来看，Boosting 算法主要关注于降低偏差，每轮的迭代都关注于训练过程中预测错误的样本，将弱学习器提升为强学习器。

Boosting 法是一些现成的机器学习算法为代表，如 AdaBoost, GBDT, XGboost, LightGBM, catBoost 等。

堆叠法 (Stacking)

Stacking 法，采用的是分阶段机制，将若干基模型的输出作为输入，再接一层主学习器，得到最终的预测。

将训练好的所有基模型对训练集进行预测，第 j 个基模型对第 i 个训练样本的预测值将作为新的训练集中第 i 个样本的第 j 个特征值，最后基于新的训练集进行训练。同理，预测的过程也要先经过所有基模型的预测形成新的测试集，最后再对测试集进行预测。

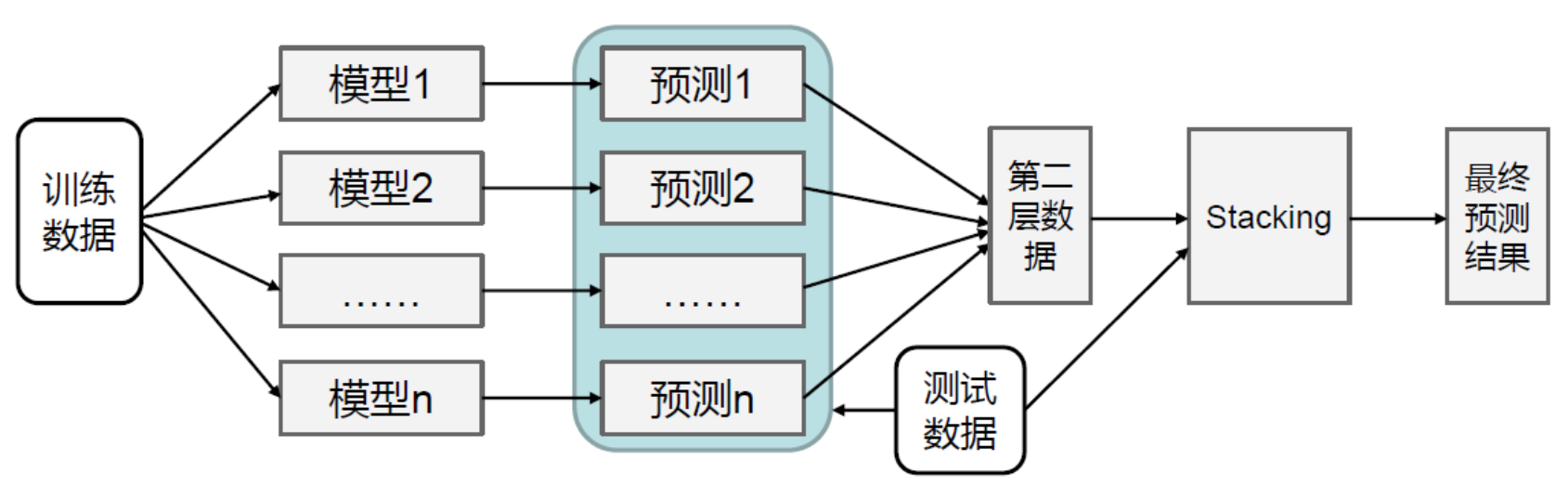


图 3: Stacking 法示意图

用于堆叠的基模型通常采用不同的模型，作用在相同的训练集上。

比如分类问题，可以选择决策树分类器、Logistic 回归和 SVM 作为基模型，并将 KNN 作为主学习器。然后，KNN 将会把三个基模型的输出作为输入，并返回基于该输入的最终预测。

为了充分利用数据，Stacking 通常采用 k 折交叉训练法 (类似 k 折交叉验证)：每个基学习器分别在各个 k-1 折数据上训练，在其剩下的 1 折数据上预测，就可以得到对任意 1 折数据的预测结果，进而用于训练元模型。

