

# 决策树与随机森林

主讲老师：高彦杰

## 课程信息

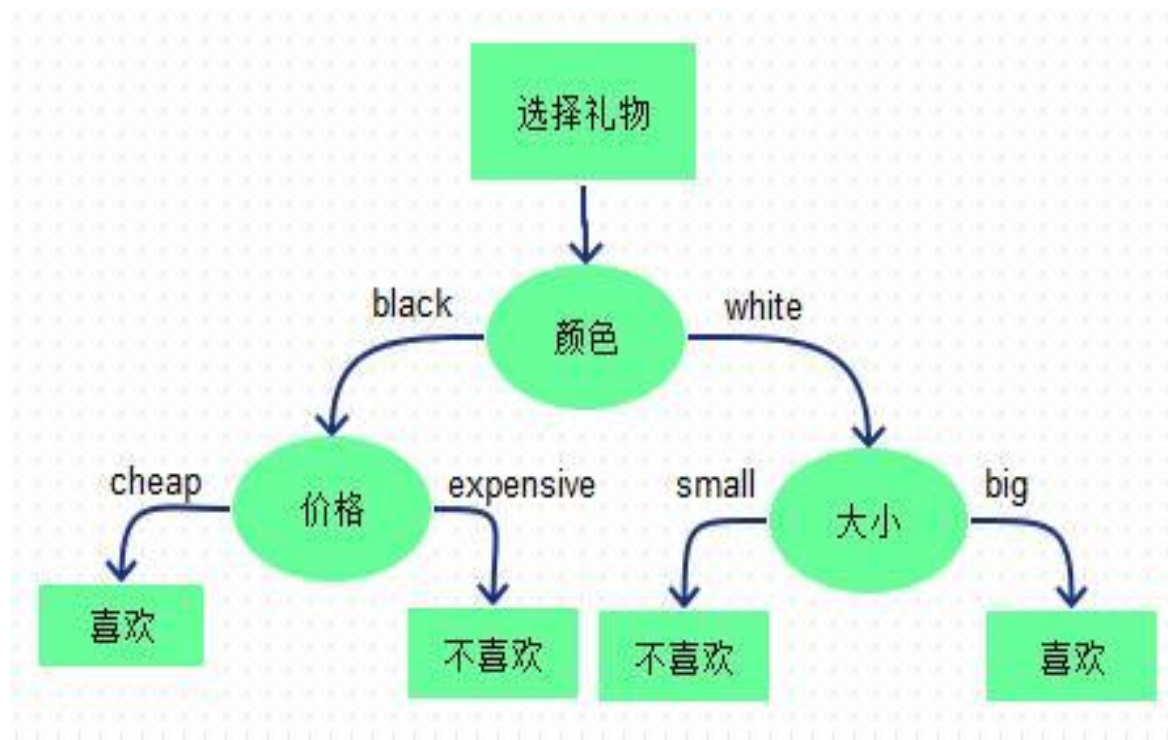
主流回归与分类算法：线性加归，逻辑回归，**Softmax**回归及其变种和扩展算法。梯度下降，牛顿法等最优化方法，正则化方法及其推导原理。通过实例掌握如何应用逻辑回等算法。

# 课程大纲

- 1) 决策树算法 (★★★)
- 2) ID3, C4.5, CART(★★★)
- 3) 随机森林(★★★)
- 4) Boost(★★)
- 5) GBDT(★★)
- 6) 集成学习(★★)
- 7) XGBoost(★★)
- 8) 案例实战(★★)

# 决策树算法

# 决策树



# 创建决策树

输入： 数据集

输出： 构造好的决策树（也即训练集）

def 创建决策树:

    if(数据集中所有样本分类一致):

        创建携带类标签的叶子节点

    else:

        寻找划分数据集最好特征

        根据最好特征划分数据集

        for 每个划分的数据集:

            创建决策子树(递归方式)

## 决策树算法衍生算法

- 决策树的关键在于当前状态下选择哪个属性作为分类条件。
- 最佳分类属性这种“最佳性”可以用非纯度 (impurity) 进行衡量。
- 如果一个数据集合中只有一种分类结果，则该集合最纯，即一致性好
- 有许多分类，则不纯，即一致性不好

## 决策树算法衍生算法

有很多指标定义不纯度，根据不同判定不纯度的目标函数：

- ✓ ID3 （信息增益）： 分类
- ✓ C4.5 （信息增益比）： 分类
- ✓ CART （GINI系数）： 分类与回归



# ID3

- ✓ 分类算法
- ✓ 熵作为衡量样本集合纯度的标准，熵越大，越不纯。
- ✓ 希望在分类以后能够降低熵的大小，变纯一些。
- ✓ 分类后熵变小可以用信息增益 (Information Gain) 来衡量。

# ID3

## 1. 计算数据集 $D$ 的经验熵 $H(D)$

$$H(D) = - \sum_{k=1}^K \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

## 2. 计算特征 $A$ 对数据集 $D$ 的经验条件熵 $H(D|A)$

$$H(D|A) = \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) = - \sum_{i=1}^n \frac{|D_i|}{|D|} \sum_{k=1}^K \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

