Boosting I

Wei Li

Syracuse University

Spring 2024

OVERVIEW

Boosting

Forward stagewise modeling

Exponential Loss for classification

Robust loss functions for classification

Robust loss functions for regression

Boosting

Boosting

Boosting: like a committee of weak learners, evolving over time, with members cast a weighted vote.

$\tau > 0$ arbitrarily small

- strong learners: for large dataset, the classifier can arbitrarily accurately learn the target function with probability $1 - \tau$
- weak learners: for large dataset, the classifier can barely learn the target function with probability $\frac{1}{2} + \tau$.

Can we construct a strong learner from weak learners and how?

Motivation of boosting: combines the outputs of many weak classifiers to produce a powerful “committee”.

- Similar to bagging and other committee-based approaches
- Originally designed for classification problems, but can also be extended to regression problem.

Practical experience suggests that boosted trees have highly competitive performance as random forests.

Forward Stagewise Linear Regression (L_2-boosting)

The goal of Forward Stagewise Linear Regression is to build upon linear regression in stagewise manner that selects variables:

Define estimators $\hat{m}^{(0)}, \ldots, \hat{m}^{(k)}, \ldots,$ as follows. Let $\hat{m}^{(0)}(x) = 0$ and then iterate the following steps:

1. Compute the residuals $U_i = Y_i - \hat{m}^{(k)}(X_i)$.
2. Regress the residuals on the $X_j'\text{s} : \hat{\beta}_j = \sum_i U_i X_{ij} / \sum_i X_{ij}^2, j = 1, \ldots, p$
3. Find $J = \operatorname{argmin}_j RSS_j$ where $RSS_j = \sum_i \left(U_i - \hat{\beta}_j X_{ij}\right)^2$
4. Set $\hat{m}^{(k+1)}(x) = \hat{m}^{(k)}(x) + \nu \hat{\beta}_J x_J, 0 < \nu \leq 1$

The version above is called L_2 **boosting** or **matching pursuit**.

A variation is to set $\hat{m}^{(k+1)}(x)\hat{m}^{(k)}(x) + \nu \operatorname{sign}\left(\hat{\beta}_J\right) x_J$ which is called forward stagewise regression.

Forward stagewise modeling

Forward stagewise modeling

A basis function expansion of f takes the form

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

- β_m 's are the expansion coefficients
- $b(x; \gamma) \in \mathbb{R}$ simple functions parameterized by γ.

Examples of additive models:

- In single-hidden-layer neural networks with continuous output, $b(x; \gamma) = \sigma\left(\gamma_0 + \gamma_1^\top x\right)$, where $\sigma(t) = 1/\left(1 + e^{-t}\right)$
- Multivariate adaptive regression splines uses truncated power spline basis functions where γ parameterizes the variables and values for the knots.
- For trees, γ parameterizes the split variables and split points at the internal nodes, and the predictions at the terminal nodes.

The model $\hat{f}$ is obtained by minimizing the loss

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i;\gamma_m\right)\right)$$

- Typically the computational cost is very high.
- Yet feasible to solve the sub-problem of fitting just a single basis function (iteratively).

Forward stagewise modeling approximate the solution to the above optimization problem by sequentially adding new basis functions to the expansion:

- At iteration m, one solves for the optimal basis function $b\left(x,\hat{\gamma}_m\right)$ and corresponding coefficient $\hat{\beta}_m$, which then is added to the current expansion $f_{m-1}(x)$.
    - Previously added terms are *not* modified.
- This process is repeated.

Forward Stagewise Additive Modeling

- Set $f_0(x) = 0$
- for $m = 1, \ldots, M$
  - Compute

$$\left(\hat{\beta}_m, \hat{\gamma}_m\right) = \underset{\beta_m, \gamma_m}{\arg\min} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} f_{m-1}(x_i) + \beta_m b(x_i; \gamma_m)\right)$$

  - Set

$$f_m(x) = f_{m-1}(x) + \hat{\beta}_m b(x; \hat{\gamma}_m)$$

The model building stops when the next term does not appreciably improve the fit of the model. Or M can be chosen using cross-validation.

Squared-error loss example

$$L(y, f(x)) = (y - f(x))^2$$

At the m th step, given the current fit $f_{m-1}(x)$, we solve

$$\begin{array}{ll}\min_{\beta,\gamma} & \sum_i L\left(y_i, f_{m-1}\left(x_i\right) + \beta b(x, \gamma)\right) \Longleftrightarrow \\ \min_{\beta,\gamma} & \sum_i \left[y_i - f_{m-1}\left(x_i\right) - \beta b\left(x_i; \gamma\right)\right]^2 = \sum_i \left[r_{im} - \beta b\left(x_i, \gamma\right)\right]^2\end{array}$$

where r_{im} is the residual of he current model on the i-th observation. The term $\hat{\beta}_m b\left(x; \hat{\gamma}_m\right)$ is the best fit to the current residual.