mlr3pipelines 管道，可以很方便地实现堆叠法。

例如，训练一个决策树并将其预测结果特征与原始特征结合起来，再在其上训练一个主模型 KNN。

为了防止过拟合，不是用基学习器对原数据整体上的进行预测，而是交叉对折外数据进行预测。要做到这一点，可以用 PipeOpLearnerCV。

PipeOpLearnerCV 对训练数据进行嵌套交叉训练，对每折拟合一个模型。然后，每个模型都用来预测其折外的数据。因此，获得了对输入数据中所有数据点的预测。

首先创建一个”0 级” 学习器，用来提取较低级别的预测特征。此外，clone() 学习器对象，以获得学习器的副本：

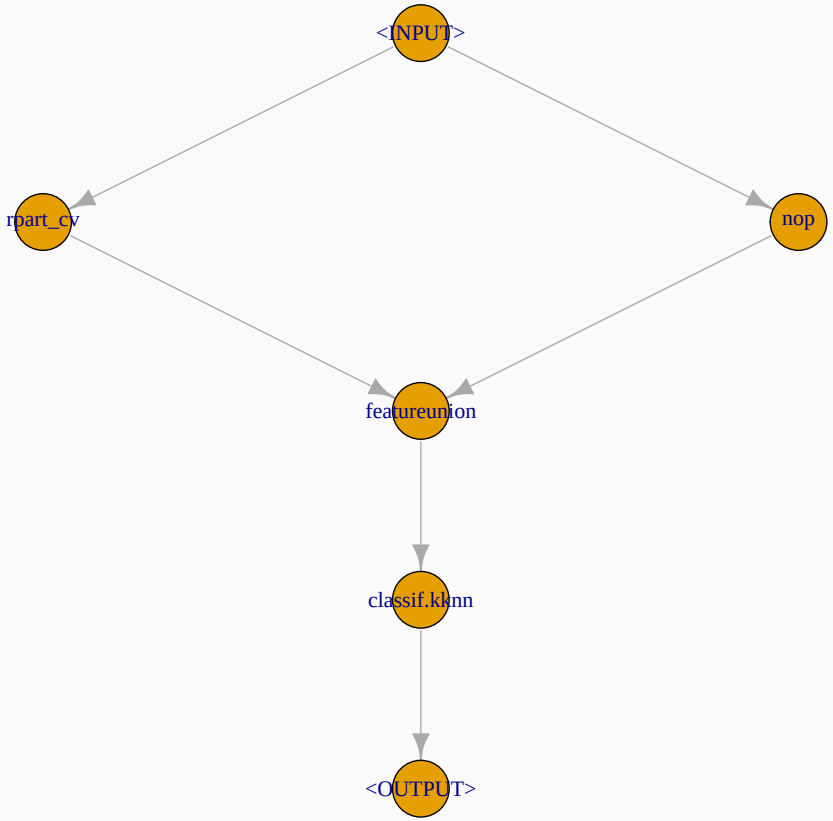
```
lrn = lrn("classif.rpart")
lrn_0 = po("learner_cv", lrn$clone())
lrn_0$id = "rpart_cv"      # 为 PipeOp 设置 ID
```

- 原始特征随未改变的任务通过 nop 管道发送到下一级，与决策树学习器的预测结果特征通过 featureunion 管道相结合，之后再接上主学习器 KNN:

```
stack = gunion(list(lrn_0, po("nop"))) %>>%
  po("featureunion", 2) %>>%
  po("learner", lrn("classif.kknn"))
```

