Tree Ensembles

树集成/集成学习

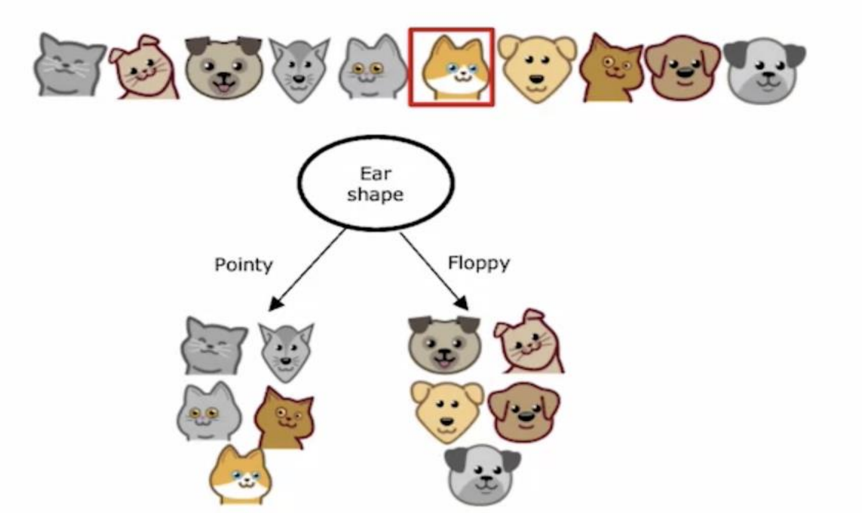
OUTLINE

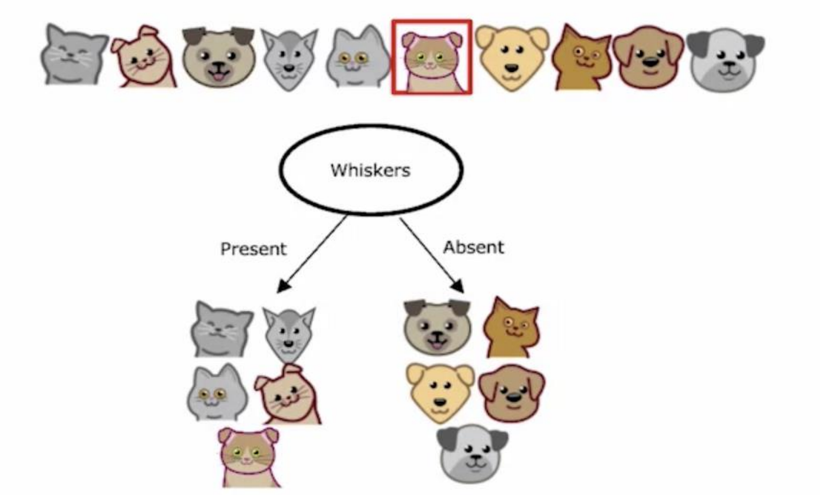
- Bagging
- Random Forest
- Boosting
- R实现

决策树

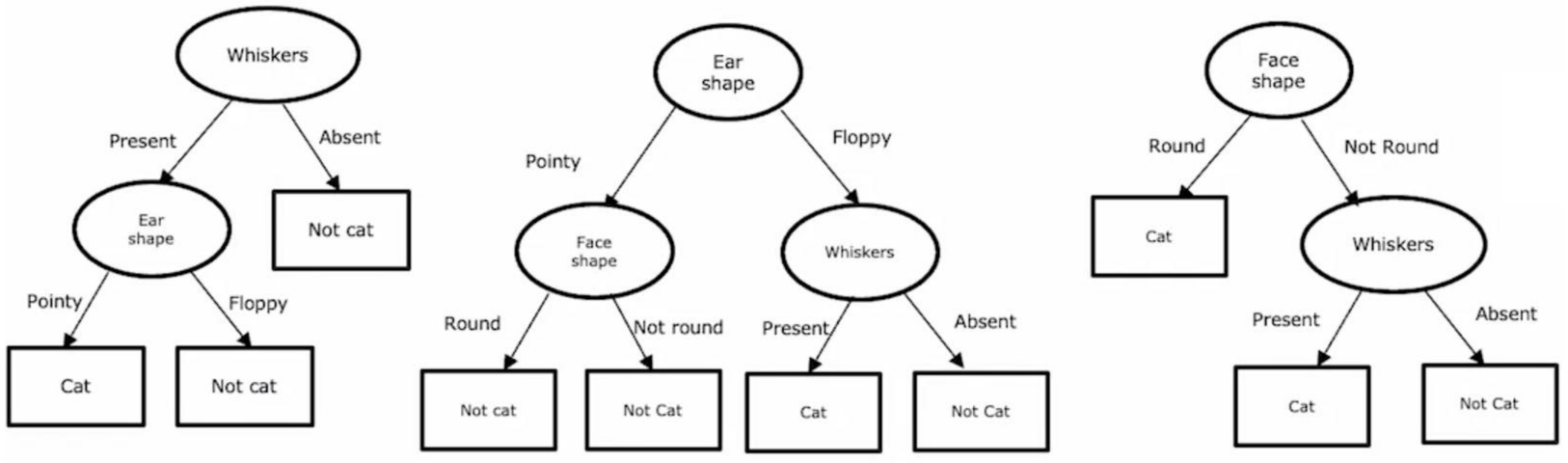
- 决策树主要存在两个问题：
  - 单个决策树的泛化能力不强
  - 单个决策树对训练数据中即使微小的变化也十分敏感
- 一个解决方式就是**构造多个决策树**，结合多个决策树的结果对数据进行预测
- 这样的方法称为tree ensemble

决策树对数据微小变化敏感





Tree Ensemble



Prediction: Cat

Prediction: Not Cat

Prediction: Cat

Final Prediction: Cat

集成学习

- 集成学习通过将多个学习器进行结合，可获得比单一学习器显著优越的泛化性能
- 在一般的经验中，如果把好坏不等的东西掺在一起，往往结果比最坏的更好，比最好的更坏。
- 集成学习把多个学习器结合起来，如何能获得比最好的单一学习器更好的性能呢？
- 考虑一个例子：两分类任务中，假定三个分类器在三个测试样本上的表现为

	测试例1	测试例2	测试例3
h_1	√	√	×
h_2	×	√	√
h_3	√	×	√
集成	√	√	√

(a) 集成提升性能

	测试例1	测试例2	测试例3
h_1	√	√	×
h_2	√	√	×
h_3	√	√	×
集成	√	√	×

(b) 集成不起作用

	测试例1	测试例2	测试例3
h_1	√	×	×
h_2	×	√	×
h_3	×	×	√
集成	×	×	×

(c) 集成起负作用

- 要想获得好的集成效果，个体学习器应该“**好而不同**”，即个体学习器要有一定的“**准确性**”，并且要有“**多样性**”

集成学习

- 如何产生并结合“**好而不同**”的个体学习器，是集成学习的核心问题
- 根据个体学习器的生成方式，目前的集成学习方法大致可分为两大类：串行、并行
- 并行：个体学习器之间不存在强依赖关系，可同时生成的并行化方法
- 串行：个体学习器之间存在强依赖关系、必须串行生成的序列化方法

