

人工智能之机器学习

集成学习：随机森林、GBDT

主讲人：李老师

课程内容

- 集成算法
- 随机森林
- 提升算法
- GBDT(迭代决策树)
- Adaboost

集成学习(Ensemble Learning)

- 集成学习的思想是将若干个学习器(分类器&回归器)组合之后产生一个新学习器。弱分类器(weak learner)指那些分类准确率只稍微好于随机猜测的分类器($\text{error rate} < 0.5$);
- 集成算法的成功在于保证弱分类器的多样性(Diversity)。而且集成不稳定的算法也能够得到一个比较明显的性能提升。
- 常见的集成学习思想有:
 - Bagging
 - Boosting
 - Stacking

集成学习(Ensemble Learning)

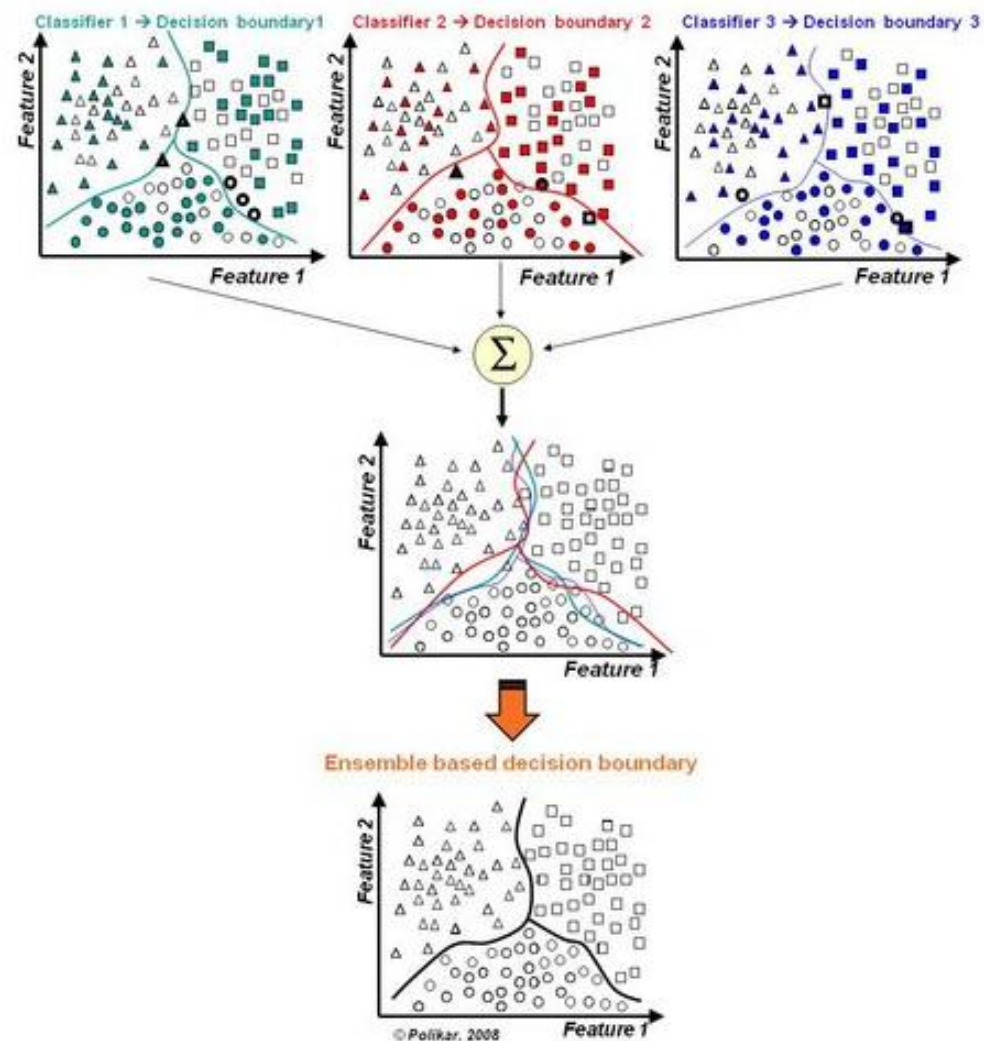


Figure 1: Combining an ensemble of classifiers for reducing classification error and/or model selection.

Why need Ensemble Learning?

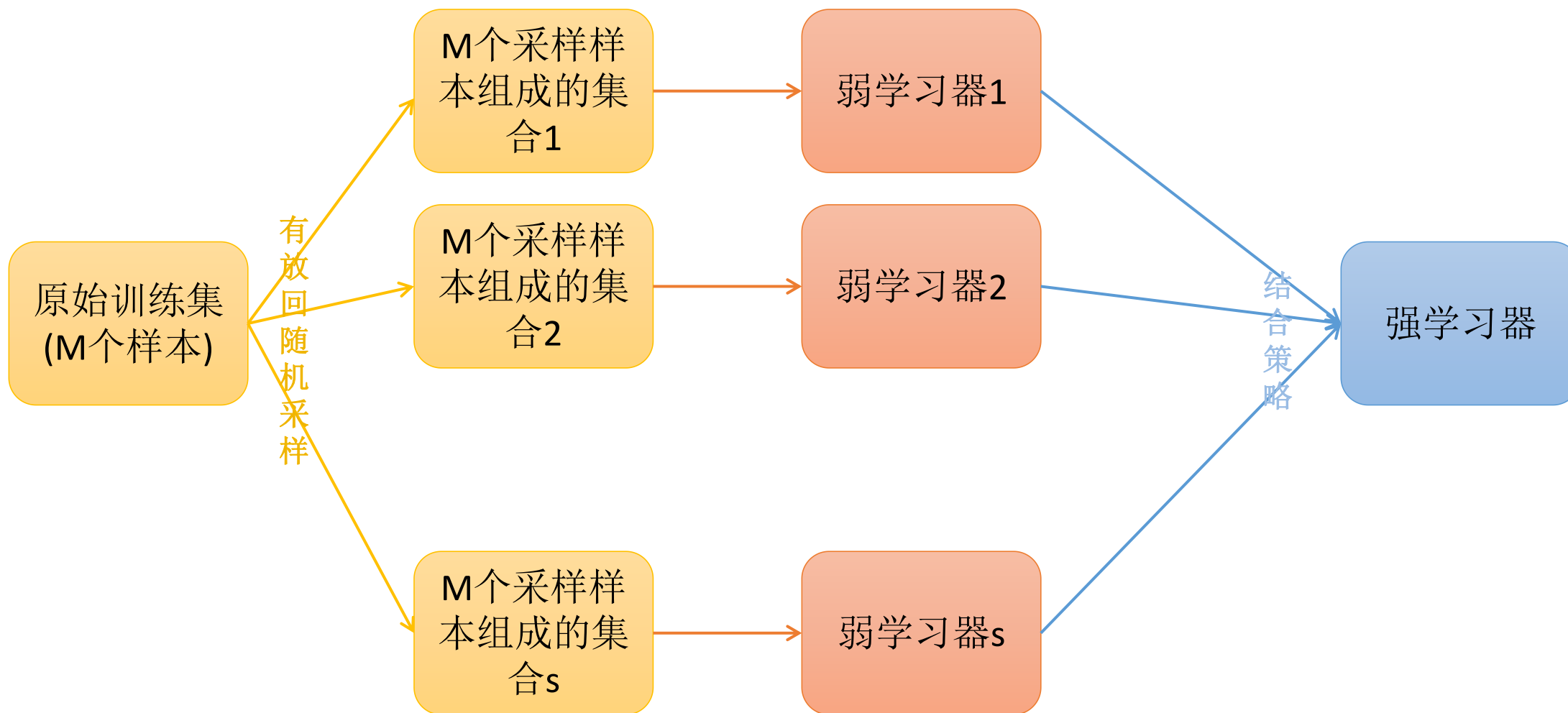
- 1. 弱分类器间存在一定的差异性，这会导致分类的边界不同，也就是说可能存在错误。那么将多个弱分类器合并后，就可以得到更加合理的边界，减少整体的错误率，实现更好的效果；
- 2. 对于数据集过大或者过小，可以分别进行划分和有放回的操作产生不同的数据子集，然后使用数据子集训练不同的分类器，最终再合并成为一个大的分类器；
- 3. 如果数据的划分边界过于复杂，使用线性模型很难描述情况，那么可以训练多个模型，然后再进行模型的融合；
- 4. 对于多个异构的特征集的时候，很难进行融合，那么可以考虑每个数据集构建一个分类模型，然后将多个模型融合。