- 可视化结构关系：

```
stack$plot()
```

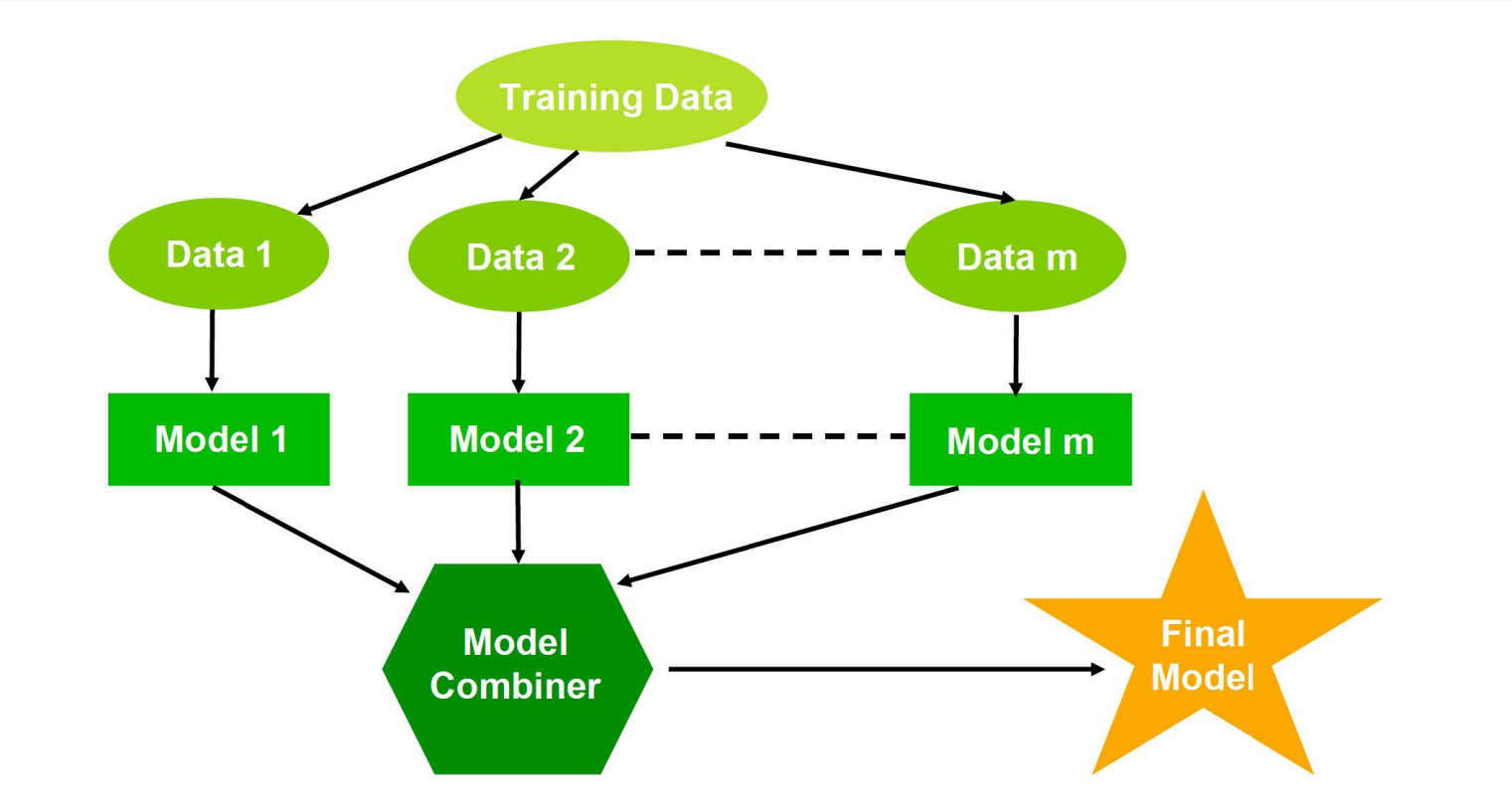


转化为图学习器，即可与普通学习器一样使用：

```
stacklrn = as_learner(stack)
stacklrn$train(tsk("iris"))
#> INFO  [20:35:36.399] [mlr3] Applying learner 'classif.rpar
#> INFO  [20:35:36.566] [mlr3] Applying learner 'classif.rpar
#> INFO  [20:35:36.593] [mlr3] Applying learner 'classif.rpar
```

