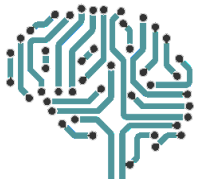
暨南大學
JINAN UNIVERSITY



机 器 学 习

第七章：集成学习

黄斐然

2022/4/2

三个臭皮匠赛过诸葛亮

什么是集成学习?

分类器1

分类器2

分类器3

分类器4

……

集成多个分类器，进行综合判断
（少数服从多数）

集成分类算法

- 什么是集成学习：
  - 集成学习的主要思路是先通过一定的规则生成多个学习器，再采用某种集成策略进行组合，最后综合判断输出最终结果。一般而言，通常所说的集成学习中的多个学习器都是同质的"弱学习器"。基于该弱学习器，通过样本集扰动、输入特征扰动、输出表示扰动、算法参数扰动等方式生成多个学习器，进行集成后获得一个精度较好的"强学习器"。
  - 目前集成学习算法主要由bagging、boosting两种思想，其他还有Stacking和Blending。

集成分类算法

- Sklearn集成学习：

```
from sklearn.ensemble import VotingClassifier

ensemble_clf = VotingClassifier(estimators=[
    ('knn_clf', KNeighborsClassifier(n_neighbors=1)),
    ('dt_clf', DecisionTreeClassifier()),
    ('nb_clf', GaussianNB())],
                                voting='hard')
```

