Ensemble Methods - Boosting

Bin Li

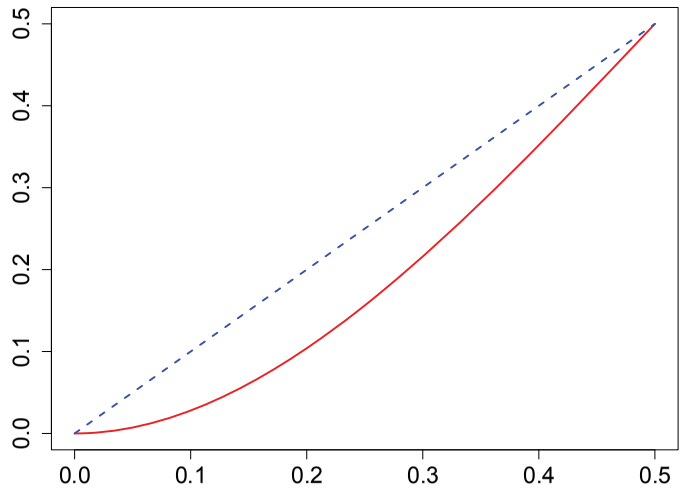
IIT Lecture Series

Boosting a weak learner

- Weak learner L produces an h with error rate $\beta = \frac{1}{2} - \epsilon < \frac{1}{2}$, with $Pr \geq 1 - \delta$ for any $\mathcal{D}$. L has access to continuous stream of training data and a class *oracle*.
  1. L learns h_1 on first N training points.
  2. L randomly filters the next batch of training points, extracting $N/2$ points correctly classified by h_1, $N/2$ incorrectly classified, and produces h_2.
  3. L builds a third training set of N points for which h_1 and h_2 disagree, and produces h_3.
  4. L ouptuts $h =$ Majority Vote(h_1, h_2, h_3)
- Theorem (Schapire, 1990): "The Strength of Weak Learnability"

$$\text{error}_{\mathcal{D}}(h) \leq 3\beta^2 - 2\beta^3 < \beta$$

Plot of error rate $3\beta^2 - 2\beta^3$



What is AdaBoost?

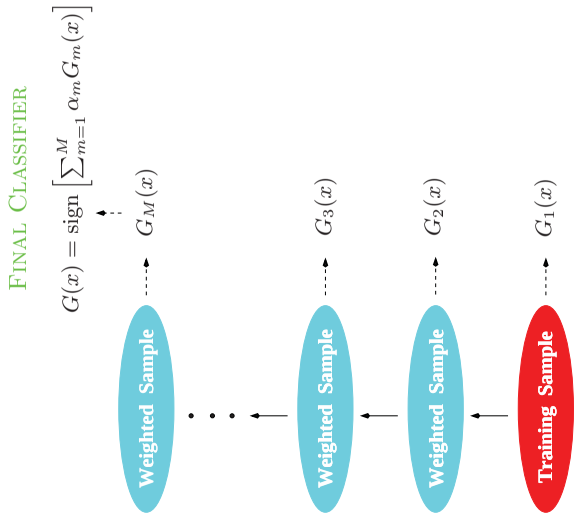
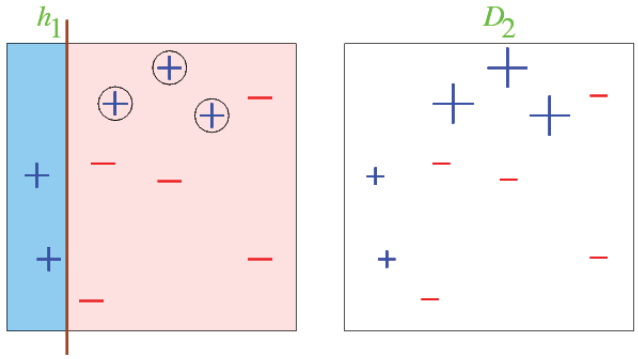


Figure from HTF 2009.

AdaBoost (Freund & Schapire, 1996)

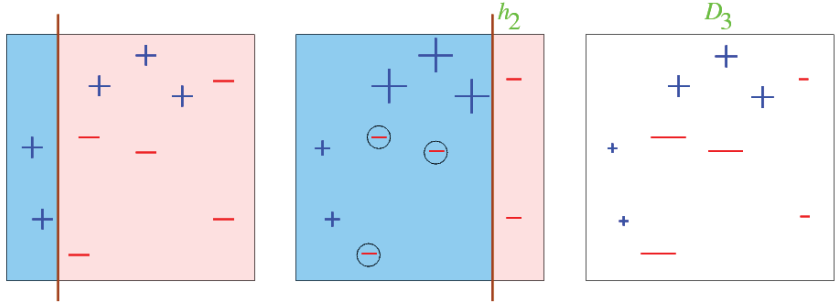
1. Initialize the observation weigths: $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M repeat steps (a)-(d):
   (a) Fit a classifier $G_m(x)$ to the training data using weigths w_i
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_m(x_i)\right)}{\sum_{i=1}^{N} w_i}$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Update weights for $i = 1, \ldots, N$:
   $$w_i \longleftarrow w_i \cdot \exp\left[\alpha_m \cdot I(y_i \neq G_m(x_i))\right]$$
   and renormalize w_i to sum to 1.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

