Statistical Learning for Data Science

MSDS534

Lecture 07: Boosting

Department of Statistics & Biostatistics
Rutgers University

Acknowledgement

- Some of the figures in this presentation are taken from *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.
- Some of the figures in this presentation are taken from *Elements of Statistical Learning* (Springer, 2009) with permission from the authors: T. Hastie, R. Tibshirani and J. Friedman.

Reading Assignments

- ESL: 10.1–10.13.1.

Outline

AdaBoost

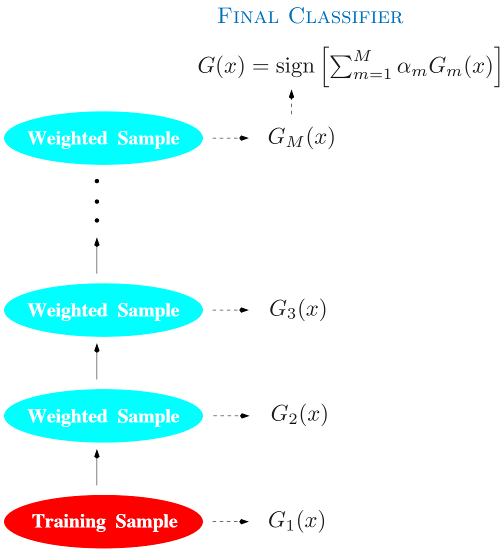


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

AdaBoost.M1.

Algorithm 10.1 *AdaBoost.M1.*

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

Example

- Ten features $X_1, \ldots, X_{10}$ which are iid $N(0,1)$.
- $$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 \geq \chi^2_{10}(.5) \approx 9.34, \\ -1 & \text{otherwise.} \end{cases}$$
- Training sample size: 2,000. Test sample size: 10,000.
- Try the following:
  - Stump: a classification tree with two terminal nodes.
  - A 21-Node tree.
  - Boost the stump up to 400 iterations.

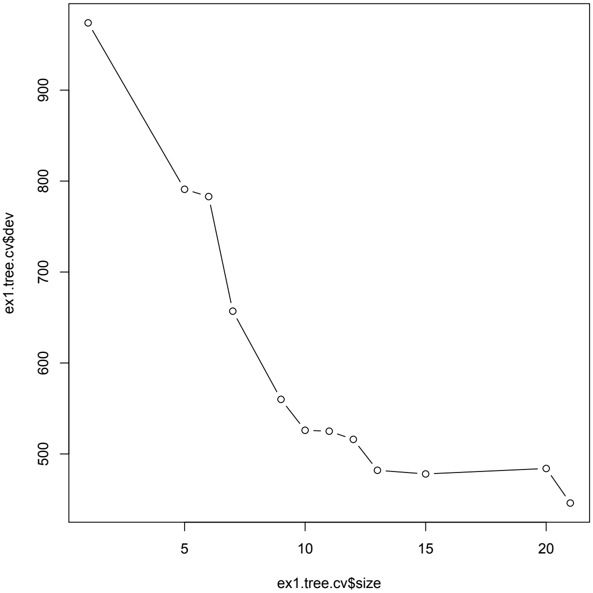
Example: Generate Data

```
library(tree)
library(gbm)

n.tr=2000
n.te=10000
p=10

set.seed(123)
X.tr=matrix(rnorm(n.tr*p),nrow=n.tr)
y=apply(X.tr^2,MAR=1,FUN="sum")
y=y>=9.34
y=as.factor(as.numeric(y))
ex1.tr=data.frame(X.tr,y)

X.te=matrix(rnorm(n.te*p),nrow=n.te)
y=apply(X.te^2,MAR=1,FUN="sum")
y=y>=9.34
y=as.factor(as.numeric(y))
ex1.te=data.frame(X.te,y)
```

Example: Trees

```
###################### 21 Node Tree
ex1.tree=tree(y~.,ex1.tr)
summary(ex1.tree)
## Classification tree:
## tree(formula = y ~ ., data = ex1.tr)
## Number of terminal nodes:  21
## Residual mean deviance:  0.8235 = 1630 / 1979
## Misclassification error rate: 0.175 = 350 / 2000

ex1.tree.pred=predict(ex1.tree,ex1.te,type="class")
table(ex1.tree.pred,ex1.te$y)
test.error.tree=sum(ex1.tree.pred!=ex1.te$y)/n.te ##.2259

###################### Stump
ex1.stump=prune.tree(ex1.tree,best=2,method="deviance")
## can replace "deviance" by "misclass"
## ex1.stump=prune.misclass(ex1.tree,best=2)
ex1.stump.pred=predict(ex1.stump,ex1.te,type="class")
table(ex1.stump.pred,ex1.te$y)
test.error.stump=sum(ex1.stump.pred!=ex1.te$y)/n.te ##.4649

###################### Cross Validation
set.seed(3)
ex1.tree.cv =cv.tree(ex1.tree,FUN=prune.misclass)
ex1.tree.cv
plot(ex1.tree.cv$size,ex1.tree.cv$dev,type="b")
```