- 前者的代表是Bagging和随机森林，后者的代表是Boosting

BAGGING

想法

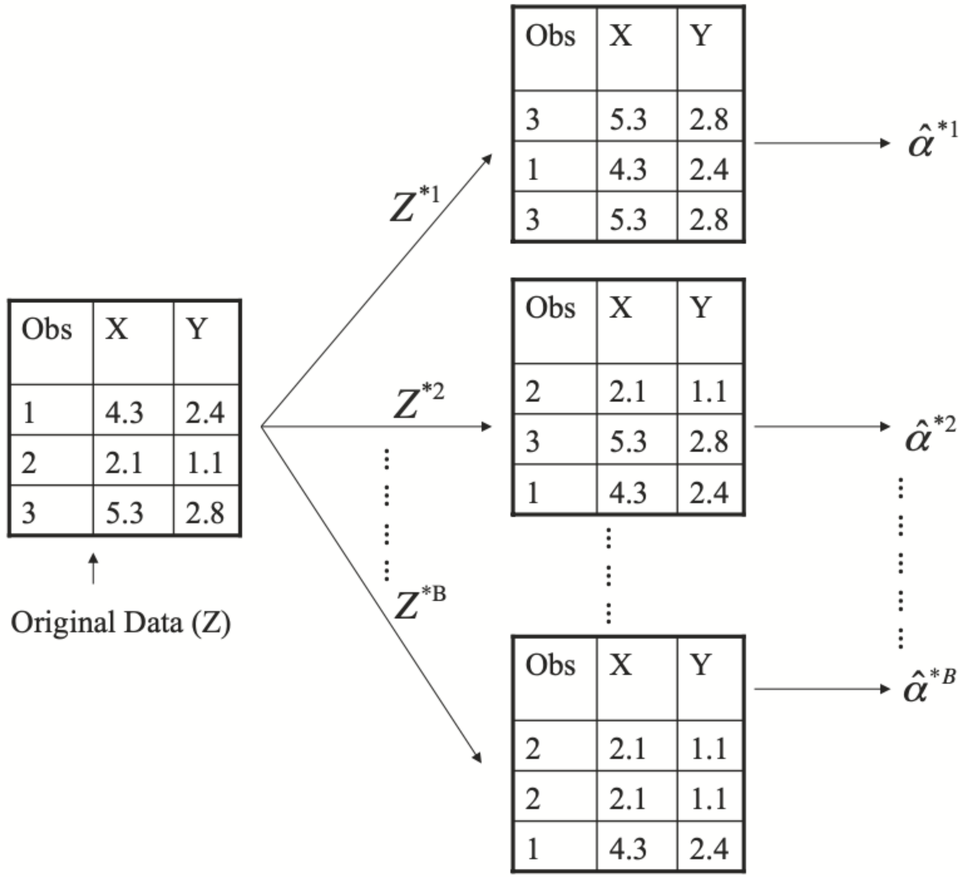
- 想要得到泛化性能强的树集成，每个决策树应尽可能相互独立
- 虽然独立在现实任务中无法做到，但可以设法使决策树之间尽可能有较大差异
- 给定一个训练数据集，可以对训练数据集进行采样，产生出若干不同子集，再对每个子集训练决策树
- 由于训练数据的不同，获得的不同决策树之间会有较大差异
- 然而，为了获得好的集成，我们同时希望决策树性能不能太差。如果采样出的每个子集都完全不同，则每个决策树只能用到一小部分训练数据，性能大打折扣
- 为此，我们可以考虑使用相互有重叠的采样子集，即可放回抽样

可放回抽样

	Ear shape	Face shape	Whiskers	Cat
	Pointy	Round	Present	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Pointy	Not round	Present	0
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Pointy	Round	Present	1
	Floppy	Not round	Present	1
	Floppy	Round	Absent	0
	Pointy	Round	Absent	1

Bootstrap Sampling

- 给定包含m个样本的数据集D，我们对其进行采样产生数据集D'：
- 每次随机从D中挑选一个样本，将其放入D'，然后再将该样本放回初始数据集D中，使得该样本在下次采样时仍有可能被采集到
- 这个过程重复m次后，我们就得到了包含m个样本的数据集D'
- 显然，D中有一部分样本会在D'中多次出现，而有一些样本可能不会出现
- 我们可以从初始数据集中产生多个不同的训练集，这对集成学习等方法有很大的好处



Bagging

- Bagging（Breiman，1996）是并行式集成学习方法最著名的代表
- 利用Bootstrap采样，我们可以采样出T个含m个训练样本的训练集D'
- Bagging：基于每个采样集，我们训练决策树，再将这些决策树的结果进行结合
- Bagging通常对分类任务使用投票法，对回归任务使用平均法

Input:

- Training dataset D
- Ensemble size T

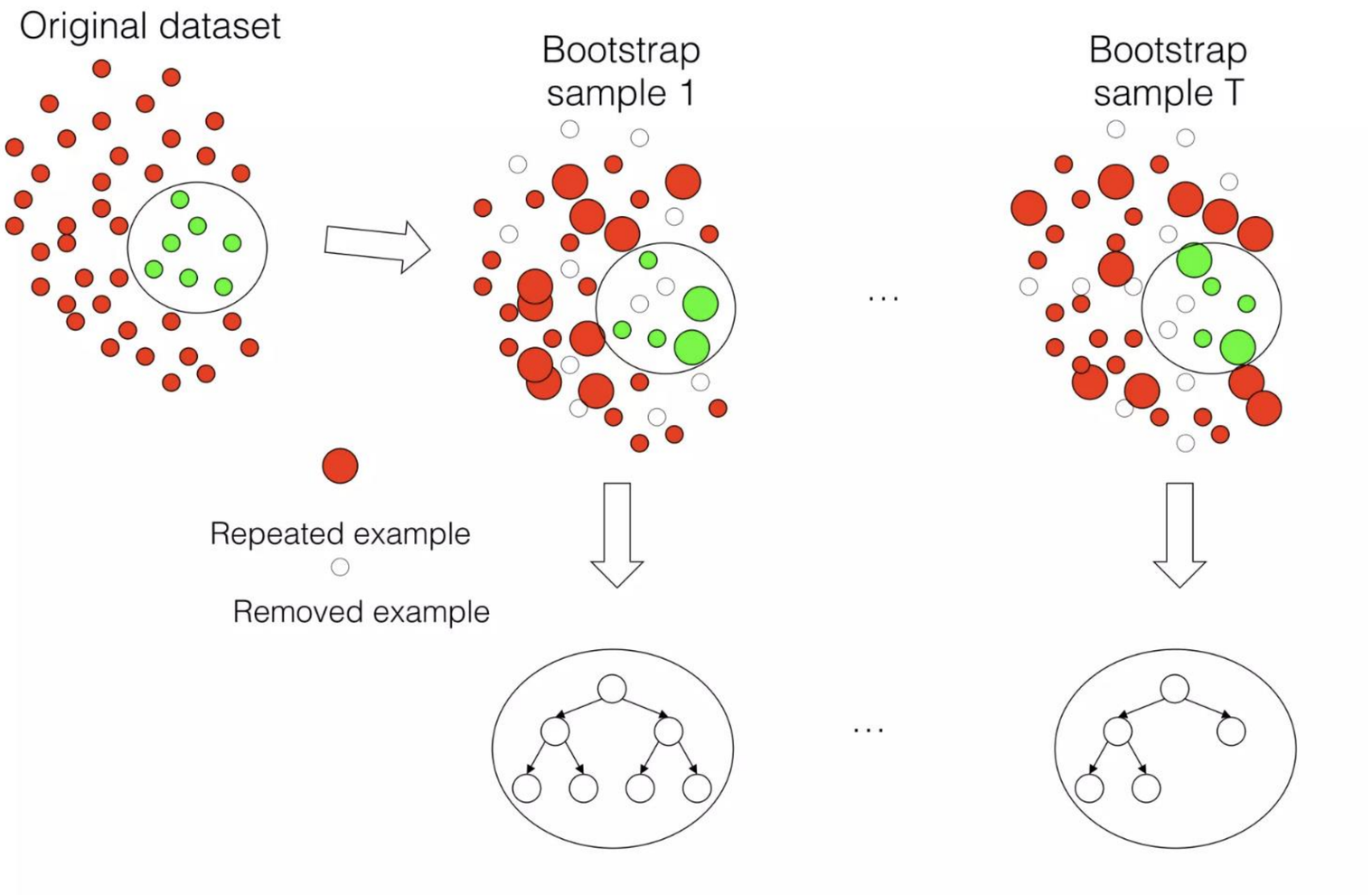
Algorithm:

1. for $t = 1$ to T :
2. sample=BootstrapSample(D)
3. f_t=Tree(sample)

Output:

$$f(\boldsymbol{x}) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} I(f_t(\boldsymbol{x}) = j) \right)$$

Bagging


Original dataset
Bootstrap sample 1
Bootstrap sample T
...
Repeated example
Removed example
...

Bagging

- Bootstrap还给Bagging带来了另一个优点：每个决策树只使用了初始训练集中约63.2%的样本：

$$\lim_{m\to\infty}\left(1-\left(1-\frac{1}{m}\right)^{m}\right)\to 1-\frac{1}{e}\approx 0.632$$

- 剩下的约36.8%的样本可用作验证集，对泛化性能进行“包外估计”（out-of-bag error estimate）

- 令D_t表示h_t实际使用的训练样本集；令$f^{OOB}(\boldsymbol{x})$表示对样本$\boldsymbol{x}$的包外预测，即只考虑那些未使用$\boldsymbol{x}$训练的决策树在$\boldsymbol{x}$上的预测：

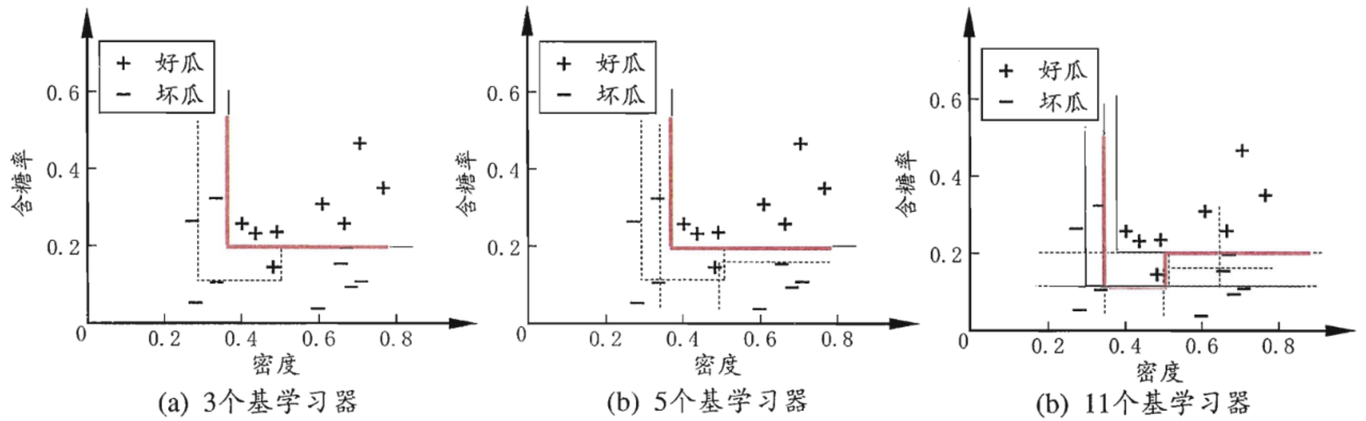
$$f^{OOB}(\boldsymbol{x})=\operatorname*{argmax}_{j}\left(\sum_{t=1}^{T}I(f_t(\boldsymbol{x})=j)\cdot I(\boldsymbol{x}\notin D_t)\right)$$

- Bagging泛化误差的包外估计为：

$$\frac{1}{|D|}\sum_{(\boldsymbol{x},y)\in D}I(f^{OOB}(\boldsymbol{x})\neq y)$$

Bagging

- 包外估计允许我们绕过交叉验证来便利的估计Bagging的测试误差
- 每一颗决策树都没有经过剪枝，长得很深。因此单颗决策树有着高方差和低偏差
- 而对T个学习器进行平均很好的降低了方差，因此Bagging在不剪枝的决策树、神经网络等易受样本扰动的学习器上效果更为明显

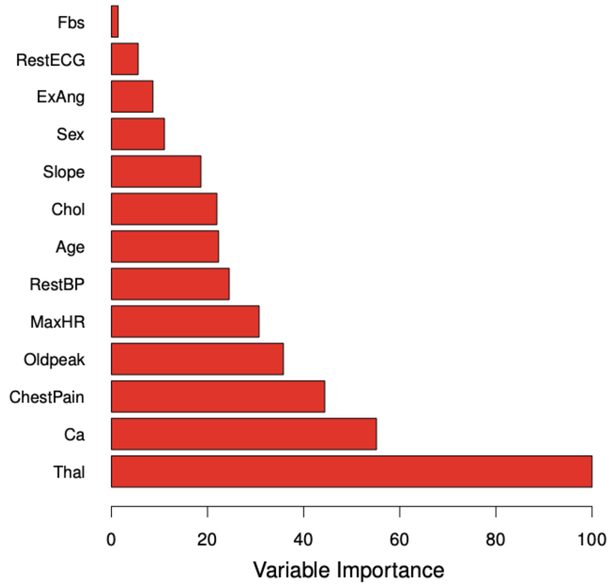


(a) 3个基学习器　(b) 5个基学习器　(b) 11个基学习器

图 8.6 西瓜数据集 3.0α 上 Bagging 集成规模为 3、5、11 时，集成(红色)与基学习器(黑色)的分类边界.