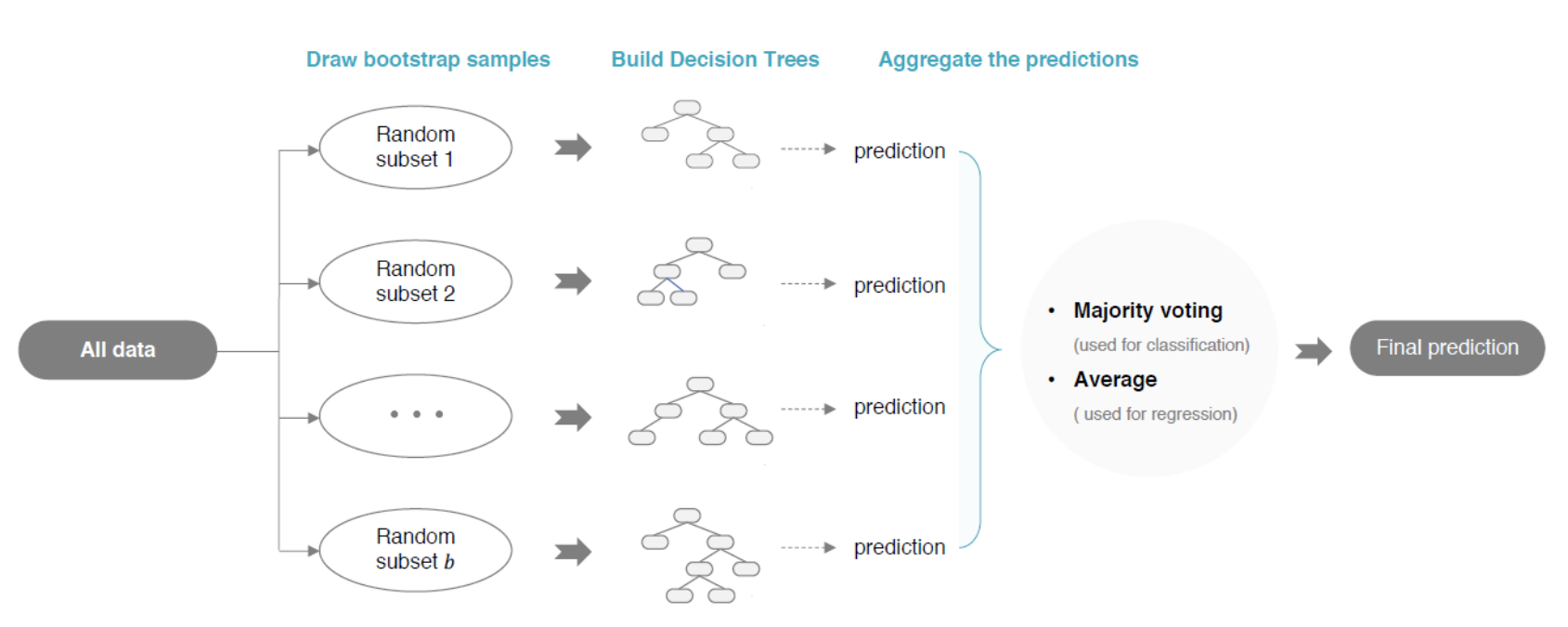
二. 随机森林原理

- 决策树是弱分类器，效果可以说很一般，但将多个弱分类器通过**集成学习**技术，即将训练好的若干学习器组合在一起，就可以实现强分类器



- 简单来说，随机森林就是将决策树按装袋法集成学习：
  - 先通过对原数据集做自助抽样（可重复）创建若干随机的数据子集
  - 对每个数据子集分别训练 CART 树
  - 将多个 CART 学习器的结果”取平均”作为最终预测。

- 随机森林算法流程示意图：



- “装袋法” 集成学习可以有效地降低模型的方差，但简单的” 装袋” 会造成” 树” 的相关性，影响降低模型方差的效果。随机森林，通过在决策树生长过程引入更多的随机性，以降低这种相关性。
- 具体来说，是在” 装袋” 生长决策树过程中，在每次决定如何切分时都只随机选择部分特征，p 个特征中的 m_{try} 个特征，一般取 $m_{try} = p/3$ (回归) 与 $m_{try} = \sqrt{p}$ (分类)，当然更好的方法是做超参数调参。
- 随机森林由于随机从训练集自助重抽样，每次切分都随机选取特征，从而会生长出更加多样的森林，这就比简单” 装袋法” 更能降低树的相关性，极大地改进模型的预测性能。

注： 若取 $m_{try} = p$，则为装袋决策森林算法。

随机森林回归/分类步骤

- 给定训练集，选择要创建的树的数量（n_trees）

for i=1 to n_trees do

生成一个原始数据的自助重抽样

根据自助重抽样数据，生长一棵回归/分类树

`for each split do`

从p个变量中随机选择m_try个变量

在m_try个变量中选择最优切分变量和切分点

将节点切分为两个子节点

`end`

根据通常的决策树停止生长准则(不做剪枝)确定何时完成创建树

end

- 这些树的预测结果做集成

- 随机森林的优点：
    - 不容易过拟合，无需做剪枝
    - 可以并行计算，单个决策树可同时训练
    - 既能做分类也能做回归，无需太多调参，就能获得很高的分类精度
    - 同样对缺失值、异常值不敏感
    - 可以处理很多变量，无需变量约减
- 随机森林的缺点：
    - 树的数量越多，性能越高，预测越稳定，但计算速度会越慢
    - 更擅长做分类，做回归差一些，因为它不能给出精确的连续预测，且预测范围只能在训练数据的范围以内
    - 属于“黑箱”模型，结果不容易解释