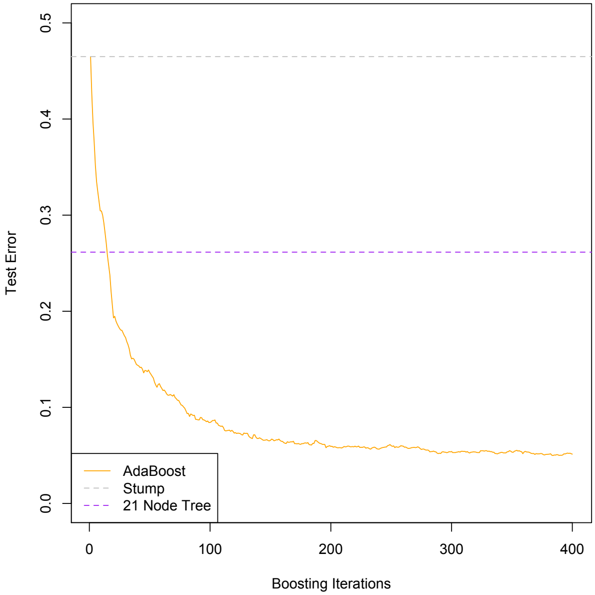
Example: Cross Validation



Example: Boost the stump

```
###################### Boosting
ex1.tr$y=as.numeric(ex1.tr$y)-1
ex1.te$y=as.numeric(ex1.te$y)-1

ntree=400
## set.seed(0) ## Can set the seed if sampling is used
ex1.boost=gbm(y~.,data=ex1.tr,distribution="adaboost",n.trees=ntree,
                  interaction.depth=1, shrinkage=1, bag.fraction=1)
summary(ex1.boost)
ex1.boost.pred=predict(ex1.boost,newdata=ex1.te[,1:p], n.trees=5)
table(ex1.boost.pred>=0,ex1.te$y)

test.error=rep(0,ntree)
for (i in 1:ntree){
    pp=predict(ex1.boost,newdata=ex1.te[,1:p], n.trees=i)
    pp=pp>=0
    pp=as.numeric(pp)
    test.error[i]=sum(pp!=ex1.te$y)/n.te
}

plot(1:ntree, test.error, type="l", col="orange",
              xlab="Boosting Iterations", ylab="Test Error" ,ylim=c(0,0.5))
abline(h=test.error.tree,lty=2,col="purple")
abline(h=test.error.stump,lty=2,col="gray")
legend("bottomleft", c("AdaBoost","Stump","21 Node Tree"),
                    col=c("orange","gray","purple"), lty=c(1,2,2))
```

Example: Test Errors



Boosting Fits an Additive Model

- Suppose we want to learn a function $f(\boldsymbol{x})$ from the data.
  - $f(\cdot)$ can be a regression function.
  - For a binary classification, the rule can be given by the sign of $f(\cdot)$.
- Express $f(\cdot)$ through the basis expansion

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}; \boldsymbol{\gamma}_m).$$

  - β_m are expansion coefficients.
  - $b(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ is usually a simple function of $\boldsymbol{x}$, governed by a parameter vector $\boldsymbol{\gamma}_m$.
- Examples of the function $b(\boldsymbol{x}; \boldsymbol{\gamma})$:
  - In neural networks, $b(\boldsymbol{x}, \boldsymbol{\gamma}) = \sigma(\gamma_0 + \boldsymbol{\gamma}_1' \boldsymbol{x})$, where $\sigma(\cdot)$ is the sigmoid function.
  - In signal processing, $b(\boldsymbol{x}, \boldsymbol{\gamma})$ are wavelets, where $\boldsymbol{\gamma}$ parameterizes the location and scale shifts from a “mother” wavelet.
  - In nonparametric function estimation, $b(\boldsymbol{x}, \boldsymbol{\gamma})$ are splines, where $\boldsymbol{\gamma}$ depends on the knots.
  - For trees, $\boldsymbol{\gamma}$ parameterizes the split variables and split points at the internal nodes, and the predictions at the terminal nodes.

Forward Stagewise Additive Modeling

- Typically these models are fit by minimizing a loss function averaged over the training data

$$\min_{\{\beta_m, \boldsymbol{\gamma}_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}; \boldsymbol{\gamma}_m)\right).$$

- For many loss functions $L(y, f(\boldsymbol{x}))$ and/or basis functions $b(\boldsymbol{x}; \boldsymbol{\gamma})$, this requires computationally intensive numerical optimization techniques.
- However, a simple alternative often can be found when it is feasible to rapidly solve the subproblem of fitting just a single basis function.
- Forward stagewise modeling approximates the solution by sequentially adding new basis functions to the expansion WITHOUT adjusting the parameters and coefficients of those that have already been added.

Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

```
1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
    (a) Compute
        $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
    (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
```

Regression and Squared Loss

- Consider the squared-error loss

$$L(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2.$$

- For the m-th step,

$$\begin{aligned} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i; \boldsymbol{\gamma}_m)) &= \left(y_i - f_{m-1}(\boldsymbol{x}_i) - \beta b(\boldsymbol{x}_i; \boldsymbol{\gamma}_m)\right)^2 \\ &= \left(r_{im} - \beta b(\boldsymbol{x}_i; \boldsymbol{\gamma}_m)\right)^2 . \end{aligned}$$

  - $r_{im} = y_i - f_{m-1}(\boldsymbol{x}_i)$ is the residual of the current model on the i-th observation.