## 3. 计算信息增益

$$g(D, A) = H(D) - H(D|A)$$

## C4.5

### ✓ 分类算法

✓ ID3的信息增益一个缺点，一般会优先选择有较多属性值的特征

✓ 解决：增加惩罚项，C4.5使用信息增益比率(gain ratio)

$$SplitInformation(D, A) = - \sum_{i=1}^N \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|}$$

$$GainRatio(D, A) = \frac{g(D, A)}{SplitInformation(D, A)}$$

# CART

## CART算法 (Classification and Regression Tree)

- ✓ 可以进行分类和回归
- ✓ 分类树使用最小GINI指数来选择划分属性
- ✓ 回归树使用 Mean Squared Error:

基尼系数

$$Gini(D) = 1 - \sum_{i=0}^n \left( \frac{D_i}{D} \right)^2$$

$$Gini(D|A) = \sum_{j=0}^k \frac{D_j}{D} Gini(D_j)$$

*MSE*

$$\min \left[ \frac{1}{n_{R1}} \sum_{x_i \in R_1} (y_i - c_1)^2 + \frac{1}{n_{R2}} \sum_{x_i \in R_2} (y_i - c_2)^2 \right]$$

# 剪枝算法

- 预剪枝

- ✓ 在构造决策树的同时进行剪枝。
- ✓ 所有决策树的构建方法，都是在无法进一步降低熵的情况下才会停止创建分支的过程，为了避免过拟合，可以设定一个阈值。
- ✓ 例如：熵减小的数量小于这个阈值，即使还可以继续降低熵，也停止继续创建分支。

- 后剪枝

- ✓ 决策树构造完成后进行剪枝。
- ✓ 剪枝的过程是对拥有同样父节点的一组节点进行检查，判断如果将其合并，熵的增加量是否小于某一阈值。
- ✓ 如果满足阈值要求，则这一组节点可以合并一个节点，其中包含了所有可能的结果。

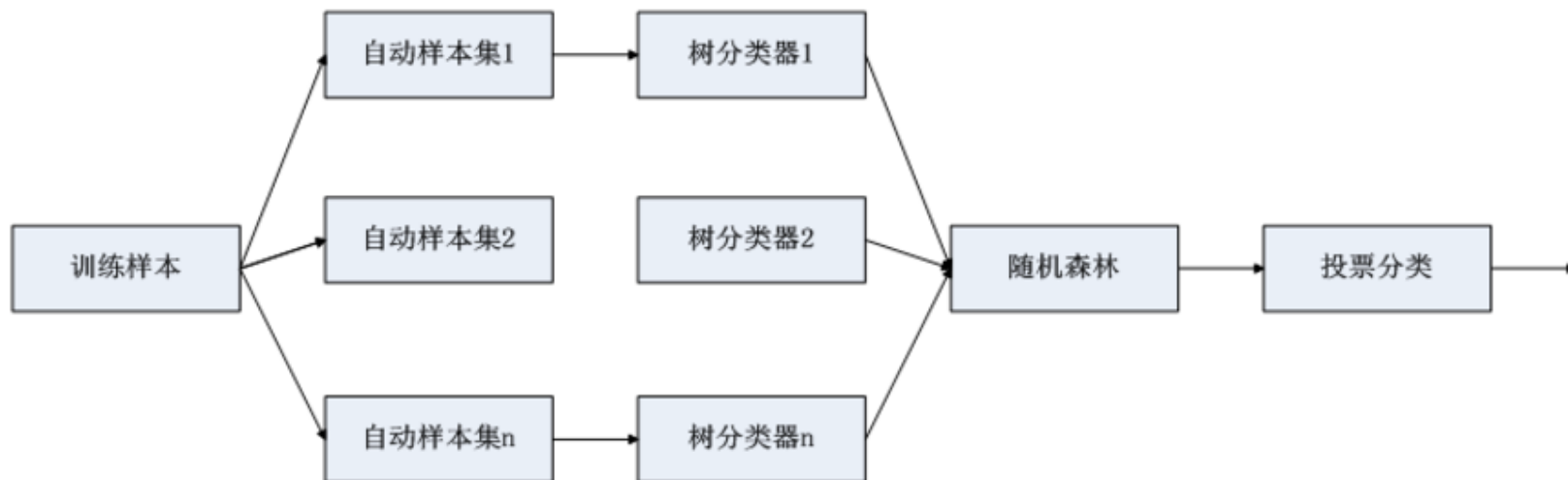
# 后剪枝-示例

- **Reduced-Error Pruning (REP, 错误率降低剪枝)**
  - 思路：决策树过度拟合后，通过测试数据集来纠正。
  - 步骤：
    1. 对于完树中每一个非叶子节点的子树，尝试着把它替换成一个叶子节点
    2. 该叶子节点的类别用子树覆盖训练样本中存在最多的那个类来代替，产生简化决策树
    3. 然后比较这两个决策树在测试数据集中的表现
    4. 简化决策树在测试数据集中的错误比较少，那么该子树就可以替换成叶子节点
    5. 以bottom-up的方式遍历所有的子树，当没有任何子树可以替换提升，算法终止