二. 随机森林案例

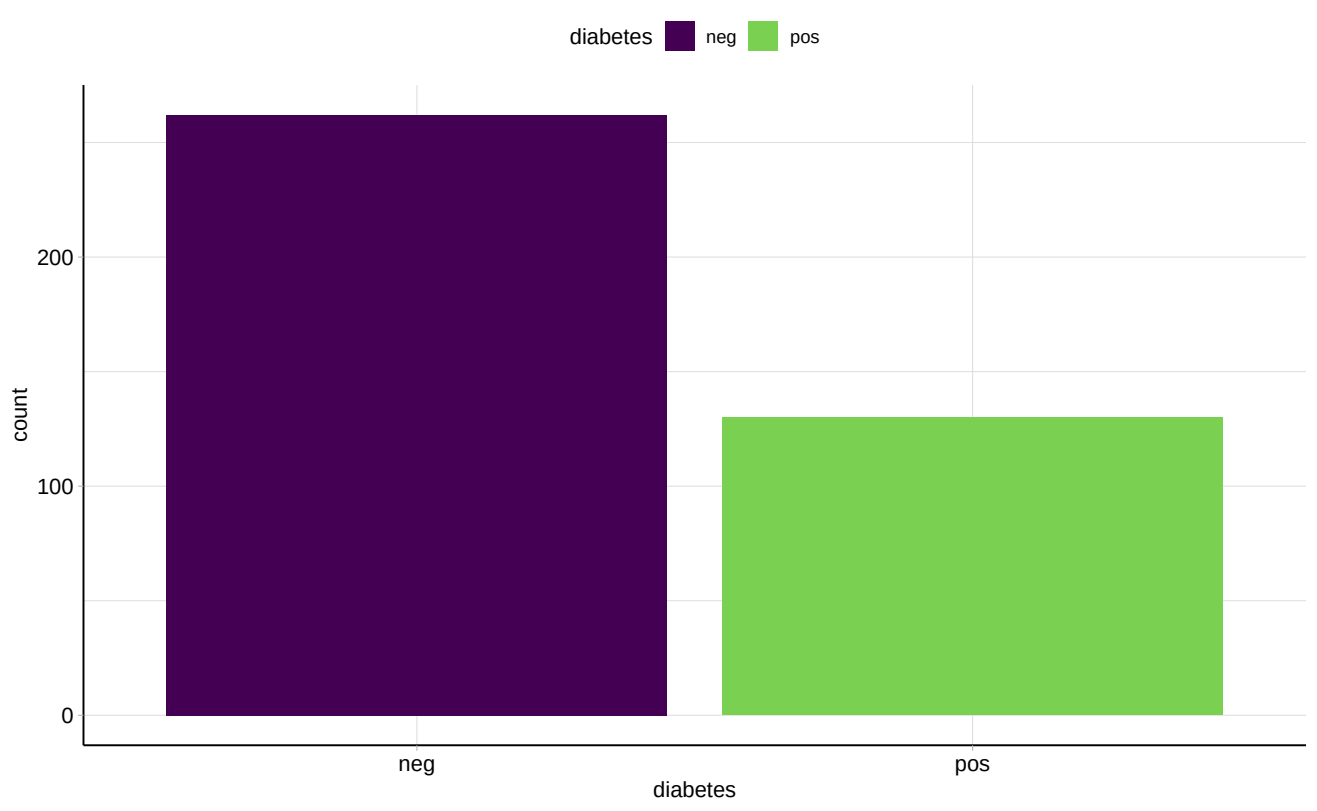
- mlr3 实现随机森林是调用 ranger 包（要求数据不能有 NA）
- 使用来自 Kaggle 的印度糖尿病数据集 pima，包含 768 个样本、9 个变量，因变量 diabetes 是二分类变量（是否患病），特征包括患者怀孕次数，BMI，胰岛素水平，年龄等。

