Notation.

- $\mathcal{A}$: the learning algorithm
- $\mathcal{H}$: hypothesis space
- $\mathcal{D}$: the set of all samples
- $\mathbb{P}$: the population distribution
- $\mathbb{E}_{X\sim\mathbb{P}}[z(X)]$: the expectation of the random variable $z(X)$, i.e. the Lebesgue integral $\int z(X)\, d\mathbb{P}$
- f: the target function
- $g^{(\mathcal{D})}$: an output hypothesis function trained on $\mathcal{D}$
- g^{-}: an output hypothesis function trained on $\mathcal{D}_{\text{train}}$
- $\overline{g}(\mathbf{x})$: $\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x})]$

Generally speaking, ensemble is the name for a set of machine learning methods which combine multiple hypothesis in hope to get a better one. It can typically improve and regularize the model at the same time.

1 Stacking & Blending

Stacking (aka. stacked generalization) and blending are two similar approaches to combine different models without resampling the dataset. The core idea is to take the outputs of these models to train a final model f.

Suppose there are n samples and M models $f_1, \ldots, f_M$ in total. In stacking, we first apply k-fold cross validations on all the models $f_1, \ldots, f_M$ to obtain the predictions for the whole training set, i.e. a matrix of size $n \times M$, and then use these predictions to train the final model. Let f_m^{-i} be the model f_m trained without the i-th fold, then the training set for the final model is

$$\bigcup_{i=1}^{k} \{(f_1^{-i}(x), \ldots, f_M^{-i}(x), y) | (x, y) \in \mathcal{D}^i\}$$

For example, if f is a linear combination of $f_1, \ldots, f_M$ and the task is regression, then

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{k} \sum_{(x,y)\in\mathcal{D}^i} \left[y - \sum_{m=1}^{M} w_m f_m^{-i}(x) \right]^2$$

Blending is a simplification of stacking: instead of a k-fold cross validation, we split the whole training set $\mathcal{D}$ into 2 parts $\mathcal{D}_1$ and $\mathcal{D}_2$, train $f_1, \ldots, f_M$ on $\mathcal{D}_1$ and obtain their predictions on $\mathcal{D}_2$, which then become the training samples for the final model f.

2 Bagging

Bagging is the abbreviation for Bootstrap AGgregation. Consider the bias-variance decomposition:

$$\mathbb{E}_{\mathcal{D}}[E_{\text{out}}(g^{(\mathcal{D})})] = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[\underbrace{\mathbb{E}_{\mathcal{D}}[(g^{(\mathcal{D})}(\mathbf{x}) - \overline{g}(\mathbf{x}))^2]}_{\text{var}(\mathbf{x})} + \underbrace{(\overline{g}(\mathbf{x}) - f(x))^2}_{\text{bias}(\mathbf{x})}]$$

When the learning algorithm $\mathcal{A}$ is sensitive to $\mathcal{D}$(i.e. the var(x) is large), the performance of $\overline{g}(\mathbf{x})$ is hopefully better than that of a simple $g^{(\mathcal{D})}$. Since it's impossible to obtain all the samples from $\mathbb{P}$, the method bootstrap is used to approximate sampling from $\mathbb{P}$: sampling from $\mathcal{D}$ with replacement T times, obtaining 'new samples' $\tilde{\mathcal{D}}_1, \ldots, \tilde{\mathcal{D}}_T$. Then we use a uniform aggregation to approximate $\overline{g}$:

$$\frac{1}{T}\sum_{i=1}^{T} g^{(\tilde{\mathcal{D}}_i)}$$

Such a process is called **bootstrap aggregation**.

3 Boosting

According to the wiki, most boosting algorithms consist of iteratively learning weak classifiers and adding them to a final strong classifier.

3.1 AdaBoost

AdaBoost is the abbreviation for adaptive boosting. The core idea of adaboost is to aggregate diverse hypotheses each of which focuses more on a specific part of the samples. Such a diversity is reflected by a weight of each sample.

Suppose, we already have a prediction model

$$g_t = \arg\min_{h\in\mathcal{H}} \left(\sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t)} \,\text{err}(y_n, g_1(\mathbf{x}_n)) \right)$$

Then for the samples $\mathcal{D}_{\text{train}}^{(t)}$ on which g_t performs bad(like $\text{err}(\mathbf{x}_n, y_n)$ larger than a certain threshold), we can somehow increase their weights $w_n^{(t+1)} > w_n^{(t)}$ and then obtain another model

$$g_{t+1} = \arg\min_{h\in\mathcal{H}} \left(\sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t+1)} \,\text{err}(y_n, g_1(\mathbf{x}_n)) \right)$$

which focus more on $\mathcal{D}_{\text{train}}^{(t)}$.
When $\text{err}(g(\mathbf{x}_n), y_n) = \mathbf{1}_{g(\mathbf{x}_n)\neq y_n}$, to make g_{t+1} distinct to g_t as much as possi-

ble, we hope that

$$\frac{\sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t+1)} \mathbf{1}_{g(\mathbf{x}_n)=y_n}}{\sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t+1)}} = \frac{1}{2}$$

i.e. g_t just functions like a random guess. This is equivalent to ensure

$$\sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t+1)} \mathbf{1}_{g(\mathbf{x}_n)=y_n} = \sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t+1)} \mathbf{1}_{g(\mathbf{x}_n)\neq y_n}$$

Denote $\epsilon_t = \sum_{n=1}^{|\mathcal{D}_{\text{train}}|} w_n^{(t)} \mathbf{1}_{g(\mathbf{x}_n)\neq y_n}$, then the above equality can be reached if we update

$$w_n^{(t+1)} \to w_n^{(t)} \begin{cases} \cdot\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} & g(\mathbf{x}_n) \neq y_n \\ /\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} & g(\mathbf{x}_n) = y_n \end{cases}$$

By iteration of T times(each time the learning algorithm $\mathcal{A}$ and the hypothesis space $\mathcal{H}$ are the same), we can obtain T diverse hypotheses $g_1, g_2, \ldots, g_T$. And to our intuition, the initial weights are the same. The last question is how to aggregate them. The idea behind adaboost is the use of linear aggregation and that more accurate the model, larger the linear weight:

$$G(x) = \sum_{t=1}^{T} \alpha_t g_t,\ \alpha_t = \ln \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

note that α_t is decreasing with ϵ_t and $\alpha_t = 0$ iff $\epsilon_t = 0.5$.

It is also guaranteed that

$$E_{\text{out}}(G) \leq E_{\text{in}}(G) + O\left(\sqrt{O(d_{\text{vc}}(\mathcal{H}).T \log T)\frac{\ln |\mathcal{D}_{\text{train}}|}{|\mathcal{D}_{\text{train}}|}}\right)$$

and $E_{\text{in}}(G) = 0$ after $T = O(\log |\mathcal{D}_{\text{train}}|)$ iterations if $\epsilon_t < 0.5$ for each iteration.

Typically, Adaboost is used along with the Decision Tree. In order to not modify either Adaboost or Decision Tree, we usually resample the training set by weight $w_n^{(t)}$ to approximate their updates through iterations. But note that ϵ is still defined on the original $\mathcal{D}_{train}$.