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

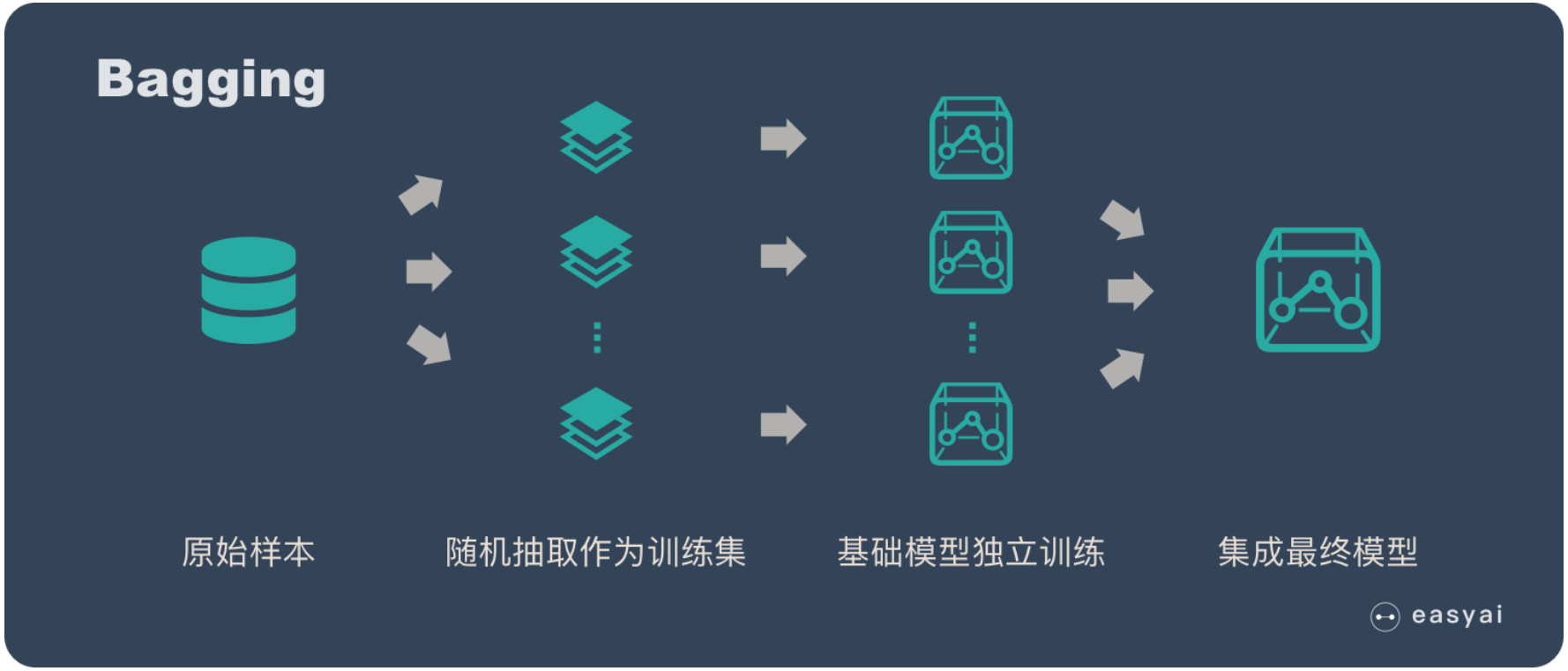
```
0.896
```

Bagging

- Bagging的算法过程：
  - 从原始样本集中使用有放回抽样（Bootstraping）方法随机抽取n个训练样本，共进行k轮抽取，得到k个训练集（k个训练集之间相互独立，元素可以有重复）。
  - 对于k个训练集，我们训练k个基础模型，（这个模型可根据具体的情况而定，可以是决策树，knn等）
  - 对于分类问题：由投票表决产生的分类结果；对于回归问题，由k个模型预测结果的均值作为最后预测的结果（所有模型的重要性相同）。

Bagging

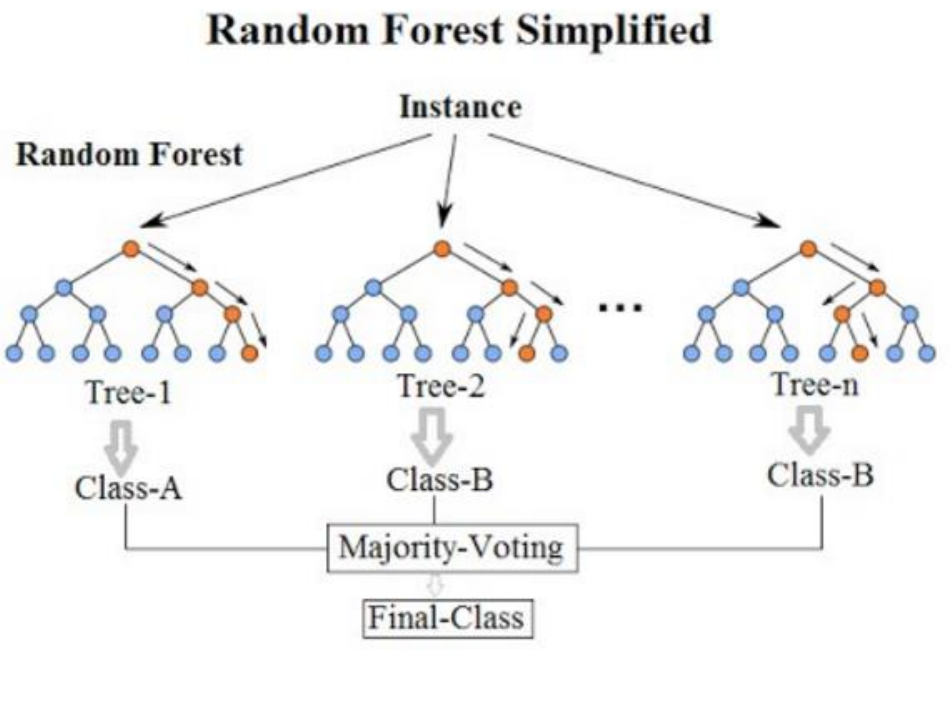
- 算法过程：



Bagging 的思路是所有基础模型都一致对待，每个基础模型手里都只有一票。然后使用民主投票的方式得到最终的结果。

随机森林

- 随机森林(Random Forest) 是Bagging算法的进化版，也就是说，它的思想仍然是bagging，但是进行了独有的改进。相对于bagging，RF一方面在引入数据随机，另一方面也引入了特征随机。
- 随机森林非常简单，易于实现，计算开销也很小，但是它在分类和回归上表现出非常惊人的性能，因此，随机森林被誉为“代表集成学习技术水平的方法”。



随机森林

- 随机森林步骤如下：
  - 从样本集中用Bootstrap采样选出n个样本；（样本随机）
  - 从所有属性中随机选择k个属性，选择最佳分割属性作为节点建立决策树；（特征随机）
  - 重复以上两步m次，即建立了m棵决策树
  - 这m个决策树形成随机森林，通过投票表决结果，决定数据属于哪一类

随机森林

随机森林使用sklearn实现：

- from sklearn.ensemble import RandomForestClassifier
- RandomForestClassifier(n_estimators=’warn’, 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)
- 重要参数：
  - n_estimators：决策树数目。默认为10。
  - criterion：特征选择方法。默认’gini’。也可以选择‘entropy’。
  - bootstrap：是否有放回采样。默认‘True’。
  - n_jobs：使用的线程数。默认1个。