随机森林

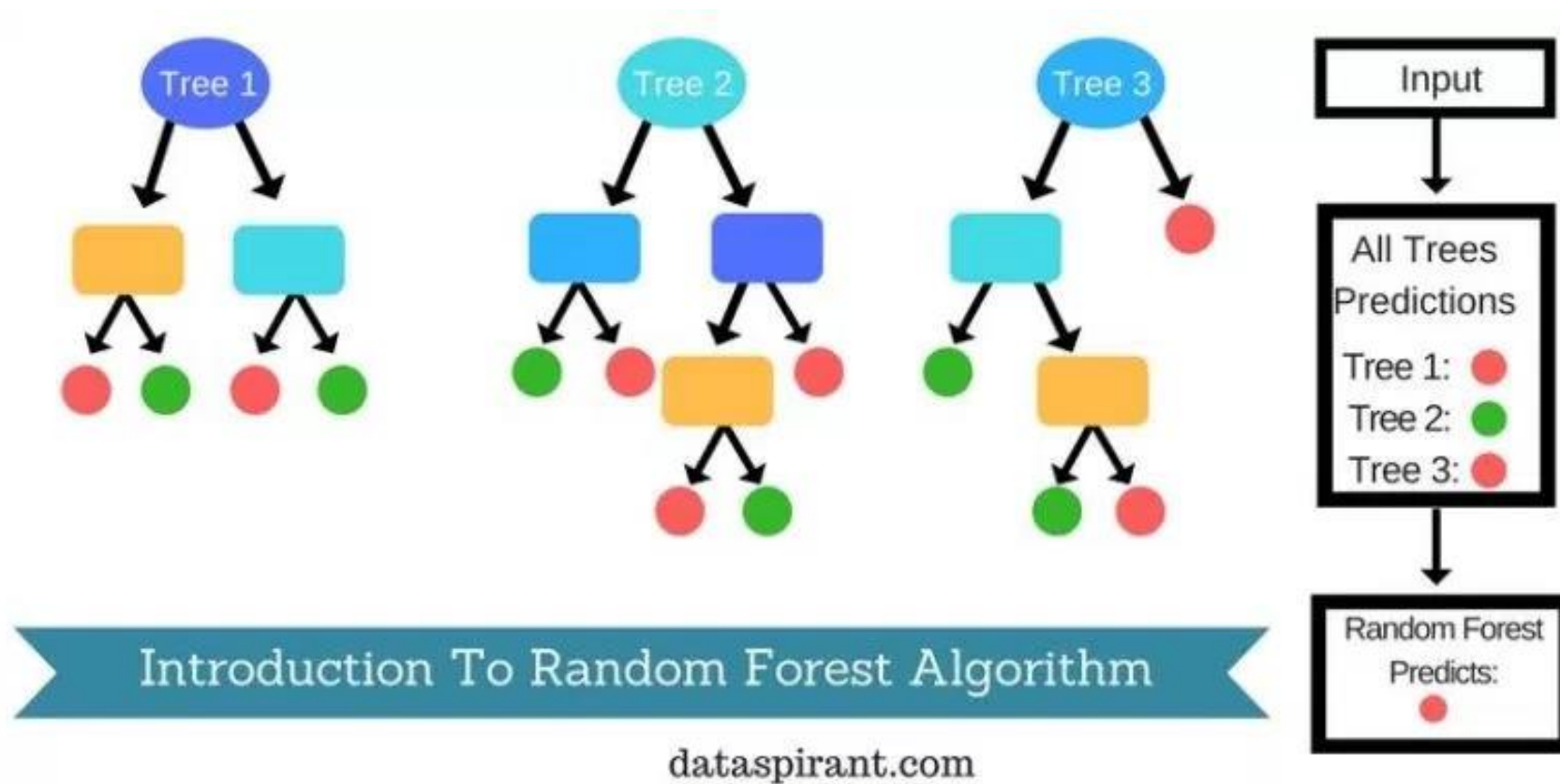
# 随机森林

- 随机森林以随机的方式建立一个森林
- 森林里有很多决策树，且每棵树之间无关联
- 当有一个新样本进入后，让森林中每棵决策树分别各自独立判断，看这个样本应该属于哪一类（分类算法）
- 然后看哪一类被选择最多，就选择预测此样本为那一类





# 随机森林



## 随机森林构造过程

- (1) 原始训练集为 $N$ ，应用bootstrap法有放回地随机抽取 $k$ 个新的自助样本集，并由此构建 $k$ 棵分类树，每次未被抽到的样本组成了 $k$ 个袋外数据**
- (2) 设有 $m$ 个变量，则在每一棵树的每个节点处随机抽取 $m_{try}$ 个变量，然后在 $m_{try}$ 中选择一个最具有分类能力的变量，变量分类的阈值通过检查每一个分类点确定**
- (3) 每棵树最大限度地生长，不做任何修剪**
- (4) 将生成的多棵分类树组成随机森林，用随机森林分类器对新的数据进行判别与分类，分类结果按树分类器的投票多少而定**