An example of AdaBoost



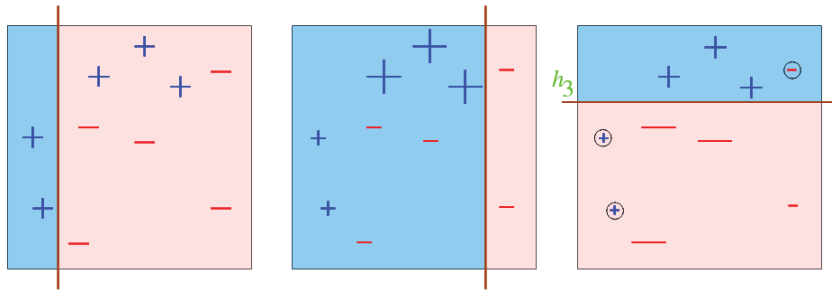
$\text{err}_1 = \sum_i w_i I(G_1(x_i) \neq y_i) = 0.3$
$\alpha_1 = \log \frac{1-\text{err}_1}{\text{err}_1} = 0.847$
w_i for misclassified points: $0.1 \times \exp(0.847) = 0.233$.
After normalizing, w_i for wroing points is $\frac{0.233}{0.233\times3+0.1\times7} = 0.1667$.
For correct points, w_i is $\frac{0.1}{0.233\times3+0.1\times7} = 0.0714$

An example of AdaBoost (cont.)



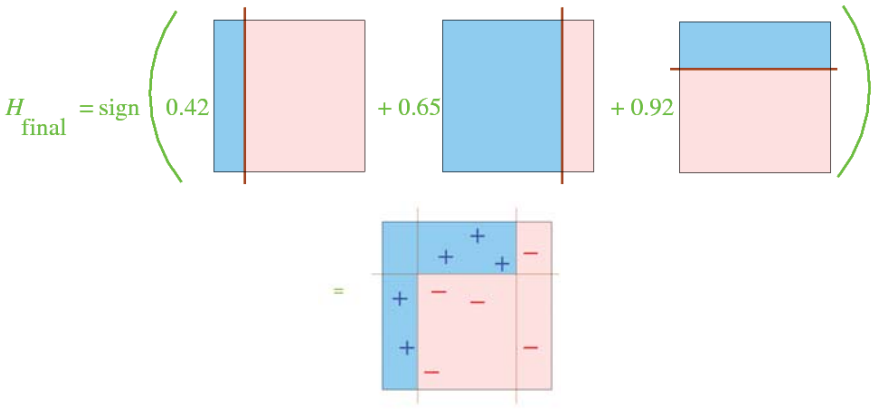
$\text{err}_2 = \sum_i w_i I(G_2(x_i) \neq y_i) = 3 \times 0.07 = 0.21$
$\alpha_2 = \log \frac{1-\text{err}_2}{\text{err}_2} \approx 1.3$

An example of AdaBoost (cont.)



$\text{err}_3 = 0.14$
$\alpha_3 = \log \frac{1-\text{err}_3}{\text{err}_3} = 1.84$

An example - final classifier



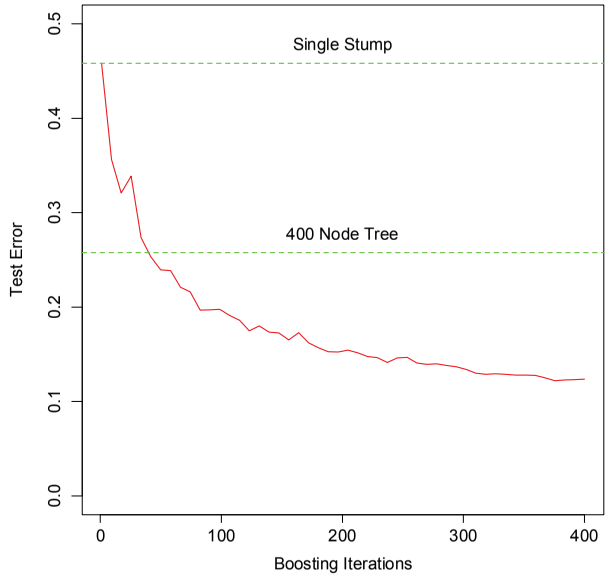
A simulation - "nested spheres" example

Ten features $X_1, \ldots, X_{10}$ are independent $N(0,1)$ r.v. The deterministic target Y is defined by

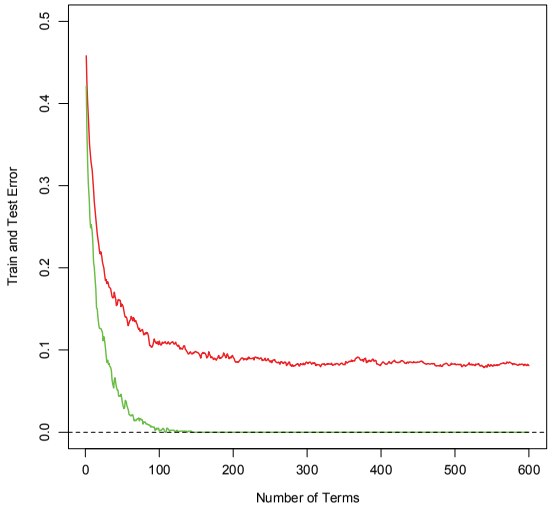
$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) = 9.34 \\ -1 & \text{otherwise} \end{cases}$$

- 2000 training observations (about 1000 cases in each class), and 10,000 test observations.
- Bayes error is 0% (noise-free).
- A *stump* is a two-node tree, after a single split.
- We use stump as the "weak learner" in the AdaBoost algorithm.