- Not only used for regression. Will also be used as an intermediate step in the gradient boosting algorithm, even for classification.

AdaBoost.M1.

Algorithm 10.1 *AdaBoost.M1.*

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

Exponential Loss and AdaBoost

- The squared-error loss is generally not a good choice for classification.
- The AdaBoost.M1 (Algorithm 10.1) is equivalent to forward stagewise additive modeling (Algorithm 10.2) using the loss function

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

  - $\boldsymbol{y} \in \{-1, 1\}$.
  - If $y_i = 1$, and $f(\boldsymbol{x}_i) > 0$, then $\exp(-y_i f(\boldsymbol{x}_i))$ is small. And the larger $f(\boldsymbol{x}_i)$ is, the better.
- For the m-the step, we need to fit the classifier $G_m(\boldsymbol{x}) \in \{-1, 1\}$:

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

- Can be re-written as

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp\left[-\beta y_i G(\boldsymbol{x}_i)\right].$$

  - $w_i^{(m)} := \exp(-y_i f_{m-1}(\boldsymbol{x}_i))$ is a weight depending on neither β nor $G(\boldsymbol{x}_i)$.

Exponential Loss and AdaBoost

- The minimization over $G(\cdot)$ does NOT depend on β:

$$G_m = \arg\min_G \sum_{i=1}^N w_i^{(m)} I(y_i \neq G(\boldsymbol{x}_i)).$$

  - We haven't specified what form $G(\cdot)$ takes. It can be, for example, a classfication tree.
- Suppose the best G_m has been obtained, let err_m be the minimized weighted error rate

$$\mathsf{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} I(y_i \neq G_m(\boldsymbol{x}_i))}{\sum_{i=1}^N w_i^{(m)}}.$$

- It can be shown that the optimal β_m is given by

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathsf{err}_m}{\mathsf{err}_m}.$$

- The classifier $f(\cdot)$ is updated as

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m G_m(\boldsymbol{x}).$$

- The weights for the $(m+1)$-th iteration becomes

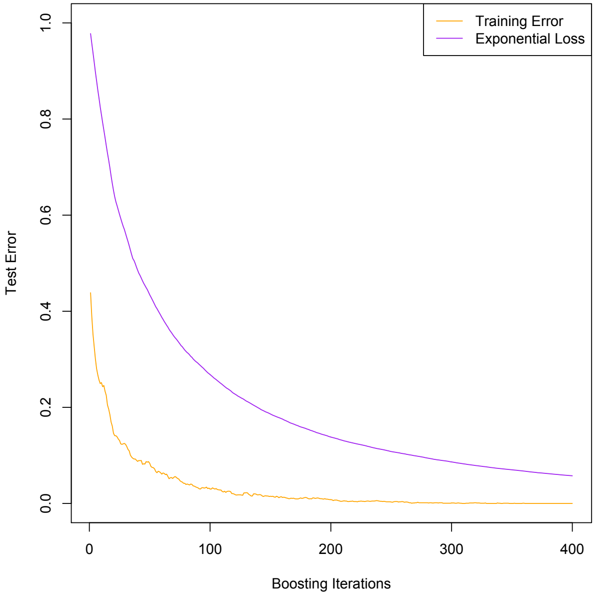
$$w_i^{(m+1)} := w_i^{(m)} \cdot e^{-\beta_m y_i G_m(\boldsymbol{x}_i)} = w_i^{(m)} \cdot e^{\alpha_m I[y_i \neq G_m(\boldsymbol{x}_i)]} \cdot e^{-\beta_m}.$$

Exponential Loss and AdaBoost

- The training-set misclassification error decreases to zero at around 250 iterations (and remains there).
- The exponential loss keeps decreasing.
- The test-set misclassification error continues to improve after iteration 250.
- Adaboost is not optimizing training misclassification error; the exponential loss is more sensitive to changes in the estimated class probabilities.

```
train.error=rep(0,ntree)
exp.loss=rep(0,ntree)
for (i in 1:ntree){
    pp=predict(ex1.boost,newdata=ex1.tr[,1:p], n.trees=i)
    exp.loss[i]=mean(exp(-pp*(2*ex1.tr$y-1)))
    pp=pp>=0
    pp=as.numeric(pp)
    train.error[i]=sum(pp!=ex1.tr$y)/n.tr
}

plot(1:ntree, train.error, type="l", col="orange",
              xlab="Boosting Iterations", ylab="Test Error" ,ylim=c(0,1))
lines(1:ntree,exp.loss,type="l",col="purple")
legend("topright", c("Training Error","Exponential Loss"),
                   col=c("orange","purple"), lty=c(1,1))
```

Example: Training Error and Exponential Loss



Why Exponential Loss?