# 随机森林

f11	f12	f13	f14	f15	t1
f21	f22	f23	f24	f25	t2
f31	f32	f33	f34	f35	t3
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
fm1	fm2	fm3	fm4	fm5	tm

Dataset

f11	f12	f13	f14	f15	t1
f81	f82	f83	f84	f85	t8
f71	f72	f73	f74	f75	t7
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
fj1	fj2	fj3	fj4	fj5	tj

Random Dataset  
for Tree-01

Random Dataset  
for Tree-02

f21	f22	f23	f24	f25	t2
f51	f52	f53	f54	f55	t5
f31	f32	f33	f34	f35	t3
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
fm1	fm2	fm3	fm4	fm5	tm

f31	f32	f33	f34	f35	t3
f61	f62	f63	f64	f65	t6
f91	f92	f73	f94	f95	t9
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
fk1	fk2	fk3	fk4	fk5	tk

Random Dataset  
for Tree-03

# 随机森林优缺点

## 优点:

- 1.适用数据集广
- 2.高维数据
- 3.Feature重要性排序
- 4.训练速度快，并行化

## 缺点:

- 1.级别划分较多的属性影响大

# Scikit-Learn RandomForest

RandomForestClassifier

RandomForestRegressor

分类与回归模型

model = fit(x, y)

**X : array-like or sparse matrix of shape = [n\_samples, n\_features]**

The training input samples. Internally, its dtype will be converted to `dtype=np.float32`.

训练-输入与输出格式

**y : array-like, shape = [n\_samples] or [n\_samples, n\_outputs]**

The target values (class labels in classification, real numbers in regression).

**Returns:** `self : object`

results = predict(x)

**X : array-like or sparse matrix of shape = [n\_samples, n\_features]**

The input samples. Internally, its dtype will be converted to `dtype=np.float32`.

预测-输入与输出格式

**Returns:** `y : array of shape = [n_samples] or [n_samples, n_outputs]`

The predicted classes.

## 企业案例分享

- ✓ 场景：入侵检测
- ✓ 挑战
- ✓ 问题建模：分类问题
- ✓ 算法选型：随机森林



**Boost**

# Boost算法

- 决策树：单决策树时间复杂度较低，模型容易展示，但是容易Over-fitting
  - ✓ 分类树
  - ✓ 回归树
- 决策树的Boost方法：迭代过程，新的训练为了改进上一次的结果
  - ✓ 传统Boost: 对正确、错误的样本进行加权，每一步结束后，增加分错点的权重，减少对分对点的权重
  - ✓ GB: 梯度迭代Gradient Boosting，每一次建立模型是在之前建立的模型损失函数的梯度下降方向



# Adaboost算法

- ✓ Adaboost的核心思想
- ✓ 关注被错分的样本，器重性能好的分类器
- ✓ 如何实现
  - ✓ 不同的训练集 -> 调整样本权重
  - ✓ 关注 -> 增加错分样本权重
  - ✓ 器重 -> 好的分类器权重大
  - ✓ 样本权重间接影响分类器权重

# Adaboost算法步骤

For  $t=1, \dots, T$

1. Train learner  $h_t$  with **min error**  $\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$

若划分正确，则不计入误差，若所有元素都被正确划分，则误差为0

若划分错误，则计入误差

2. If  $\varepsilon_t \geq 0.5$ , then stop

3. Compute the hypothesis weight  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$

The weight **Adapts**. The bigger  $\varepsilon_t$  becomes the smaller  $\alpha_t$  becomes.

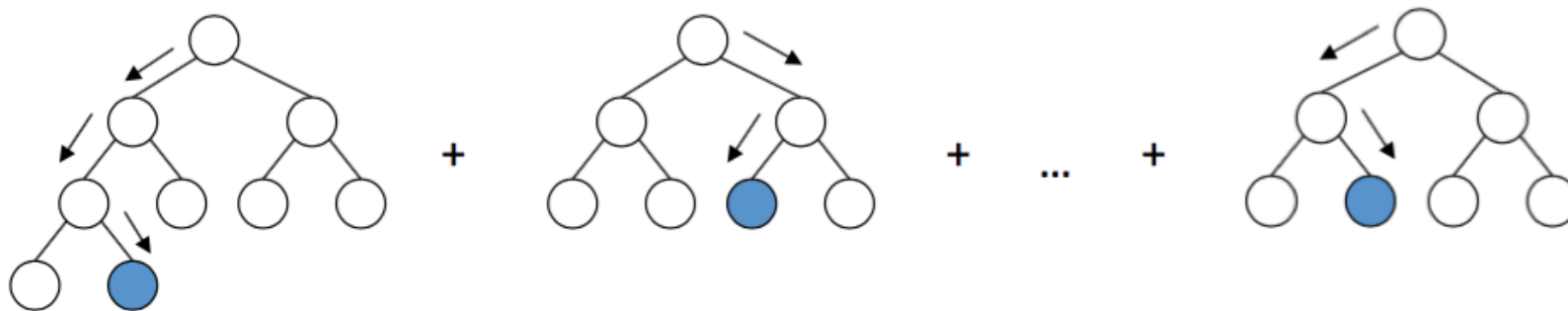
4.  $D_{t+1}(i) = D_t(i) \exp(\alpha_t * 1_{(h_t(i) \neq y_i)}) = \begin{cases} D_t(i), & \text{若 } y_i = h_t(x_i) \\ D_t(i) \frac{1 - \varepsilon_t}{\varepsilon_t}, & \text{若 } y_i \neq h_t(x_i) \end{cases}$