1. 创建任务

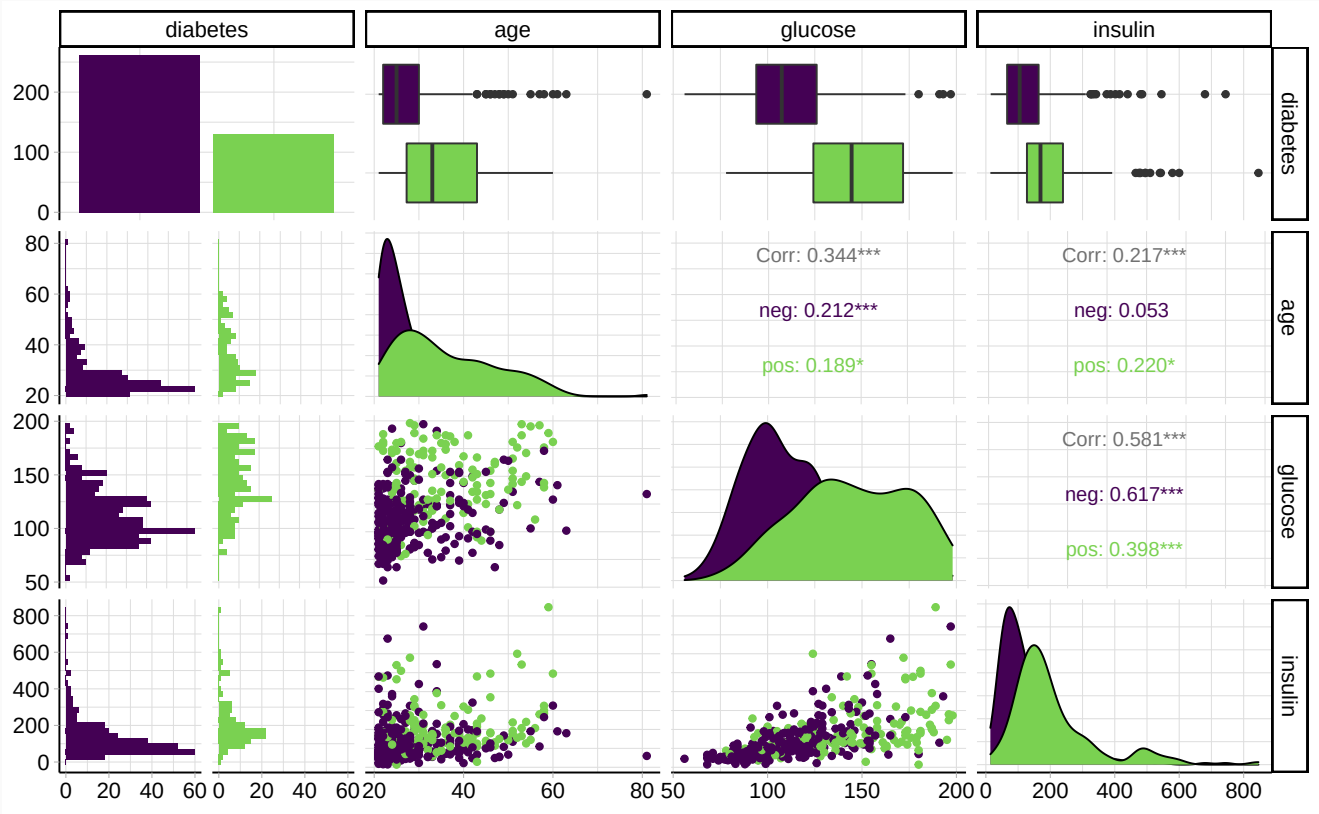
```
library(mlr3verse)
library(tidyverse)
load("datas/pima.rda")
dat = na.omit(dat)
```

```r
task = as_task_classif(dat, target = "diabetes")
task
#> <TaskClassif:dat> (392 x 9)
#> * Target: diabetes
#> * Properties: twoclass
#> * Features (8):
#>   - dbl (8): age, glucose, insulin, mass, pedigree, pregna
#>     triceps
```

```
autoplot(task)
```



```
autoplot(task$clone()$select(task$feature_names[1:3]),
         type = "pairs")          # 需要 GGally 包
```



2. 选择学习器

```
Ranger = lrn("classif.ranger")   # 需要安装 ranger 包
Ranger                           # 不能处理 NA
#> <LearnerClassifRanger:classif.ranger>
#> * Model: -
#> * Parameters: num.threads=1
#> * Packages: mlr3, mlr3learners, ranger
#> * Predict Types:  [response], prob
#> * Feature Types: logical, integer, numeric, character, fac
#> * Properties: hotstart_backward, importance, multiclass, o
#>   twoclass, weights
```

3. 划分训练集和测试集

- 做留出 (holdout) 重抽样，80% 作为训练集，其余 20% 作为测试集
- 为了保持训练集、测试集的因变量数据具有相似的分布，采用分层抽样方法
- 用 `partition()` 函数对任务做划分，默认按因变量分层，取出训练集索引和测试集索引

```
set.seed(123)
split = partition(task, ratio = 0.8)
# 默认 stratify = TRUE
```

4. 超参数调参

```
Ranger$param_set          # 查看所有超参数及默认值
#> <ParamSet>
#>                                  id    class lower upper nlev
#>  1:                           alpha ParamDbl  -Inf   Inf
#>  2:       always.split.variables ParamUty    NA    NA
#>  3:                 class.weights ParamUty    NA    NA
#>  4:                       holdout ParamLgl    NA    NA
#>  5:                    importance ParamFct    NA    NA
#>  6:                    keep.inbag ParamLgl    NA    NA
#>  7:                     max.depth ParamInt     0   Inf
#>  8:                min.node.size ParamInt     1   Inf
#>  9:                     min.prop ParamDbl  -Inf   Inf
#> 10:                      minprop ParamDbl  -Inf   Inf
#> 11:                         mtry ParamInt     1   Inf
#> 12:                   mtry.ratio ParamDbl     0     1
```