变量重要性

- 与单颗树相比，Bagging通常能够大幅提高预测准确性。遗憾的是，最终的模型很难解释
- 不过，我们可以对每个特征的重要性进行整体的总结，使用RSS（Bagging回归树）或Gini系数（Bagging分类树）
- 例如，对于Bagging分类树，我们对某给定特征在一棵树上因分裂而使Gini系数的减小量汇总，再取所有T颗树的平均，值越大说明特征越重要



特征重要性的度量

- 基于扰动的特征重要性的计算
- 对第t棵决策树，使用袋外数据计算预测误差$errOBB_t$
- 在第j个特征上打乱观测样本的顺序，再次计算预测误差$errOBB_t^*$

$$VI(j) = \frac{1}{B}\sum_{t=1}^{B}|errOOB_t - errOOB_t^*|$$

- 平均下降幅度越小，说明变量越重要

特征重要性的度量

- 基于纯洁度的下降幅度的特征重要性的计算
- 第j个特征的变量重要性是使用其进行空间切分时的信息增益之和

$$VI(j) = \frac{1}{B}\sum_{b=1}^{B} Gain(D_b, X_j)$$

RANDOM FOREST

Random Forest

- RF（Breiman，2001）通过对Bagging的微小调整进一步提升了泛化能力：**减弱树的相关性**
- RF在以决策树为基学习器构建Bagging的基础上，进一步在决策树的训练过程中引入了随机特征选择
- 具体来说，传统决策树在选择划分特征时是在当前节点的特征集合（假设有d个特征）中选择一个最优特征
- 而在RF中，对基决策树的每个节点，先从该节点的特征集合中随机抽取k个特征，从中选择最优特征用于划分
  - k控制了随机性的引入程度，若$k = d$，则决策树的构建和传统方法相同；若$k = 1$，则随机选择一个特征进行划分
  - 一般情况下，推荐$k = \log_2 d$或$k = \sqrt{d}$

Random Forest合理性

- 假设数据集当中有一个很强势的特征，还有一些中等强势的特征
- 在Bagging的过程中，大部分决策树都会使用最强势的那个特征作为根节点
- 即使数据集因为Bootstrap产生了不同，但所有的决策树都会看起来非常相似，因而这些树产生的预测也高度相关
- 对许多高度相关的量进行平均，不会像对许多不相关的量进行平均那样导致方差减少那么大
- 这意味着相较于单颗树而言，Bagging并不会带来很大的方差减小
- 随机森林通过强迫在每个节点只考虑一部分特征来解决这一问题，因此平均有$\frac{d-k}{d}$个节点没有考虑最强势的那个特征
- 这大大减弱了树之间的相关性，从而使得树的平均更加可靠

BOOSTING

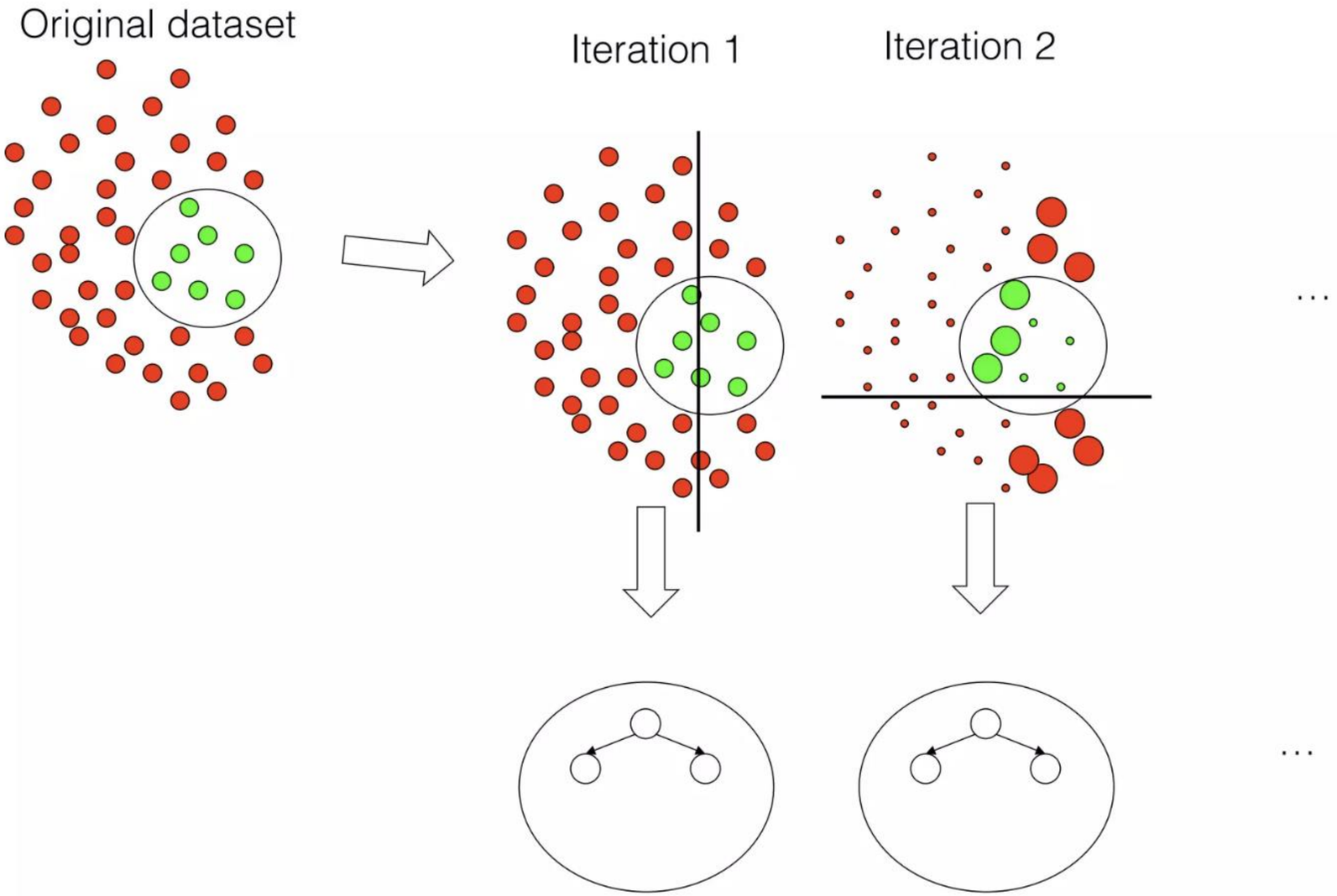
Boosting

- Bagging利用Bootstrap创造了原始训练数据集的多个复制版本，并在每个复制版本上拟合一个单独的决策树，最终结合这些决策树的结果形成单一的预测
- Boosting工作的方法类似，只不过每一棵决策树不再是独立并行，而是序列产生：

每一棵树都是基于之前生长好的树中的信息进行生长

- 以分类为例：Boosting先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练数据的权重进行调整，使得先前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本权重来训练下一个基学习器
- 如此重复进行，直到基学习器数量达到事先指定的值T，最终将这T个基学习器进行加权结合
- 核心：对做出错误或很差的预测的观测赋予更大的权重，从过去犯的错误中学习

Boosting示例：分类



AdaBoost (Freund and Schapire, 1997)

