STAT 542: Statistical Learning

Boosting

Ruoqing Zhu, Ph.D. <rqzhu@illinois.edu>

Course Website: https://teazrq.github.io/stat542/

April 10, 2022

Department of Statistics
University of Illinois at Urbana-Champaign

Boosting

- AdaBoost
- Training error bound
- Gradient boosting

AdaBoost

Boosting

- Consider producing a sequence of learners:

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

- How to train each $f_t(x)$? At the t-th iteration, given perviously estimated $f_1, \ldots, f_{t-1}$, we estimate a new function $h(x)$ to minimize the loss:

$$\min_h \sum_{i=1}^{n} L\Big(y_i, \sum_{k=1}^{t-1} f_k(x_i) + h(x_i)\Big)$$

- Instead of using the entire $h(x)$, we only use a small “fraction” of it, and add $\alpha_t h(x)$ to the current model. Then proceed to the next iteration.

Boosting

- Boosting is an additive model, but its different from **generalized additive model**, in which each weak learner only involves one variable, and we fit p of such functions. In boosting, each $f_t(x)$ can be very flexible, and we may fit a large number of functions.
- Boosting is also different from **random forests**, another additive model. In random forests, each tree is generated independently, so they can't borrow information from each other.
- AdaBoost (Freund and Schapire, 1997) is a special case of this framework with Exponential loss for classification.
- For this setting, we use labels $y_i \in \{-1, 1\}$.

AdaBoost: algorithm

1. Initiate subject weights $w_i^{(1)} = 1/n$, $i = 1, 2, \ldots, n$.
2. For $t = 1$ to T, repeat (a) – (d)
   (a) Fit a classifier $f_t(x) \in \{-1, 1\}$ to the training data, with individual weights $w_i^{(t)}$.
   (b) Compute the training error at t

   $$\epsilon_t = \sum_i w_i^{(t)} \mathbf{1}\{y_i \neq f_t(x_i)\}$$

   (c) Compute

   $$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

   ...

AdaBoost: algorithm

2. continued
   (d) Update weights

$$w_i^{(t+1)} = \frac{w_i^{(t)}}{Z_t} \exp[-\alpha_t y_i f_t(x_i)],$$

   where Z_t is a normalization factor to keep $w_i^{(t+1)}$ a distribution:

$$Z_t = \sum_{i=1}^{n} w_i^{(t)} \exp[-\alpha_t y_i f_t(x_i)],$$

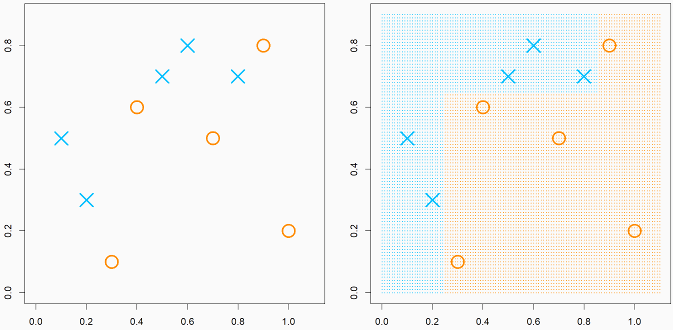
3. Output the final model

$$F_T(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

   And the classification rule: sign$(F_T(x))$

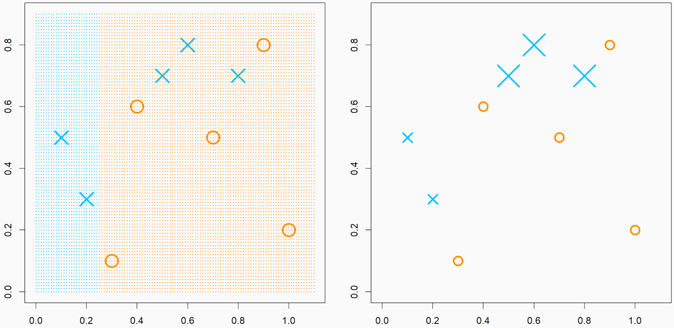
Example

- Let's look at an example with the following data
- At each iteration, we build a tree model $f_t(x)$ with just one split
- The final model is stacked with all tree models



