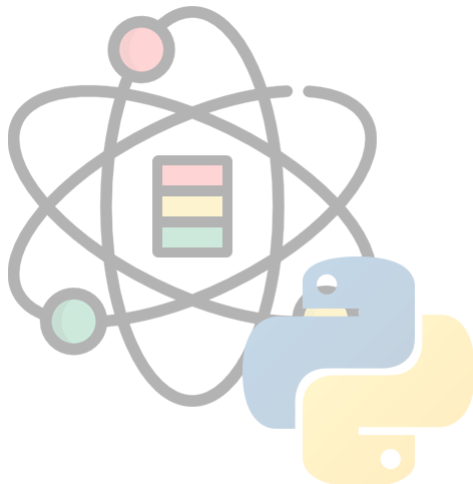
# Python 数据科学导论
# Data Science Introduction with Python

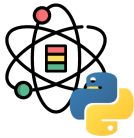
## 分类算法 (下)
## Classification Algorithms - Part 2

范叶亮



# 目录

# 集成学习

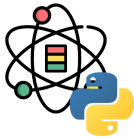


传统的机器学习算法（例如：决策树，人工神经网络，支持向量机，朴素贝叶斯等）的目标都是寻找一个最优的分类器尽可能的将训练数据分开。集成学习（Ensemble Learning）算法的基本思想就是通过将多个分类器组合，从而实现一个预测效果更好的集成分类器。集成算法可以说从一方面验证了中国的一句老话：三个臭皮匠，赛过诸葛亮。Thomas G. Dietterich [1, 2] 指出了集成算法在统计，计算和表示上的有效原因：

**统计上的原因**：一个学习算法可以理解为在一个假设空间 $\mathcal{H}$ 中选找到一个最好的假设。但是，当训练样本的数据量小到不够用来精确的学习到目标假设时，学习算法可以找到很多满足训练样本的分类器。所以，学习算法选择任何一个分类器都会面临一定错误分类的风险，因此将多个假设集成起来可以降低选择错误分类器的风险。

[1] Dietterich, Thomas G. "Ensemble methods in machine learning." *International workshop on multiple classifier systems*. Springer, Berlin, Heidelberg, 2000.
[2] Dietterich, Thomas G. "Ensemble learning." *The handbook of brain theory and neural networks* 2 (2002): 110-125.

# 集成学习



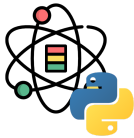
**计算上的原因**：很多学习算法在进行最优化搜索时很有可能陷入局部最优的错误中，因此对于学习算法而言很难得到一个全局最优的假设。事实上人工神经网络和决策树已经被证实为是一个 NP 问题 [1, 2]。集成算法可以从多个起始点进行局部搜索，从而分散陷入局部最优的风险。

**表示上的原因**：在多数应用场景中，假设空间 $\mathcal{H}$ 中的任意一个假设都无法表示（或近似表示）真正的分类函数 $f$。因此，对于不同的假设条件，通过加权的形式可以扩大假设空间，从而学习算法可以在一个无法表示或近似表示真正分类函数 $f$ 的假设空间中找到一个逼近函数 $f$ 的近似值。

[1] Laurent, Hyafil, and Ronald L. Rivest. "Constructing optimal binary decision trees is NP-complete." *Information processing letters* 5.1 (1976): 15-17.
[2] Blum, Avrim L., and Ronald L. Rivest. "Training a 3-node neural network is NP-complete." *Neural Networks* 5.1 (1992): 117-127.

# Bagging

# Bagging



Bagging 是由 Breiman 于 1996 年提出 [1]，基本思想如下：

1. 每次采用有放回的抽样从训练集中取出 $n$ 个训练样本组成新训练集。
2. 利用新的训练集，训练得到 $M$ 个子模型 $\{h_1, h_2, \ldots, h_M\}$。
3. 对于分类问题，采用投票的方法，得票最多子模型的分类类别为最终的类别；对于回归问题，采用简单的平均方法得到预测值。

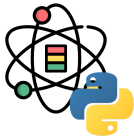
**Algorithm 1** Bagging 算法

**Require:** 学习算法，子模型个数 $M$，训练数据集 $T$
**Ensure:** Bagging 算法 $h_f(x)$

**procedure** BAGGING$(, M, T)$
  **for** $m = 1$ to $M$ **do**
    $T_m$ bootstrap sample from training set $T$
    $h_m$ $(T_m)$
  **end for**
  $h_f(x)$ $\operatorname{argmax}_{y\in} \sum_m^M h_i(x)$
  **return** $h_f(x)$
**end procedure**

[1] Breiman, Leo. "Bagging predictors." *Machine learning* 24.2 (1996): 123-140.