NS example (figure from HTF 2009)



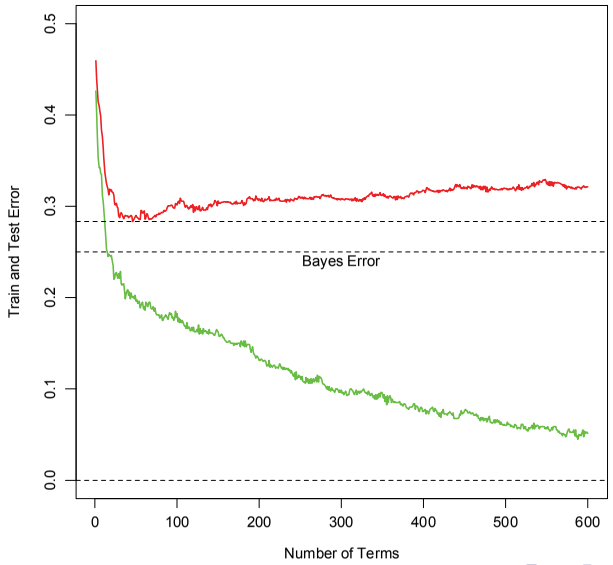
Training and testing errors (figure from HTF 2009)



Boosting drives the training error to zero. Further iterations continue to improve test error.

Boosting for noisy problems (figure from HTF)

Nested spheres example with added noise - Bayes error is 25%



Weak classifiers in AdaBoost

In the illustrative example, after fitting the first classifier the normalized weights w_i for wrong and correct points are 0.1667 and 0.0714. Then what is the sum of weights for wrong and correct points separately?

Weak classifiers in AdaBoost (cont.)

Sum of weights for wrongly classified points:

$$\begin{aligned} & \sum_{i:y_i \neq G(x_i)} w_i exp(\alpha_m) \\ = & \sum_{i:y_i \neq G(x_i)} w_i \left(\frac{1 - err_m}{err_m} \right) \\ = & \sum_{i:y_i \neq G(x_i)} w_i \left(\frac{\sum_i w_i I(y_i = G_m(x_i))}{\sum_i w_i I(y_i \neq G_m(x_i))} \right) \\ = & \sum_i w_i I(y_i = G_m(x_i)) = \sum_{i:y_i = G(x_i)} w_i \end{aligned}$$

Error rate is $1/2$ on the new weighting scheme.
The next weak classifier is *"independent"* to the previous one.

Boosting as an additive model

- Boosting builds an additive model: $f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$. For example, in boosting trees $b(x; \gamma_m)$ is a tree, and γ_m parametrizes the splits.
- We use additive models in statistics all the time!
  - Multiple linear regression: $f(x) = \sum_j \beta_j x_j$
  - Generalized additive model: $f(x) = \sum_j f_j(x_j)$
  - Basis expansions in spline model: $f(x) = \sum_{m=1}^{M} \theta_m h_m(x)$
- Traditionally, the parameters f_m, θ_m are fit **jointly** (i.e. least squares, maximum likelihood).
- Friedman *et al.* (2000) have shown that AdaBoost is equivalent to stagewise additive logistic regression using the exponential loss criterion $L(y, f(x)) = exp(-yf(x))$.
- $yf(x)$ is called "**margin**" (> 0 classified correctly, < 0 misclassified).

Stagewise additive modeling

Based on Friedman, Hastie and Tibshirani (2000)'s result, the AdaBoost is equivalent to the following modeling strategy.

Forward Stagewise Additive Modeling.

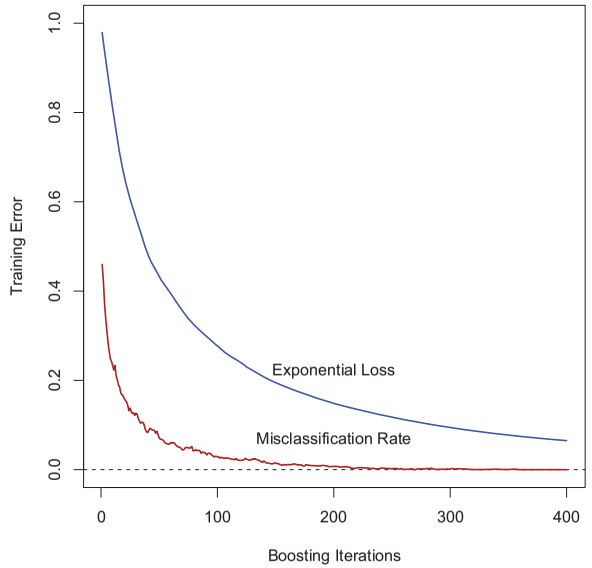
1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)\right).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

Compare with **stepwise** approach, **stagewise** additive modeling slows the process down, and tends to overfit less quickly.

