7. 集成学习

构建并结合多个模型来更好地（比起单一模型）完成任务。

线性融合：

$$y_{COM}(x) = \frac{1}{M}\sum_{m=1}^{M} y_m(x)$$

使用最简单的权重一样的组合。

这里引入一些名词：

$$\hat{E} = \overline{E} - \overline{A}$$

$\hat{E}$：集成学习错误率

$\overline{E}$：基模型平均错误率

$\overline{A}$：基模型的多样性

集成学习需要集成独立、多样、有一定效果的基模型。

– 异质集成vs 同质集成

基模型是通过同一个算法学习出来的，称为同质集成homogeneous ensemble 基模型是通过不同算法学习出来的，称为异质集成heterogenous ensemble – 平行集成vs 序贯集成

基模型的学习是独立的，可以平行完成的，称为平行集成parallel ensemble 基模型的学习不是独立的，需要按顺序完成的，称为序贯集成sequential ensemble

经典算法：

Stacking

– Stacking

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
初级学习算法 $\mathfrak{L}_1, \mathfrak{L}_2, \ldots, \mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.

过程：

```
for t = 1,2,...,T do
    h_t = L_t(D);
end for
D' = ∅;
for i = 1,2,...,m do
    for t = 1,2,...,T do
        z_it = h_t(x_i);
    end for
    D' = D' ∪ ((z_i1, z_i2, ..., z_iT), y_i);
end for
h' = L(D');
```

输出： $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
初级学习算法 $\mathfrak{L}_1, \mathfrak{L}_2, \ldots, \mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.

过程：

```
D ⟶ D_1, D_2, ..., D_k
for j = 1,2,...,k do
    D̄_j = D \ D_j
    h_t^(j) = L_t(D̄_j)
    for x_i ∈ D_j do
        for t = 1,2,...,T do
            z_it = h_t^(j)(x_i)
        end for
        D' = D' ∪ ((z_i1, z_i2, ..., z_iT), y_i)
    end for
end for
h' = L(D');
```

输出： $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$

原本的多个模型训练的精度已经到达了上限，从原始数据到target已经无法再提高了，那我们就不训练原始数据了，从上一个模型的输出的直接去拟合target。但是容易过拟合：学习基学习器的数据与学习融合器的数据发生了重合，解决办法是用学习基模型未使用的样本来形成学习融合器的训练样本（在第i个模型上不使用Di数据集训练，把Di数据集作为测试集生成zi）

Bagging

减少一个估计方差的一种方式就是对多个估计进行平均。

原始样本中有放回抽取训练集，k次抽取获得k个训练集，得到k个同类模型，分类问题采用投票方式得出分类结果，回归问题选取均值作为最后结果。

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
3: **end for**

输出： $H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y)$

支持包外估计：

$$H^{oob}(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y) \cdot \mathbb{I}(\boldsymbol{x} \notin D_t) \quad \epsilon^{oob} = \frac{1}{|D|} \sum_{(\boldsymbol{x}, y) \in D} \mathbb{I}(H^{oob}(\boldsymbol{x}) \neq y)$$

包外估计是选取该训练集没抽取到的元素的包进行检验。（没错看起来很高级的亚子但是应该就是这个意思）

随机森林类似bagging的方式，bagging获取基学习器的多样性仅仅来自于样本扰动，随机森林不仅如此，还有自属性扰动，尽可能希望森林更深更广，所以不会进行剪枝。

Boosting

通过算法集合将弱学习器转换为强学习器，先训练出一个基学习器，根据学习结果对训练样本进行调整，使错分样本得到更多关注，基于调整训练下一个学习器，直到基学习器的数目到达预期值，进行加权结合。

> 插播一句：对于三类集成学习模型的对比概括：
>
> stacking像是接力，第一步精度提不上去了那就用第一步的结果的继续提高，有点符合"堆积"的意思
>
> bagging像是从袋子里（数据里）取不同的数据用一种基模型反复做
>
> boosting是基于第一步的结果进行权重的重新分配，对好的结果权重提升，符合"提升"的意思（不太严谨，不一定是提升）
>
> bagging追求低方差，boosting追求低偏差，分类准确度高。