- Under the conditional distribution of y given $\boldsymbol{x}$

$$f^*(\boldsymbol{x}) := \arg\min_{f(\boldsymbol{x})} \mathbb{E}_{y|\boldsymbol{x}}(e^{-yf(\boldsymbol{x})}) = \frac{1}{2}\log\frac{P(y=1|\boldsymbol{x})}{P(y=-1|\boldsymbol{x})}.$$

- Or equivalently,

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

- The additive expansion produced by AdaBoost is estimating one-half of the log-odds of $P(y=1|\boldsymbol{x})$.
- The binomial negative log-likelihood, or deviance, or cross entropy is

$$-\ell(y, f(\boldsymbol{x})) = \log\left(1+e^{-2yf(\boldsymbol{x})}\right).$$

- It turns out

$$\arg\min_{f(\boldsymbol{x})} \mathbb{E}_{y|\boldsymbol{x}}\left[-\ell(y, f(\boldsymbol{x}))\right] = \arg\min_{f(\boldsymbol{x})} \mathbb{E}_{y|\boldsymbol{x}}(e^{-yf(\boldsymbol{x})}).$$

- The exponential loss and the binomial deviance lead to DIFFERENT results on finite data sets.

Loss Functions for Two-Class Classification

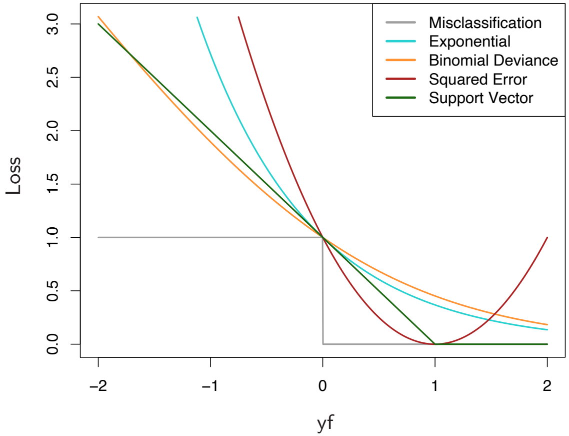


FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\mathrm{sign}(f)$*. The losses are misclassification:* $I(\mathrm{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared 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* $(0, 1)$*.*

Compare Exponential Loss and Deviance Loss

- The classification rule is $G(\boldsymbol{x}) = \operatorname{sign}[f(\boldsymbol{x})]$
  - Observations with positive margin $y_i f(\boldsymbol{x}_i) > 0$ are classified correctly.
  - Those with negative margin $y_i f(\boldsymbol{x}_i) < 0$ are misclassified.
  - The decision boundary is defined by $\{\boldsymbol{x} : \ f(\boldsymbol{x}) = 0\}$.
  - The goal of the classification algorithm is to produce positive margins as frequently as possible.
  - Any loss criterion should penalize negative margins more heavily than positive ones.
- Both the exponential and deviance loss can be viewed as monotone continuous approximations to misclassification loss.
  - The penalty associated with binomial deviance increases linearly for large increasingly negative margin.
  - The exponential criterion increases the influence of such observations exponentially.
- The exponential criterion concentrates much more influence on observations with large negative margins.
- The binomial deviance is more robust in noisy setting.
- The performance of AdaBoost has been empirically observed to dramatically degrade in such situations.

Outline

Boosting Trees

- Regression and classification trees partition the space of all joint predictor variable values into disjoint regions R_j, $j = 1, 2, ..., J$, as represented by the terminal nodes of the tree.
- Let $\Theta = \{R_j, \gamma_j\}_1^J$ be the tree parameters.

$$T(\boldsymbol{x}; \Theta) = \sum_{j=1}^{J} \gamma_j I(\boldsymbol{x} \in R_j).$$

- J is usually treated as a meta-parameter. The regions R_j and the associated γ_j are obtained by minimizing the empirical loss

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{\boldsymbol{x}_i \in R_j} L(y_i, \gamma_j).$$

- This is a formidable combinatorial optimization problem, especially because of the regions.
- The boosted tree model is a sum of such trees

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} T(\boldsymbol{x}; \Theta_m).$$

- Each tree $T(\boldsymbol{x}; \Theta_m)$ is usually simple, e.g. a stump.

Boosting Trees: AdaBoost

- The trees are induced in a forward stagewise manner (Algorithm 10.2).
- At the m-th iteration, the forward stagewise algorithm finds the m-th tree with parameters $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$, given the current $f_{m-1}(\boldsymbol{x})$.

$$\hat{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} L\left[y_i, f_{m-1}(\boldsymbol{x}_i) + T(\boldsymbol{x}_i; \Theta_m)\right].$$

- AdaBoost finds the m-th tree that minimizes the weighted error rate

$$\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq T(\boldsymbol{x}_i; \Theta_m)), \quad \text{where } w_i^{(m)} = \exp(-y_i f_{m-1}(\boldsymbol{x}_i)).$$

- In each region R_{jm}, γ_{jm} is either β_m or $-\beta_m$.
- This is a special kind of tree, which only takes two possible values on different regions. It's called a scaled classification tree.
- This is because we made the requirement that $G_m(\boldsymbol{x}) \in \{-1, 1\}$ at the beginning.