随机森林

随机森林实现：

In [15]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(5):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = RandomForestClassifier().fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(avg_scores))
```

```
0.95609022556391
```

Extra Trees

随机森林的推广——Extra Trees：

- Extra trees（Extremely randomized trees，极端随机树）是RF的一个变种, 原理几乎和RF一模一样，仅有区别有：
  - 1）对于每个决策树的训练集，RF采用的是随机采样bootstrap来选择采样集作为每个决策树的训练集，而Extra trees一般不采用随机采样，即每个决策树采用原始训练集。
  - 2）在选定了划分特征后，RF的决策树会基于基尼系数，均方差之类的原则，选择一个最优的特征值划分点，这和传统的决策树相同。但是Extra trees比较的激进，他会随机的选择一个特征值来划分决策树。
- 从第二点可以看出，由于随机选择了特征值的划分点位，而不是最优点位，这样会导致生成的决策树的规模一般会大于RF所生成的决策树。也就是说，模型的方差相对于RF进一步减少，但是偏倚相对于RF进一步增大。在某些时候，Extra trees的泛化能力比RF更好。：

Extra Trees

ExtraTrees使用sklearn实现：

- from sklearn.ensemble import ExtraTreesClassifier
- ExtraTreesClassifier(n_estimators=’warn’, 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=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None)
- 重要参数：
  - n_estimators：决策树数目。默认为10。
  - criterion：特征选择方法。默认’gini’。也可以选择‘entropy’。
  - bootstrap：是否有放回采样。默认‘True’。
  - n_jobs：使用的线程数。默认1个。

Extra Trees

- ExtraTrees实现：

In [33]:

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(5):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = ExtraTreesClassifier().fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(avg_scores))
```

```
0.9602568922305765
```

随机森林

随机森林优点：

- 1）训练可以高度并行化，对于大数据时代的大样本训练速度有优势。
- 2）由于可以随机选择决策树节点划分特征，这样在样本特征维度很高的时候，仍然能高效的训练模型。
- 3）在训练后，可以给出各个特征对于输出的重要性
- 4）由于采用了随机采样，训练出的模型的方差小，泛化能力强。
- 5）相对于Boosting系列的Adaboost和GBDT， RF实现比较简单。
- 6）对部分特征缺失不敏感。

随机森林

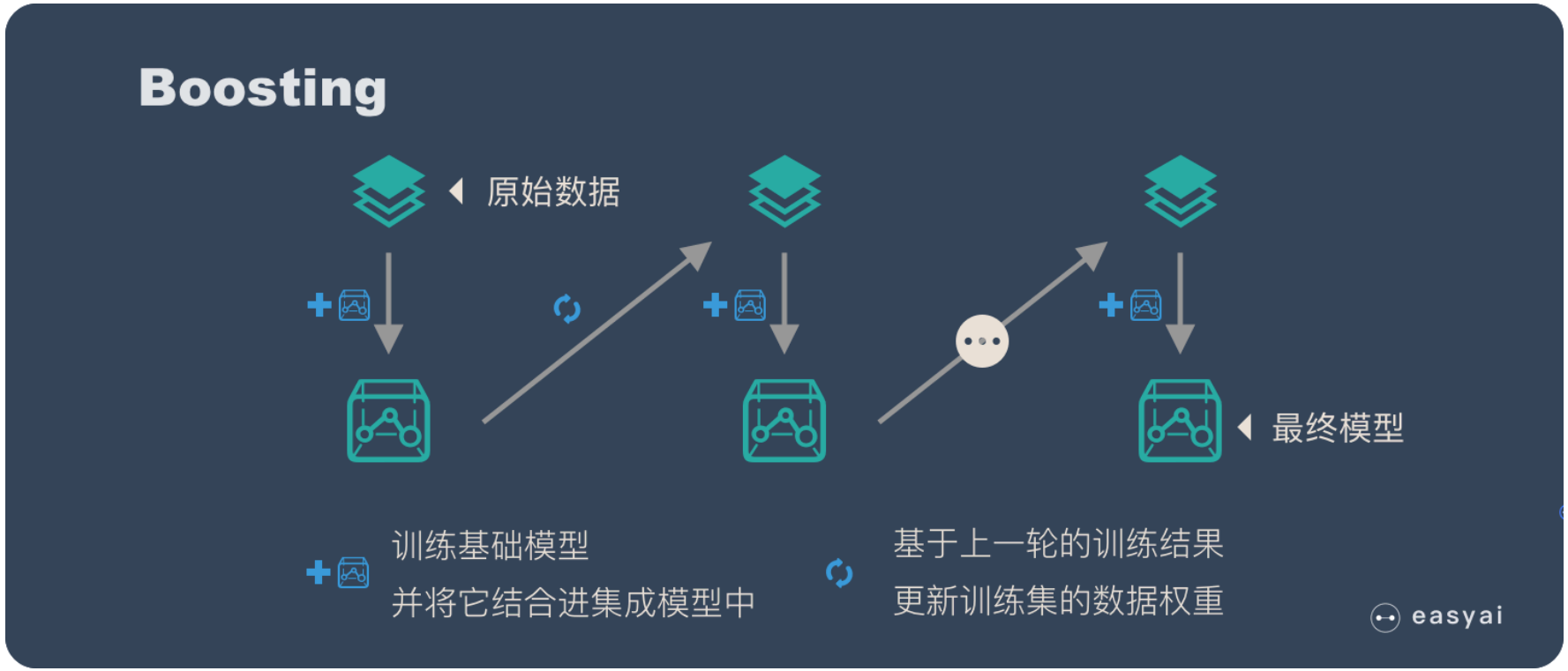
- 随机森林缺点：
  - 1）在某些噪音比较大的样本集上，RF模型容易陷入过拟合。
  - 2）对于小数据或者低维数据（特征较少的数据），可能不能产生很好的分类。（处理高维数据，处理特征遗失数据，处理不平衡数据是随机森林的长处）。