Exponential loss vs. misclassification loss (figure from HTF, 2009)



Why exponential loss?

- The AdaBoost algorithm was originally motivated from a very different perspective.
- What is the statistical property for exponential loss?
  - What does it estimate? The *population minimizer* of e^{-yf} is:

$$\frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)} = \arg\min_{f(x)} E_{Y|x}\left(e^{-Yf(x)}\right)$$

  - AdaBoost estimates one-half the log-odds of $P(Y=1|x)$ and uses its sign as the the classification rule.
- Binomial negative log-likelihood or *deviance* has the same population minimizer.

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

- Note e^{-Yf} is not a proper log-likelihood.

Loss functions for classification (figure from HTF, 2009)

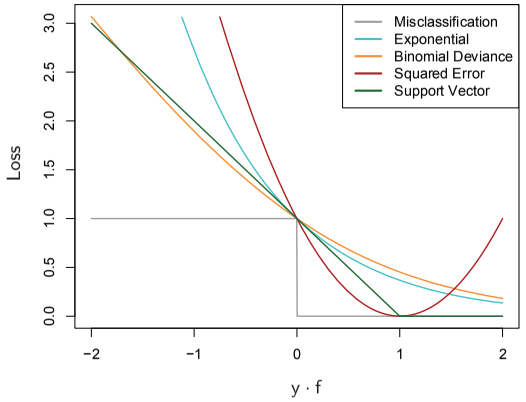


FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1+\exp(-2yf))$*; squared error:* $(y-f)^2$*; and support vector:* $(1-yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*

Loss functions for classification (cont.)

- Should penalize negative margins more heavily than positive ones.
- Both exponential and deviance loss are monotone continuous approximations to misclassification loss.
- Exponential loss concentrates much more influence on observations with large negative margins than binomial deviance.
- Binomial deviance is far more robust in noisy settings, where the Bayes error rate is not close to zero, and especially in situations where there is misspecification of the class labels in the training data.
- The performance of AdaBoost has been empirically observed to dramatically degrade in such situations.

Loss function for multicategory classification

The logistic model generalizes naturally to K classes as

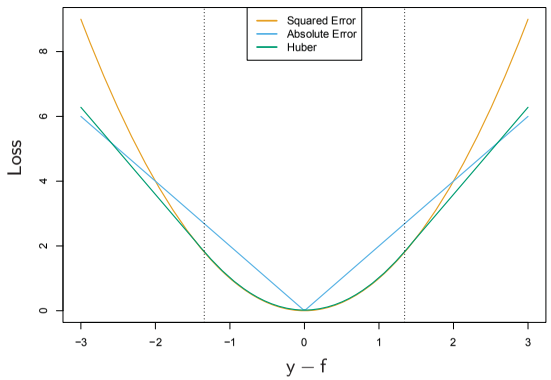
$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, \quad k = 1, \ldots, K,$$

which ensures that $0 \leq p_k(x) \leq 1$ and they sum to one. In addition, we traditioally set $f_K(x) = 0$.

The binomial deviance extends naturally to the K-class *multinomial deviance* loss function:

$$\begin{aligned} L(y, p(x)) &= -\sum_{k=1}^{K} I(y = \mathcal{G}_k) \log p_k(x) \\ &= -\sum_{k=1}^{K} I(y = \mathcal{G}_k) f_k(x) + \log \left(\sum_{l=1}^{K} e^{f_l(x)} \right) \end{aligned}$$

Loss functions for regression (figure from HTF, 2009)

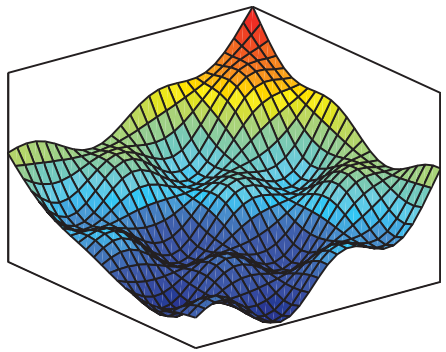


Huber loss criterion:

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

Gradient descent for optimization

The gradient of $f(x)$ (denote $\bigtriangledown f$) at a point $\langle u_1, u_2, \ldots, u_n \rangle$ can be thought of as a vector indicating which way is "uphill".



If this is an error function, we want to move "downhill" on it, i.e., in the direction opposite to the gradient.