- 对模型中超参数：树的数量 `num.trees`(做变换到以 `20` 为间隔), 最小节点数 `min.node.size` 做调参
- 使用自动调参器，需要设置学习器、重抽样方法、模型评估指标、搜索空间、终止条件、搜索方法

```
library(paradox)
search_space = ps(
  num.trees = p_int(lower = 1, upper = 20,
                    trafo = function(x) 20 * x),
  min.node.size = p_int(lower = 3, upper = 30))

at = auto_tuner(
  learner = Ranger,
  resampling = rsmp("cv", folds = 3),
  measure = msr("classif.acc"),
  search_space = search_space,
  method = "random_search",
  term_evals = 10)
```

- 在训练集上启动调参过程

```
set.seed(615)
at$train(task, row_ids = split$train)
#> INFO  [20:35:40.140] [bbotk] Starting to optimize 2 parame
#> INFO  [20:35:40.164] [bbotk] Evaluating 1 configuration(s)
#> INFO  [20:35:40.189] [mlr3] Running benchmark with 3 resam
#> INFO  [20:35:40.195] [mlr3] Applying learner 'classif.rang
#> INFO  [20:35:40.246] [mlr3] Applying learner 'classif.rang
#> INFO  [20:35:40.290] [mlr3] Applying learner 'classif.rang
#> INFO  [20:35:40.334] [mlr3] Finished benchmark
#> INFO  [20:35:40.364] [bbotk] Result of batch 1:
#> INFO  [20:35:40.366] [bbotk]  num.trees min.node.size clas
#> INFO  [20:35:40.366] [bbotk]         16            23
#> INFO  [20:35:40.366] [bbotk]
#> INFO  [20:35:40.366] [bbotk]  9b7a625f-2a59-4bb7-9903-2799
#> INFO  [20:35:40.369] [bbotk] Evaluating 1 configuration(s)
#> INFO  [20:35:40.398] [mlr3] Running benchmark with 3 resam
```

- 查看最优超参数

```
at$tuning_result                  # 调参结果
#>    num.trees min.node.size learner_param_vals  x_domain cl
#> 1:         1            14          <list[3]> <list[2]>
```

5. 训练模型

- 用调出的最优参数更新学习器的参数集，然后训练模型

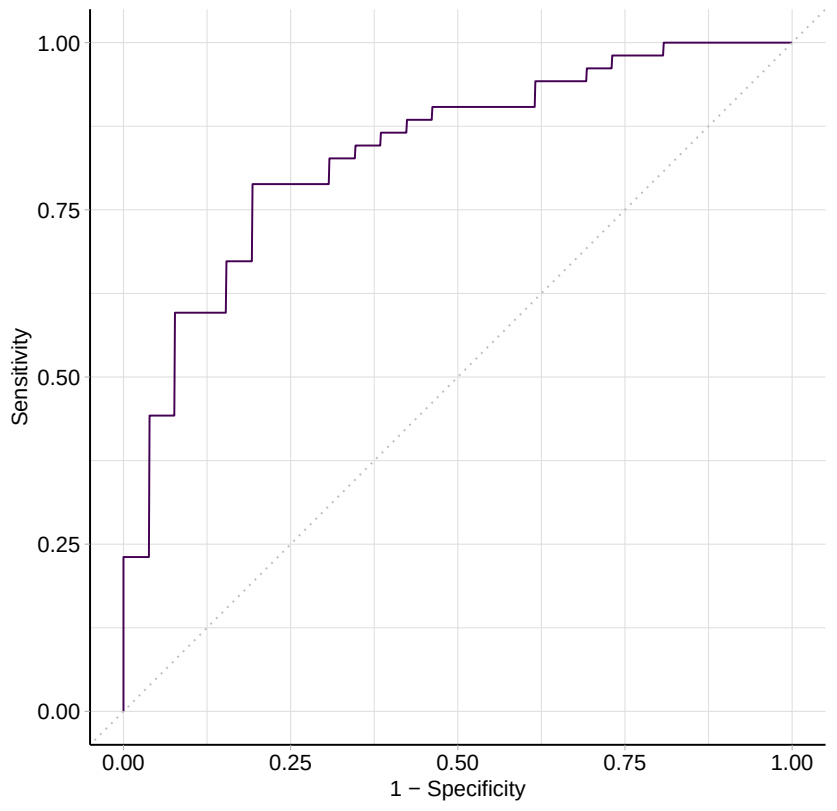
```
Ranger$param_set$values = at$tuning_result$learner_param_vals
Ranger$train(task, row_ids = split$train)
```

```
predictions = Ranger$predict(task, row_ids = split$test)
predictions
#> <PredictionClassif> for 78 observations:
#>     row_ids truth response
#>           8   neg      neg
#>          10   neg      pos
#>          25   neg      pos
#> ---
#>         369   pos      pos
#>         375   pos      neg
#>         376   pos      pos
```

```
predictions$confusion                          # 混淆矩阵
#>          truth
#> response neg pos
#>      neg  43   8
#>      pos   9  18
predictions$score(msr("classif.acc"))   # 预测准确率
#> classif.acc
#>       0.782
```