# Bagging

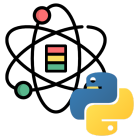


假设对于一个包含 $M$ 个样本的数据集 $T$，利用自助采样，则一个样本始终不被采用的概率是 $\left(1-\frac{1}{M}\right)^M$，取极限有：

$$\lim_{x\to\infty}\left(1-\frac{1}{M}\right)^M=\frac{1}{e}\approx 0.368 \tag{1}$$

即每个学习器仅用到了训练集中 $63.2\%$ 的数据集，剩余的 $36.8\%$ 的训练集样本可以用作验证集对于学习器的泛化能力进行包外估计（out-of-bag estimate）。

# 随机森林



随机森林（Random Forests）[1] 是一种以决策树为基学习器的 Bagging 集成学习算法。随机森林模型的构建过程如下：

- 数据采样

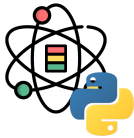
作为一种 Bagging 的集成算法，随机森林同样采用有放回的采样，对于总体训练集 $T$，抽样一个子集 $T_{sub}$ 最为训练样本集。除此之外，假设训练集的特征个数为 $d$，每次仅选择 $k\,(k<d)$ 个构建决策树。因此，随机森林处理能够做到样本扰动外，还添加了特征扰动，对于特征的选择个数，推荐值 $k=\log_2 d$。

- 树的构建

每次根据采样得到的数据和特征，构建一棵决策树。在构建决策树的过程中，会让决策树生长完全而不进行剪枝。构建出的若干棵决策树则组成了最终的随机森林。

[1] Breiman, Leo. "Random forests." *Machine learning* 45.1 (2001): 5-32.

# 随机森林

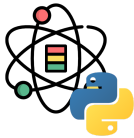


随机森林在众多分类算法中表现十分出众[1]，其主要的优点包括：

1. 由于随机森林引入了样本扰动和特征扰动，从而很大程度上提高了模型的泛化能力，尽可能地避免了过拟合现象的出现。
2. 随机森林可以处理高维数据，无需进行特征选择，在训练过程中可以得出不同特征对模型的重要性程度。
3. 随机森林的每个弱分类器采用决策树，方法简单且容易实现。同时每个弱分类器之间没有相互依赖关系，整个算法易并行化。

[1] Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *The journal of machine learning research* 15.1 (2014): 3133-3181.

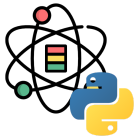
# 随机森林



```
sklearn.ensemble.RandomForestClassifier(
  n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,
  min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
  min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None,
  verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

[1] 参数列表：https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

# 随机森林



```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

X, y = load_wine(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.33, stratify=y, random_state=0)
clf = RandomForestClassifier(n_estimators=10).fit(
  X_train, y_train)
clf.feature_importances_
```

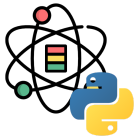
```
## array([0.17406624, 0.03205294, 0.0148181 , 0.020788
78, 0.07581662,
##        0.09529591, 0.08942977, 0.00778304, 0.
, 0.17002258,
##        0.10530024, 0.14288429, 0.0717415 ])
```

```
y_pred = clf.predict(X_test)
target_names = ['0', '1', '2']
classification_report(
  y_test, y_pred, target_names=target_names)
```

