Boosting

Egor Malkov

University of Minnesota
FRB Minneapolis

Machine Learning and Big Data Workshop
April 13, 2020

Prologue

AdaBoost with trees is the best off-the-shelf classifier in the world

(Leo Breiman, 1998)

Informally, [the hypothesis boosting] problem asks whether an efficient learning algorithm [...] that outputs a hypothesis whose performance is only slightly better than random guessing [a weak learner] implies the existence of an efficient algorithm that outputs a hypothesis of arbitrary accuracy [a strong learner]

(Michael Kearns, 1988)

General Idea

Boosting is used for fitting an adaptive basis-function model (ABM):

$$f(\boldsymbol{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\boldsymbol{x})$$

$\phi_m(\boldsymbol{x})$ — basis function learned from the data.

Boosting: $\phi_m(\boldsymbol{x})$ are generated by an algorithm called a weak learner or a base learner.

Idea of Boosting: Apply the weak learner sequentially to weighted versions of the data, where more weight is given to examples that were misclassified by earlier rounds.

- Classification and Regression Tree (CART) model is of common use.

Boosting is often considered as one of the best (or ~~one of~~ the best) classifiers (Caruana and Niculescu-Mizil, 2006).

Boosting was originally derived in the computational learning theory literature where the focus is on binary classification (*unsupervised learning*).

- Later extended for *supervised learning*: regression, robust regression, etc.

Outline

1. AdaBoost.
2. Forward Stagewise Additive Modeling.
3. L2boosting.
4. LogitBoost.
5. Gradient Boosting.
6. Sparse Boosting.
7. Boosting Trees.

Optimization Problem

$$\min_f \sum_{i=1}^{M} L\left(y_i, f(\boldsymbol{x}_i)\right)$$

$$\text{s.t.} \quad f(\boldsymbol{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\boldsymbol{x}) \qquad \text{(ABM)}$$

Name	Loss function	Derivative	Pop. minimizer, f^*	Algorithm
Squared error	$0.5\left(y_i - f(\boldsymbol{x}_i)\right)^2$	$y_i - f(\boldsymbol{x}_i)$	$\mathbb{E}\left(y \mid \boldsymbol{x}_i\right)$	L2Boosting
Absolute error	$\lvert y_i - f(\boldsymbol{x}_i)\rvert$	$\operatorname{sign}\left(y_i - f(\boldsymbol{x}_i)\right)$	$\operatorname{median}\left(y \mid \boldsymbol{x}_i\right)$	Gradient boosting
Exponential loss	$e^{-\tilde{y}_i f(\boldsymbol{x}_i)}$	$-\tilde{y}_i e^{-\tilde{y}_i f(\boldsymbol{x}_i)}$	$0.5 \log \frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log\left(1 + e^{-\tilde{y}_i f(\boldsymbol{x}_i)}\right)$	$y_i - \pi_i$	$0.5 \log \frac{\pi_i}{1-\pi_i}$	LogitBoost

Note: For binary classification problems, assume $\tilde{y}_i \in \{-1, 1\}$, $y_i \in \{0, 1\}$, and $\pi_i = \operatorname{sigm}(2f(\boldsymbol{x}_i))$. For regression problems, assume $y_i \in \mathbb{R}$. ▸ Huber Loss ▸ Gradient Boosting

Optimal Estimate

Supervised Learning ▸ Figure

With the squared error loss function,

$$f^*(\boldsymbol{x}) = \text{argmin}_{f(\boldsymbol{x})} = \mathbb{E}_{y|\boldsymbol{x}}\left[(Y - f(\boldsymbol{x}))^2\right] = \mathbb{E}\,(Y|\boldsymbol{x})$$

Don't know true distribution $p(y|\boldsymbol{x}) \Rightarrow$ need to approximate $\mathbb{E}\,(Y|\boldsymbol{x}) \Rightarrow$ use boosting.

Unsupervised Learning ▸ Figure

Alternatives for binary classification:

- **0-1 Loss**: non-differentiable $\Rightarrow$ don't use.
- **Logloss** (convex upper bound on 0-1 loss): $f^*(\boldsymbol{x}) = 0.5 \log \frac{p(\tilde{y}=1|\boldsymbol{x})}{p(\tilde{y}=-1|\boldsymbol{x})}$
- **Exponential Loss** (convex upper bound on 0-1 loss): $f^*(\boldsymbol{x}) = 0.5 \log \frac{p(\tilde{y}=1|\boldsymbol{x})}{p(\tilde{y}=-1|\boldsymbol{x})}$

AdaBoost

AdaBoost = Adaptive Boosting (*2003 Gödel Prize — Yoav Freund and Robert Schapire*).