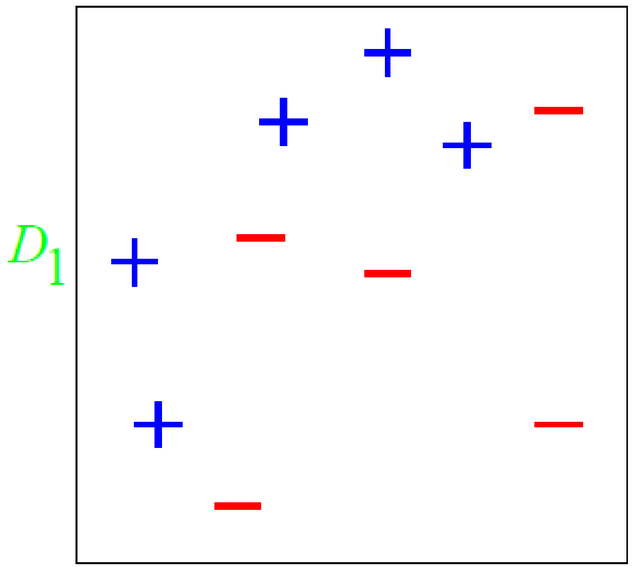
- 输入：训练数据集$D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$，决策树个数 T
- 过程：
  - $D_1^{(i)} = \frac{1}{m}, \quad i = 1, \ldots, m$
  - for $t = 1, \ldots, T$ do
    - 训练决策树：$f_t = DecisionTree(D, D_t)$
    - 计算错误率：$\epsilon_t = P_{D_t}(f_t(\boldsymbol{x}) \neq y) = \frac{\sum_{i=1}^{m} D_t^{(i)} I(f_t(\boldsymbol{x}^{(i)}) \neq y^{(i)})}{\sum_{i=1}^{m} D_t^{(i)}}$
    - 计算第t个树的权重：$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    - 调整观测权重：$D_{t+1}^{(i)} = \left(\frac{D_t^{(i)}}{Z_t}\right) \times \begin{cases} \exp(-\alpha_t), & if\ f_t(\boldsymbol{x}^{(i)}) = y^{(i)} \\ \exp(\alpha_t), & if\ f_t(\boldsymbol{x}^{(i)}) \neq y^{(i)} \end{cases}$ <1: smaller weight >1: larger weight