```
##               precision  recall  f1-score  support
##            0       0.91    1.00      0.95       20
##            1       1.00    0.91      0.95       23
##            2       1.00    1.00      1.00       16
##
##     accuracy                         0.97       59
##    macro avg       0.97    0.97      0.97       59
## weighted avg       0.97    0.97      0.97       59
```

# Boosting

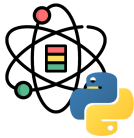
# Boosting



Boosting 是一种提升算法，可以将弱的学习算法提升（boost）为强的学习算法。其基本思路如下：

1. 利用初始训练样本集训练得到一个基学习器。
2. 提高被基学习器误分的样本的权重，使得那些被错误分类的样本在下一轮训练中可以得到更大的关注，利用调整后的样本训练得到下一个基学习器。
3. 重复上述步骤，直到得出 $M$ 个学习器。
4. 对于分类问题，采用有权重的投票方式；对于回归问题，采用加权平均得到预测值。

# Adaboost



Adaboost [1] 是 Boosting 算法中最具代表性的一个。原始的 Adaboost 算法用于解决二分类问题，因此对于一个训练集

$$T = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \tag{2}$$

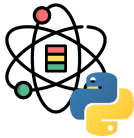
其中 $x_i \in \mathcal{X} \subseteq \mathbb{R}^n$， $y_i \in \mathcal{Y} = \{-1, +1\}$，首先初始化训练集的权重

$$\begin{aligned} D_1 =& (w_{11}, w_{12}, \ldots, w_{1n}) \\ w_{1i} =& \frac{1}{n}, i = 1, 2, \ldots, n \end{aligned} \tag{3}$$

根据每一轮训练集的权重 $D_m$，对训练集数据进行抽样得到 $T_m$，再根据 $T_m$ 可以得到每一轮的基学习器 $h_m$。

[1] Freund, Yoav, and Robert E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting." *European conference on computational learning theory*. Springer, Berlin, Heidelberg, 1995.

# Adaboost



通过计算可以得出基学习器 $h_m$ 的误差为 $\epsilon_m$，根据基学习器的误差计算得出该基学习器在最终学习器中的系数

$$\alpha_m = \frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m} \tag{4}$$

更新训练集的权重

$$\begin{aligned} D_{m+1} =& (w_{m+1,1}, w_{m+1,2}, \ldots, w_{m+1,n}) \\ w_{m+1,i} =& \frac{w_{m,i}}{Z_m}\exp\left(-\alpha_m y_i h_m\left(x_i\right)\right) \\ Z_m =& \sum_{i=1}^{n} w_{m,i}\exp\left(-\alpha_m y_i h_m\left(x_i\right)\right) \end{aligned} \tag{5}$$

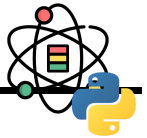
其中 $Z_m$ 为规范化因子，从而保证 $D_{m+1}$ 为一个概率分布。

# Adaboost

最终根据构建的 $M$ 个基学习器得到最终的学习器：

$$h_f(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_i h_m(x)\right) \tag{6}$$

AdaBoost 算法过程如由所示：



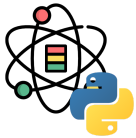
**Algorithm 2** AdaBoost 算法

**Require:** 学习算法 ， 子模型个数 $M$， 训练数据集 $T$

**Ensure:** AdaBoost 算法 $h_f(x)$

**procedure** AdaBoost$(, M, T)$
  $D_1(x)$  $1n$
  **for** $m = 1$ to $M$ **do**
    $T_{sub}$  sample from training set $T$ with weights
    $h_m$  $(T_{sub})$
    $\epsilon_m$  $rrr(h_m)$
    **if** $\epsilon_m > 0.5$ **then**
      **break**
    **end if**
    $\alpha_m$  $\frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m}$
    $D_{m+1}$  $\frac{D_m \exp(-\alpha_m y h_m(x))}{Z_m}$
  **end for**
  $h_f(x)$  $\left(\sum_{m=1}^{M} \alpha_i h_m(x)\right)$