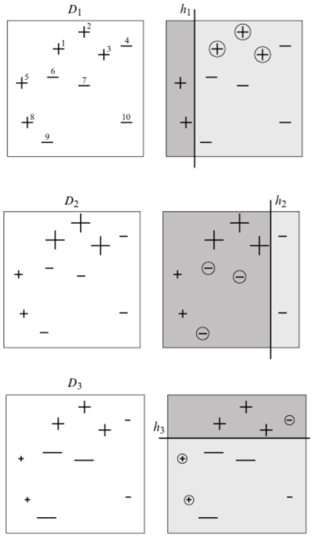
Consider a binary classification problem with exponential loss, output $\tilde{y}_i \in \{-1, 1\}$.

1. Given output $\tilde{y}$ and features $\boldsymbol{x}$, a weak classifier $\phi_m(\boldsymbol{x})$ produces a prediction.
2. Reweight the observations, giving more weight to the misclassified observations, i.e. $\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i)$.
3. Fit a new classifier on the reweighted data.
4. Repeat M times.
5. Final classification: a weighted sum of all classifiers with more accurate $\phi_m(\boldsymbol{x})$ getting greater weights.

Schematically: ▸ Figure

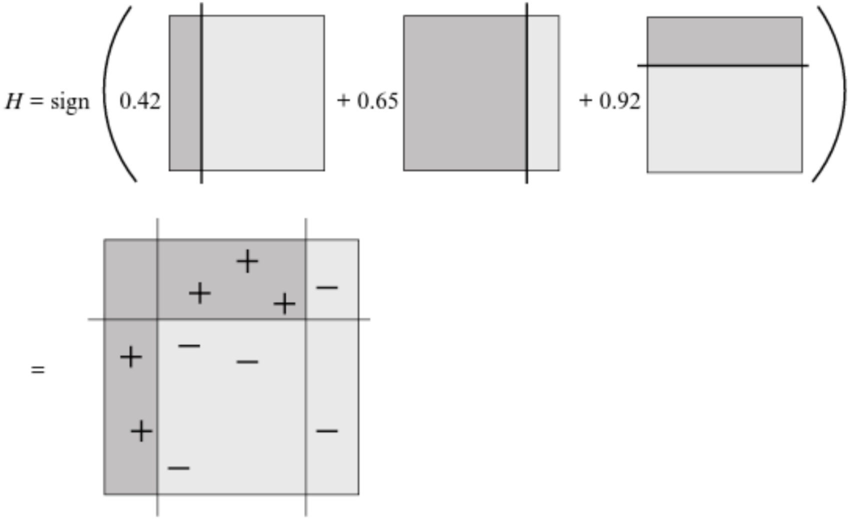
AdaBoost is very resistant to overfitting. ▸ Example

AdaBoost



Source: Cohen.

AdaBoost



Source: Cohen.

AdaBoost (Formal Algorithm)

1. Initialize the observation weights $w_i = 1/N$.

2. For each $m = 1, ..., M$:

   2.1 Using the weak learner, fit a classifier $\phi_m = \text{argmin}_\phi \sum_{i=1}^{N} w_{i,m} \mathbb{I}\{\tilde{y}_i \neq \phi(\boldsymbol{x}_i)\}$

   2.2 Compute the error of the classifier: $\text{err}_m = \frac{\sum_{i=1}^{N} w_{i,m} \mathbb{I}\{\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i)\}}{\sum_{i=1}^{N} w_{i,m}}$

   2.3 Compute $\beta_m = \log \frac{1-\text{err}_m}{\text{err}_m}$ (lower $\text{err}_m \Rightarrow$ higher β_m)

   2.4 Update $w_{i,m+1} = w_{i,m} e^{\beta_m \mathbb{I}\{\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i)\}}$

3. Final learned function: $f(\boldsymbol{x}) = \text{sign}\left[\sum_{m=1}^{M} \beta_m \phi_m(\boldsymbol{x})\right]$ ▸ Example

AdaBoost implements Forward Stagewise Additive Modeling (FSAM) using exponential loss.