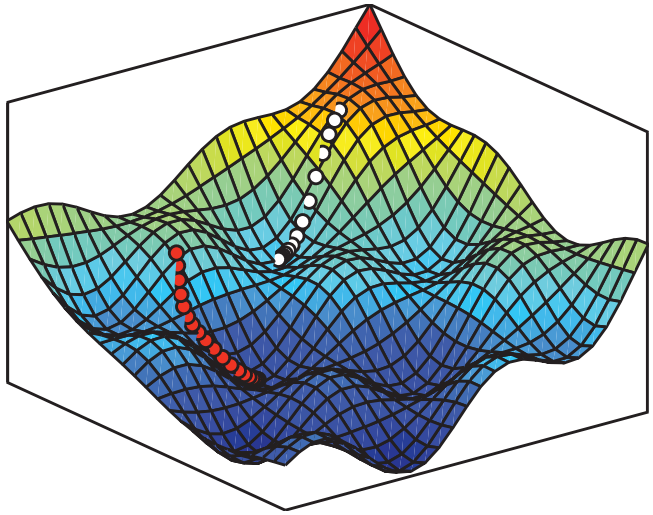
Gradient descent

- The basic algorithm for gradient descentassumes that $\bigtriangledown f$ is easily computed.
- We want to produce a sequence of vectors $\mathbf{u}^1, \mathbf{u}^2, \ldots$ with the goal that:
  - $f(\mathbf{u}^1) > f(\mathbf{u}^2) > f(\mathbf{u}^3) > \ldots$
  - $\lim_{m\to\infty} f(\mathbf{u}^m) = f(\mathbf{u}^*)$ and $f(\mathbf{u}^*)$ is locally optimal.
- The gradient descent algorithm: do for $m = 0, 1, 2, \ldots$

$$f(\mathbf{u}^{(m+1)}) = f(\mathbf{u}^m) - \alpha_i \bigtriangledown f(\mathbf{u}^m),$$

  where $\alpha_i > 0$ is the *step size* or *learning rate* for iteration m.
- If α_i is too large, oscillation may occur. If too small, the process is slow.

Example of gradient descent traces



Boosting as gradient descent optimization

Gradient Boosting Algorithm (Friedman, 2001)

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N L(y_i, \rho)$
2. For $m = 1$ to M do:
   a. $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$, $i = 1, \ldots, N$.
   b. $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
   c. $\rho_m = \arg\min_\rho \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho_m h(\mathbf{x}_i; \mathbf{a}_m))$
   d. $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
3. endFor loop
   end Algorithm

2.a: $\tilde{y}_i$ is the negative gradient shows the "downhill" direction for $L(y, f)$.
2.b Find the optimal model $h(\mathbf{x}; \mathbf{a})$ based on $\tilde{y}_i$ using least squares.
2.c Find the optimal learning rate ρ_m.

Least square gradient boosting

Response y, predictors $(x_1, x_2, \ldots, x_p)$ and loss function $L(y, f)$.

1. $F_0(\mathbf{x}) = \bar{y}$
2. For $m = 1$ to M do:
   a. $\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i)$, $i = 1, \ldots, N$.
   b. $(\rho_m, \mathbf{a}_m) = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^N [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$
   c. $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
3. endFor loop
   end Algorithm

2.a: For squared loss function $L(y, F) = (y - F)^2/2$,
$\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i)$ (i.e. current residual).
2.b Since we are using squared loss, ρ_m is simultaneously estimated in Step 2.b.

Gradient boosting for other problems

- Gradient boosting is very general. It can be applied to any problem as long as the gradient is available.
- Gradient boosting can be applied to
  - Cox proportional hazard model in survival analysis
  - Poisson regression
  - Quantile regression
  - L_1 regression using absolute loss (LAD).
- If the base learner is tree, it directly inherits nice properties from trees:
  - able to handle mix-type predictors (categorical and continuous);
  - able to handle missing values;
  - able to incorporate high order interactions;
  - able to catch non-linear effects.

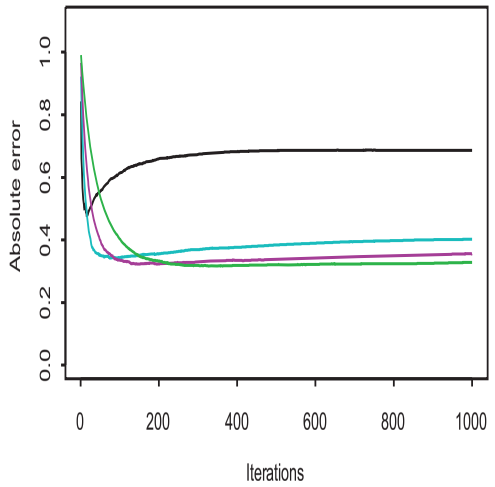
Shrinking the learning rate

Incremental gradient boosting.

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N L(y_i, \rho)$
2. For $m = 1$ to M do:
   a. $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$, $i = 1, \ldots, N$.
   b. $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
   c. $\rho_m = \arg\min_\rho \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho_m h(\mathbf{x}_i; \mathbf{a}_m))$
   d. $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \rho_m h(\mathbf{x}; \mathbf{a}_m)$
3. endFor loop
   end Algorithm

Here ν is a **shrinkage factor**, and often $0 < \nu \ll 1$. Shrinkage slows the stagewise model-building even more, and typically leads to better performance.

Shrinkage effects (figure from Friedman, 2001)



The four curves correspond to shrinkage parameter $\nu = 1, 0.25, 0.125, 0.06$ in the order (top to bottom) at the extreme right of the plot.

Least square boosting with linear regression

Below is a version of least squares boosting for linear regression: (assume predictors are standardized).

(Incremental) Forward Stagewise Linear Regression

1. Start with $r = y$, $\beta_1, \beta_2, \ldots, \beta_p = 0$.
2. Find the predictor x_j most correlated with r.
3. Update $\beta_j \leftarrow \beta_j + \delta_j$, where $\delta_j = \epsilon \cdot \text{sign}\langle r, x_j \rangle$.
4. Set $r \leftarrow r - \delta_j x_j$ and repeat Step 2 and 3 many times.