  **return** $h_f(x)$
**end procedure**

# GBM

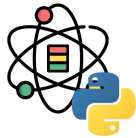


GBM（Gradient Boosting Machine）是另一种基于 Boosting 思想的集成算法，GBM 还有很多其他的叫法，例如：GBDT，GBRT，MART等等。GBM 算法由 3 个主要概念构成：Gradient Boosting（GB），Regression Decision Tree（DT 或 RT）和 Shrinkage。

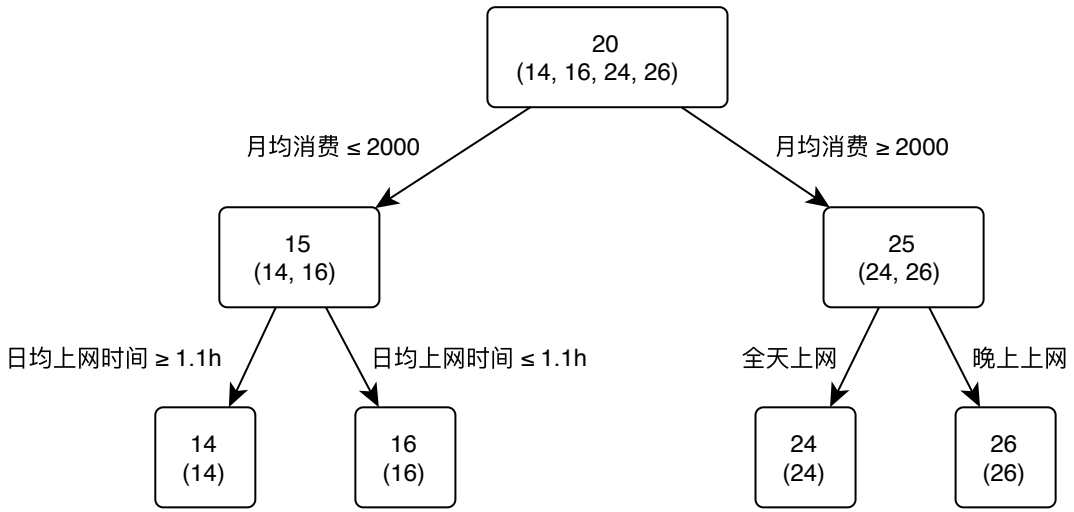
从 GBM 的众多别名中可以看出，GBM 中使用的决策树并非我们最常用的分类树，而是回归树。分类树主要用于处理响应变量为因子型的数据，例如天气（可以为晴，阴或下雨等）。回归树主要用于处理响应变量为数值型的数据，例如商品的价格。当然回归树也可以用于二分类问题，对于回归树预测出的数值结果，通过设置一个阈值即可以将数值型的预测结果映射到二分类问题标签上，即 $\mathcal{Y} = \{-1, +1\}$。

对于 Gradient Boosting 而言，首先，Boosting 并不是 Adaboost 中的 Boost 的概念，也不是 Random Forest 中的冲抽样。在 Adaboost 中，Boost 是指在生成每个新的基学习器，跟根据上一轮基学习器分类对错对训练集设置不同的权重，使得在上一轮中分类错误的样本在生成新的基学习器时更被重视。GBM 中在应用 Boost 概念时，每一轮所使用的数据集没有经过重抽样，也没有更新样本的权重，而是每一轮选择了不用的回归的目标值，即上一轮计算得出的残差（Residual）。其次，Gradient 是指在新一轮中在残差减少的梯度（Gradient）上建立新的基学习器。

# GBM

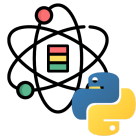


下面通过一个年龄预测的示例介绍 GBM 的工作流程。存在 4 个人 $P = \{p_1, p_2, p_3, p_4\}$，他们对应的年龄为 $14, 16, 24, 26$。其中 $p_1, p_2$ 分别是高一和高三学生， $p_3, p_4$ 分别是应届毕业生和工作两年的员工。利用决策树模型进行训练可以得到如图所示的结果：

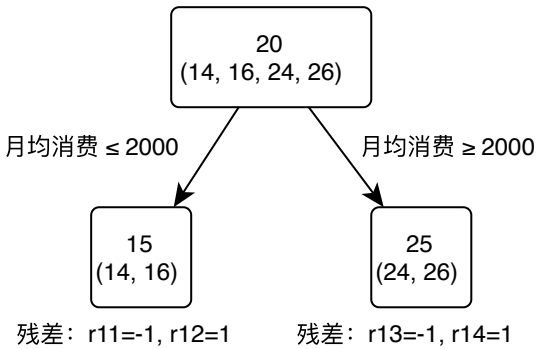


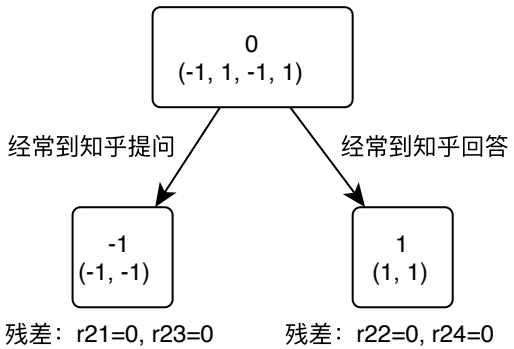

[1] 示例修改自：http://suanfazu.com/t/gbdt-die-dai-jue-ce-shu-ru-men-jiao-cheng/135

# GBM



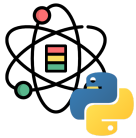
利用 GBM 训练得到模型，由于数据量少，在此限定每个基学习器中的叶子节点最多为 2 个，即树的深度最大为 1 层。训练得到的结果如图所示：




在训练第一棵树过程中，利用年龄作为预测值，根据计算可得由于 $p_1, p_2$ 年龄相近， $p_3, p_4$ 年龄相近被划分为两组。通过计算两组中真实年龄和预测的年龄的差值，可以得到第一棵树的残差 $R = \{-1, 1, -1, 1\}$。因此在训练第二棵树的过程中，利用第一棵树的残差作为预测值，最终所有人的年龄均正确被预测，即最终所有的残差均为 0。

# GBM



则对于训练集中的 4 个人利用训练得到额 GBM 模型预测的结果如下:

- $p_1$：14岁高一学生。购物较少，经常问学长问题，预测年龄 $Age = 15 - 1 = 14$。
- $p_2$：16岁高三学生。购物较少，经常被学弟问问题，预测年龄 $Age = 15 + 1 = 16$。
- $p_3$：24岁应届毕业生。购物较多，经常问师兄问题，预测年龄 $Age = 25 - 1 = 24$。
- $p_4$：26岁2年工作经验员工。购物较多，经常被师兄问问题，预测年龄 $Age = 25 + 1 = 26$。

**Algorithm 3** GBM 算法

**Require:** 子模型个数 $M$，训练数据集 $T$

**Ensure:** GBM 算法 $h_f(x)$

**procedure** GBM$(M, T)$

  $F_1(x)$ $\sum_{i=1}^{N} y_i N$

  **for** $m = 1$ to $M$ **do**

    $r_m$ $y - F_m(x)$

    $T_m$ $(x, r_m)$

    $h_m$ $RegressinTree(T_m)$

    $\alpha_m$ $\dfrac{\sum_{i=1}^{N} r_{im} h_m(x_i)}{\sum_{i=1}^{N} h_m(x_i)^2}$

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

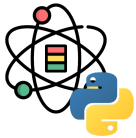
  **end for**

  $h_f(x) = F_M(x)$

  **return** $h_f(x)$

**end procedure**

# GBM

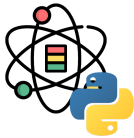


在 GBM 中也应用到了 Shrinkage 的思想，其基本思想可以理解为在每一轮利用残差学习得到的回归树仅学习到了一部分知识，即无法完全信任一棵树的结果。因此，Shrinkage 思想认为在新的一轮学习中，不利用全部残差训练模型，而只利用其中一部分，即：

$$r_m = y - sF_m(x), 0 \leq s \leq 1 \tag{7}$$

注意，这里的 Shrinkage 和学习算法中 Gradient 的步长是两个不相关的概念。Shrinkage 设置小一些可以避免发生过拟合现象；而 Gradient 中的步长如果设置太小则会陷入局部最优，如果设置过大又容易结果不收敛。

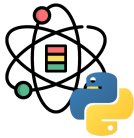
# GBM



