AdaBoost Fits an Additive Model

Leon Gu

CSD, CMU

Generalized Additive Model

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

Many classification and regression models can be written as a linear combination of some simpler models (as above), where

- x is input data;
- $\{\beta_m, \gamma_m\}$ are model parameters;
- $b_m(x; \gamma_m)$ are any arbitrary functions of x .

Typically, $\{\beta_m, \gamma_m\}$ are estimated by minimizing some loss function L, which measures the prediction errors over training data $\{x_n, y_n\}$.

$$\langle \beta_m^*, \gamma_m^* \rangle_1^M = \underset{\{\beta_m, \gamma_m\}_1^M}{\arg\min} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b_m(x; \gamma_m)\right)$$

Forward Stagewise Optimization

Directly optimizing such loss function is often difficult. However, if optimizing over one single base function

$$\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i,\, \beta b_m(x_i;\gamma)\right)$$

can be done efficiently, a simple greedy search method can be used. The basic idea is, sequentially adding new base functions to the expansion function $f(x)$ **without** changing the parameters that have been added.

For example, in the i-th stage, a new function $b_i(x;\gamma_i)$ is added to the expansion $f_{i-1}(x)$; the new coefficients β_i, γ_i are fitted by minimizing the error between $\beta_i b_i(x;\gamma_i)$ and the residue $y - f_{i-1}(x)$; then $f(x)$ is updated by $f_i(x) = f_{i-1}(x) + \beta_i^* b_i(x;\gamma_i^*)$.

Such strategy (looking for the global optimum by solving a sequence of subproblems in a greedy manner) is commonly used in practice.

AdaBoost

AdaBoost fits an additive model by forward stagewise approach, where

- the base function b_m is a binary classifier $G_m(x) : \mathcal{R}^K \to \{-1, 1\}$;
- the objective function is the exponential loss.

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

Many boosting variants are developed based on this observation.

- change base function?
- change loss function?

First we write down the exponential loss:

$$\langle \beta_m, G_m \rangle = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp\left[-y_i \left(f_{m-1}(x_i) + \beta G(x_i)\right)\right]$$

Then define weights $w_i^{(m)} = \exp\left(-y_i f_{m-1}(x_i)\right)$, and divide the data into two subsets: $\{y_i = G(x_i)\}$ and $\{y_i \neq G(x_i)\}$,

$$\begin{aligned}
& \textstyle\sum_{i=1}^{N} \exp\left[-y_i [f_{m-1}(x_i) + \beta G(x_i)]\right] \\
= \; & \textstyle\sum_{i=1}^{N} w_i^{(m)} \exp\left(-y_i \beta G(x_i)\right) \\
= \; & e^{-\beta} \textstyle\sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)} \\
= \; & e^{-\beta} \textstyle\sum_{i=1}^{N} w_i^{(m)} + \left(e^{\beta} - e^{-\beta}\right) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))
\end{aligned}$$

Then we optimize L over β and G iteratively.

When β is fixed, the optimal $G_m(x)$ is given by

$$\hat{G}_m = \arg\min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

in other words, the optimal $G_m(x)$ is the classifier which minimizes the weighted prediction error, where $w_i^{(m)}$ is viewed as a weight assigned to the i-th training data.

Given a fixed G_m, we substitute it into the loss function, take the derivative w.r.t. β_m and set it to zero,

$$d\left(e^{-\beta} \sum_{y_i = G_m(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G_m(x_i)} w_i^{(m)}\right) / d\beta_m = 0$$

that is

$$\hat{\beta}_m = \frac{1}{2} \log \frac{1 - err_m}{err_m}; \qquad err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}$$

Check the updating rule of weights $w_i^{(m)}$,

$$\begin{aligned} w_i^{(m+1)} &= \exp\left(-y_i f_m(x_i)\right) \\ &= \exp -y_i(f_{m-1}(x_i) + \beta_m G_m(x_i)) \\ &= w_i^{(m)} \exp(-\beta_m y_i G_m(x_i)) \\ &= w_i^{(m)} \exp(2\beta_m I(y_i \neq G_m(x_i))) \exp(-\beta_m) \end{aligned}$$

Now we summarize the algorithm

1. initialize weights $w_i = 1/N, i = 1, \ldots, N$.
2. for $m = 1$ to M
   - 2.1 fit a classifier G_m to training data with weights w_i.
   - 2.2 update weights by $w_i \leftarrow w_i \exp(\alpha_m I(y_i \neq G_m(x_i)))$, where $\alpha_m = \log \frac{1-err_m}{err_m}$ and $err_m = \frac{\sum_{i=1}^N w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i^{(m)}}$
3. output the final classifier $f(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right)$

That is exactly **AdaBoost**.

