- AdaBoost

  What exactly does it do? The resulting classifier will always have a good prediction accuracy?

- Forward stage-wise optimization for fitting an additive model

  AdaBoost is a special case of this framework with Exponential loss for classification. Similarly we can develop Boosting algorithms for regression/classification with other loss functions.

Weak Classifiers

- Consider a binary classification problem where $x \in \mathbb{R}^p$ and y is coded as ± 1. A classifier g maps the p-dim feature to $\{-1, 1\}$, namely,

$$g : x \longrightarrow \{-1, 1\}.$$

- g is a weak classifier, if its performance is just slightly better than random guessing. E.g., decision stumps (classification tress with only two leaf nodes).

  It's possible that the mis-classification rate of g is more than $1/2$, i.e., worse than random guessing. Then we just use $-g(x)$ for prediction.

Boost Weak Classifiers

Aim : use a combination of weak classifiers to improve the performance.

- Sequentially modify the weights on the training data $\{w_i\}_{i=1}^n$;
- Sequentially pick classifiers $g_t(x)$;[a]
- Output the weighted version

$$G(x) = \mathsf{sign}\Big(\sum_{t=1}^{T} \alpha_t g_t(x)\Big).$$

[a] The algorithm still works if $g_t(x)$'s are chosen randomly.

AdaBoost

1. Initialize the weights $w_i^{(1)} = 1/n$, $i = 1, 2, \ldots, n$.

2. For $t = 1$ to T:

   (a) Fit a classifier $g_t(x)$;

   (b) Compute the training error wrt weights $w_i^{(t)}$'s

   $$\epsilon_t = \sum_i \overset{= \frac{1}{n}}{w_i^{(t)}} I\big(y_i \neq g_t(x_i)\big)$$

   weighted error rate

   (c) Compute $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$;

   (d) Update weights $w_i^{(t+1)} = w_i^{(t)} \frac{\exp[-\alpha_t y_i g_t(x_i)]}{Z_t}$, where Z_t is the normalizing constant to ensure that $\boxed{\sum_i w_i^{(t+1)} = 1.}$

   ✓ $w_i^{(t)} \searrow$ ✗ $w_i^{(t)}$

3. Output $G_T(x) = \text{sign}\big(\sum_{t=1}^{T} \alpha_t g_t(x)\big)$.

Next we show that the Training Error (measured by mis-classification rate) will go to 0 (not necessarily monotonically) when $T \to \infty$.

number of mis classified

update

$$
\begin{aligned}
\text{Training-Err}(G_T) &= \sum_i \frac{1}{n} I\Big(y_i \neq \text{sign}\big(\sum_{t=1}^{T} a_t g_t(x_i)\big)\Big) \\
&= \sum_i \frac{1}{n} I\Big(\sum_{t=1}^{T} y_i a_t g_t(x_i) < 0\Big) \\
&\leq \sum_i \frac{1}{n} \exp\Big(- \sum_{t=1}^{T} \alpha_t y_i g_t(x_i)\Big) && (1) \\
&\leq \prod_{t=1}^{T} Z_t \leq \exp\Big\{ - 2 \sum_t (\frac{1}{2} - \epsilon_t)^2 \Big\}. && (2)
\end{aligned}
$$

results of each classifier

average

y_i and $\sum_{t=1}^{T} \cdots$ are not both pos or neg

We use the following results

because $w_i^{(1)}$ is $\frac{1}{n}$

- At (1), $I(z < 0) < \exp(-z)$ where z is any number in $\mathbb{R}$.

there is no $\frac{1}{n}$ in here

- At (2),

weight

sum

$$\sum_{i=1}^{n} \frac{1}{n} \exp\Big(-\sum_{t=1}^{T} \alpha_t y_i g_t(x_i) \Big)$$

$$= \sum_i \frac{1}{n} \prod_{t=1}^{T} \exp\Big(-\alpha_t y_i g_t(x_i) \Big) = \sum_i w_i^{(1)} \prod_{t=1}^{T} \frac{w_i^{(t+1)}}{w_i^{(t)}} Z_t$$

$$= \sum_i \cancel{w_i^{(1)}} \frac{\cancel{w_i^{(2)}}}{\cancel{w_i^{(1)}}} \cdots \frac{\cancel{w_i^{(T)}}}{\cancel{w_i^{(T-1)}}} \frac{w_i^{(T+1)}}{\cancel{w_i^{(T)}}} \Big(\prod_{t=1}^{T} Z_t \Big) = \Big(\prod_{t=1}^{T} Z_t \Big) \sum_i w_i^{(T+1)}$$