```
sklearn.ensemble.GradientBoostingClassifier(
  loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse',
  min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
  min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None,
  verbose=0, max_leaf_nodes=None, warm_start=False, presort='deprecated', validation_fraction=0.1,
  n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

[1] 参数列表：https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

# XGBoost



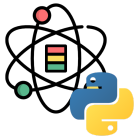
XGBoost 是由 Chen 等人 [1] 提出的一种梯度提升树模型框架。XGBoost 的基本思想同 GBDT 一样，对于一个包含 $n$ 个样本和 $m$ 个特征的数据集 $\mathcal{D}=\{(\mathbf{x}_i, y_i)\}$，其中 $|\mathcal{D}|=n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}$，一个集成树模型可以用 $K$ 个加法函数预测输出：

$$\hat{y}_i=\phi\left(\mathbf{x}_i\right)=\sum_{k=1}^K f_k\left(\mathbf{x}_i\right), f_k \in \mathcal{F} \tag{8}$$

其中，$\mathcal{F}=\left\{f\left(\mathbf{x}\right)=w_{q(\mathbf{x})}\right\}\left(q: \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T\right)$ 为回归树 (CART)，$q$ 表示每棵树的结构，其将一个样本映射到最终的叶子节点，$T$ 为叶子节点的数量，每个 $f_w$ 单独的对应一棵结构为 $q$ 和权重为 $w$ 的树。不同于决策树，每棵回归树的每个叶子节点上包含了一个连续的分值，我们用 $w_i$ 表示第 $i$ 个叶子节点上的分值。

[1] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794).

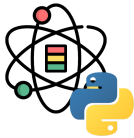
# XGBoost



```
xgboost.train(
  params, dtrain, num_boost_round=10, evals=(), obj=None, feval=None, maximize=False,
  early_stopping_rounds=None, evals_result=None, verbose_eval=True, xgb_model=None, callbacks=None)

xgboost.XGBClassifier(objective='binary:logistic', **kwargs)
```

[1] 参数列表：https://xgboost.readthedocs.io/en/latest/python/python_api.html

# XGBoost



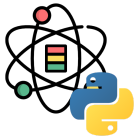
```
import xgboost as xgb

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, stratify=y, random_state=42)

clf = xgb.XGBClassifier().fit(X_train, y_train)
y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```

```
## 0.9210526315789473
```

```
from sklearn.metrics import classification_report

target_names = [
  'Setosa', 'Versicolour', 'Virginica']
classification_report(
  y_test, y_pred, target_names=target_names)
```

```
##               precision  recall  f1-score  support

##       Setosa       1.00    1.00      1.00       12
##  Versicolour       0.86    0.92      0.89       13
##    Virginica       0.92    0.85      0.88       13

##     accuracy                         0.92       38
##    macro avg       0.92    0.92      0.92       38
## weighted avg       0.92    0.92      0.92       38
```

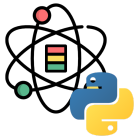
# LightGBM



```
lightgbm.train(
  params, train_set, num_boost_round=100, valid_sets=None, valid_names=None, fobj=None,feval=None,
  init_model=None, feature_name='auto', categorical_feature='auto', early_stopping_rounds=None,
  evals_result=None, verbose_eval=True, learning_rates=None, keep_training_booster=False, callbacks=None)

lightgbm.LGBMClassifier(
  boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=100,
  subsample_for_bin=200000, objective=None, class_weight=None, min_split_gain=0.0, min_child_weight=0.001,
  min_child_samples=20, subsample=1.0, subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0,reg_lambda=0.0,
  random_state=None, n_jobs=-1, silent=True, importance_type='split', **kwargs)
```

[1] 参数列表：https://lightgbm.readthedocs.io/en/latest/Python-API.html

# LightGBM



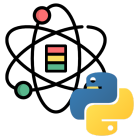
```python
import lightgbm as lgb

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, stratify=y, random_state=42)

clf = lgb.LGBMClassifier(verbosity=-1).fit(
  X_train, y_train)
y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```

```
## 0.8421052631578947
```

```python
from sklearn.metrics import classification_report

target_names = [
  'Setosa', 'Versicolour', 'Virginica']
classification_report(
  y_test, y_pred, target_names=target_names)
```

```
##               precision  recall  f1-score  support
##       Setosa       1.00    1.00      1.00       12
##  Versicolour       0.75    0.92      0.83       13
##    Virginica       0.90    0.69      0.78       13
##
##     accuracy                         0.87       38
##    macro avg       0.88    0.87      0.87       38
## weighted avg       0.88    0.87      0.87       38
```

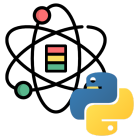
# CatBoost



```
catboost.train(
  pool=None, params=None, dtrain=None, logging_level=None, verbose=None, iterations=None,
  num_boost_round=None, evals=None, eval_set=None, plot=None, verbose_eval=None, metric_period=None,
  early_stopping_rounds=None, save_snapshot=None, snapshot_file=None, snapshot_interval=None, init_model=None)

catboost.CatBoostClassifier(
  iterations=None, learning_rate=None, depth=None, l2_leaf_reg=None, model_size_reg=None,
  rsm=None, loss_function=None, border_count=None, feature_border_type=None, per_float_feature_quantization=None,
  input_borders=None, output_borders=None, fold_permutation_block=None, od_pval=None, od_wait=None,
  od_type=None, nan_mode=None, counter_calc_method=None, leaf_estimation_iterations=None,
  leaf_estimation_method=None, thread_count=None, random_seed=None, use_best_model=None, verbose=None,
  logging_level=None, metric_period=None, ctr_leaf_count_limit=None, store_all_simple_ctr=None,
  max_ctr_complexity=None, has_time=None, allow_const_label=None, classes_count=None, class_weights=None,
  one_hot_max_size=None, random_strength=None, name=None, ignored_features=None, train_dir=None, custom_loss=None,
  custom_metric=None, eval_metric=None, bagging_temperature=None, save_snapshot=None, snapshot_file=None,
  snapshot_interval=None, fold_len_multiplier=None, used_ram_limit=None, gpu_ram_part=None, ...)
```