- 5.最后得到的强分类器:  $H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$

**GBDT**

# GBDT(Gradient Boosting Decision Tree)算法

- ✓ 用一个初始值来学习一棵决策树，叶子处可以得到预测的值，以及预测之后的残差，然后后面的决策树就要基于前面决策树的残差来学习，直到预测值和真实值的残差为零。
- ✓ 最后对于测试样本的预测值，就是前面许多棵决策树预测值的累加。



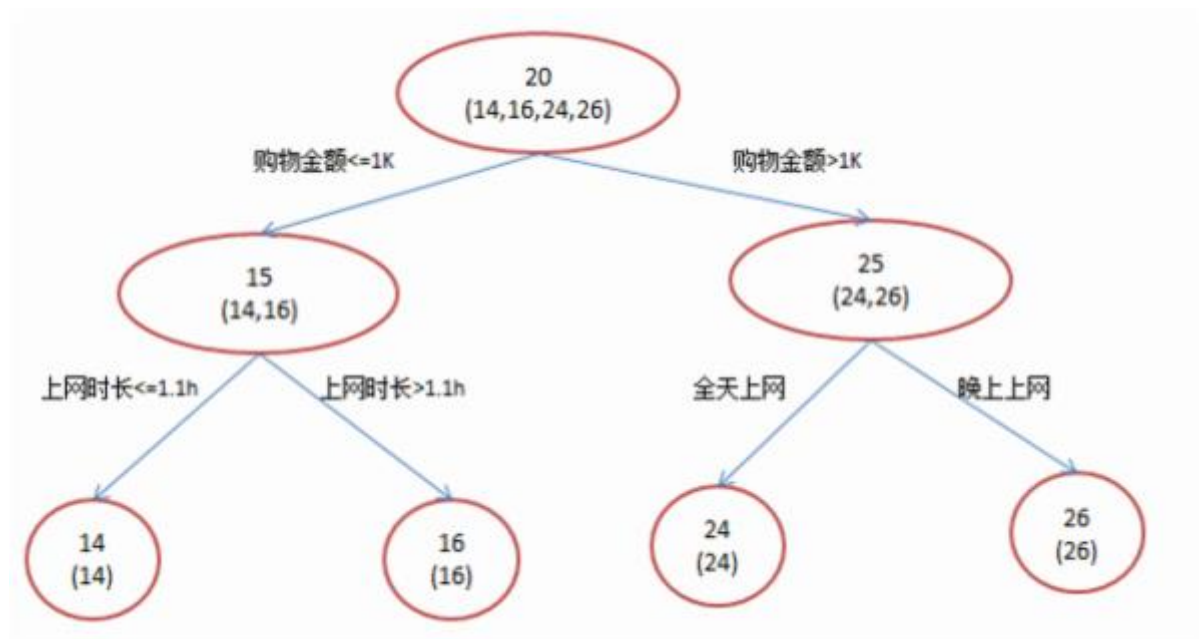
# GBDT算法应用场景

- ✓ 适用问题广，可扩展性好
- ✓ GBDT几乎可以用于所有回归问题（线性、非线性）
- ✓ 分类问题
- ✓ 排序问题
- ✓ 常用于各大数据挖掘竞赛
- ✓ 广告推荐