Bagging方法

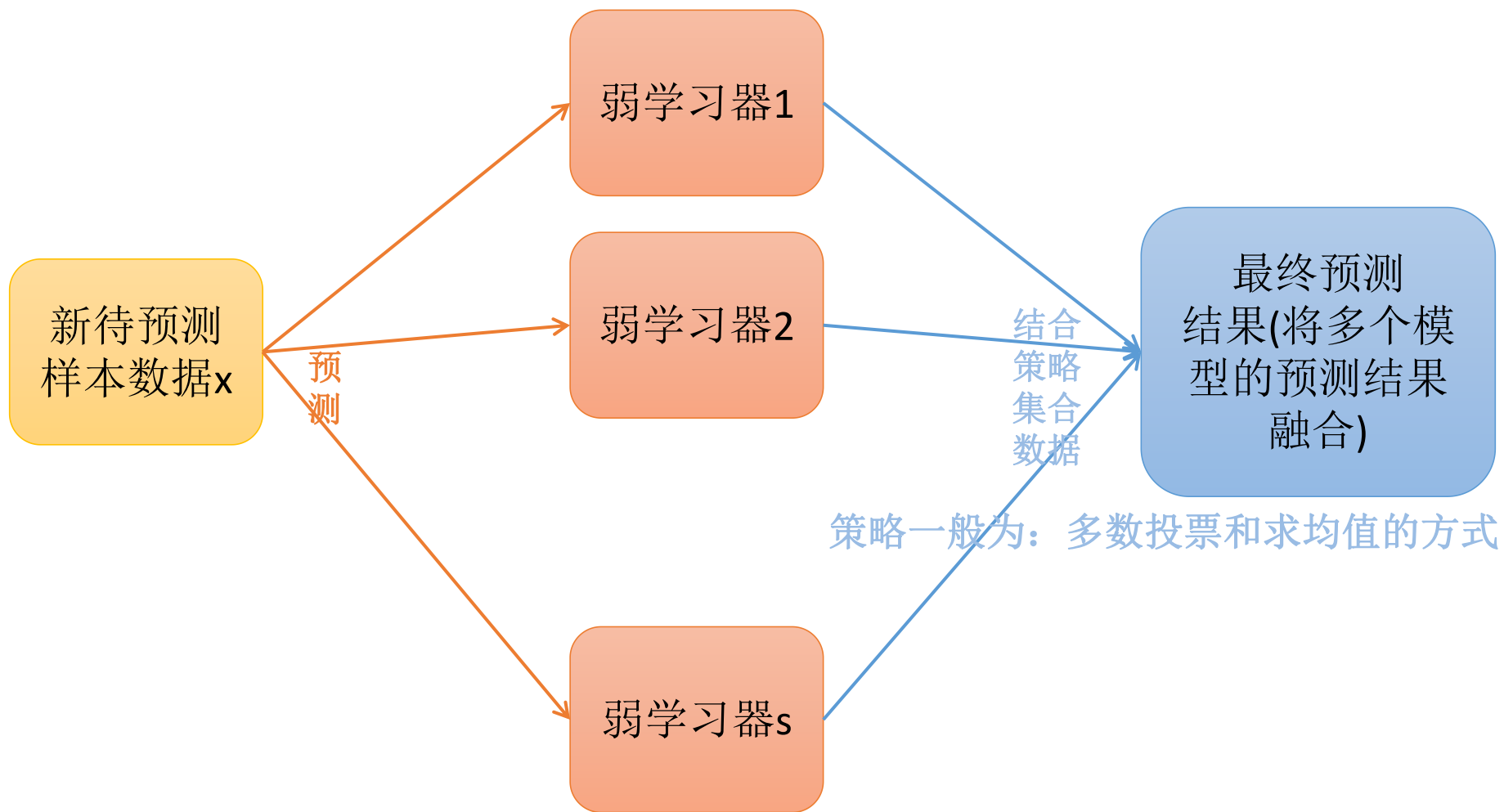
- Bagging方法又叫做自举汇聚法(Bootstrap Aggregating), 思想是: 在原始数据集上通过**有放回的抽样**的方式, 重新选择出S个新数据集来分别训练S个分类器的集成技术。
- Bagging方法训练出来的模型在预测新样本分类/回归的时候, 会使用**多数投票**或者**求均值**的方式来统计最终的分类/回归结果。
- Bagging方法的弱学习器可以是基本的算法模型, eg: Linear、Ridge、Lasso、Logistic、Softmax、ID3、C4.5、CART、SVM、KNN等。
- NOTE: Bagging方式是有放回的抽样, 并且每个子集的样本数量必须和原始样本数量一致, 所以抽取出来的子集中是存在重复数据的, **模型训练的时候允许存在重复数据**。
- NOTE: 差不多有1/3的样本数据是不在Bagging的每个子模型的训练数据中的。

$$\lim_{m \rightarrow +\infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e} \approx 0.368$$

Bagging方法_训练过程



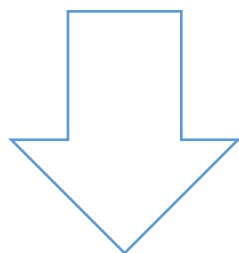
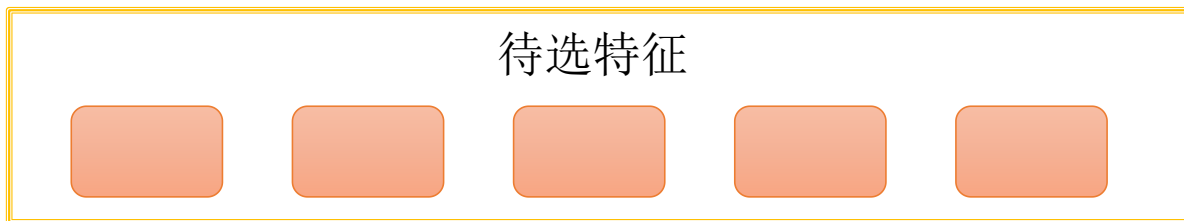
Bagging方法_预测过程



随机森林(Random Forest)

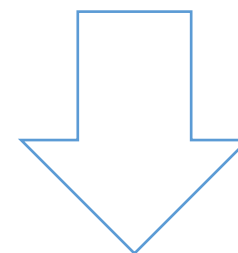
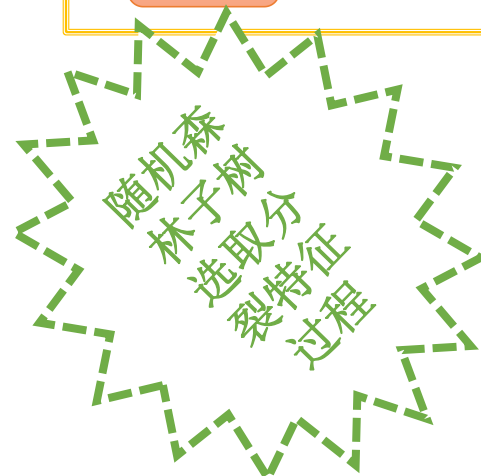
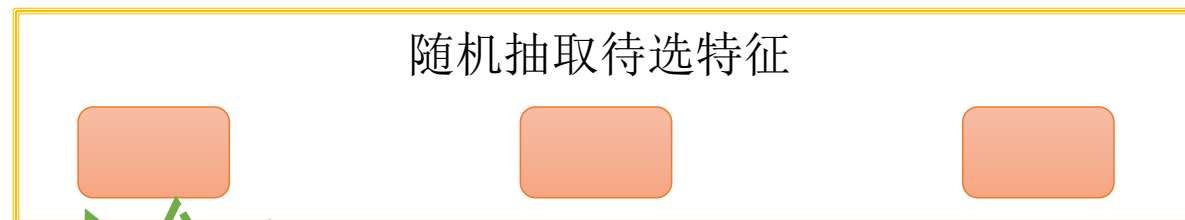
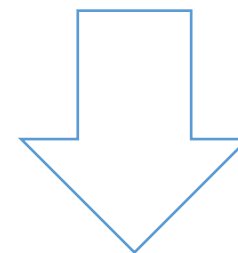
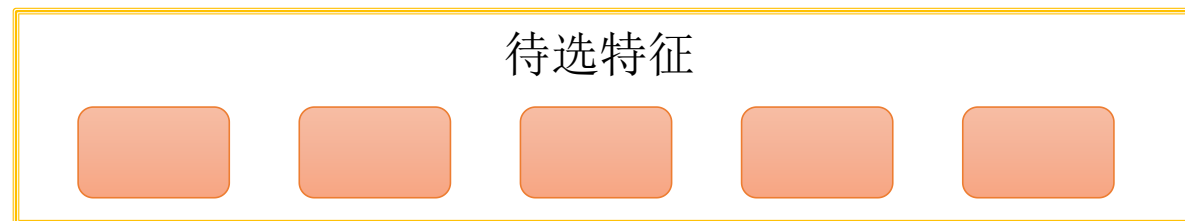
- 在Bagging策略的基础上进行修改后的一种算法
 - 1. 从原始样本集(n 个样本)中用Bootstrap采样(有放回重采样)选出 n 个样本;
 - 2. 使用抽取出来的子数据集(存在重复数据)来训练决策树; 从所有属性中随机选择 K 个属性, 从 K 个属性中选择出最佳分割属性作为当前节点的划分属性, 按照这种方式来迭代的创建决策树。
 - 3. 重复以上两步 m 次, 即建立 m 棵决策树;
 - 4. 这 m 个决策树形成随机森林, 通过投票表决结果决定数据属于那一类

随机森林(Random Forest)



分裂
特征

An orange rounded rectangle containing the text '分裂特征' (Split Feature).