$$\exp(\alpha_t) = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} > 1$$

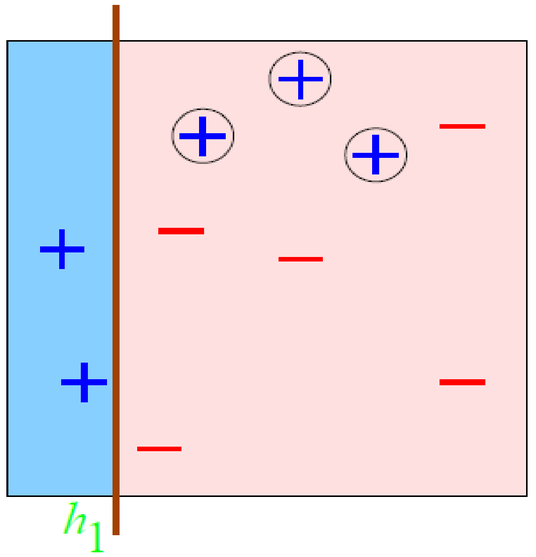
单调递减

- 输出：$f(\boldsymbol{x}) = sign(\sum_{t=1}^{T} \alpha_t f_t(\boldsymbol{x}))$

AdaBoost示例

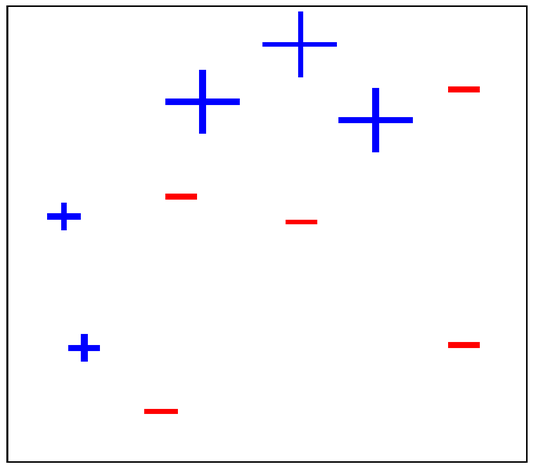


Round 1

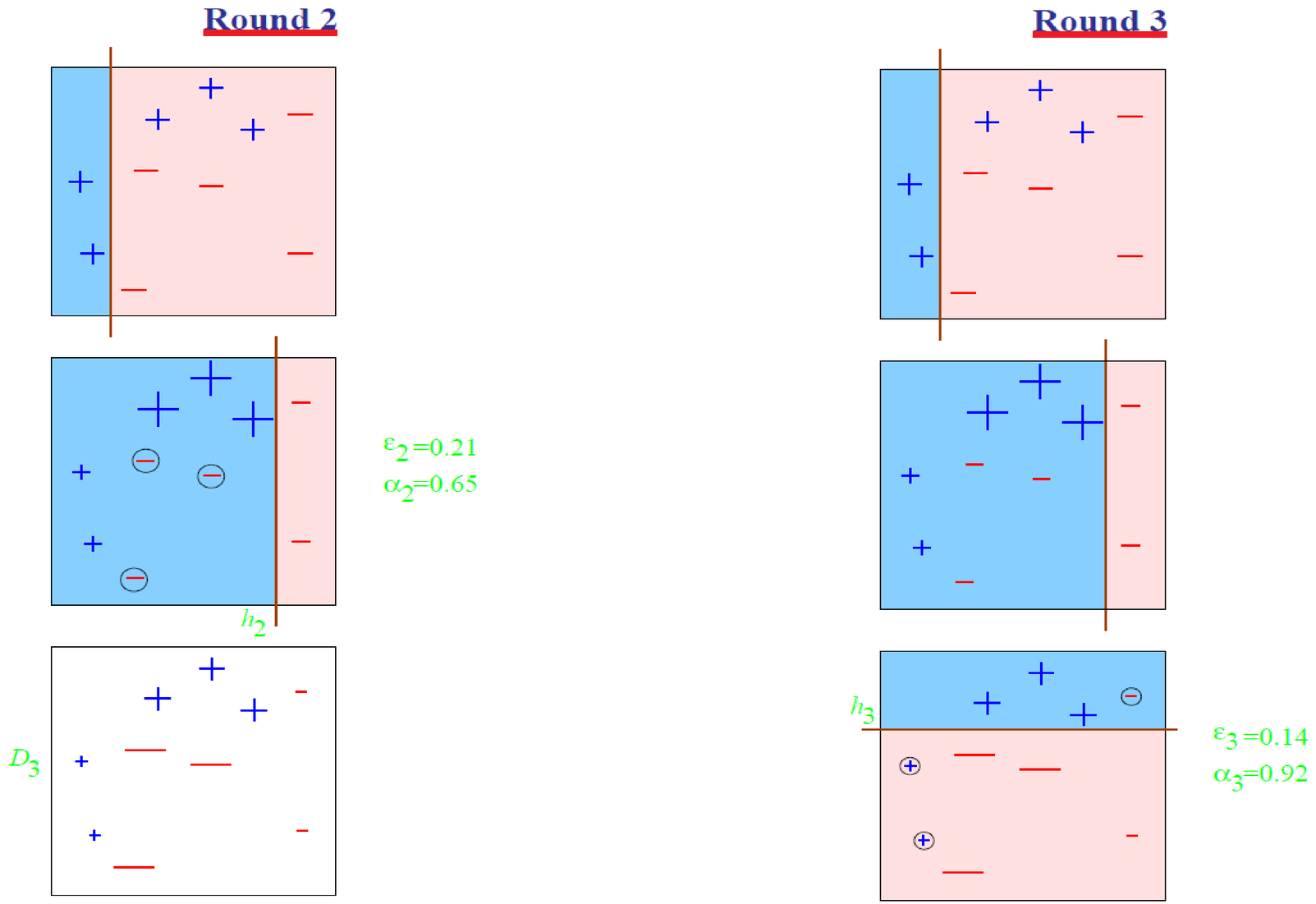


$\varepsilon_1 = 0.30$

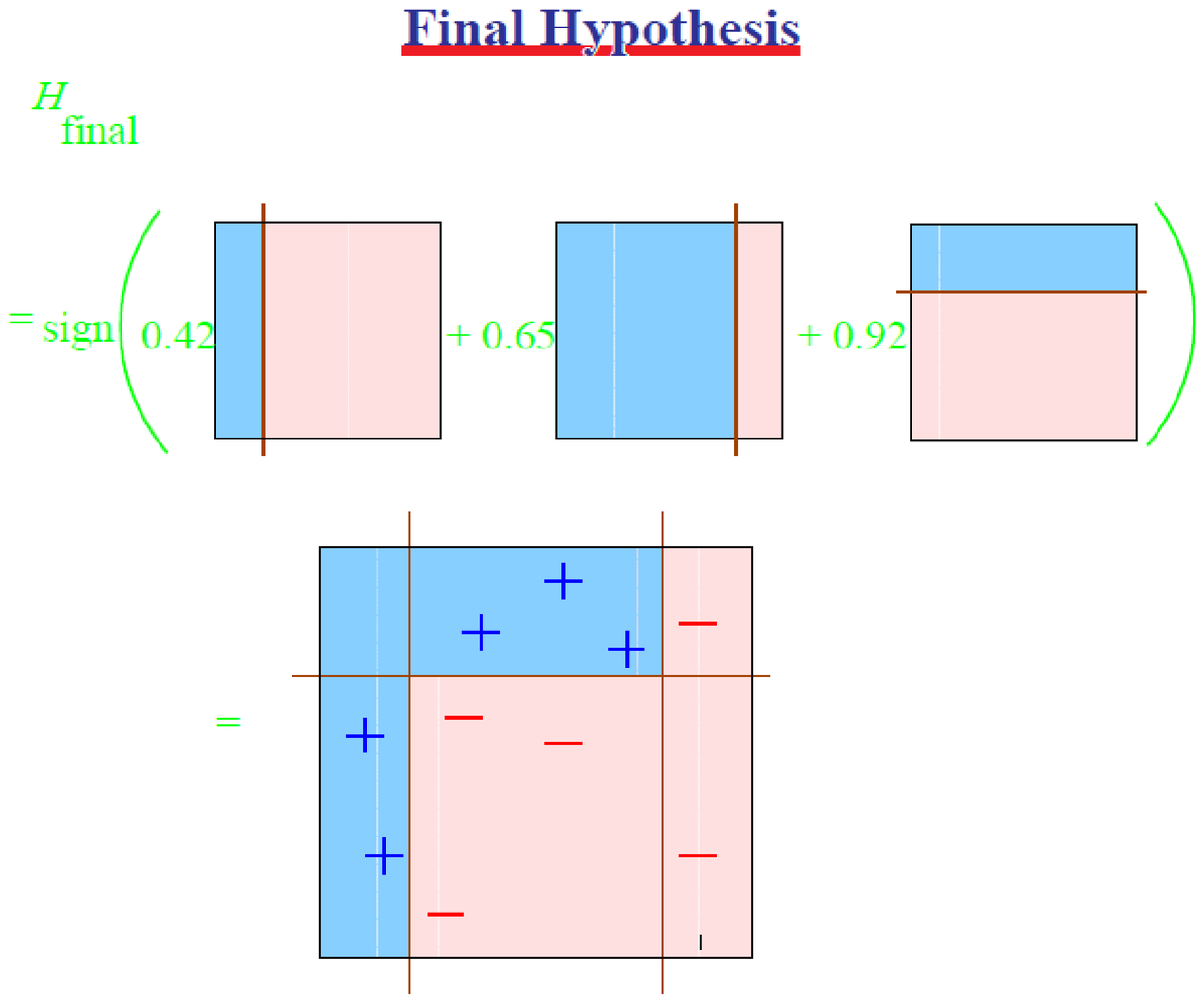
$\alpha_1 = 0.42$



AdaBoost示例


Round 2
ε_2=0.21
α_2=0.65
h_2
D_3
Round 3
h_3
ε_3=0.14
α_3=0.92

AdaBoost示例


Final Hypothesis
H_{final}
$= \text{sign}\left(0.42 \right.$
$+ 0.65$
$+ 0.92 \left. \right)$
$=$

AdaBoost的统计解释

- Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting. The annals of statistics, 2000, 28(2): 337-407.
- AdaBoost可解释为在加法模型下，基于指数损失函数的向前逐步算法
- 加法模型：

$$H_M(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m h_m(\boldsymbol{x})$$

- 指数损失函数：

$$L\big(y, H(\boldsymbol{x})\big) = \exp\{-yH(\boldsymbol{x})\}$$

AdaBoost的统计解释

- 给定训练样本$S^n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$，其中$\boldsymbol{x} \in \mathbb{R}^P$，　$y_i \in \{-1,1\}$
  - 初始化$H_0(\boldsymbol{x}) = 0$
  - 从$m = 1$到M：
    1. 最小化损失函数

$$(\alpha_m, h_m) = \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, H_{m-1}(\boldsymbol{x}_i) + \alpha h(\boldsymbol{x}_i))$$

       得到α_m, h_m

    2. 更新：

$$H_m(\boldsymbol{x}) = H_{m-1}(\boldsymbol{x}) + \alpha_m h_m(\boldsymbol{x})$$

    3. 输出$H_M(x) = \sum_{m=1}^{M} \alpha_m h_m(\boldsymbol{x})$

AdaBoost的统计解释

- 在第m步求解h_m时，考虑指数损失函数下的优化问题：

$$(h_m, \alpha_m) \leftarrow \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1}^{n} \exp(-[H_{m-1}(\boldsymbol{x}_i) + \alpha h(\boldsymbol{x}_i)]y_i)$$

- 化简可得：

$$(h_m, \alpha_m) \leftarrow \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^m \exp(-\alpha h(\boldsymbol{x}_i)y_i)$$