$\delta_j = \langle r, x_j \rangle$ gives usual forward stagewise; different from forward stepwise.

Stagewise least square regression vs. Lasso

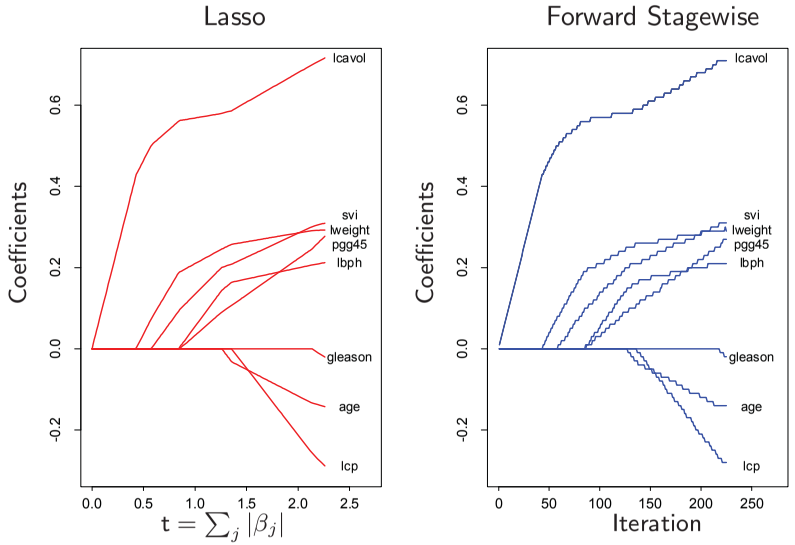


Figure from HTF (2009)

Least square boosting with trees

Response y and predictors $(x_1, x_2, \ldots, x_p)$.

1. Start with function $F(x) = 0$ and residual $r = y$.
2. Fit a CART regression tree to r giving $f(x)$.
3. Set $F(x) \leftarrow F(x) + \epsilon f(x)$
4. Set $r \leftarrow r - \epsilon f(x)$ and repeat Step 2 and 3 many times.

Subsampling

- Bagging improves the performance of a noisy classifier by subsampling.
- Subsampling in *stochastic gradient boosting* (Friedman, 1999.) improves performance and computational efficiency.
- At each iteration, we sample a fraction η of training observations (without replacement) and grow the tree (or other learners) using that subsample.
- A typical value for η is 0.5, although for large N, η can be substantially smaller than 0.5.
- Figure (on next slide) shows the effect of subsampling in a classification and a regression example.
  - Subsampling along with shrinkage slightly outperforms the rest in both cases.
  - Subsampling without shrinkage does poorly.

Subsampling example

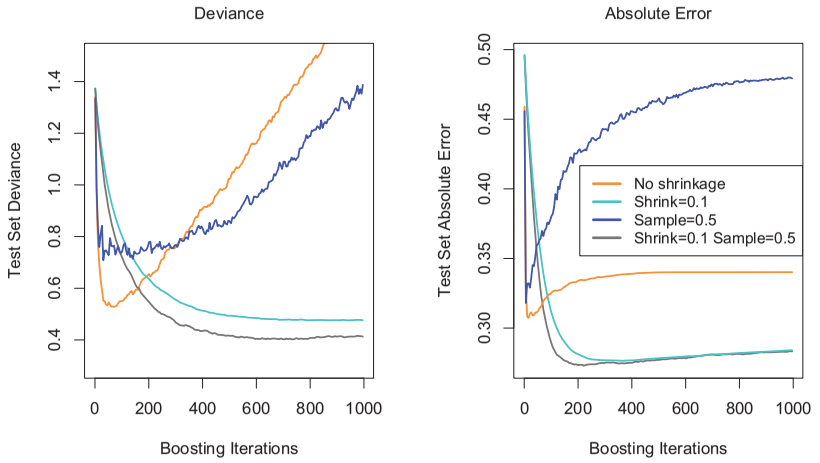


Figure from HTF (2009)

Right-sized trees for boosting trees

- A complicated approach.
  - For each iteration, grow a very large tree (oversized) first.
  - Then a bottom-up procedure procedure is employed to prune it to the estimated optimal number of terminal nodes (tree size).
  - Trees tend to be too large, especially in the early iterations.
  - Degrades performance and increases computation.
- A simple approach.
  - Restrict all trees to be the same size J.
  - The interaction level is limited by the tree size J.
    - $J = 2$ (single split "stump"), produces boosted models with only main effects
    - With $J = 3$ or $J = 4$, two-variable interactions are allowed.
    - Experience so far indicates that $4 \leq J \leq 8$ works well. It is unlikely that $J > 10$ is required.
- In "nested spheres" example (next slide), the generative model is additive. Boosting model with $J > 2$ incurs unnecessary variance and hence the higher test error.