Forward Stagewise Additive Modeling

Solve the optimization problem sequentially:

1. Initialize $f_0(\boldsymbol{x}) = \text{argmin}_{\boldsymbol{\gamma}} \sum_{i=1}^{N} L\left(y_i, f^*(\boldsymbol{x}_i; \boldsymbol{\gamma})\right)$
   - **Example:** Use $f_0(\boldsymbol{x}) = \bar{y}$ for squared error loss.
2. At iteration m, compute $(\beta_m, \boldsymbol{\gamma}_m) = \text{argmin}_{\beta, \boldsymbol{\gamma}} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta\phi(\boldsymbol{x}_i; \boldsymbol{\gamma})\right)$
   - With exponential loss, use AdaBoost. Other examples to come!
3. Next, set $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m \phi(\boldsymbol{x}; \boldsymbol{\gamma}_m)$
   - Don't update earlier parameters $\Rightarrow$ **Forward Stagewise Additive Modeling**
4. Continue for M iterations.
   - How to choose M? Early stopping (use the validation set), AIC/BIC criteria.

Shrinkage: Better (test set) performance can be obtained by using

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu\beta_m \phi(\boldsymbol{x}; \boldsymbol{\gamma}_m), \quad \nu \in (0, 1]$$

In practice, Hastie, Tibshirani, and Friedman suggest to use $\nu < 0.1$ Figure

L2boosting

L2boosting — use squared error loss.

With the squared error loss function, at iteration m,

$$L\left(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta\phi(\boldsymbol{x}_i;\boldsymbol{\gamma})\right) = \left(r_{im} - \beta\phi(\boldsymbol{x}_i;\boldsymbol{\gamma})\right)^2$$

- Current residual: $r_{im} \triangleq y_i - f_{m-1}(\boldsymbol{x}_i)$

Use some weak learner to predict $r_{im} \Rightarrow$ find the new basis function, $\phi(\boldsymbol{x}_i;\boldsymbol{\gamma})$.

The basis function that best fits the residuals from the *previous iteration* minimizes the loss and is added to the model.

LogitBoost

Problems with exponential loss:

1. Misclassified examples are overweighted $\Rightarrow$ sensitive to outliers. ▸ Figure
2. $e^{-\tilde{y}f}$ is not the logarithm of any probability mass function for $\tilde{y} \in \{-1, 1\} \Rightarrow$ cannot recover probability estimates from $f(\boldsymbol{x})$.

Logloss

- Misclassified examples are punished linearly.
- Can recover probability estimates from $f(\boldsymbol{x})$:

$$p(y = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-2f(\boldsymbol{x})}}$$

At iteration m, minimize

$$L_m(\phi) = \sum_{i=1}^{N} \log\left(1 + e^{-2\tilde{y}_i(f_{m-1}(\boldsymbol{x}) + \phi(\boldsymbol{x}_i))}\right)$$

LogitBoost

1. Initialize the observation weights $w_i = 1/N$ and probability estimates $\pi_i = 1/2$.
2. For each $m = 1, ..., M$:
   2.1 Compute the working response: $z_i = \frac{y_i - \pi_i}{\pi_i(1-\pi_i)}$
   2.2 Compute the weights: $w_i = \pi_i(1 - \pi_i)$
   2.3 Fit by a weighted least-squares regression: $\phi_m = \text{argmin}_\phi \sum_{i=1}^{N} w_i \left(z_i - \phi(\boldsymbol{x}_i)\right)^2$
   2.4 Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + 0.5\phi_m(\boldsymbol{x})$
   2.5 Compute $\pi_i = 1/\left(1 + e^{-2f(\boldsymbol{x}_i)}\right)$
3. Final learned function: $f(\boldsymbol{x}) = \text{sign}\left[\sum_{m=1}^{M} \phi_m(\boldsymbol{x})\right]$

Useful paper on AdaBoost and LogitBoost: *"Additive Logistic Regression: A Statistical View of Boosting"* (Friedman, Hastie, and Tibshirani, 2000).

Gradient Boosting

So far, we had to derive new versions of boosting for every different loss function (AdaBoost — exponential loss, LogitBoost — logloss, etc.).