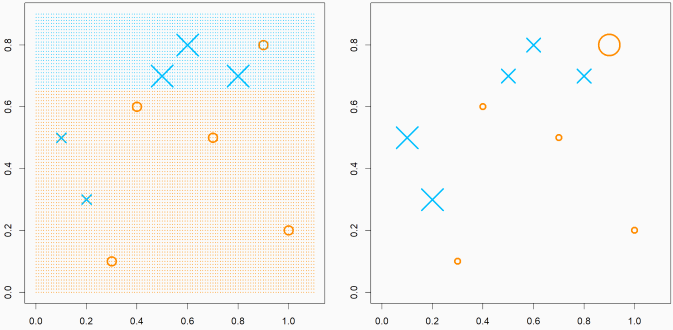
Example

- At the first iteration, the tree splits at 0.25 for X_1
- This makes the three positive cases on the right hand side to increase their weights



Example

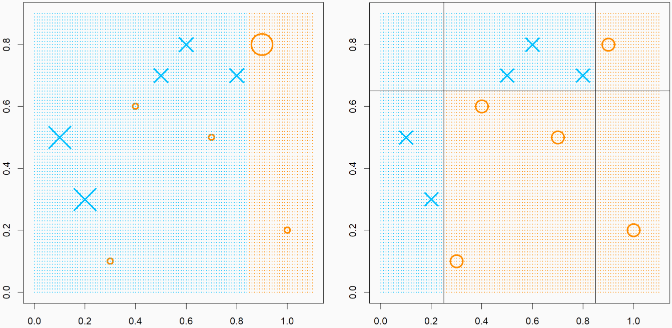
- At the second iteration, the tree splits at 0.65 for X_2
- This further adjusts the weights, along with calculating α_t at each step.



Example

- At the second iteration, the tree splits at 0.85 for X_1
- This produces the final model:

$$F_3(x) = 0.4236 \cdot f_1(x) + 0.6496 \cdot f_2(x) + 0.9229 \cdot f_3(x)$$



AdaBoost: intuition

- At the initial step, we treat all subject with equal weight
- Learn a classifier $f_t(x)$ and inspect which subjects got mis-classified.
- Put more weights on the mis-classified subjects for the next iteration
- Add $\alpha_t f_t(x)$ to the existing model and train the next iteration using the updated weights

AdaBoost: intuition

- Why α_t is choosing this way $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$?
- Why the weak classifier is chosen to minimize the weighted error?
- What can we say about the performance of the final model $F_T(x)$

Training Error Bound

The Subject Weights

- Let's start with analyzing the weight after the final iteration:

$$w_i^{(T+1)} = \frac{1}{Z_T} w_i^{(T)} \exp[-\alpha_t y_i f_T(x_i)]$$

- Note that for $w_i^{(T)}$, we can also further back-track it into $T-1$.

$$w_i^{(T)} = \frac{1}{Z_{T-1}} w_i^{(T-1)} \exp[-\alpha_t y_i f_{T-1}(x_i)]$$

- Hence, we can track this all the way back to the first iteration

The Subject Weights

- This gives

$$w_i^{(T+1)} = \frac{1}{Z_1 \cdots Z_T} w_i^{(1)} \prod_{t=1}^{T} \exp[-\alpha_t y_i f_t(x_i)]$$

$$= \frac{1}{Z_1 \cdots Z_T} \frac{1}{n} \prod_{t=1}^{T} \exp[-\alpha_t y_i f_t(x_i)]$$

$$= \frac{1}{Z_1 \cdots Z_T} \frac{1}{n} \exp\big[-y_i \sum_{t=1}^{T} \alpha_t f_t(x_i)\big]$$

- Note that $\sum_{t=1}^{T} \alpha_t f_t(x_i)$ is the just the final model at the T-th iteration, i.e., $F_T(x_i)$.

The Subject Weights

- Noticing that the weights sum up to 1, we have

$$1 = \sum_{i=1}^{n} w_i^{(T+1)} = \frac{1}{Z_1 \cdots Z_T} \frac{1}{n} \sum_{i=1}^{n} \exp\big\{ - y_i F_T(x_i) \big\}$$

- or

$$Z_1 \cdots Z_T = \frac{1}{n} \sum_{i=1}^{n} \exp\big\{ - y_i F_T(x_i) \big\}$$

- On the right-hand side, it is the exponential loss.