分裂
特征

An orange rounded rectangle containing the text '分裂特征' (Split Feature).

RF的推广算法

- RF算法在实际应用中具有比较好的特性，应用也比较广泛，主要应用在：分类、回归、特征转换、异常点检测等。常见的RF变种算法如下：
 - Extra Tree
 - Totally Random Trees Embedding(TRTE)
 - Isolation Forest

Extra Tree

- Extra Tree是RF的一个变种，原理基本和RF一样，区别如下：
 - 1. RF会随机重采样来作为子决策树的训练集，而Extra Tree每个子决策树采用原始数据集训练；
 - 2. RF在选择划分特征点的时候会 and 传统决策树一样，会基于信息增益、信息增益率、基尼系数、均方差等原则来选择最优特征值；而Extra Tree会随机的选择一个特征值来划分决策树。
- Extra Tree因为是随机选择特征值的划分点，这样会导致决策树的规模一般大于RF所生成的决策树。也就是说Extra Tree模型的方差相对于RF进一步减少。在某些情况下，Extra Tree的泛化能力比RF的强。

Totally Random Trees Embedding(TRTE)

- TRTE是一种非监督的数据转化方式。将低维的数据集映射到高维，从而让映射到高维的数据更好的应用于分类回归模型。
- TRTE算法的转换过程类似RF+KDTree算法的方法，建立T个决策树来拟合数据(是类似KD-Tree一样基于特征属性的方差选择划分特征)。当决策树构建完成后，数据集里的每个数据在T个决策树中叶子节点的位置就定下来了，将位置信息转换为向量就完成了特征转换操作。
- 案例：有3棵决策树，各个决策树的叶子节点数目分别为:5,5,4，某个数据x划分到第一个决策树的第3个叶子节点，第二个决策树的第一个叶子节点，第三个决策树的第四个叶子节点，那么最终的x映射特征编码为:(0,0,1,0,0, 1,0,0,0,0, 0,0,0,1)

0 0 1 0 0

1 0 0 0 0

0 0 0 1

Isolation Forest(IForest)

- IForest是一种异常点检测算法，使用类似RF的方式来检测异常点；IForest算法和RF算法的区别在于：
 - 1. 在随机采样的过程中，一般只需要少量数据即可；
 - 2. 在进行决策树构建过程中，IForest算法会随机选择一个划分特征，并对划分特征随机选择一个划分阈值；
 - 3. IForest算法构建的决策树一般深度max_depth是比较大的。
- 区别原因：目的是异常点检测，所以只要能够区分异常的即可，不需要大量数据；

Isolation Forest(IForest)

- 对于异常点的判断，则是将测试样本 x 拟合到 m 棵决策树上。计算在每棵树上该样本的叶子节点的深度 $h_t(x)$ 。从而计算出平均深度 $h(x)$ ；然后就可以使用下列公式计算样本点 x 的异常概率值， $p(s,m)$ 的取值范围为 $[0,1]$ ，越接近于1，则是异常点的概率越大。备注：如果落在的叶子节点为正常样本点，那么当前决策树不考虑，如果所有决策树上都是正常样本点，那么直接认为异常点概率为0.

$$p(x, m) = 2^{-\frac{h(x)}{c(m)}}$$

$$c(m) = 2 \ln(m-1) + \xi - 2 \frac{m-1}{m}; \quad m \text{ 为样本个数, } \xi \text{ 为欧拉常数}$$

RF随机森林总结

- RF的主要优点：
 - 1. 训练可以并行化，对于大规模样本的训练具有速度的优势；
 - 2. 由于进行随机选择决策树划分特征列表，这样在样本维度比较高的时候，仍然具有比较高的训练性能；
 - 3. 给以给出各个特征的重要性列表；
 - 4. 由于存在随机抽样，训练出来的模型方差小，泛化能力强，能够缓解过拟合的情况；
 - 5. RF实现简单；
 - 6. 对于部分特征的缺失不敏感。
- RF的主要缺点：
 - 1. 在某些噪音比较大的特征上（数据特别异常情况），RF模型容易陷入过拟合；
 - 2. 取值比较多的划分特征对RF的决策会产生更大的影响，从而有可能影响模型的效果。

RF scikit-learn相关参数

参数	RandomForestClassifier	RandomForestRegressor
criterion	指定划分标准，默认为gini，不支持其它参数	指定划分标准，可选"mse"和"mae"; 默认mse
loss	不支持	指定误差的计算方式，可选参数"linear", "square", "exponential", 默认为"linear"; 一般不用改动
n_estimators	最大迭代次数，也就是最多允许的决策树的数目，值过小可能会导致欠拟合，值过大可能会导致过拟合，一般50~100比较适合，默认10	
max_features	给定在进行最佳特征划分的时候，选择多少个特征进行考虑；默认为auto；max_features=sqrt(n_features)；一般不建议改动，具体参数见官网文档。	
max_depth	给定树的深度，默认为None，表示一致扩展到叶子节点足够纯或者样本数小于min_samples_split	
min_samples_split	给定树构建过程中，叶子节点中最少样本数量，默认为2	
min_samples_leaf	给定每个叶子节点中，最少的样本数目是多少，默认为2	
bootstrap	是否进行有放回的重采样，默认为True	

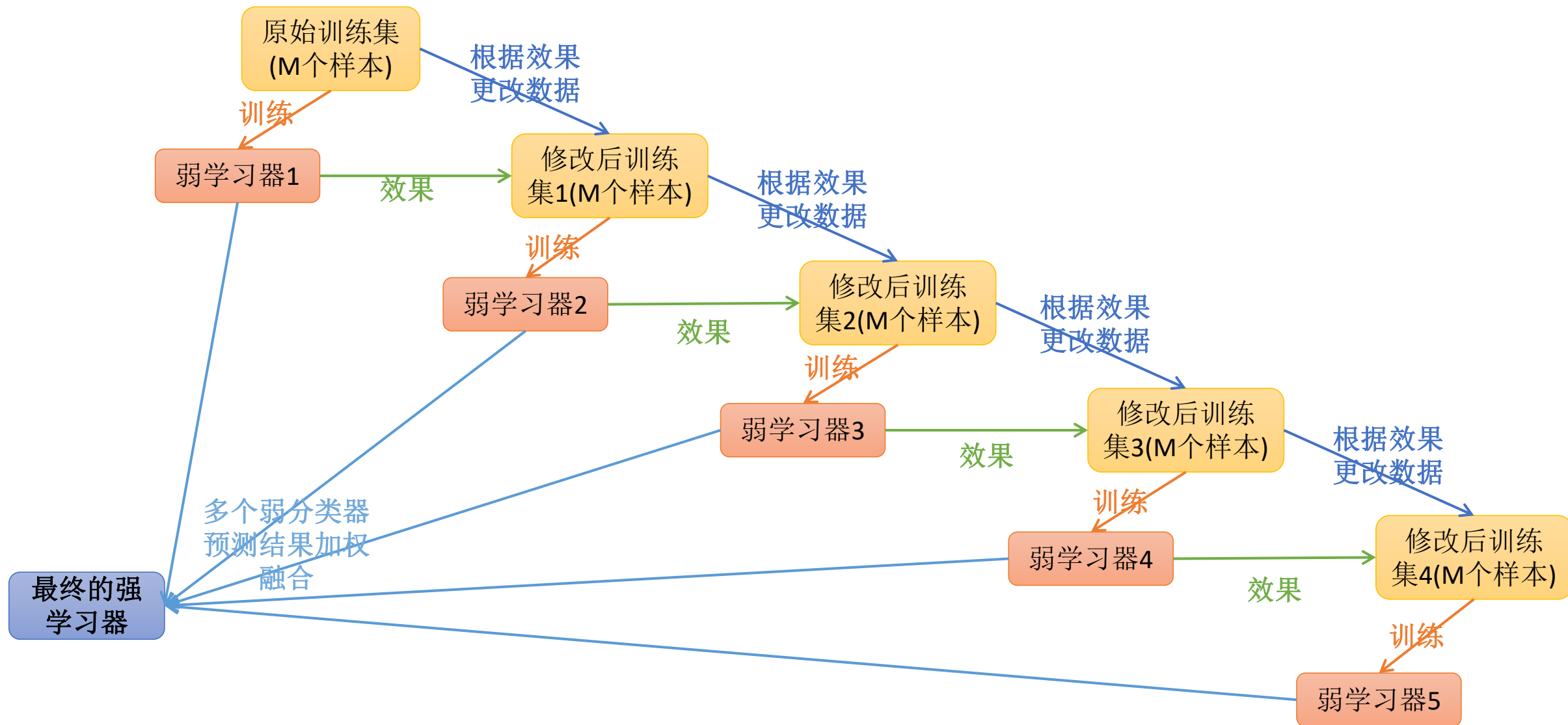
随机森林的思考

- 在随机森林的构建过程中，由于各棵树之间是没有关系的，相对独立的；在构建的过程中，构建第 m 棵子树的时候，不会考虑前面的 $m-1$ 棵树。
- 思考：
 - 如果在构建第 m 棵子树的时候，考虑到前 $m-1$ 棵子树的结果，会不会对最终结果产生有益的影响？
 - 各个决策树组成随机森林后，在形成最终结果的时候能不能给定一种既定的决策顺序呢？（也就是那颗子树先进行决策、那颗子树后进行决策）