[1] 参数列表：https://catboost.ai/docs/concepts/python-reference_catboostclassifier.html

# CatBoost



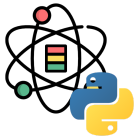
```
from catboost import CatBoostClassifier

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, stratify=y, random_state=42)

clf = CatBoostClassifier(verbose=0).fit(
  X_train, y_train)
y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```

```
## 0.9210526315789473
```

```
from sklearn.metrics import classification_report

target_names = [
  'Setosa', 'Versicolour', 'Virginica']
classification_report(
  y_test, y_pred, target_names=target_names)
```

```
##               precision  recall  f1-score  support
##       Setosa       1.00    1.00      1.00       12
##  Versicolour       0.86    0.92      0.89       13
##    Virginica       0.92    0.85      0.88       13
##
##     accuracy                         0.92       38
##    macro avg       0.92    0.92      0.92       38
## weighted avg       0.92    0.92      0.92       38
```

# 实现对比



针对 scikit-learn，XGBoost，LightGBM 和 CatBoost 4 种 GBM 的具体实现，下表汇总了各自的相关特性 [1]：

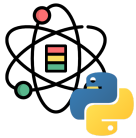
| | scikit-learn | XGBoost | LightGBM | CatBoost |
|---|---|---|---|---|
| **当前版本** [2] | 0.22.1 | 1.0.0 | 2.3.2 | 0.21 |
| **实现语言** | C, C++, Python | C, C++ | C, C++ | C++ |
| **API 语言** | Python | Python, R, Java, Scala, C++ and more | Python, R | Python, R |
| **模型导出** | JPMML | JPMML, ONNX | ONNX | CoreML, Python, C++, JSON |
| **多线程** | No | Yes | Yes | Yes |
| **GPU /多 GPU** | No / No | Yes / Yes | Yes / No | Yes / Yes |

[1] https://leovan.me/cn/2018/12/ensemble-learning/

[2] 数据截止 2020 年 2 月 1 日

(接下表)

# 实现对比

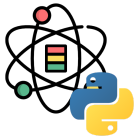


(接上表)

| | scikit-learn | XGBoost | LightGBM | CatBoost |
|---|---|---|---|---|
| **Boosting 类型** | Gradient Boosted Tree (**GBDT**) | **GBDT** (booster: gbtree) Generalized Linear Model, **GLM** (booster: gbliner) Dropout Additive Regression Tree, **DART** (booster: dart) | **GBDT** (boosting: gbdt) **Random Forest** (boosting: rf) **DART** (boosting: dart) Gradient-based One-Side Sampling, **GOSS** (bossting: goss) | **Ordered** (boosting_type: Ordered) **Plain** (bossting_type: Plain) |
| **Level-wise (Depth-wise) Split** | Yes | Yes (grow_policy: depthwise) | No | Yes |
| **Leaf-wise (Best-first) Split** | No | Yes (grow_policy: lossguide) | Yes | No |

(接下表)

# 实现对比

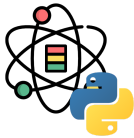


(接上表)

| | scikit-learn | XGBoost | LightGBM | CatBoost |
|---|---|---|---|---|
| **Histogram-based Split** | No | Yes (tree_method: hist / gpu_hist) | Yes | Yes |