The Exponential Loss

- Let's check some facts:
  - Correctly classified: sign$(y) =$ sign$(f(x))$, and $\exp[-yf(x)] > 0$
  - Incorrectly classified: sign$(y) = -$sign$(f(x))$ the $\exp[-yf(x)] > 1$
- Hence, the exponential loss is larger than 0/1 loss:

$$
\begin{aligned}
& Z_1 \cdots Z_T \\
&= \frac{1}{n} \sum_{i=1}^{n} \exp\big\{ - y_i F_T(x_i) \big\} \\
&> \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{y_i \neq F_T(x_i)\}
\end{aligned}
$$

The Z_t's

- On the other hand, we can further break down each Z_t
- Notice that $f_t(x_i)$ is a classification model with output 1 or -1, this either matches or not matches y_i:

$$\begin{aligned} Z_t &= \sum_{i=1}^{n} w_i^{(t)} \exp[-\alpha_t y_i f_t(x_i)] \\ &= \sum_{y_i = f_t(x_i)} w_i^{(t)} \exp[-\alpha_t] + \sum_{y_i \neq f_t(x_i)} w_i^{(t)} \exp[\alpha_t] \\ &= \exp[-\alpha_t] \sum_{y_i = f_t(x_i)} w_i^{(t)} + \exp[\alpha_t] \sum_{y_i \neq f_t(x_i)} w_i^{(t)} \end{aligned}$$

The Z_t's

- By our definition,

$$\epsilon_t = \sum_i w_i^{(t)} \mathbf{1}\{y_i \neq f_t(x_i)\}$$

  is the proportion of weights for mis-classified samples.

- Hence,

$$Z_t = (1 - \epsilon_t) \exp[-\alpha_t] + \epsilon_t \exp[\alpha_t]$$

- Since we want to minimize $Z_1 \cdots Z_t$, we can simply minimize Z_t by choosing α_t

The Z_t's

- Take a derivative with respect to α_t, we have

$$-(1-\epsilon_t)\exp[-\alpha_t] + \epsilon_t \exp[\alpha_t] = 0$$

- This gives

$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$$

- And plug this back into Z_t

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

- Since $\epsilon_t(1-\epsilon_t)$ can only attain maximum of $1/4$, Z_t must be smaller than 1. And $Z_1 \cdots Z_t$ should converge to 0.

The Training Error

- Alternatively, if we let $\gamma_t = \frac{1}{2} - \epsilon_t$ as the improvement from a random model

$$\begin{aligned} Z_t =& 2\sqrt{\epsilon_t(1-\epsilon_t)} \\ =& \sqrt{1-4\gamma_t^2} \\ \leq & \exp\Big[-2\gamma_t^2\Big] \end{aligned}$$

- The last equation uses the Taylor expansion that

$$\exp\Big[-4\gamma_t^2\Big] = 1 - 4\gamma_t^2 + \cdots$$

The Training Error

- Hence, the AdaBoost training error is bounded above by

$$
\begin{aligned}
\text{Training Error} =& \sum_{i=1}^{n} \mathbf{1}\{y_i \neq \text{sign}(F_T(x_i))\} \\
=& \sum_{i=1}^{n} \exp\big[-y_i \neq F_T(x_i)\big] \\
=& Z_1 \cdots Z_T \\
\leq& \exp\big[-2\sum_{t=1}^{T} \gamma_t^2\big] \\
\to& \ 0
\end{aligned}
$$

as long as $f_t(x)$ at each iteration t is better than random guess.

Remarks

- The Adaboost outputs a classifier $F_T(x)$ with small testing error? No. We need to tune T. Careful! — You can easily overfit.
- The training error of $F_T(x)$ decreases w.r.t. T? No. Its only the upper bound of $0/1$ training error
  - After each iteration, Adaboost decreases a particular upper-bound of the 0/1 training error. So in a long run, the training error is going to zero, but not necessarily monotonically.
- We can use a classifier that is worse than random guessing? Yes. The reverse of that classier can be used ($\alpha_t < 0$)
- In practice, a classification tree model is used as the weak learner.