Z_{t+1}

$$= \prod_{t=1}^{T} Z_t = \exp\Big\{ -2 \sum_t \big(\frac{1}{2} - \epsilon_t\big)^2 \Big\}$$

which decreases with T if $\epsilon_t < 1/2$, where

$$
\begin{aligned}
Z_t &= \underline{\sum_i w_i^{(t)} \exp\left(-\alpha_t y_i g_t(x_i)\right)} \\
&= \underline{\sum_{i:y_i g_t(x_i)=1} w_i^{(t)} \exp\left(-\alpha_t\right)} + \sum_{\underline{i:y_i g_t(x_i)=-1}} w_i^{(t)} \exp\left(\alpha_t\right) \\
&= (1-e_t)\exp\left(\alpha_t\right) + \epsilon_t \exp\left(-\alpha_t\right) \\
&= (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)} < 1,
\end{aligned}
$$

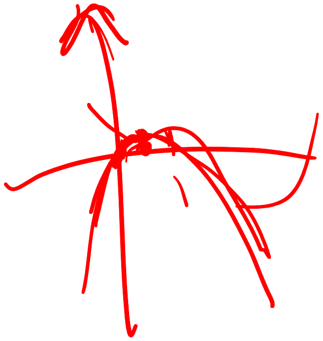
if $\epsilon_t < 1/2$.

$-\frac{b}{2a}$ $\frac{1}{2}$

$-\varepsilon_t^2 + \varepsilon_t$ $-\frac{1}{4} + \frac{1}{2}$

$\varepsilon_t - \varepsilon_t^2$

$\frac{1}{1}$



- We can use a classifier $g_t(x)$ whose error rate $\epsilon_t > 1/2$ (i.e., worse than random-guessing).

  Then $\alpha_t < 0$, and Adaboost basically uses $-g_t(x)$.

- The training error of the combined classifier G_T (from Adaboost) is **not** monotonically decreasing with T.

  After each iteration, Adaboost decreases a particular upper-bound of the 0/1 training error. So in a long run, the training error will be pushed to zero.

- The Adaboost algorithm outputs a classifier G_T with small generalization error. No.

What Does AdaBoost Do?

- Combine weak classifiers to reduce the 0/1 training error (or more specifically, reduce an upper bound of the training error).

- The classifier returned by AdaBoost is not guaranteed to have a good performance on test sets.

- In fact AdaBoost is prone to overfitting, unless it stops early.

Boosting: Forward Stagewise Additive Modeling

- Consider an Additive model:

$$f(x) = \sum_{t=1}^{T} \beta_t b(x; \gamma_t),$$

  where $b(x;\gamma)$ is a classifier or a regression function characterized by parameter γ. For example, $b(x;\gamma)$ could be a linear function with coefficient γ or a small tree with parameter γ.

- It is difficult to optimize over all T pairs of parameters:

$$\min_{\{\beta_t,\gamma_t\}_1^T} \sum_{i=1}^{n} L(y_i, f(x_i)).$$

Forward Stagewise Optimization

(1) $f_0(x) = 0$

(2) For $t = 1$ to T,

- Given f_{t-1}, choose (β_t, γ_t) to minimize

$$\sum_i L\Big(y_i, f_{t-1}(x_i)\Big) + \beta b(x_i; \gamma); \tag{3}$$

- Update $f_t(x) = f_{t-1}(x) + \beta_t b(x; \gamma_t)$.

Boosting algorithms can take various forms, depending on the choice of the base model $b(\cdot; \gamma)$, the choice of the loss function $L(y, f(x))$, and how optimization is done at (3).