Nested spheres example

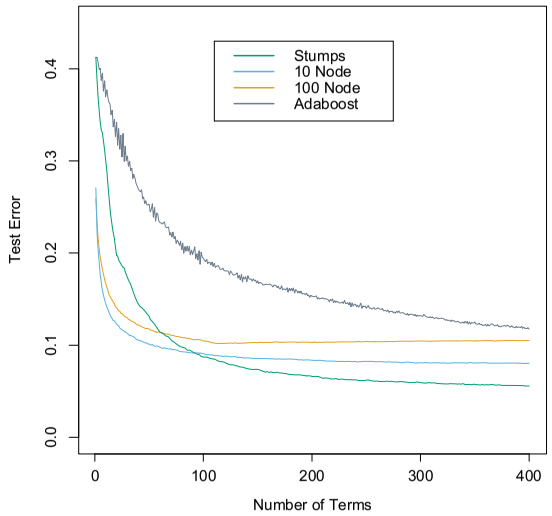


Figure from HTF (2009)

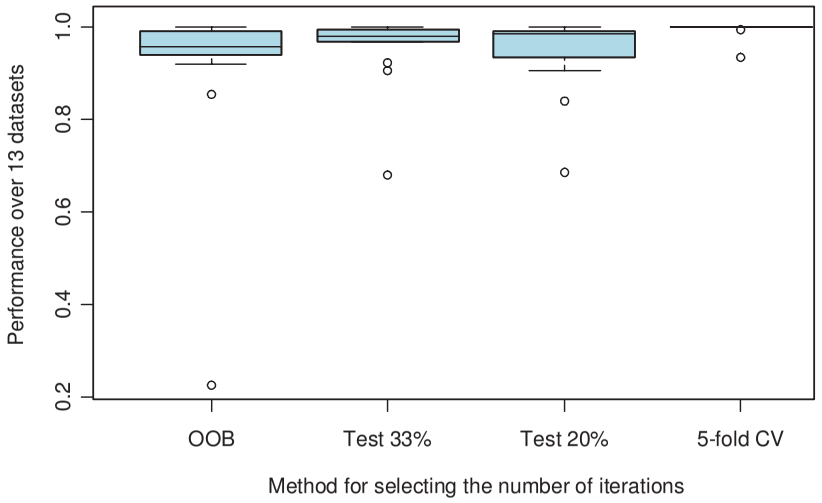
Estimate the optimal number of iterations

There are three ways to estimate the optimal number of iterations after a boosting model has been fit.

- Monitering the performance based on an independent test set.
- Cross-validation.
- Out-off-bag (OOB) approach. It evaluates the reduction in deviance on those observations not used in constructing the current regression tree. Studies show that the OOB estimator (of the optimal number of iteration) tends to under-estimates the reduction in deviance. As a result, it is usually too conservative in its selection for the optimal number of iterations.

The boxplots on next slide show the comparative performance relative to the best method on each of the 13 datasets. 5-fold CV is consistently the best. OOB, using a 33% and 20% test set, all have datasets for which they perform considerably worse than the best method.

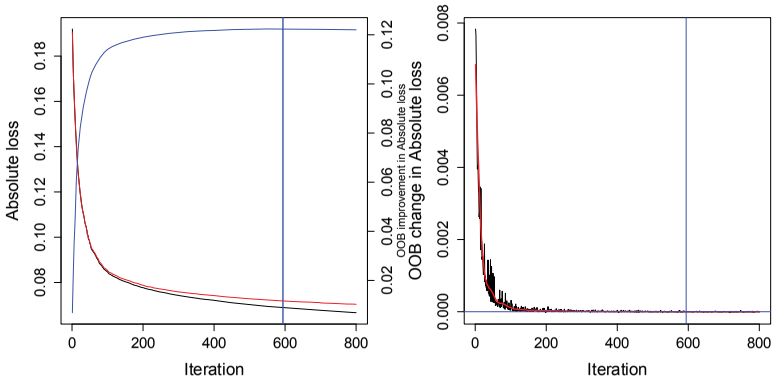
A comparative study (from Ridgeway, 2007)



California housing data

- Available at CMU *StatLib* repository (`http://lib.stat.cmu.edu/datasets/`).
- Originally used by Pace and Barry (1997).
- Consist of aggregated data from each of 20640 neighborhoods (1990 census block groups) in California.
- The response variable is the median house value in each neighborhood measured in units of \$100,000.
- Eight continuous input variables, which are demographics (e.g. median income), housing density and occupancy, housing properties (e.g. number of rooms/bedrooms), and location of each neighborhood.
- Randomly divide the dataset into a training set (80%) and a test set (20%).

CA house example - error curves



Red (black) curve on the left plot is the test (train) error (average of absolute error). The optimal number of iterations based on OOB improvement is around 600. The red (black) curve on the right plot is the smoothed (raw) OOB changes in AE.