Remarks

- We may also roughly calculate the estimated probability
- Consider the (upper bound) exponential loss $\mathsf{E}(\exp\{-yF(x)\})$, which is

$$e^{-F(x)}P(Y=1|x)+e^{F(x)}P(Y=-1|x)$$

- The best $F(x)$ that minimize this expectation should be

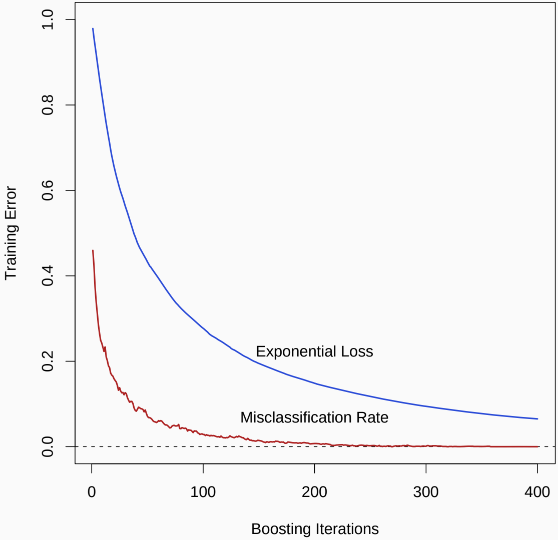
$$-e^{-F(x)}P(Y=1|x)+e^{F(x)}P(Y=-1|x)=0$$

- This leads to

$$F(x)=\frac{1}{2}\log\frac{\mathsf{P}(y=1|x)}{\mathsf{P}(y=-1|x)}$$

$$\mathsf{P}(y=1|x)=\frac{e^{2F(x)}}{1+e^{2F(x)}}$$

AdaBoost

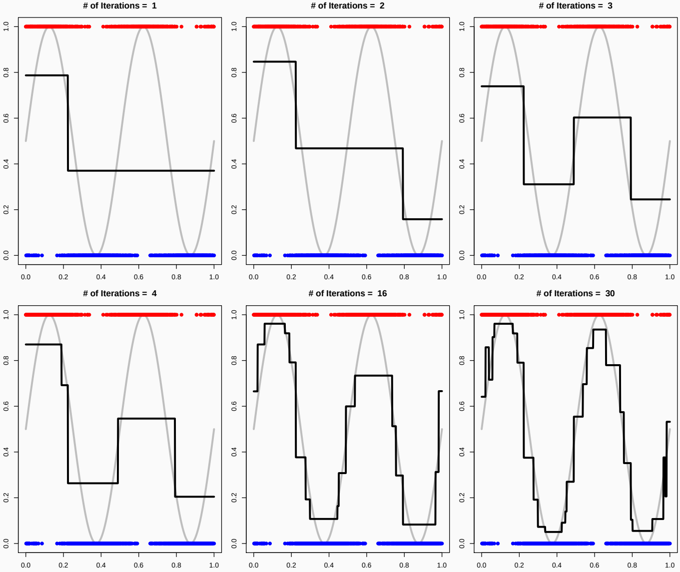


Training Error
0.0
0.2
0.4
0.6
0.8
1.0
Exponential Loss
Misclassification Rate
0
100
200
300
400
Boosting Iterations

R implementation

- Use R package gbm : function gbm
- Tuning parameters:
  - Specify distribution = “adaboost”
  - n.trees controls the number of iterations T
  - shrinkage : further set a shrinkage factor on each $f_t(x)$. The default is 0.01. The original AdaBoost uses 1, however, can be less stable. A small value of this will require a large number of trees.
  - bag.fraction : each $f_t(x)$ uses a bootstrapped sample. If set to < 1, two different runs will produce slightly different models
  - cv.folds : number of cross validations
- Other parameters to consider: interaction.depth $= 1$ means stumps (additive model), > 1 allows interations

An Example



# of Iterations = 1
# of Iterations = 2
# of Iterations = 3
# of Iterations = 4
# of Iterations = 16
# of Iterations = 30
0.0
0.2
0.4
0.6
0.8
1.0

Gradient Boosting

Forward Stage-wise Additive Model

- In more general framework, consider additive structure:

$$F_T(x) = \sum_{t=1}^{T} \alpha_t f(x; \boldsymbol{\theta}_t)$$

- Fit model by minimizing the loss function

$$\min_{\{\alpha_t, \boldsymbol{\theta}_t\}_{t=1}^{T}} \sum_{i=1}^{n} L\big(y_i, F_T(x_i)\big)$$

- We may choose
  - Loss function L, suitable for the problem
  - Base learner $f(x; \boldsymbol{\theta})$ with parameter $\boldsymbol{\theta}$, such as linear, tree, etc.