# GBDT实例

训练集：(A, 14岁)、(B, 16岁)、(C, 24岁)、(D, 26岁)；

下面为决策树示例



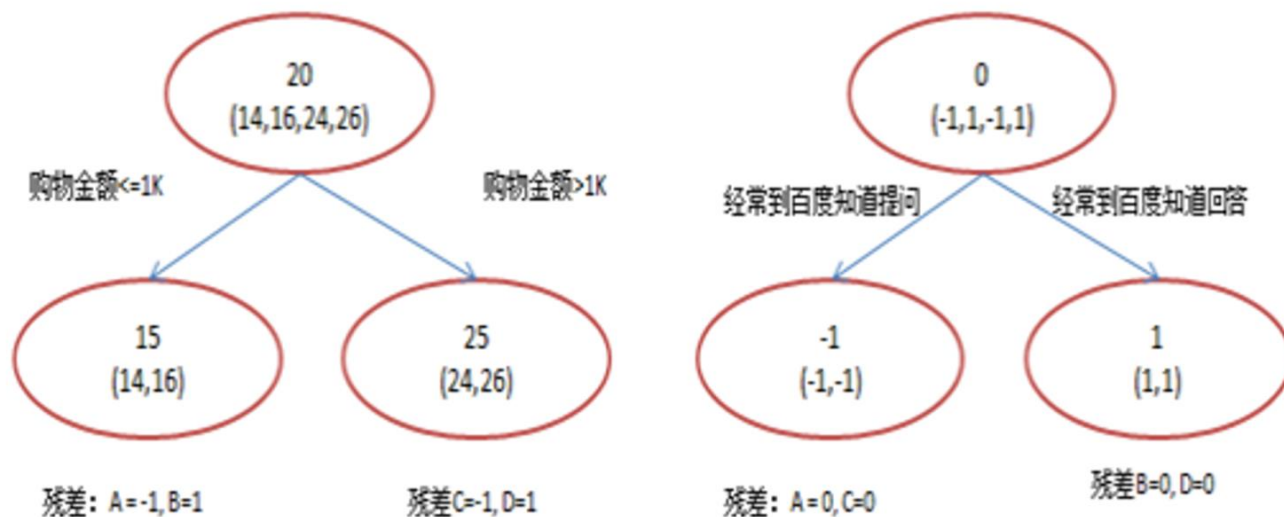
# GBDT实例

A: 14岁高一学生, 购物较少, 经常问学长问题; 预测年龄 $A = 15 - 1 = 14$

B: 16岁高三学生; 购物较少, 经常被学弟问问题; 预测年龄 $B = 15 + 1 = 16$

C: 24岁应届毕业生; 购物较多, 经常问师兄问题; 预测年龄 $C = 25 - 1 = 24$

D: 26岁工作两年员工; 购物较多, 经常被师弟问问题; 预测年龄 $D = 25 + 1 = 26$



## GBDT算法

### Algorithm 1: Gradient\_Boost

```
1  $F_0(\mathbf{x}) = \arg \min_{\rho} \sum_{i=1}^N L(y_i, \rho)$   
2 For  $m = 1$  to  $M$  do:  
3    $\tilde{y}_i = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, N$   
4    $\mathbf{a}_m = \arg \min_{\mathbf{a}, \beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$   
5    $\rho_m = \arg \min_{\rho} \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$   
6    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$   
7 endFor  
end Algorithm
```



# Random Forest VS GBDT

- **准确度与拟合：** GBDT > RF better accuracy with less trees. RF are harder to overfit than GBDT
- **建模能力：** GBDT > RF because boosted trees are derived by optimizing a objective function, basically it can be used to solve almost all objective you can write gradient out.
- **并行化：** Random Forests can be easily deployed in a distributed fashion due to the fact that they can run in parallel, whereas Gradient Boosted Machines only run trial after trial.

# 企业案例分享

✓ 场景：

Facebook互联网广告CTR预估

✓ 挑战：

✓ LR模型中的特征组合

✓ 依靠人工经验，耗时耗力同时并不一定会带来效果提升

✓ 如何自动发现有效的特征、特征组合，弥补人工经验不足



## 企业案例分享

✓ 背景:



In sponsored search advertising, the user query is used to retrieve candidate ads, which explicitly or implicitly are matched to the query. At Facebook, ads are not associated with a query, but instead specify demographic and interest targeting. As a consequence of this, the volume of ads that are eligible to be displayed when a user visits Facebook can be larger than for sponsored search.

# 企业案例分享

## ✓ 解决方案

index of the leaf an instance ends up falling in. We use 1-of-K coding of this type of features. For example, consider the boosted tree model in Figure 1 with 2 subtrees, where the first subtree has 3 leafs and the second 2 leafs. If an instance ends up in leaf 2 in the first subtree and leaf 1 in second subtree, the overall input to the linear classifier will be the binary vector  $[0, 1, 0, 1, 0]$ , where the first 3 entries correspond to the leaves of the first subtree and last 2 to those of the second subtree. The boosted decision trees we