| **过拟合控制** | Yes (max_depth, ...) | Yes (max_depth, max_leaves, gamma, reg_alpha, reg_lamda, ...) | Yes (max_depth, num_leaves, gamma, reg_alpha, reg_lamda, drop_rate, ...) | Yes (max_depth, reg_lambda, ...) |
| **分类特征** | No | No | Yes (categorical_feature) | Yes (cat_features) |
| **缺失值处理** | No | Yes | Yes (use_missing) | Yes |
| **不均衡数据** | No | Yes (scale_pos_weight, max_delta_step) | Yes (scale_pos_weight, poisson_max_delta_step) | Yes (scale_pos_weight) |

# Stacking

# Stacking



Stacking 本身是一种集成学习方法，同时也是一种模型组合策略，我们首先介绍一些相对简单的模型组合策略：**平均法**和**投票法**。

对于 数值型的输出 $h_i(\mathbf{x}) \in \mathbb{R}$,

- 简单平均法 (Simple Averaging)

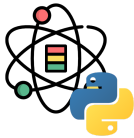
$$H(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} h_i(\mathbf{x}) \tag{9}$$

- 加权平均法 (Weighted Averaging)

$$H(\mathbf{x}) = \sum_{i=1}^{M} w_i h_i(\mathbf{x}) \tag{10}$$

其中， $w_i$ 为学习器 $h_i$ 的权重，且 $w_i \geq 0, \sum_{i=1}^{T} w_i = 1$。

# Stacking



对于 分类型的任务，学习器 $h_i$ 从类别集合 $\{c_1, c_2, \ldots, c_N\}$ 中预测一个标签。我们将 $h_i$ 在样本 $\mathbf{x}$ 上的预测输出表示为一个 $N$ 维向量 $\left(h_i^1(\mathbf{x}); h_i^2(\mathbf{x}); \ldots, h_i^N(\mathbf{x})\right)$，其中 $h_i^j(\mathbf{x})$ 为 $h_i$ 在类型标签 $c_j$ 上的输出。

- 绝对多数投票法 (Majority Voting)

$$H(\mathbf{x}) = \begin{cases} c_j, & \sum_{i=1}^{M} h_i^j(\mathbf{x}) > 0.5 \sum_{k=1}^{N} \sum_{i=1}^{M} h_i^k(\mathbf{x}) \\ \text{拒绝}, & \text{其他情况} \end{cases} \tag{11}$$

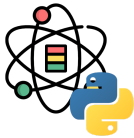
即如果一个类型的标记得票数过半，则预测为该类型，否则拒绝预测。

- 相对多数投票法 (Plurality Voting)

$$H(\mathbf{x}) = c_{\arg\max_j \sum_{i=1}^{M} h_i^j(\mathbf{x})} \tag{12}$$

即预测为得票数最多的类型，如果同时有多个类型获得相同最高票数，则从中随机选取一个。

# Stacking



- 加权投票法 （Weighted Voting）

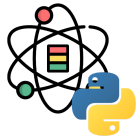
$$H(\mathbf{x}) = c_{\arg\max_j \sum_{i=1}^{M} w_i h_i^j(\mathbf{x})} \tag{13}$$

其中， $w_i$ 为学习器 $h_i$ 的权重，且 $w_i \geq 0, \sum_{i=1}^{M} w_i = 1$。

绝对多数投票提供了“拒绝预测”，这为可靠性要求较高的学习任务提供了一个很好的机制，但如果学习任务要求必须有预测结果时则只能选择相对多数投票法和加权投票法。在实际任务中，不同类型的学习器可能产生不同类型的 $h_i^j(\boldsymbol{x})$ 值，常见的有：

- 类标记， $h_i^j(\mathbf{x}) \in \{0, 1\}$，若 $h_i$ 将样本 $\mathbf{x}$ 预测为类型 $c_j$ 则取值为 1，否则取值为 0。使用类型标记的投票称之为“硬投票” (Hard Voting)。
- 类概率， $h_i^j(\mathbf{x}) \in [0, 1]$，相当于对后验概率 $P(c_j \mid \mathbf{x})$ 的一个估计。使用类型概率的投票称之为“软投票” (Soft Voting)。

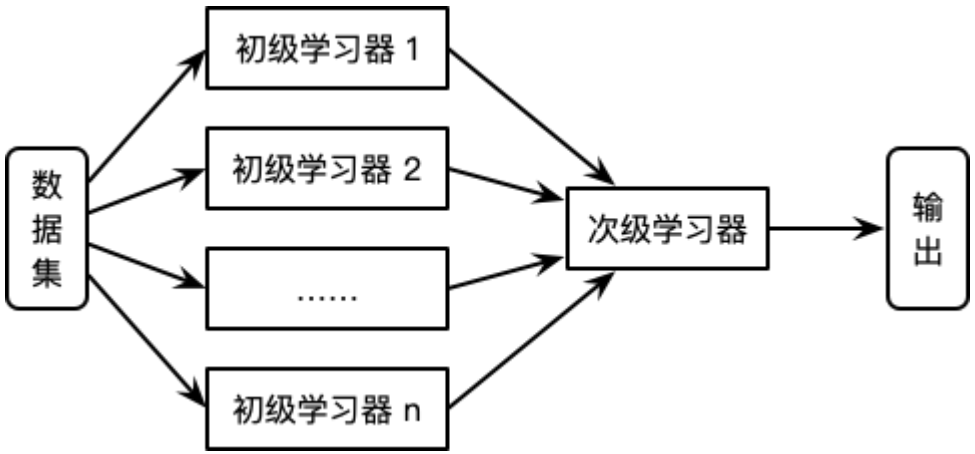
# Stacking



Stacking [1, 2] 方法又称为 Stacked Generalization，是一种基于分层模型组合的集成算法。Stacking 算法的基本思想如下：

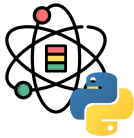
1. 利用初级学习算法对原始数据集进行学习，同时生成一个新的数据集。
2. 根据从初级学习算法生成的新数据集，利用次级学习算法学习并得到最终输出。

对于初级学习器，可以是相同类型也可以是不同类型，同时在生成新的数据集的过程中，其相应变量仍为原始数据集中的相应变量。Stacking 算法流程如图所示：




[1] Wolpert, David H. "Stacked generalization." *Neural networks* 5.2 (1992): 241-259.
[2] Breiman, Leo. "Stacked regressions." *Machine learning* 24.1 (1996): 49-64.