Boosting: General Case

- Without this restriction, the optimal constant γ_{jm} for the region R_{jm} is given by (typically easy to obtain)

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma_{jm}).$$

- Under the exponential loss,

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp[-y_i T(\boldsymbol{x}_i; \Theta_m)].$$

- Given R_{jm}, the parameters γ_{jm} are then given by

$$\hat{\gamma}_{jm} = \frac{1}{2} \log \frac{\sum_{\boldsymbol{x}_i \in R_{jm}} w_i^{(m)} I(y_i = 1)}{\sum_{\boldsymbol{x}_i \in R_{jm}} w_i^{(m)} I(y_i = -1)}.$$

- Finding the region requires a specialized tree-growing algorithm. In practice, we prefer the approximation presented below that uses a weighted least squares regression tree.

Gradient Boosting

- An approximate algorithm for adding a tree with any differentiable loss criterion.
- The loss of using $f(\boldsymbol{x})$ to predict y on the training set is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i)).$$

- The goal is to minimize $L(f)$ with respect to f, where here $f(x)$ is constrained to be a sum of trees.
- Let $\boldsymbol{f} = [f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_N)]'$. Then the problem becomes

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} L(\boldsymbol{f}).$$

- Using gradient descent, we try to reach or approximate $\hat{\boldsymbol{f}}$ by

$$\boldsymbol{f}_M = \sum_{m=0}^{M} \boldsymbol{h}_m, \quad \boldsymbol{h}_m \in \mathbb{R}^N.$$

  - $\boldsymbol{f}_0 = \boldsymbol{h}_0$ is an initial guess.
  - Each $\boldsymbol{h}_m$ is induced based on the current $\boldsymbol{f}_{m-1}$.

Fit a Regression Tree to the Gradient

- Given the current $\boldsymbol{f}_{m-1}$, steepest descent (gradient descent) seeks to update $\boldsymbol{f}$ along the direction $\boldsymbol{g}_m$, where

$$g_{im} = \left. \frac{\partial L(y_i, f_i)}{\partial f_i} \right|_{f_i = f_{m-1}(\boldsymbol{x}_i)}$$

- For optimization, can take $\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m$The scalar ρ_m is determined by

$$\rho_m = \arg\min_{\rho} L(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m).$$

- However, we need a $\boldsymbol{f}_m$ that is able to make predictions on new data.
- Induce a regression tree $T(\boldsymbol{x}; \Theta_m)$ by

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} [-g_{im} - T(\boldsymbol{x}_i; \Theta)]^2.$$

- After finding the regions from $\tilde{\Theta}_m$, the constants in each region are

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma_{jm}).$$

- Now update $\boldsymbol{f}_m(\boldsymbol{x}) = \boldsymbol{f}_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$.

Gradient Tree Boosting Algorithm

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm},\ j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Tree Size

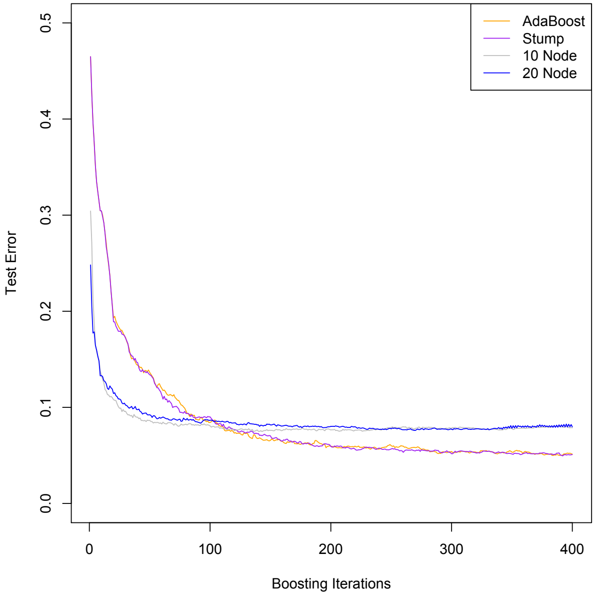
- The simplest strategy is to restrict all trees to be of the same size J.
- No interaction effects of level greater than $J - 1$ are possible.
  - Setting $J = 2$ produces boosted models with only main effects.
  - With $J = 3$, two-variable interaction effects are also allowed.
  - The `gbm()` function has a parameter `interaction.depth` for the allowed level of interactions.
- Experience so far indicates that $4 \leq J \leq 8$ works well in the context of boosting, with results being fairly insensitive to particular choices in this range.

```
error.rate=function(m,newdata,ntree){
err=array(0,c(3,ntree))
rownames(err)=c("Mis","Exp","Dev")
for (i in 1:ntree){
    p=dim(newdata)[2]-1
    pp=predict(m,newdata=newdata[,1:p], n.trees=i)
    err[2,i]=mean(exp(-pp*(2*newdata$y-1)))
    err[3,i]=mean(log(1+exp(-2*pp*(2*newdata$y-1))))
    pp=pp>=0
    pp=as.numeric(pp)
    err[1,i]=mean(pp!=newdata$y)
}
err
}
```