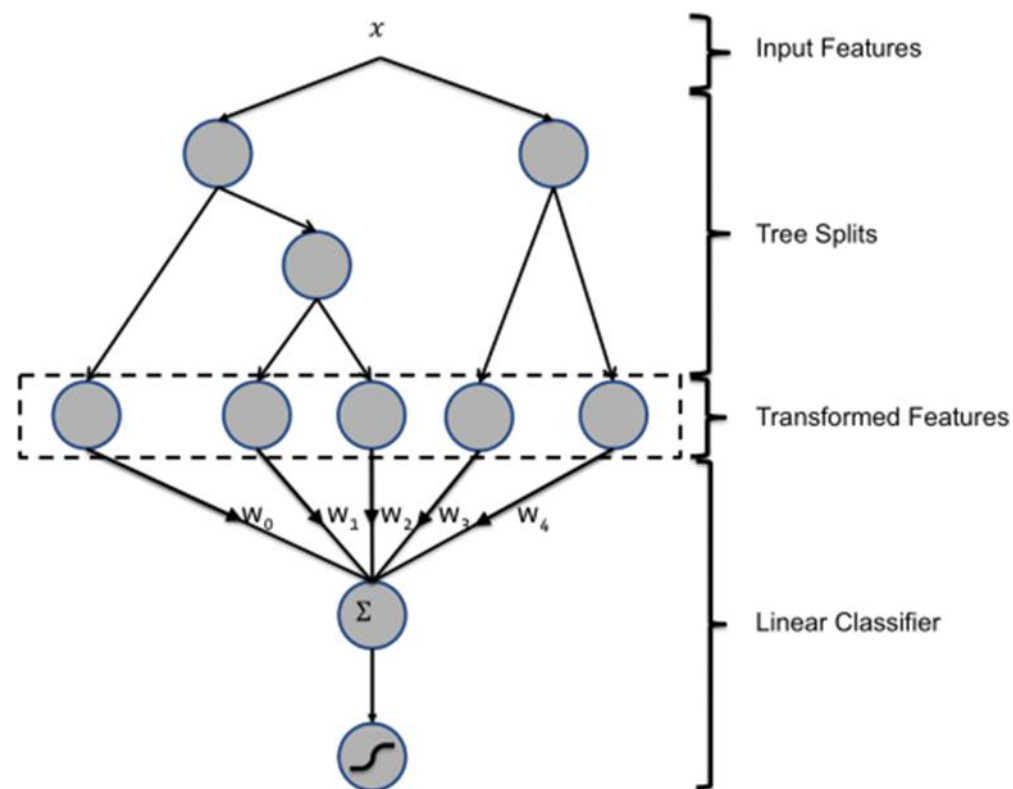
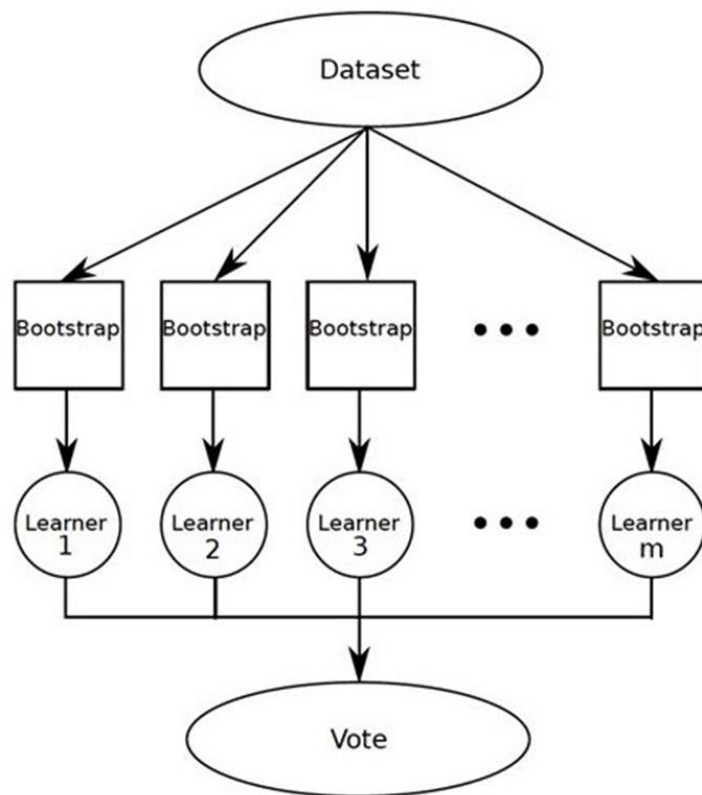


Figure 1: Hybrid model structure. Input features are transformed by means of boosted decision trees. The output of each individual tree is treated as a categorical input feature to a sparse linear classifier. Boosted decision trees prove to be very powerful feature transforms.

# 集成学习

# 集成学习 Bagging

让该学习算法训练多轮，每轮的训练集由从初始的训练集中随机取出的 $n$ 个训练样本组成，某个初始训练样本在某轮训练集中可以出现多次或根本不出现，训练之后可得到一个预测函数序列 $h_1, \dots, h_n$ ，最终的预测函数 $H$ 对分类问题采用投票方式，对回归问题采用简单平均方法对新示例进行判别。