Boosting

- 提升学习 (Boosting) 是一种机器学习技术, 可以用于**回归**和**分类**的问题, 它每一步产生**弱预测模型**(如决策树), 并**加权累加**到总模型中; 如果每一步的弱预测模型的生成都是依据损失函数的梯度方式的, 那么就称为梯度提升(Gradient boosting);
- 提升技术的意义: 如果一个问题存在**弱预测模型**, 那么可以通过提升技术的办法得到一个**强预测模型**;
- 常见的模型有:
 - Adaboost
 - Gradient Boosting(GBT/GBDT/GBRT)

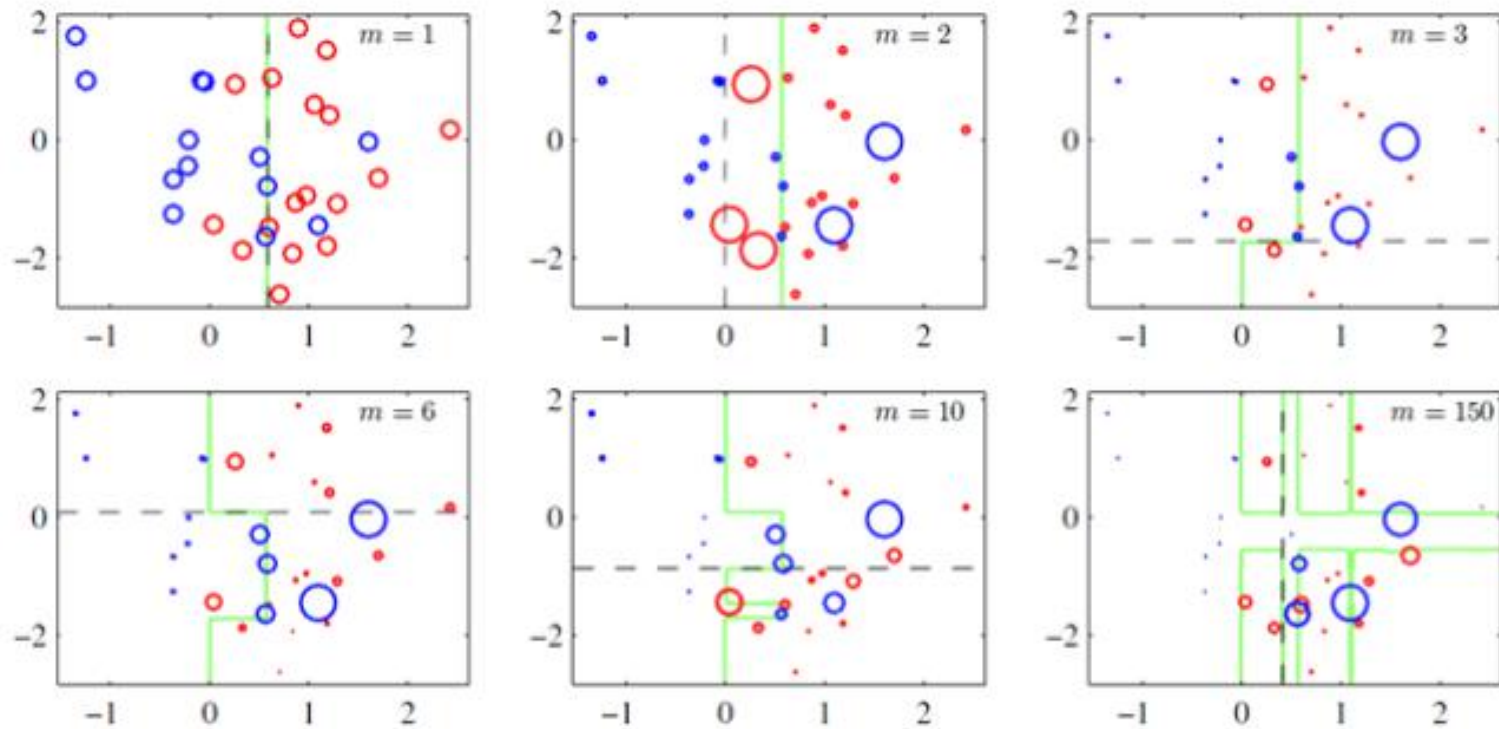
Boosting



AdaBoost算法原理

- Adaptive Boosting是一种迭代算法。每轮迭代中会在训练集上产生一个新的学习器，然后使用该学习器对所有训练样本进行预测，以评估每个样本的重要性(Informative)。换句话说来讲就是，算法/子模型会为每个样本赋予一个权重，每次用训练好的学习器标注/预测各个样本(训练数据)，**如果某个样本点被预测的越正确，则将样本权重降低；否则提高样本的权重。权重越高的样本在下一个迭代训练中所占的权重就越大，也就是说越难区分的样本在训练过程中会变得越重要；**
- 整个迭代过程直到错误率足够小或者达到一定的迭代次数为止。

样本加权



Adaboost算法

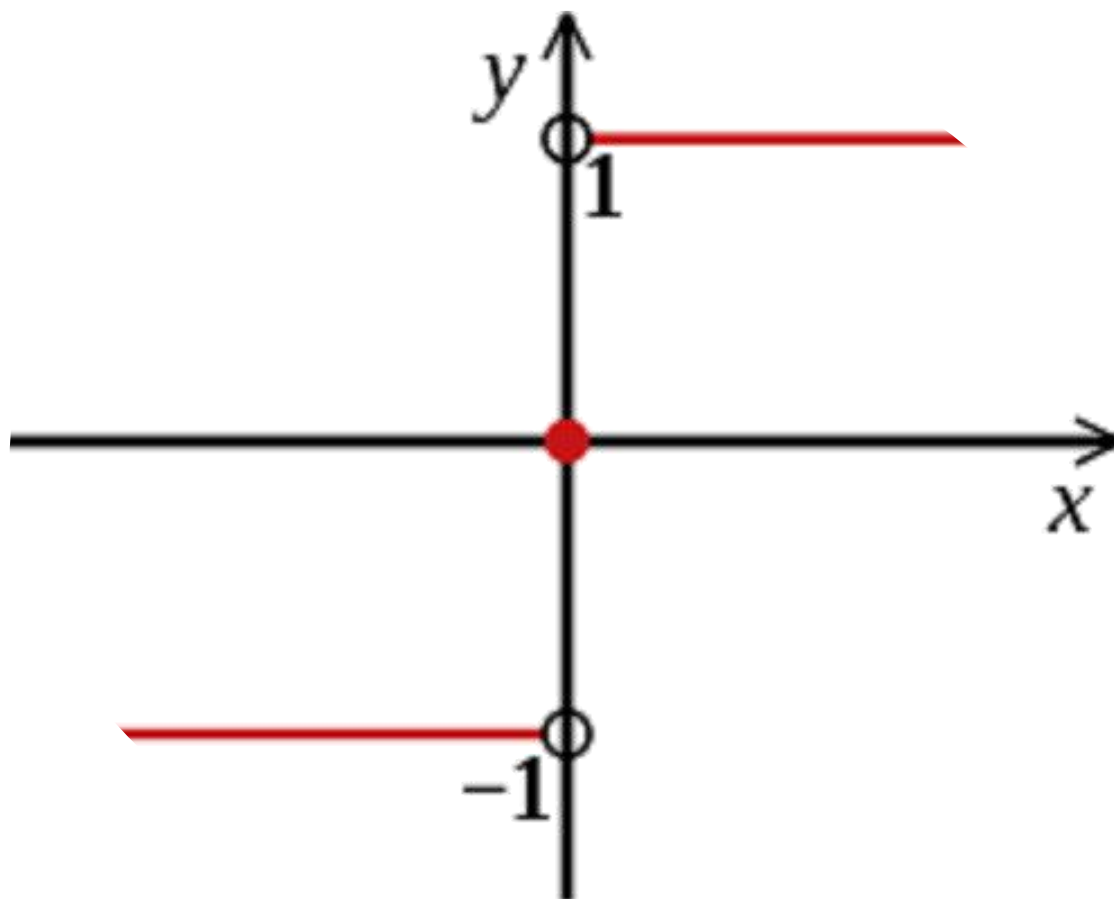
- Adaboost算法将基分类器的线性组合作为强分类器，同时给分类误差率较小的基本分类器以大的权值，给分类误差率较大的基分类器以小的权重值；构建的线性组合为：