Adaboost

提高前一轮弱分类器错误分类样本的权值，降低分类正确的样本权值；同时提高准确度高的弱分类器权值，降低准确度低的弱分类器权值，误差率小的分类器有更多票数。

- Adaboost

给定数据 $(x_i, y_i), i = 1, 2, \ldots, n$

初始化权重 $w_i = 1/n \ , i = 1, 2, \ldots, n$

For $m = 1, \ldots, M$

利用带权重 w_i 的数据学习第 m 个基模型 $G_m \in -1,1$

然后计算它的错误率

$$\text{err}_m = \frac{\sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i^{(m)}} \quad \text{(分母能令权重归一化)}$$

设定

$$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

$$w_i^{(m+1)} = w_i^{(m)} \, e^{-y_i \beta_m G_m(x)} \quad , i = 1, 2, \ldots, n$$

获得AdaBoost 模型 $G(x) = \text{sgn}\left(\sum_{m=1}^{M} \beta_m G_m(x) \right)$

（上面的Ind就是I，sgn就是符号函数）看起来有些莫名其妙：为什么这样设定β与w的权重?

β：基分类器在最终分类器上面所占比重，因为每个分类器效果要好于随机分类器，所以err均小于0.5，并且随着err的减少，β增大，故误差率小的分类器在最终权重最大。

w的权重：（这里进行了标准化）

$$W_{m+1,i} = \frac{w_i^{(m)}}{\sum_{i=1}^{N} w_i^{(m)}} exp\{-y_i \beta_m G_m(x)\} = \begin{cases} \frac{w_i^{(m)}}{\sum_{i=1}^{N} w_i^{(m)}} e^{-\beta_m}, & G_m(x_i) = y_i \\ \frac{w_i^{(m)}}{\sum_{i=1}^{N} w_i^{(m)}} e^{\beta_m}, & G_m(x_i) \neq y_i \end{cases}$$

分对分错权重相差

$$e^{2\beta_m} = \frac{1 - err_m}{err_m}$$

由于βm>0，故而exp(-βm)<1，当样例被基本分类器正确分类时，其权重在减小，反之权重在增大。通过增大错分样例的权重，让此样例在下一轮的分类器中被重点关注，通过这种方式，慢慢减小了分错样例数目，使得基分类器性能逐步改善。

算法的另一种解释：

是模型为加法模型(b函数可以看成G_m函数)

$$f(x)=\sum_{m=1}^{M}\beta_m b(x_i;\gamma_m)$$

损失函数为指数函数，学习算法采取的是向前分步算法时的二分类学习方法。

证明采取的是向前分布算法：

算法 8.2（前向分步算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$；损失函数 $L(y,f(x))$；基函数集 $\{b(x;\gamma)\}$；

输出：加法模型 $f(x)$.

（1）初始化 $f_0(x)=0$

（2）对 $m=1,2,\cdots,M$

（a）极小化损失函数

$$(\beta_m,\gamma_m)=\arg\min_{\beta,\gamma}\sum_{i=1}^{N}L\left(y_i,f_{m-1}(x_i)+\beta b(x_i;\gamma)\right) \tag{8.16}$$

得到参数 β_m，γ_m

（b）更新

$$f_m(x)=f_{m-1}(x)+\beta_m b(x;\gamma_m) \tag{8.17}$$

（3）得到加法模型

$$f(x)=f_M(x)=\sum_{m=1}^{M}\beta_m b(x;\gamma_m) \tag{8.18}$$

成功讲向前分步算法将求解1-M所有参数的β，γ问题变成了求解各个β、γ的优化问题。

下面讲为什么损失函数

$$L(y,f(x))=exp\{-yf(x)\}$$

是指数函数时的向前分步算法等价于adaboost算法：

下式是损失函数为指数函数的向前分步算法

$$(\beta_m,G_m(x))=\arg\min_{\beta,G}\sum_{i=1}^{N}\exp\{-y_i(f_{m-1}(x_i)+\beta G(x_i))\}$$