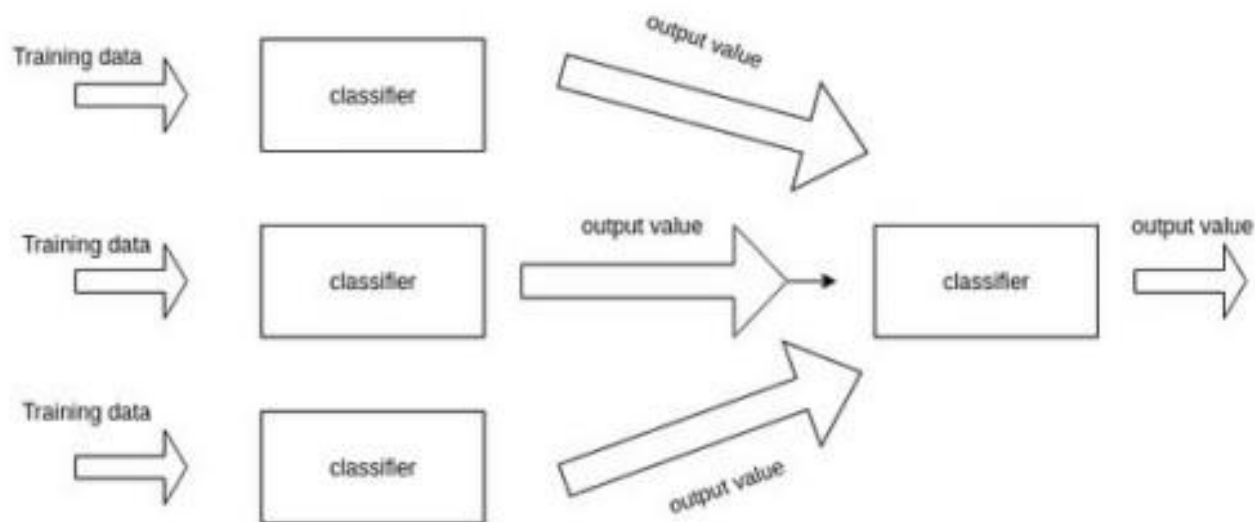
# 集成学习Boosting

- 初始化时对每一个训练例赋相等的权重  $1 / n$ ,
- 然后用该学算法对训练集训练  $t$  轮
- 每次训练后, 对训练失败的训练例赋以较大的权重, 也就是让学习算法在后续的学习中集中对比较难的训练例进行学习, 从而得到一个预测函数序列  $h_1, \dots, h_m$ , 其中  $h_i$  也有一定的权重, 预测效果好的预测函数权重较大, 反之较小。
- 最终的预测函数  $H$  对分类问题采用有权重的投票方式, 对回归问题采用加权平均的方法对新示例进行判别。

# 集成学习 Stacking

将训练好的所有基模型对训练基进行预测，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





**XGboost**

# XGBoost

- XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient, flexible and portable**.
- It implements machine learning algorithms under the Gradient Boosting framework.
- XGBoost provides a **parallel tree boosting** (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.
- The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

官网: <https://github.com/dmlc/xgboost>

dmlc  
**XGBoost** eXtreme Gradient Boosting

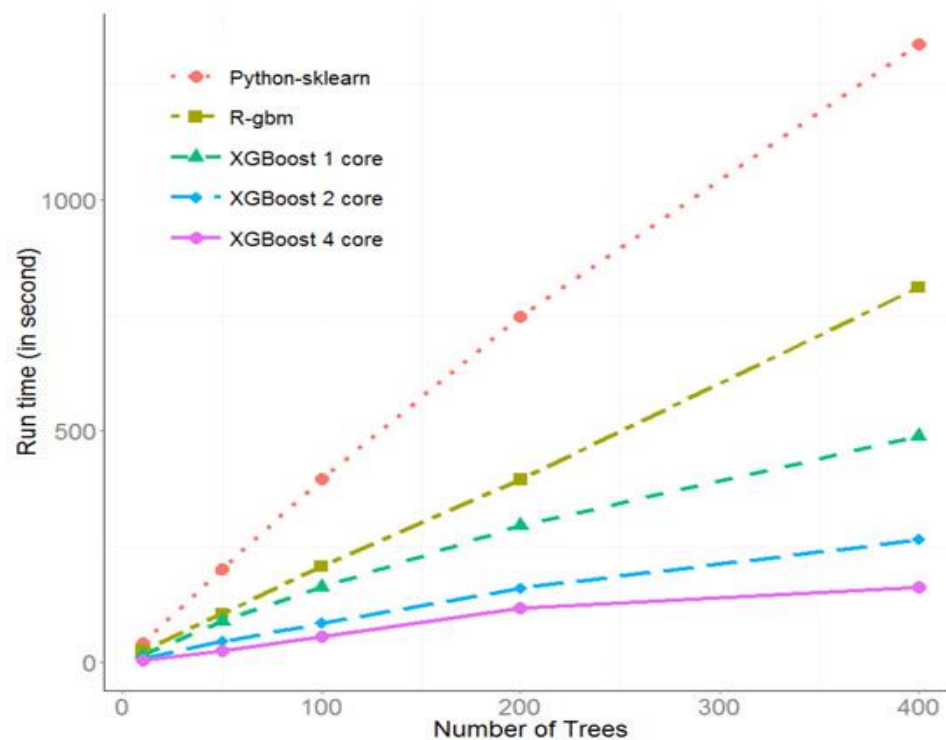
# XGBoost

- ✓ Boosting分类器将成百上千个分类准确率较低的树模型组合起来，成为一个准确率很高的模型。
- ✓ 数据集较大较复杂的时候，我们可能需要几千次迭代运算，这将造成巨大的计算瓶颈。
- ✓ XGboost正是为了解决这个瓶颈而提出。单机它采用多线程来加速树的构建，并可以进行分布式计算。
- ✓ xgboost提供了 Python和R语言接口。

# XGBoost

Kaggle的希格斯子竞赛数据上

- ✓ 单线程xgboost比其他两个包均要快出50%
- ✓ 在多线程上xgboost更是有接近线性的性能提升



## 案例实战

# 预习/复习内容重点与难点

✓ 重点:

✓ 1. 线性回归

✓ 2. 梯度下降

✓ 3. 逻辑回归

✓ 4. Softmax函数

✓ 难点:

✓ 随机梯度下降



## 参考资料

- Induction of Decision Trees
- An introduction to random forests
- Ensemble Learning
- 统计学习方法