# Stacking



**Algorithm 4** Stacking 算法

**Require:**
初级学习算法 $= \{_1, _2, \cdots _M\}$
次级学习算法 $'$
训练数据集 $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$

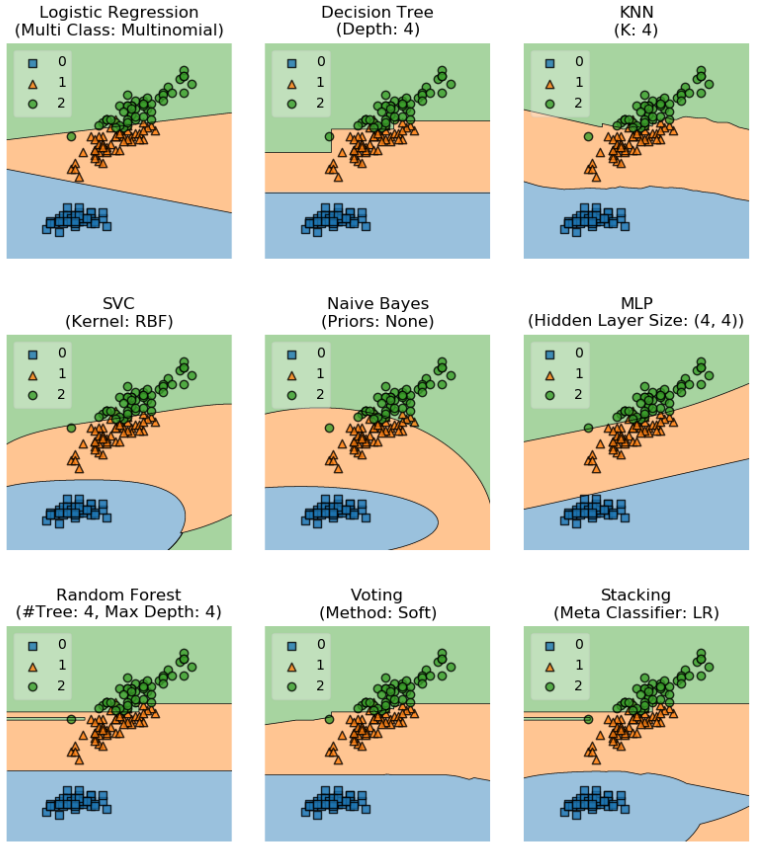
**Ensure:** Stacking 算法 $h_f(x)$

1: **procedure** STACKING$(, ', T)$
2: **for** $m = 1$ to $M$ **do**
3: $h \quad _m(T)$
4: **end for**
5: $T'$
6: #接下文
7: **end procedure**

1: **procedure** STACKING$(, ', T)$
2: #接上文
3: **for** $i = 1$ to $N$ **do**
4: **for** $m = 1$ to $M$ **do**
5: $z_{im} \quad h_m(\mathbf{x}_i)$
6: **end for**
7: $T' \quad T' \quad ((z_{i1}, z_{i2}, \ldots, z_{iM}), y_i)$
8: **end for**
9: $h' \quad '(T')$
10: $h_f(\mathbf{x}) \quad h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_M(\mathbf{x}))$
11: **return** $h_f(\mathbf{x})$
12: **end procedure**

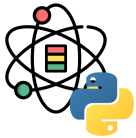
# Stacking

次级学习器的训练集是由初级学习器产生的，如果直接利用初级学习器的训练集生成次级学习器的训练集，则过拟合风险会比较大 [1]。因此，一般利用在训练初始学习器中未使用过的样本来产生次级学习器的训练样本。以 $k$ 折交叉检验为例：初始的训练集 $T$ 被随机划分为 $k$ 个大小相近的集合 $T_1, T_2, \ldots, T_k$。令 $T_j$ 和 $\bar{T}_j$ 表示第 $j$ 折的测试集和训练集。则对于 $M$ 个初级学习算法，其学习器 $h_m^{(j)}$ 是根据训练集 $\bar{T}_j$ 生成的，对于测试集 $T_j$ 中的每个样本 $x_i$，得到 $z_{im} = h_m^{(j)}(x_i)$。则根据 $x_i$ 所产生的次级学习器的训练样本为 $((z_{i1}, z_{i2}, \ldots, z_{iM}), y_i)$。最终利用 $M$ 个初级学习器产生的训练集 $T' = \{(z_i, y_i)\}_{i=1}^{N}$ 训练次级学习器。




[1] 周志华, *机器学习*. 清华大学出版社, 2016.

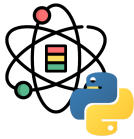
# Stacking



```
sklearn.ensemble.StackingClassifier(
  estimators, final_estimator=None, cv=None, stack_method='auto', n_jobs=None, passthrough=False, verbose=0)
```

[1] 参数列表：https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

# Stacking



```
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, stratify=y, random_state=42)
estimators = [
  ('rf', RandomForestClassifier(
    n_estimators=10, random_state=42)),
  ('svr', make_pipeline(
    StandardScaler(), LinearSVC(random_state=42)))]
clf = StackingClassifier(
  estimators=estimators,
  final_estimator=LogisticRegression())
```

```
clf.fit(X_train, y_train).score(X_test, y_test)
```

```
## 0.9473684210526315
```

```
target_names = [
  'Setosa', 'Versicolour', 'Virginica']
classification_report(
  y_test, y_pred, target_names=target_names)
```

```
##                precision  recall  f1-score  support
##       Setosa        1.00    1.00      1.00       12
##  Versicolour        0.86    0.92      0.89       13
##    Virginica        0.92    0.85      0.88       13
##
##     accuracy                           0.92       38
##    macro avg        0.92    0.92      0.92       38
## weighted avg        0.92    0.92      0.92       38
```

# 感谢倾听