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x)$$

- 最终分类器是在线性组合的基础上进行Sign函数转换：

$$G(x) = \text{sign}(f(x)) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$$

Sign函数



AdaBoost算法原理

- 最终的强学习器：
$$G(x) = \text{sign}(f(x)) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$$

- 损失函数(以错误率作为损失函数):

$$\sum_{i=1}^n w_i = 1 \quad \text{loss} = \sum_{i=1}^n w_i I(G(x_i) \neq y_i)$$

- 损失函数(上界):

$$\text{loss} = \sum_{i=1}^n w_i I(G(x_i) \neq y_i) \leq \sum_{i=1}^n w_i e^{(-y_i f(x))}$$

AdaBoost算法原理

$$loss = \sum_{i=1}^n w_i e^{(-y_i f(x_i))}$$

- 第k-1轮的强学习器:

$$f_{k-1}(x) = \sum_{j=1}^{k-1} \alpha_j G_j(x)$$

- 第k轮的强学习器:

$$f_k(x) = \sum_{j=1}^k \alpha_j G_j(x) \quad f_k(x) = f_{k-1}(x) + \alpha_k G_k(x)$$

- 损失函数:

$$loss(\alpha_m, G_m(x)) = \sum_{i=1}^n w_{1,i} e^{(-y_i (f_{m-1}(x) + \alpha_m G_m(x)))}$$

AdaBoost算法原理

$$\text{loss}(\alpha_m, G_m(x)) = \sum_{i=1}^n w_{1,i} e^{(-y_i(f_{m-1}(x) + \alpha_m G_m(x)))}$$

$$= \sum_{i=1}^n w_{1,i} e^{-y_i f_{m-1}(x)} e^{(-y_i \alpha_m G_m(x))}$$

$$\xrightarrow{\text{令 } \bar{w}_{mi} = w_{1,i} e^{-y_i f_{m-1}(x)}} \rightarrow$$

$$= \sum_{i=1}^n \bar{w}_{mi} e^{(-y_i \alpha_m G_m(x))}$$

AdaBoost算法原理

- 使下列公式达到最小值的 α_m 和 G_m 就是AdaBoost算法的最终解

$$\text{loss}(\alpha_m, G_m(x)) = \sum_{i=1}^n \bar{w}_{mi} e^{(-y_i \alpha_m G_m(x))}$$

- G这个分类器在训练的过程中，是为了让误差率最小，所以可以认为G越好其实就是误差率越小。

$$G_m^*(x) = \min_{G_m(x)} \sum_{i=1}^n \bar{w}_{mi} I(y_i \neq G_m(x_i))$$

$$\varepsilon_m = P(G_m(x) \neq y) = \sum_{i=1}^n \bar{w}_{mi} I(y_i \neq G_m(x_i)) = \sum_{y_i \neq G_m(x_i)} \bar{w}_{mi}$$

- 对于 α_m 而言，通过求导然后令导数为零，可以得到公式（log对象可以以e为底也可以以2为底）：

$$\alpha_m^* = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$

扩展_AdaBoost算法子模型权重系数求解

$$\begin{aligned} \text{loss}(\alpha_m, G_m(x)) &= \sum_{i=1}^n \bar{w}_{mi} e^{(-y_i \alpha_m G_m(x))} & \sum_{i=1}^n \bar{w}_{mi} &= 1 & \mathcal{E}_m &= \sum_{y_i \neq G_m(x_i)} \bar{w}_{mi} \\ &= \sum_{y=G(x)} \bar{w}_{mi} e^{-\alpha_m} + \sum_{y \neq G(x)} \bar{w}_{mi} e^{\alpha_m} \\ &= \sum_{y=G(x)} \bar{w}_{mi} e^{-\alpha_m} + \mathcal{E}_m e^{\alpha_m} \\ &= \sum_{y=G(x)} \bar{w}_{mi} e^{-\alpha_m} + \mathcal{E}_m e^{\alpha_m} + \sum_{y \neq G(x)} \bar{w}_{mi} e^{-\alpha_m} - \sum_{y \neq G(x)} \bar{w}_{mi} e^{-\alpha_m} \\ &= \sum_{i=1}^n \bar{w}_{mi} e^{-\alpha_m} + \mathcal{E}_m e^{\alpha_m} - \mathcal{E}_m e^{-\alpha_m} \\ &= e^{-\alpha_m} + \mathcal{E}_m e^{\alpha_m} - \mathcal{E}_m e^{-\alpha_m} \end{aligned}$$

扩展_AdaBoost算法子模型权重系数求解

$$loss = e^{-\alpha_m} + \varepsilon_m e^{\alpha_m} - \varepsilon_m e^{-\alpha_m} \quad \frac{\partial loss}{\partial \alpha_m} = -e^{-\alpha_m} + \varepsilon_m e^{\alpha_m} + \varepsilon_m e^{-\alpha_m}$$

$$\Rightarrow \frac{\partial loss}{\partial \alpha_m} = 0 \Rightarrow -e^{-\alpha_m} + \varepsilon_m e^{\alpha_m} + \varepsilon_m e^{-\alpha_m} = 0$$

$$\Rightarrow (\varepsilon_m - 1)e^{-\alpha_m} + \varepsilon_m e^{\alpha_m} = 0 \Rightarrow \varepsilon_m e^{\alpha_m} = (1 - \varepsilon_m)e^{-\alpha_m}$$

$$\Rightarrow \frac{e^{\alpha_m}}{e^{-\alpha_m}} = \frac{(1 - \varepsilon_m)}{\varepsilon_m} \Rightarrow e^{2\alpha_m} = \frac{(1 - \varepsilon_m)}{\varepsilon_m}$$

$$\Rightarrow \ln e^{2\alpha_m} = \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right) \Rightarrow 2\alpha_m = \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$

$$\Rightarrow \alpha_m = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$

Adaboost算法构建过程一

- 1. 假设训练数据集 $T = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$

- 2. 初始化训练数据权重分布

$$D_1 = (w_{11}, w_{12}, \dots, w_{1i}, \dots, w_{1n}), \quad w_{1i} = \frac{1}{n}, \quad i = 1, 2, \dots, n$$

- 3. 使用具有权值分布 D_m 的训练数据集学习, 得到基本分类器

$$G_m(x): x \rightarrow \{-1, +1\}$$

- 4. 计算 $G_m(x)$ 在训练集上的分类误差

$$\varepsilon_m = P(G_m(x) \neq y) = \sum_{i=1}^n \bar{w}_{mi} I(y_i \neq G_m(x_i)) = \sum_{y_i \neq G_m(x_i)} \bar{w}_{mi}$$

- 5. 计算 $G_m(x)$ 模型的权重系数 α_m :

$$\alpha_m = \frac{1}{2} * \ln\left(\frac{1 - \varepsilon_m}{\varepsilon_m}\right)$$

Adaboost算法构建过程二

$$f_{m-1}(x) = \sum_{j=1}^{m-1} \alpha_j G_j(x) \quad \bar{w}_{mi} = w_{1,i} e^{-y_i f_{m-1}(x)}$$

- 6. 权重训练数据集的权值分布

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,n}) \quad w_{m+1,i} = \frac{w_{m,i}}{Z_m} e^{-\alpha_m y_i G_m(x_i)}$$

- 7. 这里 Z_m 是规范化因子(归一化)

$$Z_m = \sum_{i=1}^n w_{m,i} e^{-\alpha_m y_i G_m(x_i)}$$

- 8. 构建基本分类器的线性组合

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x)$$

- 9. 得到最终分类器

$$G(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right)$$

Adaboost算法的直观理解

- 使用下列样本作为训练数据，试图使用AdaBoost算法学习一个强分类器

序号	1	2	3	4	5	6	7	8	9	10
X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1

- 初始化训练数据集的权值分布

$$D_1 = (w_{11}, w_{12}, \dots, w_{1i}, \dots, w_{1n}), \quad w_{1i} = \frac{1}{N}, \quad i = 1, 2, \dots, N$$

$$w_{1i} = 0.1$$

Adaboost算法的直观理解

- 对于m=1

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

- 在权值分布为D1的训练数据上，阈值v取2.5时误差率最低，故基本分类器为：

$$G_1(x) = \begin{cases} 1, x < 2.5 \\ -1, x > 2.5 \end{cases}$$

- G1(x)在训练数据集上的误差率： $\varepsilon_1 = P(G_1(x_i) \neq y_i) = 0.3$

- 计算G1的系数：
$$\alpha_1 = \frac{1}{2} \ln \frac{1 - \varepsilon_1}{\varepsilon_1} = 0.4236$$