Gradient Boosting — generic version of boosting:

$$\hat{f}(\boldsymbol{x}) = \text{argmin}_{f(\boldsymbol{x})} L(f(\boldsymbol{x}))$$

Gradient boosting is analogous to the *steepest descent* numerical optimization procedure.

Steepest descent chooses the direction in which the function most rapidly decreases.

In gradient boosting, we want to not only minimize loss on the training set but also generalize to new data.

Use some weak learner: for example, at each iteration m, grow a tree whose predictions are as close as possible to the negative gradient.

Gradient Boosting

1. Initialize $f_0(\boldsymbol{x}) = \operatorname{argmin}_\gamma \sum_{i=1}^N L\left(y_i, \phi(\boldsymbol{x}_i; \boldsymbol{\gamma})\right)$
2. For each $m = 1, ..., M$:
   - 2.1 Compute the gradient at the last update: $g_{im} = \left[\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f(\boldsymbol{x}_i)=f_{m-1}(\boldsymbol{x}_i)}$

     Need a differentiable loss function! ▸ Table
   - 2.2 Using the weak learner, compute $\boldsymbol{\gamma}_m = \operatorname{argmin}_\gamma \sum_{i=1}^N (-g_{im} - \phi(\boldsymbol{x}_i; \boldsymbol{\gamma}))^2$
   - 2.3 Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu\phi(\boldsymbol{x}; \boldsymbol{\gamma}_m)$
3. Final learned function: $f(\boldsymbol{x}) = f_M(\boldsymbol{x})$

Applying this algorithm to specific loss functions, we recover the ‘specific’ boosting cases.

Sparse Boosting

Search over all possible variables j, and pick $j(m)$ that best predicts the residual vector:

$$j(m) = \text{argmin}_j \sum_{i=1}^{N} (r_{im} - \hat{\beta}_{jm} x_{ij})^2$$

where

$$\hat{\beta}_{jm} = \frac{\sum_{i=1}^{N} x_{ij} r_{im}}{\sum_{i=1}^{N} x_{ij}^2}$$

We obtain

$$\phi_m(\boldsymbol{x}) = \hat{\beta}_{j(m),m} x_{j(m)}$$

Sparse boosting $\Rightarrow$ sparse estimate.

Notice close association with lasso!

- Can modify the algorithm to make it equivalent to the Least Angle Regression (LARS) algorithm that is used for the lasso problem.

Boosting Trees

Classification and Regression Tree (CART) models are often used as weak learners.

- Use a shallow tree (a small-depth tree) $\Rightarrow$ low variance + high bias.
- High bias is compensated for in subsequent rounds of boosting.

Can we use the trees that have larger depth?

- In addition to M (rounds of boosting) and ν (shrinkage factor), the height of the tree, J, is another tuning parameter.
- Decision stump: $J = 2$.
- It is recommended to use $J \simeq 6$ (up to five-variable interactions).

Gradient boosting + shallow trees $\Rightarrow$ Multivariate Adaptive Regression Trees (MART) model.

Boosting Trees

Trees partition the space of all joint predictor variable values into disjoint regions R_j.

A constant γ_j is assigned to each R_j and the predictive rule is $x \in R_j \Rightarrow f(x) = \gamma_j$

Formal expression of a tree:

$$T(\boldsymbol{x}, \boldsymbol{\Theta}) = \sum_{j=1}^{J} \gamma_j \mathbb{I}\{\boldsymbol{x} \in R_j\}, \qquad \boldsymbol{\Theta} = \{R_j, \gamma_j\}_{j=1}^{J}$$

Find parameters by solving

$$\hat{\boldsymbol{\Theta}} = \text{argmin}_{\boldsymbol{\Theta}} \sum_{j=1}^{J} \sum_{\boldsymbol{x}_i \in R_j} L(y_i, \gamma_j)$$

Finding γ_j given $R_j \Rightarrow$ easy! (e.g., $\hat{\gamma}_j = \bar{y}_j$ for a regression)

Finding $R_j \Rightarrow$ challenging!

Boosting Trees

Boosted tree model:

$$f_M(\boldsymbol{x}) = \sum_{m=1}^{M} T(\boldsymbol{x}; \boldsymbol{\Theta}_m)$$

Growing the trees in a FSAM manner, solve at each iteration:

$$\hat{\boldsymbol{\Theta}}_m = \text{argmin}_{\boldsymbol{\Theta}} \sum_{i=1}^{N} L(y_i, f_{m-1}(\boldsymbol{x}_i) + T(\boldsymbol{x}_i; \boldsymbol{\Theta}))$$

Again, finding γ_{jm} is easy if we know R_{jm}... but finding R_{jm} if difficult.