Tree Size

```
############## Number of Nodes
m0=gbm(y~.,data=ex1.tr,distribution="adaboost",n.trees=ntree,
                       interaction.depth=1, shrinkage=1, bag.fraction=1)
m1=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
                       interaction.depth=1, shrinkage=1, bag.fraction=1)
m2=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
                       interaction.depth=9, shrinkage=1, bag.fraction=1)
m3=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
                       interaction.depth=19, shrinkage=1, bag.fraction=1)

err0=error.rate(m0,newdata=ex1.te,ntree=ntree)
err1=error.rate(m1,newdata=ex1.te,ntree=ntree)
err2=error.rate(m2,newdata=ex1.te,ntree=ntree)
err3=error.rate(m3,newdata=ex1.te,ntree=ntree)

par(mar=c(4.5,4.5,.5,.4))
plot(1:ntree, err0[1,], type="l", col="orange", xlab="Boosting Iterations",
                        ylab="Test Error" ,ylim=c(0,0.5))
lines(1:ntree,err1[1,],type="l",col="purple")
lines(1:ntree,err2[1,],type="l",col="gray")
lines(1:ntree,err3[1,],type="l",col="blue")
legend("topright", c("AdaBoost","Stump","10 Node", "20 Node"),
                   col=c("orange","purple","gray","blue"), lty=rep(1,4))
```

Tree Size



Test Error
0.0
0.1
0.2
0.3
0.4
0.5
Boosting Iterations
0
100
200
300
400
AdaBoost
Stump
10 Node
20 Node

Regularization: Shrinkage

- As with ridge regression and neural networks, shrinkage techniques can be employed as well.
- Scale the contribution of each tree by a factor $0 < \nu \leq 1$, i.e. line 2(d) of Algorithm 10.3 is replaced by

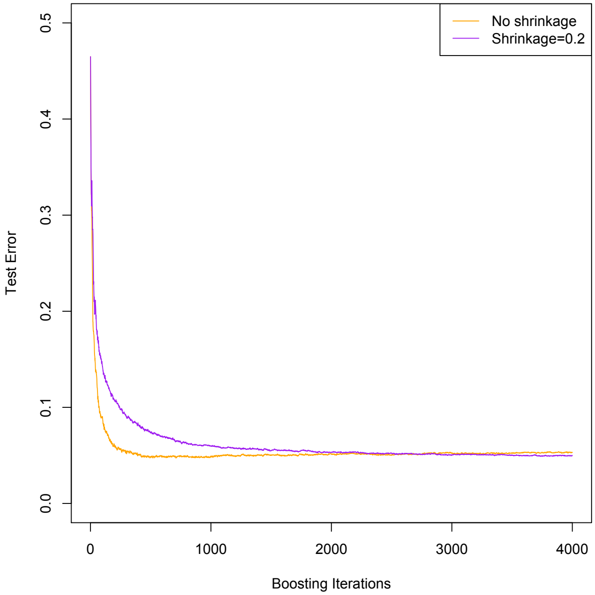
$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} I(\boldsymbol{x} \in R_{jm}).$$

- Smaller values of ν favor better test error, and require correspondingly larger values of M.
- The best strategy appears to be to set ν to be very small ($\nu < 0.1$) and then choose M by early stopping.
- The `gbm()` function has a parameter `shrinkage`, with default value 0.1.
- Example: the benefits of shrinkage are evident, especially when the binomial deviance is tracked.

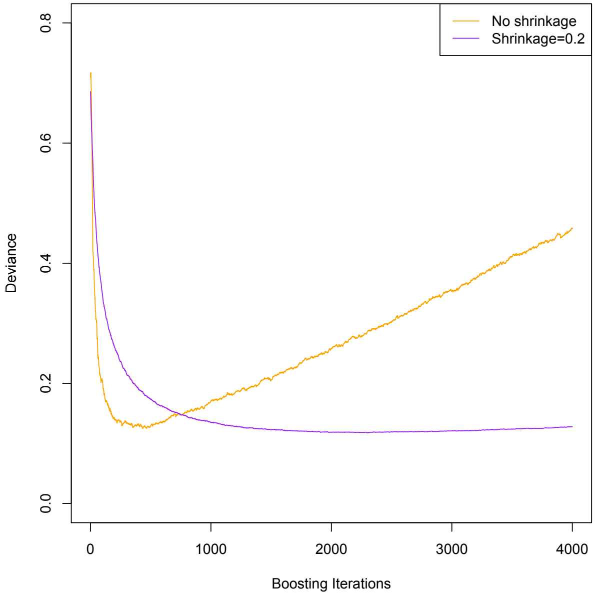
Regularization: Shrinkage

```
###### Stump
###### Set interaction.depth=5 to experiment with the trees of 6 nodes
ntree=4000
m1=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1, shrinkage=1, bag.fraction=1)
err1=error.rate(m1,newdata=ex1.te,ntree=ntree)
m4=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1, shrinkage=.2, bag.fraction=1)
err4=error.rate(m4,newdata=ex1.te,ntree=ntree)

par(mar=c(4.5,4.5,.5,.4))
### Misclassfication Error
plot(1:ntree, err1[1,], type="l", col="orange", xlab="Boosting Iterations",
                        ylab="Test Error" ,ylim=c(0,0.5))
lines(1:ntree,err4[1,],type="l",col="purple")
legend("topright", c("No shrinkage","Shrinkage=0.2"),
                   col=c("orange","purple"), lty=rep(1,2))
### Deviance
plot(1:ntree, err1[3,], type="l", col="orange", xlab="Boosting Iterations",
                        ylab="Deviance" ,ylim=c(0,0.8))
lines(1:ntree,err4[3,],type="l",col="purple")
legend("topright", c("No shrinkage","Shrinkage=0.2"),
                   col=c("orange","purple"), lty=rep(1,2))
```