The updated fit

$$f_m(x) = f_{m-1}(x) + \hat{\beta}_m b\left(x; \hat{\gamma}_m\right)$$

Binary classification: exponential loss

Consider the two-class problem $Y \in \{-1, 1\}$:

The general paradigm of boosting:

- Sequentially apply the weak classification algorithm to repeatedly modified versions of the data (re-weighted data)
- produces a sequence of weak classifiers

$$G_m(x), \quad m = 1, 2, \dots, M$$

- The predictions from G_m 's are then combined through a weighted majority vote to produce the final prediction

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \in \{-1, 1\}$$

Here $\alpha_1, \dots, \alpha_M \geq 0$ are computed by the boosting algorithm.

AdaBoost.M1 or discrete AdaBoost

AdaBoost algorithm (Freund and Schapire, 1997):

1. Initially the observation weights $w_i = 1/n, i = 1, \ldots, n$.
2. For m = 1 to M
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i. For example, $G_m = \arg min_G \sum_{i=1}^{n} w_i 1(y_i \neq G(x_i))$
   (b) Compute the *weighted error rate*
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{n} w_i 1\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{n} w_i}$$
   (c) Compute the *importance* of G_m as
   $$\alpha_m = \log\left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}\right)$$
   (d) Update $w_i \leftarrow w_i \exp\Big(\alpha_m 1\big(y_i \neq G_m(x_i)\big)\Big), i = 1, \ldots, n$
3. Output $G(x) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$.

Intuition

At step m, those observations that were misclassified by the classifier $G_{m-1}(x)$ induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly.

- The weights α_m 's weigh the contribution of each G_m.
- The higher (lower) err $_m$, the smaller (larger) α_m.
- The w_i increases for x_i misclassified by G_m. The more important classifier G_m is (thus smaller α_i), the smaller increase of the weight in this w_i.

But why this algorithm works?

Exponential Loss for classification

Exponential Loss for classification

For classification $Y \in \{-1, 1\}$:

Boosting fits an additive expansion in a set of elementary basis functions:

$$G(x) = \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

where the basis functions are the weak classifiers $G_m(x) \in \{-1, 1\}$.

AdaBoost is equivalent to forward stagewise additive modeling using the exponential loss function

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

using individual classifiers $G_m(x) \in \{-1, 1\}$ as basis functions

One can prove that AdaBoost.M1 is a method that approximates minimizing

$$\arg\min_{\beta_1, G_1, \ldots \beta_M, G_M} \sum_{i=1}^{n} \exp\left(-y_i \sum_{m=1}^{M} \beta_m G_m\left(x_i\right)\right)$$

via a forward-stagewise additive modeling approach.

$$\begin{aligned}
(\beta_m, G_m) &= \arg\min_{\beta, G} \textstyle\sum_{i=1}^{n} \exp\left(-y_i\left(f_{m-1}\left(x_i\right) + \beta G\left(x_i\right)\right)\right) \\
&= \arg\min_{\beta, G} \textstyle\sum_{i=1}^{n} w_i^{(m)} \exp\left(-y_i \beta G(x_i)\right)
\end{aligned}$$

where $w_i^{(m)} = \exp\left(-y_i f_{m-1}\left(x_i\right)\right)$.

Why exponential loss?

Let $Y \in \{-1, 1\}$. The (population) minimizer of the exponential loss function is

$$\begin{aligned} f^*(x) &= \arg\min_f \mathrm{E}_{Y|x}\left(e^{-Yf(x)}\right) \\ &= \frac{1}{2}\log\frac{\Pr(Y=1 \mid x)}{\Pr(Y=-1 \mid x)} \end{aligned}$$

which is equal to one half of the log-odds.

So AdaBoost can be regarded as an additive logistic regression model.

binomial deviance

Define $Y' = (Y+1)/2 \in \{0,1\}$, let

$$p(x) = \Pr(Y' = 1 \mid x) = \frac{e^{f(x)}}{e^{-f(x)} + e^{f(x)}}$$

Then the binomial log-likelihood loss function (for Y') is

$$l(Y', p(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x))$$

Equivalently the deviance (-log likelihood) is

$$-l(Y', f(x)) = \log\left(1 + e^{-2Yf(x)}\right)$$

So $\arg\min_f \mathrm{E}_{Y'|x}(-l(Y', f(x))) = \arg\min_f \mathrm{E}_{Y|x}\left(e^{-Yf(x)}\right)$ has the same solution

$$\frac{1}{2}\log\left(\frac{P(Y=1|x)}{P(Y \neq 1|x)}\right).$$

K-Class classification with exponential loss

For a K -class classification problem, class label $Y \in \{1, \dots, K\}, K \geq 3$.