Adaboost算法的直观理解

- 更新数据集的权值分布

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \dots, w_{m+1,i}, \dots, w_{m+1,n}) \quad w_{m+1,i} = \frac{w_{m,i}}{Z_m} e^{-\alpha_m y_i G_m(x_i)}$$

$$\begin{aligned} D_2 &= (w_{21}, w_{22}, \dots, w_{2n}) \\ &= (0.0714, 0.0714, 0.0714, 0.0714, 0.0714, 0.0714, 0.1667, 0.1667, 0.1667, 0.0714) \end{aligned}$$

$$f_1(x) = 0.4236 G_1(x)$$

- 分类器 $\text{sign}(f_1(x))$ 在训练数据集上有3个误分类点

Adaboost算法的直观理解

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
G_1	1	1	1	-1	-1	-1	-1	-1	-1	-1
G_{R1}	1	1	1	-1	-1	-1	-1	-1	-1	-1

alpha	value
α_1	0.4236

$$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases} \quad f_1(x) = 0.4236 G_1(x)$$

Adaboost算法的直观理解

- 对于m=2

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w2	0.0714	0.0714	0.0714	0.0714	0.0714	0.0714	0.1667	0.1667	0.1667	0.0714

- 在权值分布为D2的训练数据上，阈值v取8.5时误差率最低，故基

本分类器为：

$$G_2(x) = \begin{cases} 1, & x \leq 8.5 \\ -1, & x > 8.5 \end{cases}$$

- $G_2(x)$ 在训练数据集上的误差率

$$\varepsilon_2 = P(G_2(x_i) \neq y_i) = 0.0714 * 3 = 0.2142$$

Adaboost算法的直观理解

- 计算 G_2 的系数

$$\alpha_2 = \frac{1}{2} \ln \frac{1 - \varepsilon_2}{\varepsilon_2} = 0.6496$$

- 更新数据集的权值分布

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w3	0.0455	0.0455	0.0455	0.1667	0.1667	0.1667	0.1061	0.1061	0.1061	0.0455

$$f_2(x) = 0.4236G_1(x) + 0.6496G_2(x)$$

分类器 $\text{sign}(f_2(x))$ 在训练数据集上有3个误分类点

Adaboost算法的直观理解

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
G1	1	1	1	-1	-1	-1	-1	-1	-1	-1
w2	0.0714	0.0714	0.0714	0.0714	0.0714	0.0714	0.1667	0.1667	0.1667	0.0714
G2	1	1	1	1	1	1	1	1	1	-1
G_R2	1	1	1	1	1	1	1	1	1	-1

alpha	value
α_1	0.4236
α_2	0.6496

$$G_2(x) = \begin{cases} 1, & x \leq 8.5 \\ -1, & x > 8.5 \end{cases} \quad f_2(x) = 0.4236G_1(x) + 0.6496G_2(x)$$

Adaboost算法的直观理解

- 对于m=3

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w3	0.0455	0.0455	0.0455	0.1667	0.1667	0.1667	0.1061	0.1061	0.1061	0.0455

- 在权值分布为D3的训练数据上，阈值v取5.5时误差率最低，故基

本分类器为：

$$G_3(x) = \begin{cases} -1, & x \leq 5.5 \\ 1, & x > 5.5 \end{cases}$$

- $G_3(x)$ 在训练数据集上的误差率

$$\varepsilon_3 = P(G_3(x_i) \neq y_i) = 0.0455 * 4 = 0.182$$

Adaboost算法的直观理解

- 计算G3的系数

$$\alpha_3 = \frac{1}{2} \ln \frac{1 - \varepsilon_3}{\varepsilon_3} = 0.7520$$

- 更新数据集的权值分布

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w4	0.125	0.125	0.125	0.1019	0.1019	0.1019	0.0648	0.0648	0.0648	0.125

$$f_3(x) = 0.4236G_1(x) + 0.6496G_2(x) + 0.7520G_3(x)$$

- 分类器 $\text{sign}(f_3(x))$ 在训练数据集上有0个误分类点

Adaboost算法的直观理解

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
G1	1	1	1	-1	-1	-1	-1	-1	-1	-1
w2	0.0714	0.0714	0.0714	0.0714	0.0714	0.0714	0.1667	0.1667	0.1667	0.0714
G2	1	1	1	1	1	1	1	1	1	-1
w3	0.0455	0.0455	0.0455	0.1667	0.1667	0.1667	0.1061	0.1061	0.1061	0.0455
G3	-1	-1	-1	-1	-1	-1	1	1	1	1
G_R3	1	1	1	-1	-1	-1	1	1	1	-1

alpha	value
α_1	0.4236
α_2	0.6496
α_3	0.7520

$$G_1(x) = \begin{cases} 1, x < 2.5 \\ -1, x > 2.5 \end{cases} \quad G_2(x) = \begin{cases} 1, x \leq 8.5 \\ -1, x > 8.5 \end{cases} \quad G_3(x) = \begin{cases} -1, x \leq 5.5 \\ 1, x > 5.5 \end{cases}$$

$$f_3(x) = 0.4236G_1(x) + 0.6496G_2(x) + 0.7520G_3(x)$$

作业

- 推导 α 求解过程中底数为2的情况下，该案例的最终参数情况：

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1
w1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

$$\alpha_m = \frac{1}{2} * \log_2 \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right) \quad w_{m+1,i} = \frac{w_{m,i}}{Z_m} 2^{-\alpha_m y_i G_m(x_i)}$$

AdaBoost scikit-learn相关参数

参数	AdaBoostClassifier	AdaBoostRegressor
base_estimator	弱分类器对象，默认为CART分类树 DecisionTreeClassifier;	弱回归器对象，默认为CART回归树DecisionTreeRegressor;
algorithm	SAMME和SAMME.R; SAMME表示构建过程中使用样本集分类效果作为弱分类器的权重; SAMME.R使用对样本集分类的预测概率大小作为弱分类器的权重。由于SAMME.R使用了连续的概率度量值，所以一般迭代比SAMME快，默认参数为SAMME.R; 强调：使用SAMME.R必须要求base_estimator指定的弱分类器模型必须支持概率预测，即具有predict_proba方法。	不支持
loss	不支持	指定误差的计算方式，可选参数“linear”, “square”, “exponential”, 默认为“linear”; 一般不用改动
n_estimators	最大迭代次数，值过小可能会导致欠拟合，值过大可能会导致过拟合，一般50~100比较适合，默认50	
learning_rate	指定每个弱分类器的权重缩减系数v，默认为1；一般从一个比较小的值开始进行调参；该值越小表示需要更多的弱分类器	

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x) \xrightarrow{\text{添加缩减系数}v} f(x) = \sum_{m=1}^M v \alpha_m G_m(x)$$

AdaBoost总结

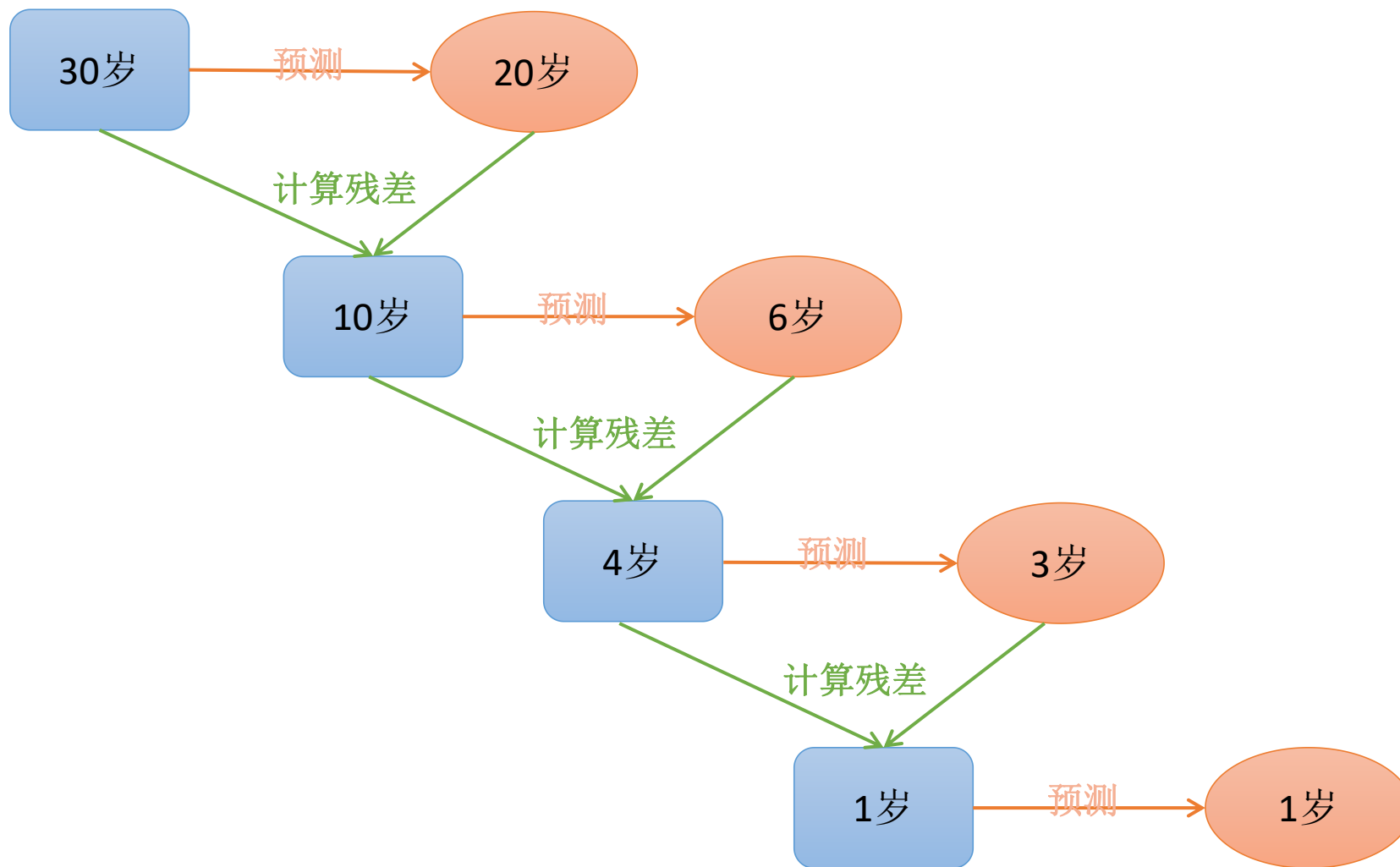
- AdaBoost的优点如下：
 - 可以处理连续值和离散值；
 - 模型的鲁棒性比较强；
 - 解释强，结构简单。
- AdaBoost的缺点如下：
 - 对异常样本敏感，异常样本可能会在迭代过程中获得较高的权重值，最终影响模型效果。