Shrinkage: Stump



Shrinkage: Stump

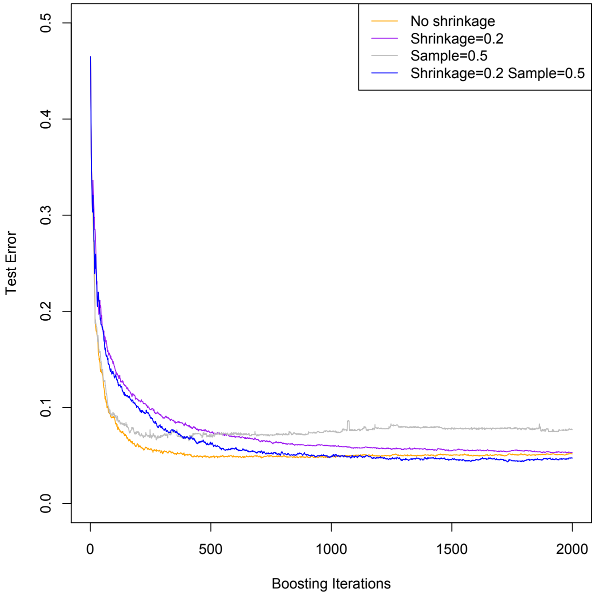


Regularization: Subsampling

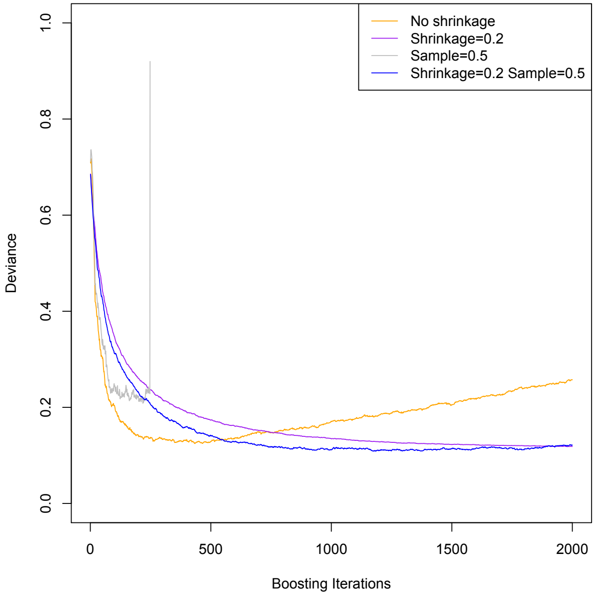
- With stochastic gradient boosting, at each iteration we sample a fraction η of the training observations (without replacement), and grow the next tree using that subsample.
- A typical value for η can be 0.5.
- For large N, η can be substantially smaller than 0.5.
- The `gbm()` function has a parameter `bag.fraction`, with default value 0.5
- Subsampling can reduce the computing time.
- In many cases it actually produces a more accurate model.

```
ntree=2000
m1=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1, shrinkage=1, bag.fraction=1)
err1=error.rate(m1,newdata=ex1.te,ntree=ntree)
m7=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1, shrinkage=0.2, bag.fraction=1)
err7=error.rate(m7,newdata=ex1.te,ntree=ntree)
m8=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1, shrinkage=1, bag.fraction=0.5)
err8=error.rate(m8,newdata=ex1.te,ntree=ntree)
m9=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1, shrinkage=0.2, bag.fraction=0.5)
err9=error.rate(m9,newdata=ex1.te,ntree=ntree)

### Misclassification Error
plot(1:ntree, err1[1,], type="l", col="orange", xlab="Boosting Iterations",
                        ylab="Test Error" ,ylim=c(0,0.5))
lines(1:ntree,err7[1,],type="l",col="purple")
lines(1:ntree,err8[1,],type="l",col="gray")
lines(1:ntree,err9[1,],type="l",col="blue")
legend("topright",
       c("No shrinkage","Shrinkage=0.2","Sample=0.5", "Shrinkage=0.2 Sample=0.5"),
       col=c("orange","purple","gray","blue"), lty=rep(1,4))

### Deviance
plot(1:ntree, err1[3,], type="l", col="orange", xlab="Boosting Iterations",
                        ylab="Deviance" ,ylim=c(0,0.8))
lines(1:ntree,err7[3,],type="l",col="purple")
lines(1:ntree,err8[3,],type="l",col="gray")
lines(1:ntree,err9[3,],type="l",col="blue")
legend("topright",
       c("No shrinkage", "Shrinkage=0.2", "Sample=0.5", "Shrinkage=0.2 Sample=0.5"),
       col=c("orange","purple","gray","blue"), lty=rep(1,4))
```