其中$w_i^m = \exp(-H_{m-1}(\boldsymbol{x}_i)y_i)$

- 注意到

$$w_i^m \exp(-\alpha h(\boldsymbol{x}_i)y_i) \propto w_i^m \exp(2\alpha \mathbb{I}_{\{h(\boldsymbol{x}_i)\neq y_i\}})$$

- 因此对于变量h的优化等价于：

$$h_m \leftarrow \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^m \mathbb{I}_{\{h(\boldsymbol{x}_i)\neq y_i\}}$$

AdaBoost的统计解释

- 将得到的h_m代入可以解得

$$\alpha_m = \frac{1}{2}\log(\frac{1-err_m}{err_m})$$

其中err_m是加权的分类误差：

$$err_m = \frac{\sum_{i=1}^{n} w_i^m \mathbb{I}_{\{h(x_i)\neq y_i\}}}{\sum_{i=1}^{n} w_i^m}$$

- 第$m+1$步的权重更新为：

$$\begin{aligned} w_i^{m+1} &= \exp(-H_m(\boldsymbol{x}_i)y_i) \\ &= \exp(-[H_{M-1}(\boldsymbol{x}_i) + \alpha_m h_m(\boldsymbol{x_i})]y_i) \\ &= \exp(-H_{M-1}(\boldsymbol{x}_i)y_i) \cdot \exp(-\alpha_m h_m(\boldsymbol{x_i})y_i) \\ &= w_i^m \cdot \exp(-\alpha_m h_m(\boldsymbol{x_i})y_i) \end{aligned}$$

AdaBoost的统计解释

- 给定训练样本$S^n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$，其中$\boldsymbol{x} \in \mathbb{R}^P$，　$y_i \in \{-1,1\}$，在指数损失下
  - 初始化$H_0(\boldsymbol{x}) = 0$，　$w_i^1 = \frac{1}{n}(i = 1, \dots, n)$
  - 从$m = 1$到M:
    1. $h_m \leftarrow \underset{h}{\operatorname{argmin}} \sum_{i=1}^n w_i^m \mathbb{I}_{\{h(\boldsymbol{x}_i) \neq y_i\}}$
    2. $\alpha_m = \frac{1}{2}\log(\frac{1-err_m}{err_m})$，其中$err_m = \frac{\sum_{i=1}^n w_i^m \mathbb{I}_{\{h(\boldsymbol{x}_i) \neq y_i\}}}{\sum_{i=1}^n w_i^m}$
    3. $w_i^{m+1} = w_i^m \cdot \exp(-\alpha_m h_m(\boldsymbol{x}_i) y_i)$
    4. 更新$H_m(\boldsymbol{x}) = H_{m-1}(\boldsymbol{x}) + \alpha_m h_m(\boldsymbol{x})$
  - 输出$H_M(x) = \sum_{m=1}^M \alpha_m h_m(\boldsymbol{x})$

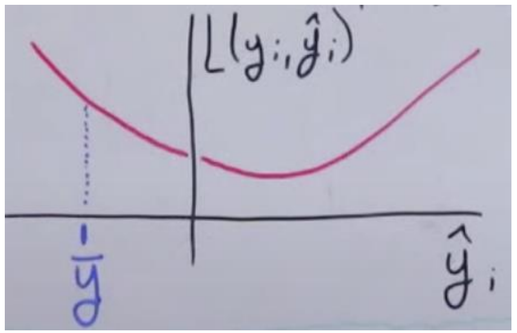
Boosting：回归

- 设定$f(\boldsymbol{x}^{(i)})=0$，$r^{(i)}=y^{(i)}, i=1,\ldots,m$
- for $t=1,\ldots,T$ do
  - 根据训练数据$(\boldsymbol{x}^{(i)}, r^{(i)})$拟合回归树$f_t$
  - 更新响应变量预测值：$f(\boldsymbol{x}^{(i)})=f(\boldsymbol{x}^{(i)})+\lambda\cdot f_t(\boldsymbol{x}^{(i)})$
  - 更新残差：$r^{(i)}=r^{(i)}-\lambda\cdot f_t(\boldsymbol{x}^{(i)})$
- 输出：$f(\boldsymbol{x})=\sum_{t=1}^{T}\lambda\cdot f_t(\boldsymbol{x})$
- λ：学习率

- $r^{(i)}$为每次决策树学习剩余的残差
- 什么叫做残差，比如小明的年龄是10岁，第一次我们的决策树拟合的值为6，那么第二次我们拟合的目标值为10-6=4
- 第二次决策树拟合值为2，那么第三次我们拟合的目标值为10-6-2=2
- 通过不断的对残差进行拟合，我们可以缓慢的提升f尚未拟合充分的区域，最终得到一个很好的集成结果
- 函数估计/近似：在函数空间上做数值优化，而不是在参数空间

Gradient Boost Decision Tree

- GBDT将负梯度作为残差值来学习基学习器，在迭代的每一步构建一个能够沿着梯度最陡的方向降低损失的学习器来弥补已有模型的不足，稳健性强
- Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.
- Step 0：定义可微的损失函数$L(y,\hat{y})$ ，最好能很高效的可微
- Step 1：以回归为例，初始化$f(\boldsymbol{x}) = f_1(\boldsymbol{x}) = \bar{y}$
- Step 2：计算残差$r_1^{(i)} = -\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial \hat{y}^{(i)}}|_{\hat{y}^{(i)}=f(\boldsymbol{x}^{(i)})},\ i = 1,\dots,m$
- Step 3：针对训练数据$\left(\boldsymbol{x}^{(i)}, r_1^{(i)}\right)_{i=1}^{m}$训练新的弱学习器$f_2(\boldsymbol{x})$



GBDT

- Step 0：定义可微损失函数$L(y,\hat{y})$
- Step 1：初始化$f(\boldsymbol{x})=f_1(\boldsymbol{x})=\bar{y}$
- Step 2：计算残差

$$r_1^{(i)}=-\frac{\partial L\left(y^{(i)},\hat{y}^{(i)}\right)}{\partial \hat{y}^{(i)}}\Big|_{\hat{y}^{(i)}=f\left(\boldsymbol{x}^{(i)}\right)}$$
$$i=1,\ldots,m$$

- Step 3：针对训练数据$\left(\boldsymbol{x}^{(i)},r_1^{(i)}\right)_{i=1}^{m}$训练新的弱学习器$f_2(\boldsymbol{x})$
- Step 4：Figure out “how much” of $f_2(\boldsymbol{x})$ to add:

$$\gamma_2=\underset{\gamma}{\operatorname{argmin}}\sum_{i=1}^{m}L\left(y^{(i)},f_1\left(\boldsymbol{x}^{(i)}\right)+\gamma f_2\left(\boldsymbol{x}^{(i)}\right)\right)$$

- Step 5：$f(\boldsymbol{x})=f_1(\boldsymbol{x})+\gamma_2 f_2(\boldsymbol{x})$；回到Step 2，直到共生成$T$个弱学习器