梯度提升迭代决策树GBDT

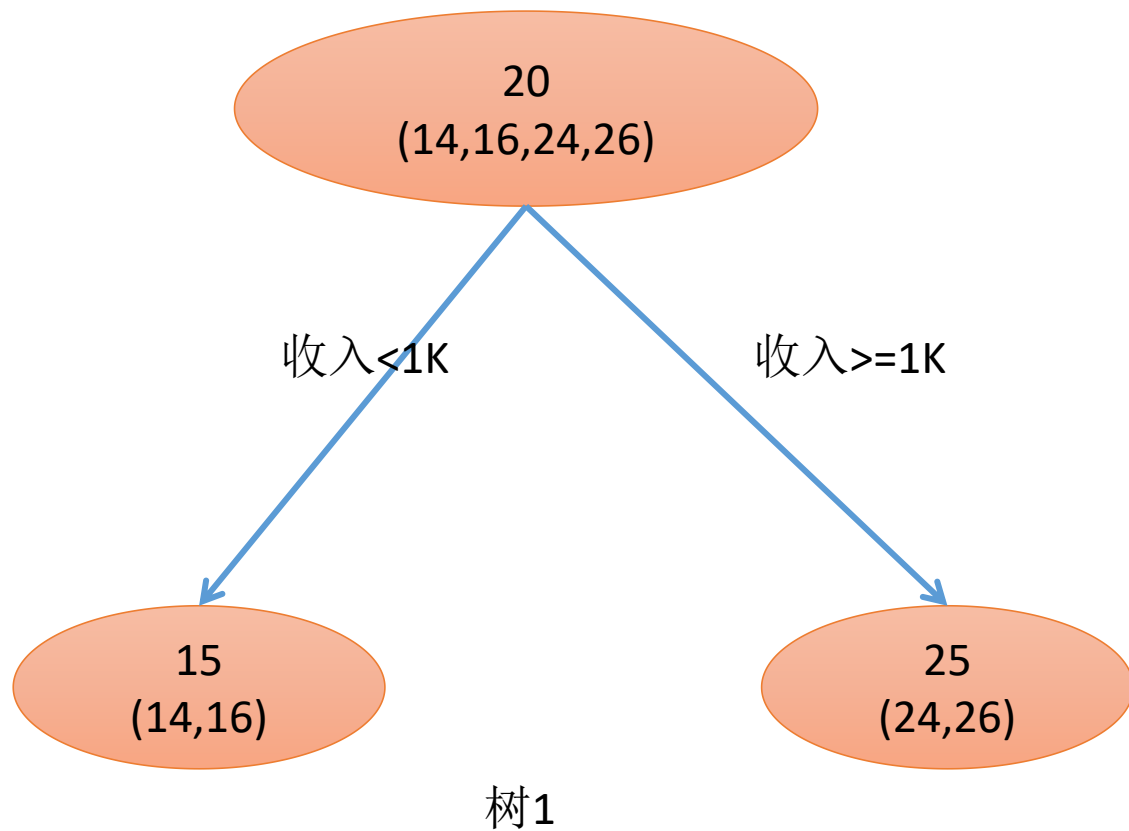
- GBDT也是Boosting算法的一种，但是和AdaBoost算法不同；区别如下：
AdaBoost算法是利用前一轮的弱学习器的误差来更新样本权重值，然后一轮一轮的迭代；GBDT也是迭代，但是GBDT要求弱学习器必须是回归CART模型，而且GBDT在模型训练的时候，是要求模型预测的样本损失尽可能的小。
- 备注：所有GBDT算法中，底层都是回归树。
- 别名：GBT(Gradient Boosting Tree)、GTB(Gradient Tree Boosting)、GBRT(Gradient Boosting Regression Tree)、GBDT(Gradient Boosting Decision Tree)、MART(Multiple Additive Regression Tree)

$$\begin{array}{ccc} f_{t-1}(x) & & f_t(x) \\ L(y, f_{t-1}(x)) & \xrightarrow{\text{训练弱学习器: } h_t(x)} & L(y, f_{t-1}(x) + h_t(x)) \end{array}$$

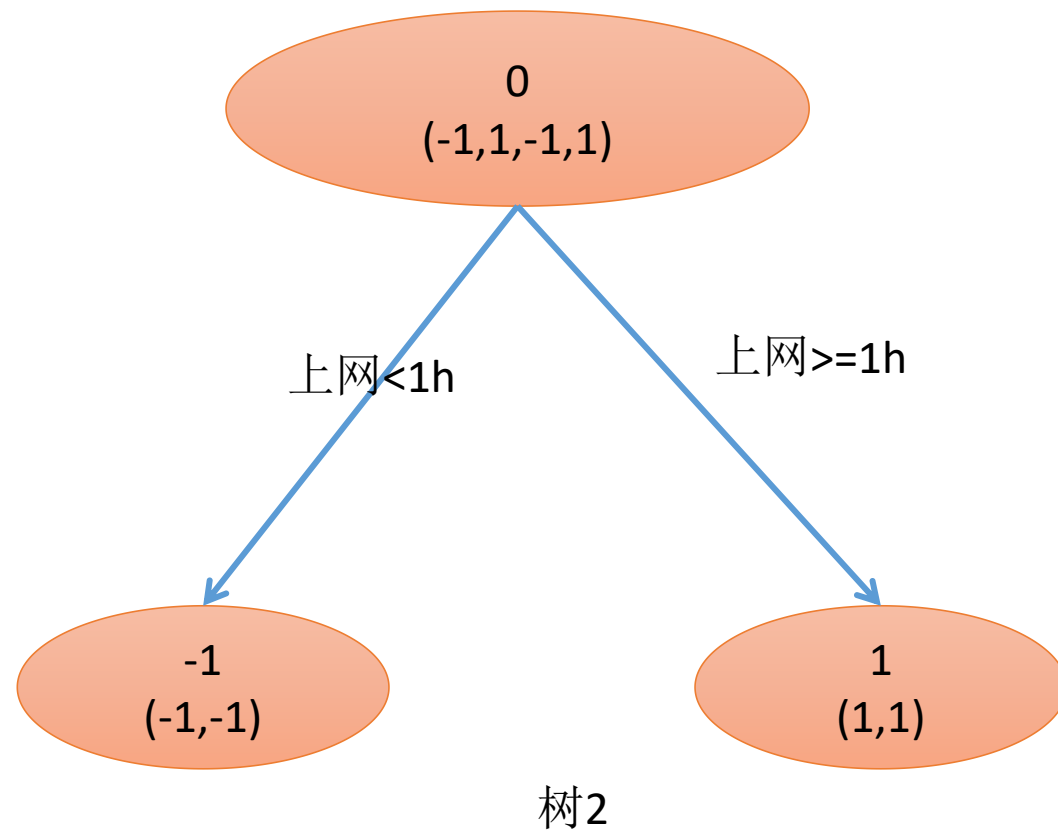
GBDT直观理解



GBDT直观理解



r



- 当给定步长时候, 给定一个步长step, 在构建下一棵树的时候使用step*残差值作为输入值, 这种方式可以减少过拟合的发生

梯度提升迭代决策树GBDT

- GBDT由三部分构成：DT(Regression Decision Tree)、GB(Gradient Boosting)和Shrinkage(衰减)
- 由多棵决策树组成，所有树的结果累加起来就是最终结果
- 迭代决策树和随机森林的区别：
 - 随机森林使用抽取不同的样本构建不同的子树，也就是说第m棵树的构建和前m-1棵树的结果是没有关系的
 - 迭代决策树在构建子树的时候，使用之前子树构建结果后形成的残差作为输入数据构建下一个子树；然后最终预测的时候按照子树构建的顺序进行预测，并将预测结果相加