Consider the coding $Y' = (Y'_1, \dots, Y'_K)^\top$ with

$$Y'_k = \begin{cases} 1, & \text{if } Y = k \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases}$$

Let $f = (f_1, \dots, f_K)^\top$ with $\sum_{k=1}^K f_k = 0$, and define

$$L(Y, f) = \exp\left(-\frac{1}{K} {Y'}^\top f\right)$$

Then the minimizers f^* satisfies

$$\Pr(Y = k \mid x) = \frac{e^{\frac{f_k^*(x)}{K-1}}}{\sum_{l=1}^K e^{\frac{f_l^*(x)}{K-1}}}$$

Classical AdaBoost: $f_M(x) = \sum_{m=1}^{M} \beta_m f_m(x)$, where $f_m(x) \in \{-1, 1\}$.

K-Class AdaBoost: $f_M(x) = \sum_{m=1}^{M} \beta_m f_m(x)$ - $f_m(x)$ take values in the support of Y', i.e., - the collection of K-dim vectors, where only one entry being 1, the remaining $K-1$ entries being $-1/(K-1)$

Adaboost algorithm is used to solve the optimization problem

$$
\begin{aligned}
(\beta_m, f_m) &\in \arg\min_{\beta, f} \sum_{i}^{n} L\left(y_i', f_{m-1}(x_i) + \beta f(x_i)\right) \\
&= \arg\min_{\beta, f} \sum_{i=1}^{n} \exp\left(-\frac{1}{K} y_i'^{\top} (f_{m-1}(x_i) + \beta f(x_i))\right)
\end{aligned}
$$

Robust loss functions for classification

Robust loss functions for classification

Suppose $Y \in \{-1, 1\}$. The Margin of $f(x)$ is defined as $yf(x)$.

- The negative margin plays a similar role to the residuals $y - f(x)$ in regression.

Consider the classification rule is $G(x) = \text{sign}(f(x))$. Then for i:

- positive margin $y_i f(x_i) > 0$: correctly classified
- negative margin $y_i f(x_i) < 0$: incorrectly classified
- The decision boundary is $f(x) = 0$

The goal of a classification algorithm is to produce positive margins as frequently as possible.

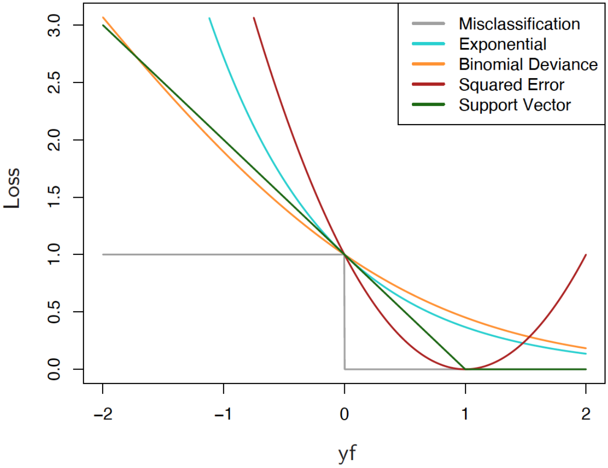
Any loss function should penalize negative margins *more heavily* than positive margins.

various loss functions: Monotone increasing loss functions of the margin $-yf$:

- Misclassification loss: $1(\text{sign}(f(x)) \neq y) = 1(-fy > 0)$
- Exponential loss: $\exp(-yf)$ (not robust against mislabeled samples)
- Binomial deviance: $\log\{1 + \exp(-yf)\}/\log(2)$ (more robust against influential points)
- SVM loss: $(1 - yf)_+$

Other loss functions (not monotonic):

- Squared error: $(y - f)^2 = (1 - yf)^2$ (penalize large positive margins heavily)

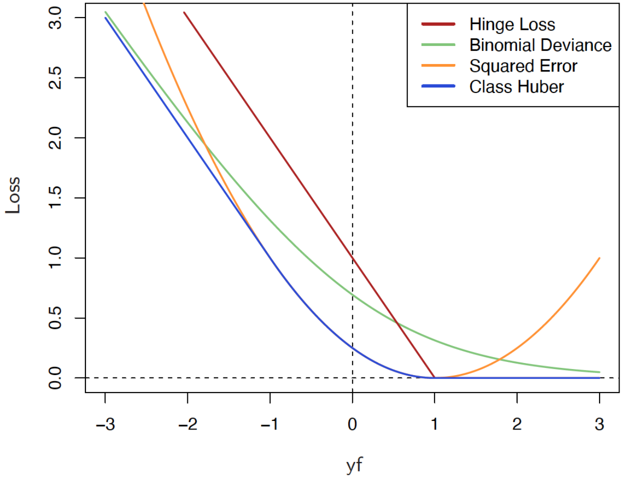


ESL Fig. 10.4. Loss functions for two-class classification. Each function has been scaled so that it passes through the point (0,1).