Why GBDT works

- 在第k次循环：$f(\boldsymbol{x}^{(i)}) := f(\boldsymbol{x}^{(i)}) + \gamma_k f_k(\boldsymbol{x}^{(i)})$
- $f(\boldsymbol{x}^{(i)}) + \gamma_k f_k(\boldsymbol{x}^{(i)}) \approx f(\boldsymbol{x}^{(i)}) + \gamma_k r_k^{(i)} = f(\boldsymbol{x}^{(i)}) - \gamma_k \cdot \frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \hat{y}^{(i)}}|_{\hat{y}^{(i)}=f(\boldsymbol{x}^{(i)})}$
- $f(\boldsymbol{x}^{(i)}) := f(\boldsymbol{x}^{(i)}) - \gamma_k \cdot \frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \hat{y}^{(i)}}|_{\hat{y}^{(i)}=f(\boldsymbol{x}^{(i)})}$
- New model=Prev model + Step in Direction of Decreasing Loss
- 针对f做梯度下降，γ_k为学习率

XGBoost

- Extreme Gradient Boosting，可看作是GBDT算法的改进
- 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*.
- XGBoost在Kaggle等数据科学竞赛中获得了很好的表现
- 如果不知道怎样开始你的项目，XGBoost是一个不错的选择，常被作为基准方法
- 精度高、速度快、引入正则化不会过拟合

XGBoost

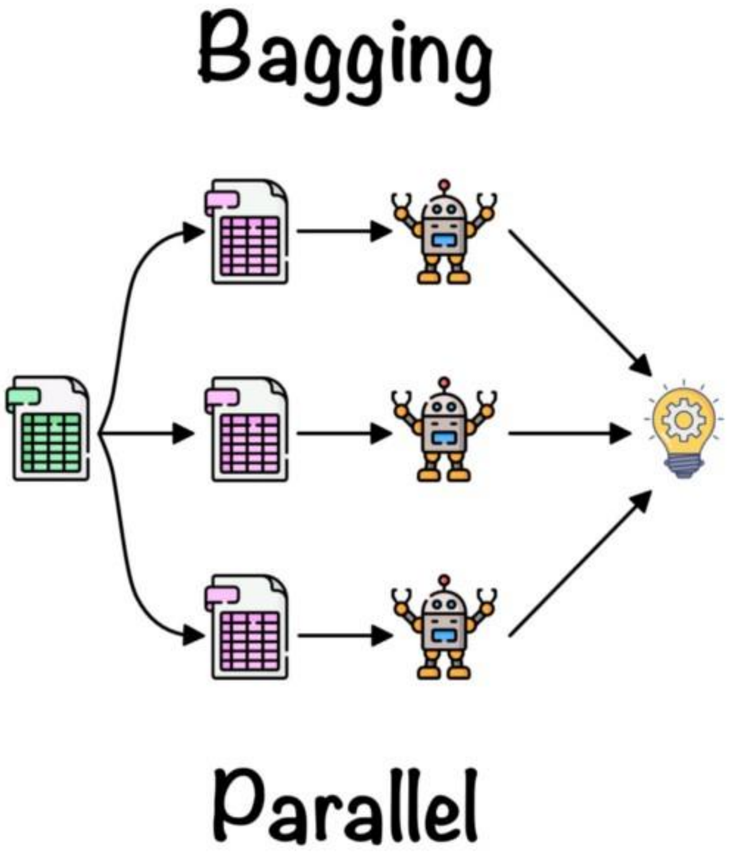
- 最终模型： $f(\boldsymbol{x}) = \sum_{t=1}^{T} f_t(\boldsymbol{x})$
- 定义目标函数： $\mathcal{L} = \sum_{i=1}^{m} L\left(y^{(i)}, \hat{y}^{(i)}\right) + \sum_t \Omega(f_t)$
- 正则化项Ω惩罚了树的复杂度，可能的定义方式包括：树中的节点数量、叶节点权重的L1/L2范数等
- 当正则化项为0时，XGBoost退化为GBDT

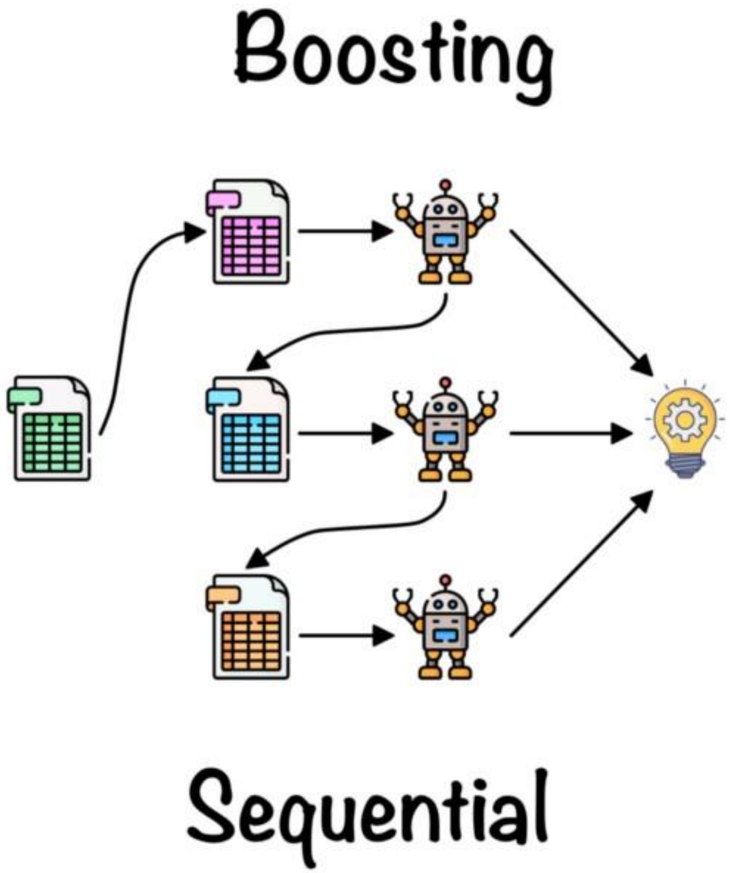
XGBoost

- $\hat{y}_0^{(i)} = 0$
- $\hat{y}_1^{(i)} = f_1(\boldsymbol{x}^{(i)}) = \hat{y}_0^{(i)} + f_1(\boldsymbol{x}^{(i)})$
- …
- $\hat{y}_T^{(i)} = \hat{y}_{T-1}^{(i)} + f_T(\boldsymbol{x}^{(i)})$
- f_t的选取要最小化目标函数：

$$\mathcal{L}_t = \sum_{i=1}^{m} L\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(\boldsymbol{x}^{(i)})\right) + \Omega(f_t)$$

BAGGING & BOOSTING





本章小节

- Bagging
- Random Forest

- Boosting
- AdaBoost
- GBDT
- XGBoost

- 纸上得来终觉浅，绝知此事要躬行

期末安排

- 16周课程：TBD
- 答疑时间： TBD
- 考试时间： TBD

期末考试

- 题型： TBD
- 解答题包括： TBD
- 考试范围： TBD
- 预祝大家考试顺利！