Boosting

- Boosting的算法过程：
  - 通过加法模型将基础模型进行线性的组合。
  - 每一轮训练都提升那些错误率小的基础模型权重，同时减小错误率高的模型权重。
  - 在每一轮改变训练数据的权值或概率分布，通过提高那些在前一轮被弱分类器分错样本的权值，减小前一轮分对样本的权值，来使得分类器对误分的数据有较好的效果。

Boosting

- 算法过程：

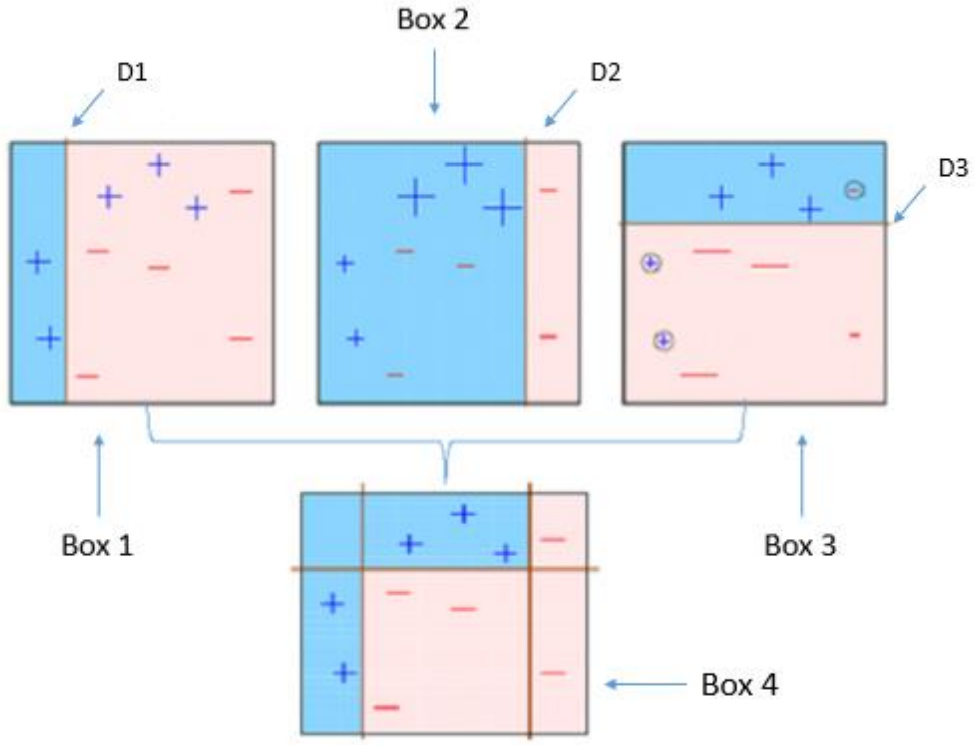


经过不停的考验和筛选来挑选出精英，然后给精英更多的投票权，表现不好的基础模型则给较少的投票权，然后综合所有人的投票得到最终结果。

AdaBoost

AdaBoost：

- Adaboost是boosting的主流算法之一。
- Adaboost 算法很有名，全称是adaptive boosting。 曾被称为数据挖掘十大算法之一。
- Adaboost是一种基于boosting思想的一种自适应的迭代式算法。通过在同一个训练数据集上训练多个弱分类器， 然后把这一组弱分类器ensmble起来， 产生一个强分类器。



AdaBoost

Adaboost使用sklearn实现：

- from sklearn.ensemble import AdaBoostClassifier
- AdaBoostClassifier(base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)
- 重要参数：
  - base_estimator：基本分类器。默认为DecisionTreeClassifier(max_depth=1)。
  - n_estimators：分类器个数。默认50。
  - learning_rate：学习率。默认1。
  - algorithm：学习算法。默认'SAMME.R' ，SAMME.R算法比SAMME算法收敛速度更快。

AdaBoost

- Adaboost使用sklearn实现：

In [10]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(5):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = AdaBoostClassifier().fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(avg_scores))
```

```
0.9627882205513784
```

提升树

- Boosting主流方法：
  - 2）**提升树**：
    - 在Adaboost算法的框架下，以决策树为基函数的提升方法称为提升树。由于树的线性组合可以很好的拟合训练数据，即使数据中的输入与输出之间的关系很复杂也是如此，提升树被认为是统计学习中性能最好的方法之一。
    - AdaBoost + 决策树 = 提升树。Adaboost使用sklearn实现时，默认使用决策树作为基础模型。所以默认即是提升树。

GBDT

- Boosting主流方法：
  - 3）**GBDT（梯度提升树）**：
    - GBDT(Gradient Boosting Decision Tree) 又叫 MART（Multiple Additive Regression Tree)，是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的结论累加起来做最终答案。
    - 提升树利用加法模型与前向分布算法实现学习的优化过程，当损失函数是平方损失或指数损失函数时，每一步优化是很简单的。但对一般损失函数而言，往往每一步优化并不容易，针对这一问题Friedman在2000年提出了梯度提升方法，该方法是最速下降法的近似方法，利用损失函数的负梯度作为回归树提升问题中残差的近似值，拟合一个回归树。

GBDT

- Boosting主流方法：
  - 3）**GBDT**：
    - GBDT在被提出之初就和SVM一起被认为是泛化能力较强的算法。近些年更因为被用于搜索排序的机器学习模型而引起大家关注。
    - GBDT主要由三个概念组成：Regression Decistion Tree（即DT），Gradient Boosting（即GB），Shrinkage (缩减，算法的一个重要演进分枝，目前大部分源码都按该版本实现）。

GBDT

GBDT使用sklearn实现：

- from sklearn.ensemble import GradientBoostingClassifier
- 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=’auto’, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001)
- 重要参数：
  - loss：GBDT算法中的损失函数。分类模型和回归模型的损失函数是不一样的。对于分类模型，有对数似然损失函数"deviance"和指数损失函数"exponential"两者输入选择。默认是对数似然损失函数"deviance"。在原理篇中对这些分类损失函数有详细的介绍。一般来说，推荐使用默认的"deviance"。它对二元分离和多元分类各自都有比较好的优化。而指数损失函数等于把我们带到了Adaboost算法。
  - learning_rate：学习率。默认0.1。
  - n_estimators：决策树数目。默认100。
  - subsample：子采样，取值为(0,1]。这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。

GBDT

- GBDT使用sklearn实现：

In [21]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(5):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = GradientBoostingClassifier().fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(avg_scores))
```

```
0.9606203007518797
```

XGBoost

- Boosting主流方法：
  - 4）**XGBoost**：
    - XGBoost全名叫（eXtreme Gradient Boosting）极端梯度提升，经常被用在一些比赛中，其效果显著。它是大规模并行boosted tree的工具，它是目前最快最好的开源boosted tree工具包。XGBoost 所应用的算法就是 GBDT的改进，既可以用于分类也可以用于回归问题中。