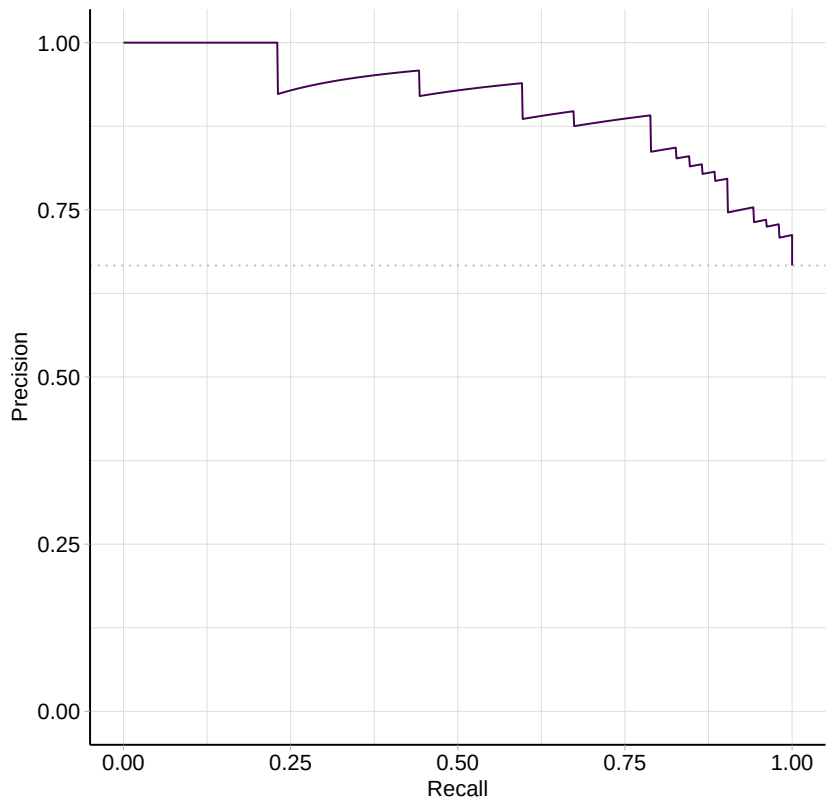
```
Ranger$predict_type = "prob"
Ranger$train(task, row_ids = split$train)
predictions = Ranger$predict(task, row_ids = split$test)
predictions
#> <PredictionClassif> for 78 observations:
#>     row_ids truth response prob.neg prob.pos
#>           8   neg      neg    0.829    0.171
#>          10   neg      neg    0.671    0.329
#>          25   neg      pos    0.224    0.776
#> ---
#>         369   pos      pos    0.106    0.894
#>         375   pos      neg    0.911    0.089
#>         376   pos      pos    0.432    0.568
```

```r
autoplot(predictions, type = "roc")      # ROC 曲线
```



```r
predictions$score(msr("classif.auc"))    # AUC 値
#> classif.auc
#>       0.837
```

```
autoplot(predictions, type = "prc")      # PR 曲线
```



7. 预测新数据

```
newdata = dat[1:5,-1]
Ranger$predict_newdata(newdata)
#> <PredictionClassif> for 5 observations:
#>  row_ids truth response prob.neg prob.pos
#>        1  <NA>      neg    0.973   0.0266
#>        2  <NA>      pos    0.136   0.8638
#>        3  <NA>      neg    0.644   0.3557
#>        4  <NA>      pos    0.190   0.8097
#>        5  <NA>      pos    0.167   0.8331
```

主要参考文献

[1] mlr3book. 2021. https://mlr3book.mlr-org.com/

[2] Jim Liang(梁劲). Getting Started with Machine Learning, 2019.

[3] 周志华. 机器学习 (西瓜书), 清华大学出版社, 2016.

[4] 李航. 统计学习方法 (第二版). 清华大学出版社, 2019.

[5] 黄海广. 机器学习课件全集 (v1). 温州大学, 2021.

[6] 刘顺祥. 从零开始学 Python: 数据分析与挖掘. 清华大学出版社, 2018.

[7] 刘建平. 集成学习原理小节, 博客园