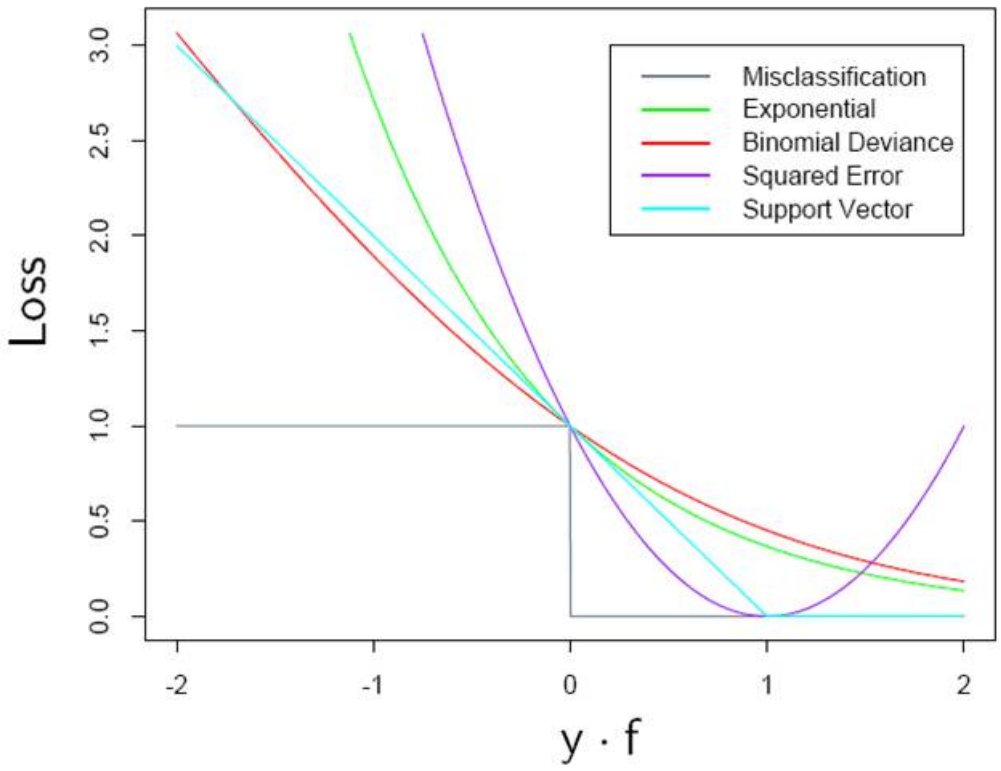
Other Loss Functions

The choice of loss function is directly related to computation complexity and robustness of the algorithm.

- $|y - f(x)|$ (called "**residual error**") is used to represent the goodness of regression.
- $yf(x)$ (called "**margin**") is used to represent the goodness of classification. The loss criterion should penalize large negative margin and encourage positive margin.

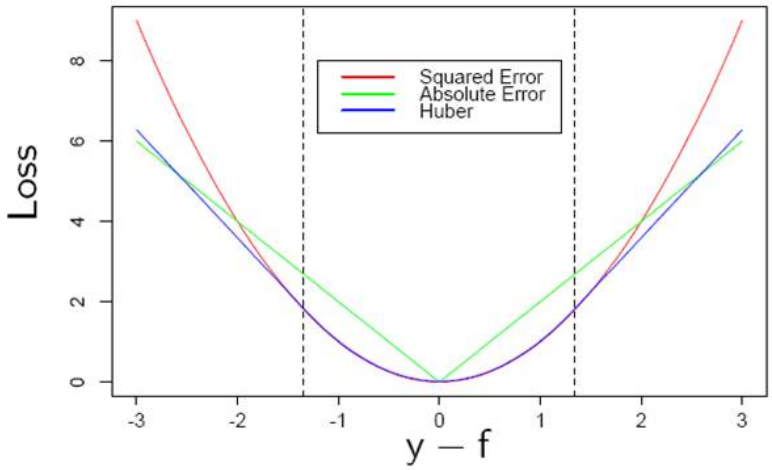
Apparently square error loss $(y - f(x))^2$ is not suitable for classification, because it penalizes correctly classified data as heavily as misclassified ones.

Typical Loss Functions for Classification



- $0/1$ loss: $\mathrm{I}(\mathrm{sign}(\mathrm{f}) \neq \mathrm{y})$
- exponential loss: $\exp(-yf(x))$ (adaboost)
- binomial deviance: $\log(1 + \exp(-2yf))$
- soft-margin loss: $(1 - yf)\mathrm{I}(\mathrm{yf} > 1)$ (SVM)

Typical Loss Functions for Regression



- squared-error loss: $(y - f(x))^2$
- absolute loss: $|y - f(x)|$
- Huber loss:

$$L(y, f) = \begin{cases} (y - f(x))^2 & \texttt{for } |y - f(x)| < \delta \\ \delta(|y - f(x)| - \delta/2) & \texttt{otherwise} \end{cases}$$

  1. quadratically increasing with the residual error while $|y - f(x)| < \delta$;
  2. linear increasing while $|y - f(x)| < \delta$;
  3. differentiable, that is, the tangent of the quadratic loss part at δ equals to the linear loss.