XGBoost

- 相对于GBDT，XGBoost的改进：
  - 1）将树模型的复杂度加入到正则项中，来避免过拟合，因此泛化性能会优于GBDT。
  - 2）损失函数是用泰勒展开式展开的，同时用到了一阶导和二阶导，可以加快优化速度。
  - 3）和GBDT只支持CART作为基分类器之外，还支持线性分类器，在使用线性分类器的时候可以使用L1，L2正则化。
  - 4）引进了特征子采样，像RandomForest那样，这种方法既能降低过拟合，还能减少计算。

XGBoost

- 相对于GBDT，XGBoost的改进：
  - 5）在寻找最佳分割点时，考虑到传统的贪心算法效率较低，实现了一种近似贪心算法，用来加速和减小内存消耗，除此之外还考虑了稀疏数据集和缺失值的处理，对于特征的值有缺失的样本，XGBoost依然能自动找到其要分裂的方向。
  - 6）XGBoost支持并行处理，XGBoost的并行不是在模型上的并行，而是在特征上的并行，将特征列排序后以block的形式存储在内存中，在后面的迭代中重复使用这个结构。这个block也使得并行化成为了可能，其次在进行节点分裂时，计算每个特征的增益，最终选择增益最大的那个特征去做分割，那么各个特征的增益计算就可以开多线程进行。

XGBoost

使用XGBoost包实现：

- 安装：pip install xgboost 或 conda install xgboost
- from xgboost import XGBClassifier
- XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)
- 重要参数：
  - booster ：gbtree 树模型做为基分类器（默认），gblinear 线性模型，dart引入dropout思想
  - colsample_bytree：训练每棵树时，使用的特征占全部特征的比例。默认值为1，典型值为0.5-1。
  - learning_rate：学习率，控制每次迭代更新权重时的步长，默认0.3。
  - gamma：惩罚项系数，指定节点分裂所需的最小损失函数下降值。
  - alpha：L1正则化系数，默认为1。　　lambda: L2正则化系数，默认为1

XGBoost

- 使用XGBoost包实现：

In [38]:

```
from xgboost import XGBClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

import warnings
warnings.filterwarnings(module='sklearn*', action='ignore', category=DeprecationWarning)

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(1):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = XGBClassifier().fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(score))
```

```
0.9701441102756894
```

LightGBM

- Boosting主流方法：
  - 4）**LightGBM**：
    - LightGBM（Light Gradient Boosting Machine）是一款基于决策树算法的分布式梯度提升框架。
    - 为了满足工业界缩短模型计算时间的需求，LightGBM的设计思路主要是两点：1. 减小数据对内存的使用，保证单个机器在不牺牲速度的情况下，尽可能地用上更多的数据；2. 减小通信的代价，提升多机并行时的效率，实现在计算上的线性加速。由此可见，LightGBM的设计初衷就是提供一个快速高效、低内存占用、高准确度、支持并行和大规模数据处理的数据科学工具。。

LightGBM

使用LightGBM包实现：

- 安装：pip install lightgbm 或 conda install lightgbm
- from lightgbm import LGBMClassifier
- 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)
- 重要参数：
  - boosting_type ：提升树的类型 gbdt,dart,goss,rf。
  - max_depth=-1：最大树的深度。默认-1，表示没有限制
  - learning_rate：学习率。默认0.1
  - n_estimators：拟合的树的棵树。默认100颗。

LightGBM

- 使用LightGBM包实现：

In [62]:

```
from lightgbm import LGBMClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(5):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = LGBMClassifier().fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(avg_scores))
```

```
0.9644674185463659
```

Bagging和Boosting的区别

- 1）**样本选择**：
  - Bagging：训练集是在原始集中有放回选取的，从原始集中选出的各轮训练集之间是独立的。
  - Boosting：每一轮的训练集不变，只是训练集中每个样本在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。
- 2）**样例权重**：
  - Bagging：使用均匀取样，每个样例的权重相等
  - Boosting：根据错误率不断调整样例的权值，错误率越大则权重越大。

集成分类算法

- 3）**预测函数**：
  - Bagging：所有弱分类器的权重相等。
  - Boosting：每个弱分类器都有相应的权重，对于分类误差小的分类器会有更大的权重。
- 4）**并行计算**：
  - Bagging：各个预测函数可以并行生成。
  - Boosting：各个预测函数只能顺序生成，因为后一个模型参数需要前一轮模型的结果。

问题？

暨南大學
JINAN UNIVERSITY