Loss Function	$L[y, f(x)]$	Minimizing Function
Binomial Deviance	$\log[1 + e^{-yf(x)}]$	$f(x) = \log \frac{\Pr(Y=+1\|x)}{\Pr(Y=-1\|x)}$
SVM Hinge Loss	$[1 - yf(x)]_+$	$f(x) = \text{sign}[\Pr(Y=+1\|x) - \frac{1}{2}]$
Squared Error	$[y - f(x)]^2 = [1 - yf(x)]^2$	$f(x) = 2\Pr(Y=+1\|x) - 1$
"Huberised" Square Hinge Loss	$-4yf(x)$, $\quad yf(x) < -1$ $[1 - yf(x)]_+^2$ $\quad$ otherwise	$f(x) = 2\Pr(Y=+1\|x) - 1$

ESL Table 12.1. Loss functions for two-class classification.

The "Huberized" square hinge loss, which enjoys the favorable properties of the binomial deviance, squared-error loss and the SVM hinge loss.



ESL Table 12.4. Loss functions for two-class classification. All are scaled to have the limiting left-tail slope of -1.

K-Class classification (multinomial deviance)

- Class label $Y \in \{1, \dots, K\}, K \geq 3$.
- The classifier $h : \mathbb{R}^p \longrightarrow \{1, \dots, K\}$.

Using the 0-1 loss, the Bayes rule h^* is

$$h^*(x) = \arg \max_{k=1,\dots,K} P(Y = k \mid x)$$

i.e., assign x to the most probable class using $P(Y \mid x)$.

One can model $p_k(x) := P(Y = k|x)$ by logistic model:

$$p_k(x) := P(Y = k|x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(z)}}$$

For identification, set $\sum_{k=1}^{K} f_k(x) = 0$, thus

$$f_k(x) = \log p_k(x) - \frac{1}{K} \sum_{l=1}^{K} \log p_l(x).$$

The K-class multinomial deviance is $-\sum_{k=1}^{K} 1(y=k)\log p_k$ (i.e., -2 negative log-likelihood),

$$\begin{aligned}
L(y, p(x)) &= -\sum_{k=1}^{K} 1\,(y=k)\log p_k(x) \\
&= -\sum_{k=1}^{K} 1\,(y=k)\, f_k(x) + \log\left(\sum_{\ell=1}^{K} e^{f_\ell(x)}\right)
\end{aligned}$$

Note: as for other robust loss functions, no direct solution can be derived for the multinomial deviance loss, but some numerical algorithm caleld *gradient boosting* can be applied.

Robust loss functions for regression

Robust loss functions for regression

- Squared error loss

$$L(y, f(x)) = (y - f(x))^2$$

Population optimum for this loss function: $f(x) = E(Y \mid x)$

- Absolute loss

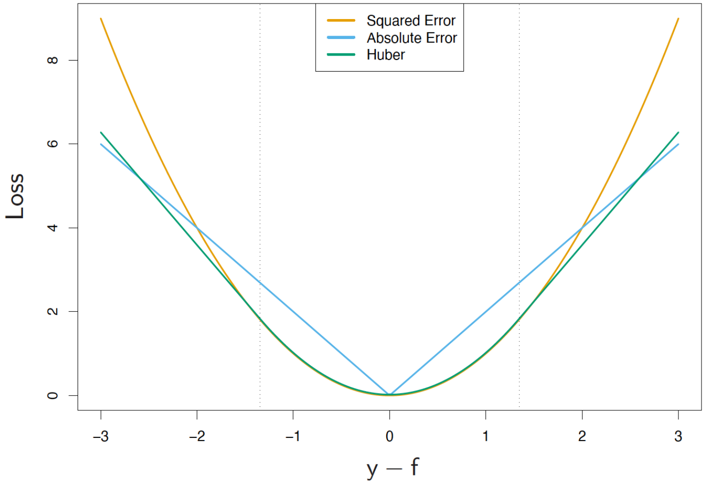
$$L(y, f(x)) = |y - f(x)|$$

Population optimum for this loss function: $f(x) = \text{median}(Y \mid x)$

- Huber loss

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

Such loss provides strong resistance to gross outliers while being nearly as efficient as least squares for Gaussian errors. It combines the good properties of squared-error loss near zero and absolute error loss when $|y - f|$ is large.



ESL Fig 10.5.