GBDT算法原理

- 给定输入向量 X 和输出变量 Y 组成的若干训练样本 $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$, 目标是找到近似函数 $F(X)$, 使得损失函数 $L(Y, F(X))$ 的损失值最小。
- 损失函数一般采用最小二乘损失函数或者绝对值损失函数。

$$L(y, F(X)) = \frac{1}{2} (y - F(X))^2 \quad L(y, F(X)) = |y - F(X)|$$

- 最优解为:

$$F^*(X) = \arg \min_F L(y, F(X))$$

- 假定 $F(X)$ 是一族最优基函数 $f_i(X)$ 的加权和:

$$F(X) = \sum_{i=0}^M f_i(X) \xrightarrow[\text{防止每个学习器能力过强, 可能导致过拟合; 给定一个缩放系数 } v]{\quad} F(X) = v \sum_{i=0}^M f_i(X)$$

GBDT算法原理

- 以贪心算法的思想扩展得到 $F_m(X)$ ，求解最优 f

$$F_m(X) = F_{m-1}(X) + \arg \min_f \sum_{i=1}^n L(y_i, F_{m-1}(X_i) + f(X_i))$$

- 以贪心法在每次选择最优基函数 f 时仍然困难，使用梯度下降的方法近似计算
- 给定常数函数 $F_0(X)$

$$F_0(X) = \arg \min_c \sum_{i=1}^n L(y_i, c)$$

GBDT算法原理

$$L(y, F(X)) = \frac{1}{2} (y - F(X))^2$$

- 计算损失函数的负梯度值：

$$y_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

- 使用数据 (x_i, y_{im}) ($i=1, \dots, n$) 计算拟合残差找到一个CART回归树，得到第m棵树

$$c_{mj} = \arg \min_c \sum_{x_i \in \text{leaf}_j} L(y_{im}, c) \quad f_m(x) = \sum_{j=1}^{|\text{leaf}|_m} c_{mj} I(x \in \text{leaf}_{mj})$$

- 更新模型

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{|\text{leaf}|_m} c_{mj} I(x \in \text{leaf}_{mj}) \quad \Rightarrow \quad F(x) = F_0(x) + \sum_{m=1}^M \sum_{j=1}^{|\text{leaf}|_m} c_{mj} I(x \in \text{leaf}_{mj})$$

GBDT回归算法和分类算法的区别

- 两者唯一的区别就是选择不同的损失函数、以及对应的负梯度值和模型初值采用不一样的值。
- 回归算法选择的损失函数一般是均方差(最小二乘)和绝对值误差, 分类算法中一般选择对数损失函数来表示。

$$\alpha_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

GBDT回归算法和分类算法的区别

- 均方差损失函数

- 损失函数: $L(y, F_m(x)) = \frac{1}{2}(y - F_m(x))^2$
- 负梯度值: $y_{im} = y_i - F_{m-1}(x)$
- 初始值: 一般采用均值作为初始值。

- 绝对误差损失函数

- 损失函数: $L(y, F_m(x)) = |y - F_m(x)|$
- 负梯度值: $y_{im} = \text{sign}(y_i - F_{m-1}(x))$
- 初始值: 一般采用中值作为初始值。

GBDT回归算法和分类算法的区别

- 对数损失函数(二分类)

- 损失函数: $L(y, F_m(x)) = -(y \ln(p_m) + (1 - y) \ln(1 - p_m))$ $p_m = \frac{1}{1 + e^{-F_m(x)}}$
- 负梯度值: $y_{im} = y_i - p_m$
- 初始值: 一般采用 $\ln(\text{正样本个数}/\text{负样本个数})$ 作为初始值。

- 对数损失函数(多分类K)

- 损失函数: $L(y, F_{ml}(x)) = -\sum_{k=1}^K y_k \ln p_k(x)$ $p_k(x) = \exp(f_k(x)) / \sum_{l=1}^K \exp(f_l(x))$
- 负梯度值: $y_{iml} = y_{il} - p_{ml}(x)$
- 初始值: 一般采用0作为初始值。

GBDT总结

- GBDT的优点如下：
 - 可以处理连续值和离散值；
 - 在相对少的调参情况下，模型的预测效果也会不错；
 - 模型的鲁棒性比较强。
- GBDT的缺点如下：
 - 由于弱学习器之间存在关联关系，难以并行训练模型。也就是模型训练的速度慢。

GBDT scikit-learn相关参数

参数	GradientBoostingClassifier	GradientBoostingRegressor
alpha	不支持	当使用huber或者quantile损失函数的时候，需要给定分位数的值，默认为0.9；如果噪音数据比较多，可以适当的降低该参数值
loss	给定损失函数，可选对数似然函数deviance和指数损失函数exponential；默认为deviance；不建议修改	给定损失函数，可选均方差ls、绝对损失lad、Huber损失huber、分位数损失quantile；默认ls；一般采用默认；如果噪音数据比较多，推荐huber；如果是分段预测，推荐quantile
n_estimators	最大迭代次数，值过小可能会导致欠拟合，值过大可能会导致过拟合，一般50~100比较适合，默认50	
learning_rate	指定每个弱分类器的权重缩减系数v，默认为1；一般从一个比较小的值开始进行调参；该值越小表示需要更多的弱分类器	
subsample	给定训练模型的时候，进行子采样的比例值，取值范围(0,1], 默认为1，表示不采用子采样；给值小于1表示采用部分数据进行模型训练，可以降低模型的过拟合情况；推荐[0.5,0.8]；采样采用的方式是不放回采样	
init	给定初始化的模型，可以不给定	

Bagging、Boosting的区别

- 样本选择: Bagging算法是有放回的随机采样; Boosting算法是每一轮训练集不变, 只是训练集中的每个样例在分类器中的权重发生变化或者目标属性 y 发生变化, 而权重& y 值都是根据上一轮的预测结果进行调整;
- 样例权重: Bagging使用随机抽样, 样例是等权重; Boosting根据错误率不断的调整样例的权重值, 错误率越大则权重越大(Adaboost);
- 预测函数: Bagging所有预测模型的权重相等; Boosting算法对于误差小的分类器具有更大的权重(Adaboost)。
- 并行计算: Bagging算法可以并行生成各个基模型; Boosting理论上只能顺序生产, 因为后一个模型需要前一个模型的结果;
- **Bagging是减少模型的variance(方差); Boosting是减少模型的Bias(偏度)。**
- Bagging里每个分类模型都是强分类器, 因为降低的是方差, 方差过高需要降低是过拟合; Boosting里每个分类模型都是弱分类器, 因为降低的是偏度, 偏度过高是欠拟合。

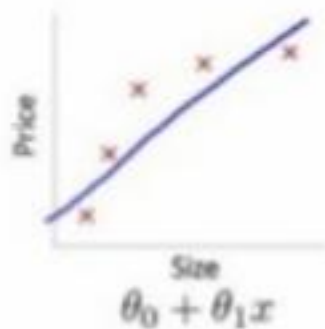
Bagging、Boosting的区别

- error = Bias + Variance

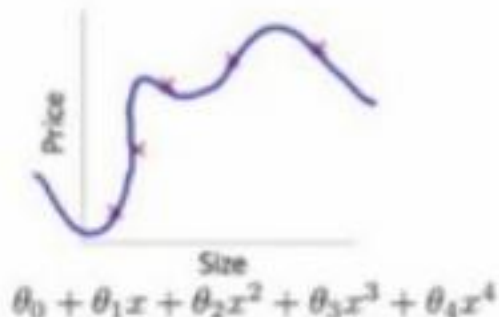
7) Bias/Variance Trade-off

High Variance (Overfitting)

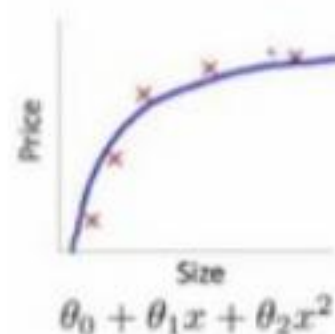
High Bias (Underfitting)



High bias
(underfit)



High variance
(overfit)



"Just right"