CA house example - relative variable importance

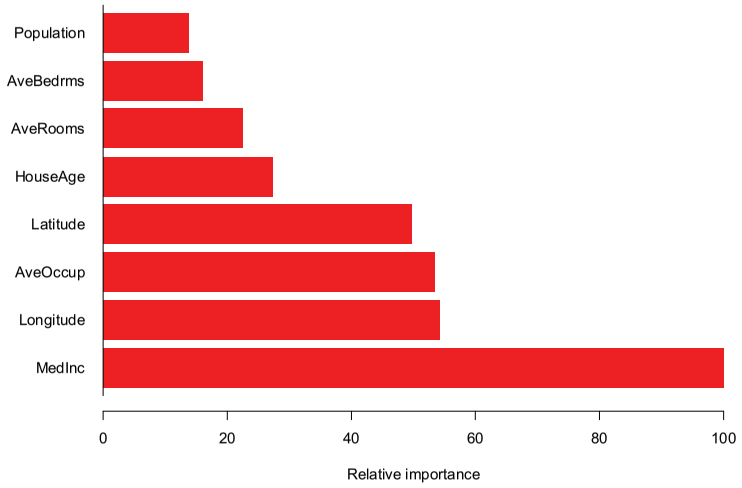
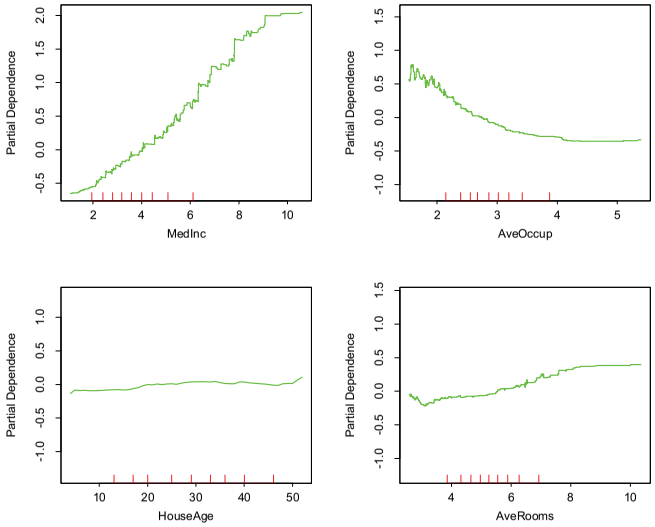


Figure from HTF (2009)

CA house example - partial dependence plots



The red ticks at the base of the plot are deciles of the input variables.

Figure from HTF (2009)

CA house example - partial dependence plots

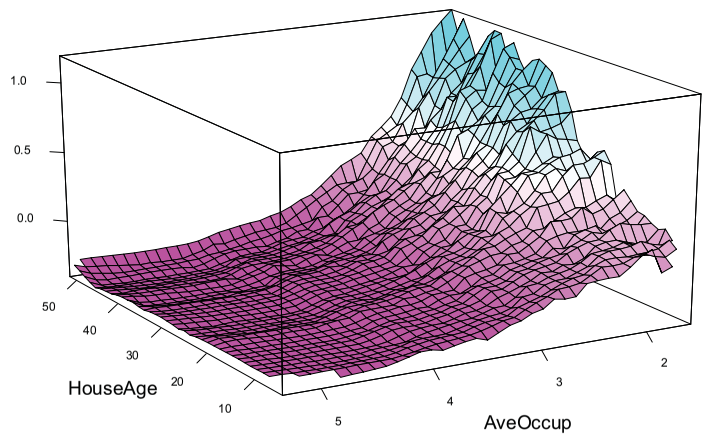


Figure from HTF (2009)

CA house example - spatial effects

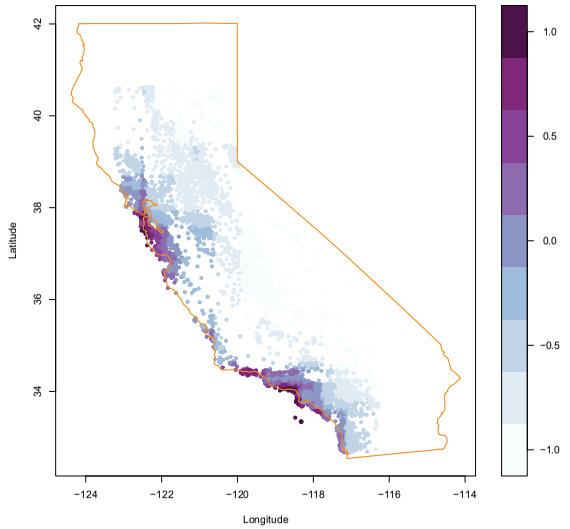


Figure from HTF (2009)

Packages in R for boosting

- `gbm` package provides an internal regression tree engine, marginal plots and additional utilities for optimizing a wider range of loss functions outside of classification.
- `mboost` package is an advanced boosting tool for processing several base learners and arbitrary loss functions.
- `ada` package provides an R implementation for discrete, real, and gentle stochastic boosting (see Friedman, Hastie and Tibshirani, 2000) under both logistic and exponential loss for classification, ideally suited for small to moderate-sized data sets.
- `xgoost` (eXtreme Gradient BOOSTing) package is an efficient and scalable implementation of gradient boosting. It can automatically do parallel computation. Input data can be from sparse matrix format. Regularization is used to avoid overfitting.

Reference

- Friedman, J.H., (1999) Stochastic Gradient Boosting, *Computational Statistics and Data Analysis*, **38**, 367-378.
- Friedman, J.H., Hastie, T. and Tibshirani, R. (2000) Additive logistic regression: a statistical view of boosting (with discussion), *The Annals of Statistics*, **28**, 337-407.
- Friedman, J.H., (2001) Greedy function approximation: A gradient boosting machine, *The Annals of Statistics*, **29**, 1189-1232.
- Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The Elements of Statistical Learning*, Springer.
- Ridgeway,G. (2007) Generalized Boosted Models: A guide to the `gbm` package.