AdaBoost is equivalent to forward stagewise additive modeling using an exponential loss

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

$$\begin{aligned}
& \arg\min_{\beta,\gamma} \sum_i L(y_i, f_{t-1}(x_i) + \beta b(x_i;\gamma)) \\
= \quad & \arg\min_{\beta,\gamma} \sum_i \exp[-y_i f_{t-1}(x_i) - y_i \beta b(x_i;\gamma)] \\
= \quad & \arg\min_{\beta,\gamma} \sum_i w_i^{(t)} \exp(-\beta y_i b(x_i;\gamma)).
\end{aligned}$$

Instead of optimizing over both β and $b(\cdot, \gamma)$, AdaBoost just randomly picks a classifier $b(\cdot; \gamma)$, and then optimize over β.

For any given $b(\cdot; \gamma)$, denote the corresponding weighted empirical error rate by ϵ, then the optimal β is given by

$$\beta = \frac{1}{2} \log \frac{1 - \epsilon}{\epsilon}.$$

For regression, we can use L_2-Boosting.

- Loss function is the squared error,

$$\begin{aligned} & (y_i - f_{t-1}(x_i) - \beta b(x_i;\gamma))^2 \\ = & (r_{it} - \beta b(x_i;\gamma))^2. \end{aligned}$$

- At the t-th iteration,

$$f_t(x) = f_{t-1}(x) + \hat{\beta}_t x^{(t)},$$

  where $x^{(t)}$ denotes the variable (possibly random) chosen at the t-th iteration, and $\hat{\beta}_t$ is the estimated coefficient based on the partial residuals r_{it}.

When doing the optimization at the t-th iteration,

- for exponential loss, the effect of the previous $(t-1)$ functions becomes weights;
- for squared loss, the effect of the previous $(t-1)$ functions becomes partial residuals.

For many other loss functions, we don't have such a simple form for the effect of the previous $(t-1)$ functions. Instead we use Gradient Boosting.

Gradient Boosting

- Goal is to minimize $L(f)$ w.r.t f,

$$L(f) = \sum_{i=1}^{n} L(y_i, f(x_i))$$

  where f is constrained to be in the space spanned by base classifers/regressors $b(x; \gamma)$'s, e.g., trees or linear functions or even SVM's.

- At the $(t + 1)$-th iteration, we have already had an estimate of f based on the previous t iterations f_t.

- For some loss functions, it is not easy to solve

$$\text{argmin}_{\beta,\gamma} \sum_i L(y_i, f_t(x_i) + \beta b(x_i; \gamma)).$$

In Gradient Boosting, at step $(t+1)$:

- View the loss function (sum of the training error at each fitted value $f_t(x_i)$) as a function evaluated at an n-dim vector

  $$\mathbf{f}_t = (f_t(x_1), f_t(x_2), \ldots, f_t(x_n))^t.$$

  Calculate the corresponding gradient (that is also an n-dim vector)

  $$\mathbf{g}_t = \left[\frac{\partial L(y_i, f_i)}{\partial f_i} \right]_{i=1:n, \mathbf{f}=\mathbf{f}_t}.$$

  If we move the n fitted values along the direction "$-\mathbf{g}_t$", then we can reduce the (training) error.

- Choose a base function $b(x; \gamma_{t+1})$ whose predictions at the n data points are as close as possible to $-\mathbf{g}_t$. In order words, the negative gradient at the n samples, $-\mathbf{g}_t$, becomes our working response when picking up $b(x; \gamma)$.

- Then choose step length (how far we'll move along the direction "$-\mathbf{g}_t$"),

$$\rho_{t+1} = \text{argmin}_\rho \sum_{i=1}^{n} L(y_i, f_t(x_i) + \rho b(x_i; \gamma_{t+1})),$$

which is just a one-dimensional optimization problem.

- Finally update

$$f_{t+1}(x) = f_t(x) + \rho_{t+1} b(x; \gamma_{t+1}).$$

When to Stop?

- Boosting is prone to overfitting.
- Early stopping based on CV or test error.
- Regularized step size, i.e., take a small step along the gradient (for Gradient Boosting).
- Some packages, e.g., `gbm` and `XGBoost`, even add a bootstrap component: at the t-th iteration, the optimization is done based on a subset of the training data.