$$=\arg\min_{\beta,G}\sum_{i=1}^{N}\overline{w}_{mi}\exp\{-y_i\beta G(x_i)\},\overline{w}_{mi}=\exp\{-y_i f_{m-1}(x_i)\}$$

w依赖于f_m-1，每一轮迭代发生改变，不依赖于β与G，所以问题现在变成了证明这个式子中最小的那个β和G就是adaboost里面的β和G

先求最小的G：继续对原式进行变化：

$$G_m^*(x) = \arg\min_G \Big(\sum_{y_i = G_m(x_i)} \overline{w}_{mi} e^{-\beta} + \sum_{y_i \neq G_m(x_i)} \overline{w}_{mi} e^{\beta} \Big)$$

$$= \arg\min_G \Big((e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} \overline{w}_{mi} I(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{N} w_{mi} \Big)$$

$$= \arg\min_G \sum_{i=1}^{N} \overline{w}_{mi} I(y_i \neq G(x_i))$$

所以该G函数就是adaboost算法里面的基本分类器，是使第m轮加权训练之后分类误差率最小的基本分类器。（看看多么巧）

现在再求最小的β：(其实跟上面差不太多，后面多了求导)

$$\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta\, G(x_i)\} = \min_{\beta} \left(\min_{G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta\, G(x_i)\} \right)$$

$$\beta^* = \arg\min_{\beta} \left((e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \mathsf{Ind}(y_i \neq G_m(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \right)$$

对β的偏导置零可得：

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathsf{err}_m}{\mathsf{err}_m} \qquad \mathsf{err}_m = \frac{\sum_{i=1}^{n} w_i^{(m)} \mathsf{Ind}(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i^{(m)}}$$

Gradient boosting

adaboost是第一个成功的提升算法，是基于某种特定损失函数的梯度下降算法。

便于理解梯度提升的场景：

想象你被给了一组训练样本$(x_1,y_1)\cdots(x_n,y_n)$ 当前任务是学习到一个模型 $F(x)$去拟合它们.
再想象此时已经有一个朋友为你提供了一个模型 F.
然而当你对这个模型做check之后发现这个模型并不完美，因为它会犯几个错误：
F(x1) = 0.8, while y1 = 0.9, and F(x2) = 1.4 while y2 = 1.3...
那么你会考虑如何改进模型F呢 (当你被限制在不能移除当前模型F的任何东西或者修改F里的参数)

解决方案: 我们考虑若添加一个额外的子模型h在F的后面，即此时的模型将被更新为F(x) + h(x).
对于h的表现: F(xi) + h(xi) = yi, i =1...n. 或者说: h(xi) = yi − F (xi).
所以，我们应该令 h 去拟合数据 (x1, y1− F(x1)), (x2, y2− F(x2)), ..., (xn, yn− F(xn)).
然后令 $F(x)$ = F(x) + h(x).

之后 $F(x)$ 可以看作前面的F来被继续更新

基于前向分步方式学习加法模型，在每一步，算法会引入一个基模型来补充现有的模型的缺点，这种缺点是被梯度刻画的——负梯度就是模型当前的缺点。

负梯度就提示了模型当前的 *shortcomings*，用新的基模型来拟合它加入到旧模型
更一般地，梯度提升算法可以被总结为:

Step 1. 初始化$F(x)$ ，例如 $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
Step 2. 不断更新F, 直到收敛:
计算负梯度$-g(x_i) = -\partial L(y_i, F(x_i)) / \partial F(x_i)$
以$-g(x_i)$为目标学习拟合函数h
更新$F := F + \rho h, \rho = 1$ 或线搜索 *line search* 来设定

上面的框架也适用于其他损失函数，例如当$L(y, F) = |y - F|$，$-g(x_i) = sign(y_i - F(x_i))$

Gradient boosting Decision Trees

决策函数：F为基模型

$$\hat{y}_i = \phi(X_i) = \sum_{k=1}^{K} f_k(X_i), \quad f_k \in F$$

$$F = \{f(x) = w_{q(x)}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$$

损失函数:

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$$