Forward Stage-wise Additive Model

- It is difficult to minimize over all $\{\alpha_t, \boldsymbol{\theta}_t\}_{t=1}^T$.
- Instead, we do this in a stage-wise fashion. (recall the connection between Lasso and stage-wise regression)
- The algorithm:
  (1) Set $F_0(x) = 0$
  (2) For $t = 1, \ldots, T$
    - Choose $(\alpha_t, \boldsymbol{\theta}_t)$ to minimize the loss

$$\min_{\alpha, \boldsymbol{\theta}} \sum_{i=1}^n L\big(y_i, F_{t-1}(x_i) + \alpha f(x_i; \boldsymbol{\theta})\big)$$

    - Update $F_t(x) = F_{t-1}(x) + \alpha_t f(x; \boldsymbol{\theta}_t)$

Forward Stage-wise Additive Model

- AdaBoost is forward stage-wise using exponential loss.
- It doesn't pick an optimal $f(x;\boldsymbol{\theta})$ at each step: the tree model is not optimized, we just need some model that is better than random.
- Only the step size α_t is optimized at each t given the fitted $f(x;\boldsymbol{\theta}_t)$

Forward Stage-wise Additive Model

- Another example is the forward stage-wise linear regression
- For each step we use a single variable linear model:

$$f(x, j) = \text{sign}\big(\text{Cor}(X_j, \mathbf{r})\big) X_j$$

  - $\mathbf{r}$ is the residual, as $r_i = y_i - F_{t-1}(x_i)$
  - j is the index that has the largest absolute correlation with $\mathbf{r}$
- Then we give a very small step size α_t, say, $\alpha_t = 10^{-5}$, and with sign equal to the correlation between X_j
- $F_t(x)$ is almost equivalent to the Lasso solution path (as t changes)

An Alternative View

- r_i is in fact the gradient to the squared-error loss:

$$r_{it} = -\left[\frac{\partial \left(y_i - F(x_i)\right)^2}{\partial F(x_i)}\right]_{F(x_i)=F_{t-1}(x_i)}$$

- We then fit the weak leaner $f_t(x)$ to the residuals
- Update the fitted model F_t

An Alternative View

- This can be generalized into any loss function L
- At each iteration t, calculate "pseudo-residuals", i.e., the negative gradient for each observation

$$g_{it} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{t-1}(x_i)}$$

- Fit $f_t(x, \boldsymbol{\theta}_t)$ to pseudo-residual g_{it}'s
- Search for a step length

$$\alpha_t = \arg\min_{\alpha} \sum_{i=1}^{n} L\big(y_i, F_{t-1}(x_i) + \alpha f(x_i; \boldsymbol{\theta}_t)\big)$$

- Update $F_t(x) = F_{t-1}(x) + \alpha_t f(x; \boldsymbol{\theta}_t)$

Gradient Boosting

- Hence, the only change when modeling different outcomes is to choose the loss function, and derive the pseudo-residuals

Setting	Loss	Negative Gradient
Regression	$\frac{1}{2}(y - f(x))^2$	$y_i - f(x_i)$
Regression	$\lvert y - f(x) \rvert$	$\text{sign}(y_i - f(x_i))$
Classification	Deviance	$y_i - p(x_i)$

- For gradient boosting using CART as base classifier, we can make it more sophisticated by optimizing α_t at each terminal node

R Implementation

- Boosting is prone to over-fitting
- Fit a large number of iterations n.trees, then select T using CV or test set.
- It is better to take small steps: shrinkage = 0.01 as default
- Use gbm package by specifying the distribution:
    - “gaussian”, “bernoulli”, “laplace”, “huberized”, “multinomial”, etc.