Cases when the problem simplifies:

1. **Squared error loss**: Grow a tree that best predicts current residuals, $y_i - f_{m-1}(\boldsymbol{x}_i)$, and $\hat{\gamma}_{jm}$ is the mean of the residuals in each corresponding region R_{jm}.
2. **Exponential loss**: With binary classification (under some conditions) we can use AdaBoost. In general, the $\hat{\gamma}_{jm}$ will be the weighted log-odds in each region R_{jm}.

We can also do gradient boosting for trees.

Concluding Remarks

Boosting works very well, especially for classifiers.

- Boosting a-la L_1 regularization (we use one in lasso).

The "art" of boosting.

- Trade-off between boosting iterations, M, and shrinkage parameter, ν (smaller $\nu \Rightarrow$ slower learning $\Rightarrow$ higher M).

Boosting is actively used in real-world applications.

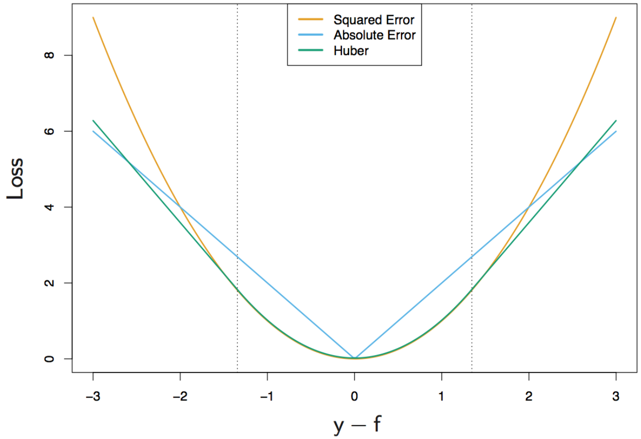
- Viola-Jones face detection algorithm uses AdaBoost with decision stumps as weak learners.

Boosting algorithms can be based on convex or non-convex optimization algorithms.

- Today we have captured convex optimization algorithms.
- BrownBoost — example of a boosting algorithm based on non-convex optimization.

Appendix

Loss Functions for Regressions ◂ Back



Note: The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when $|y - f|$ is large.

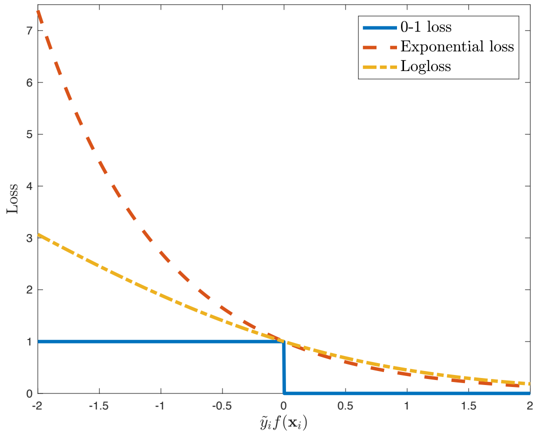
Huber Loss Function

Back

$$L(y_i, f(\boldsymbol{x}_i)) = \begin{cases} (y_i - f(\boldsymbol{x}_i))^2, & \text{for } |y_i - f(\boldsymbol{x}_i)| \leq \delta \\ 2\delta|y_i - f(\boldsymbol{x}_i)| - \delta^2, & \text{otherwise} \end{cases}$$

Loss Functions for Binary Classification

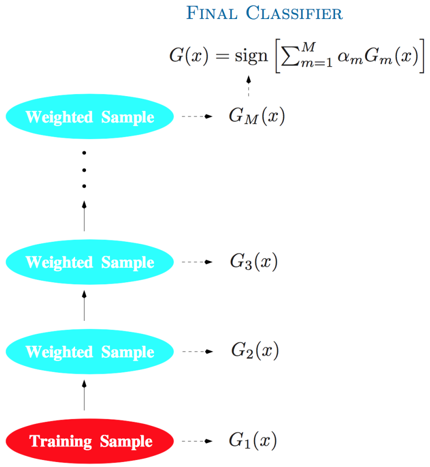
Back



Note: Misclassifications (negative margins, $\tilde{y}f(\boldsymbol{x})$) are penalized more heavily than correct classifications (positive margins, $\tilde{y}f(\boldsymbol{x})$). Logloss uses log base 2.