Shrinkage and Subsampling



Shrinkage and Subsampling



Relative Importance of Predictor Variables

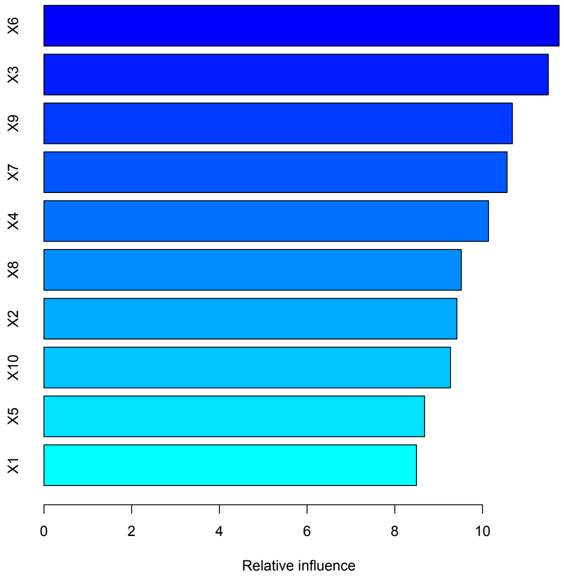
- Often only a few of the predictors have substantial influence on the response; the vast majority are irrelevant and could just as well have not been included.
- For a single decision tree T with J nodes, Breiman et al. proposed

$$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 I[v(t) = \ell]$$

  as a measure of relevance for each predictor variable X_ℓ.
  - The sum is over the $J-1$ internal nodes of the tree.
  - At each internal node t, one of the input variables $X_{v(t)}$ is used to partition the region, which leads to the maximal improvement $\hat{\imath}_t^2$ in squared error loss.
- For a additive tree, the relative importance is defined as

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T).$$

Relative Importance

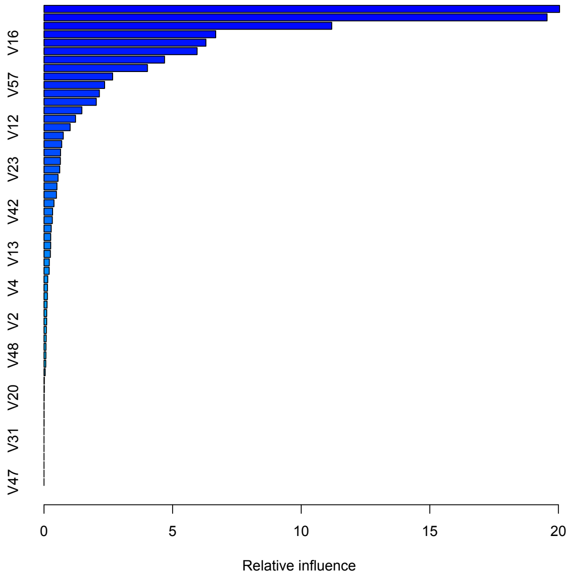


Example: Spam Data

```
Sources:
   (a) Creators: Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt
        Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304
   (b) Donor: George Forman (gforman at nospam hpl.hp.com)  650-857-7835
   (c) Generated: June-July 1999

-- Number of Instances: 4601 (1813 Spam = 39.4%)
-- The spam_ind.txt gives the label of training (0) and testing (0) data.
-- The last column of denotes whether the e-mail was considered spam (1)
   or not (0).
-- Number of Attributes (predictor variables): 57
-- 48 continuous real [0,100] attributes of type word_freq_WORD =
   percentage of words in the e-mail that match WORD.
-- 6 continuous real [0,100] attributes of type char_freq_CHAR =
   percentage of characters in the e-mail that match CHAR.
-- 1 continuous real [1,...] attribute of type
   capital_run_length_average = average length of uninterrupted
   sequences of capital letters.
-- 1 continuous integer [1,...] attribute of type
   capital_run_length_longest = length of longest uninterrupted
   sequence of capital letters.
-- 1 continuous integer [1,...] attribute of type
   capital_run_length_total = total number of capital letters in the e-mail.
```

```
###################### Example: Spam
spam=read.table("spam.txt",header=F)
colnames(spam)[58]="Y"
ind=scan("spam_ind.txt")
spam.tr=subset(spam,ind==0) ## size: 3065
dim(spam.tr)
spam.te=subset(spam,ind==1) ## size: 1536
dim(spam.te)

ntree=400
m1=gbm(Y~.,data=spam.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=1,shrinkage=.1,bag.fraction=.5)
spam.pred=predict(m1,newdata=spam.te, n.trees=ntree)
table(spam.pred>=0,spam.te$Y) ## .054

ntree=400
m2=gbm(Y~.,data=spam.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=3,shrinkage=.1,bag.fraction=.5)
spam.pred=predict(m2,newdata=spam.te, n.trees=ntree)
table(spam.pred>=0,spam.te$Y) ## .050

ntree=800
m3=gbm(Y~.,data=spam.tr,distribution="bernoulli",n.trees=ntree,
           interaction.depth=4,shrinkage=.1,bag.fraction=.5)
spam.pred=predict(m3,newdata=spam.te, n.trees=ntree)
table(spam.pred>=0,spam.te$Y) ## .042
summary(m3)
```

Spam: Relative Importance


V16
V57
V12
V23
V42
V13
V4
V2
V48
V20
V31
V47
Relative influence