AdaBoost

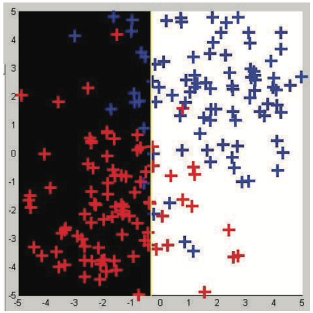
Back

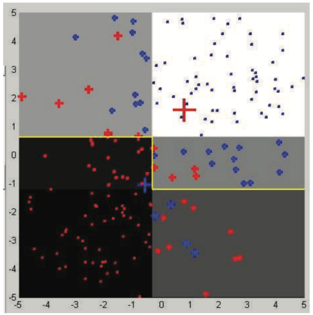


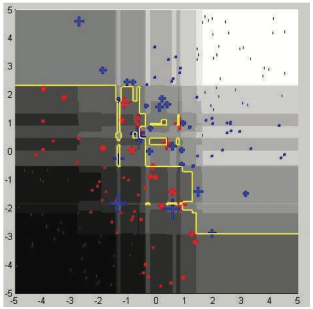
Note: Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction. Source: Hastie, Tibshirani, and Friedman.

AdaBoost Example

Back


(a)

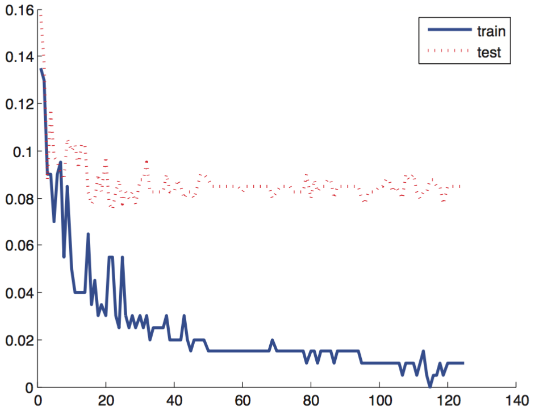

(b)


(c)

Note: Example of AdaBoost using a *decision stump* (decision tree with one internal node which is immediately connected to the terminal nodes) as a weak learner. The degree of blackness represents the confidence in the red class. The degree of whiteness represents the confidence in the blue class. The size of the datapoints represents their weight. Decision boundary is in yellow. (a) After 1 round. (b) After 3 rounds. (c) After 120 rounds. Source: Murphy.

AdaBoost Example

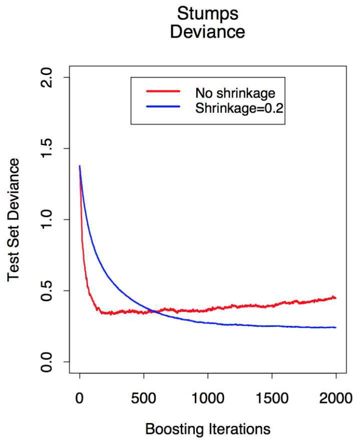
Back

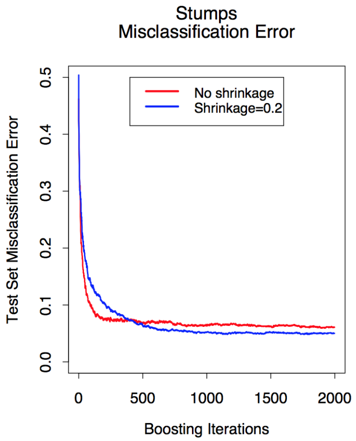


Note: Example of AdaBoost using a *decision stump* (decision tree with one internal node which is immediately connected to the terminal nodes) as a weak learner. Training (solid blue) and test (dotted red) error are plotted against the number of iterations. Source: Murphy.

Shrinkage

Back





Note: The left panel reports test deviance (regression example), while the right panels show misclassification error (classification example). The beneficial effect of shrinkage can be seen in both cases, especially for deviance in the left panel. Source: Hastie, Tibshirani, and Friedman.