Overview

1. AdaBoost
   - Forward Stagewise Additive Modeling
   - The Loss Function

2. Selecting a Loss Function
   - Classification
   - Regression

3. Boosting Trees
   - Brief Background on CART

4. Boosting Trees

5. Tuning and Metaparameter Values
   - Tree Size
   - Regularization

6. Implementation in R
AdaBoost
- Forward Stagewise Additive Modeling
- The Loss Function

Selecting a Loss Function

Boosting Trees

Gradient Boosting

Tuning and Metaparameter Values

Implementation in R
AdaBoost

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, $X$, a classifier $G(X)$ produces a prediction taking one of the two values of $Y$.

1 A weak classifier is one that performs only slightly better than random guessing.
AdaBoost

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, $X$, a classifier $G(X)$ produces a prediction taking one of the two values of $Y$.
- We begin with a "weak" classifier fit to the data.\(^1\)

---

\(^1\) A weak classifier is one that performs only slightly better than random guessing.
Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, $X$, a classifier $G(X)$ produces a prediction taking one of the two values of $Y$.
- We begin with a ”weak” classifier fit to the data.\(^1\)
- We then reweight the observations so that those that were misclassified in this round, $y_i \neq G(x_i)$ are upweighted.

\(^1\)A weak classifier is one that performs only slightly better than random guessing.
AdaBoost

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, \( Y \in \{-1, 1\} \) and a vector of predictor variables, \( X \), a classifier \( G(X) \) produces a prediction taking one of the two values of \( Y \).
- We begin with a ”weak” classifier fit to the data.\(^1\)
- We then reweight the observations so that those that were misclassified in this round, \( y_i \neq G(x_i) \), are upweighted.
- We then fit a new classifier and repeat this \( M \) times.

\(^1\)A weak classifier is one that performs only slightly better than random guessing.
Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, $X$, a classifier $G(X)$ produces a prediction taking one of the two values of $Y$

- We begin with a "weak" classifier fit to the data.\(^1\)

- We then reweight the observations so that those that were misclassified in this round, $y_i \neq G(x_i)$ are upweighted

- We then fit a new classifier and repeat this $M$ times.

- The final classification is given by a weighted vote of all classifiers, with those $G_m(x)$ that are more accurate receiving higher weights.

---

\(^1\)A weak classifier is one that performs only slightly better than random guessing.
AdaBoost, Visually

Figure 1: (Hastie et al. 2009:338)
AdaBoost, More Formally

1. Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \ldots, N$
AdaBoost, More Formally

1. Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, ..., N$
2. For $m = 1$ to $M$:

   - Compute the error of the classifier as $\text{err}_m = \frac{\sum_{i=1}^{N} w_i 1(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
   - Compute $\alpha_m = \log \frac{1 - \text{err}_m}{\text{err}_m}$
   - Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot G_m(x_i)]$
AdaBoost, More Formally

1. Initialize the observation weights \( w_i = \frac{1}{N} \), for \( i = 1, 2, \ldots, N \)

2. For \( m = 1 \) to \( M \):
   - Fit a classifier \( G_m(x) \) to the training data using weights \( w_i \)
AdaBoost, More Formally

1. Initialize the observation weights \( w_i = \frac{1}{N} \), for \( i = 1, 2, \ldots, N \)

2. For \( m = 1 \) to \( M \):
   - Fit a classifier \( G_m(x) \) to the training data using weights \( w_i \)
   - Compute the error of the classifier as

\[
err_m = \frac{\sum_{i=1}^{N} w_i 1(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}
\]
1. Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, ..., N$

2. For $m = 1$ to $M$:
   - Fit a classifier $G_m(x)$ to the training data using weights $w_i$
   - Compute the error of the classifier as
     \[
     err_m = \frac{\sum_{i=1}^{N} w_i 1(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}
     \]
   - Compute $\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$

Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
AdaBoost, More Formally

1. Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \ldots, N$

2. For $m = 1$ to $M$:
   - Fit a classifier $G_m(x)$ to the training data using weights $w_i$
   - Compute the error of the classifier as
     $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$
   - Compute $\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right)$
   - Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot G_m(x_i)]$
AdaBoost, More Formally

1. Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \ldots, N$

2. For $m = 1$ to $M$:
   - Fit a classifier $G_m(x)$ to the training data using weights $w_i$
   - Compute the error of the classifier as
     \[
     \text{err}_m = \frac{\sum_{i=1}^N w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}
     \]
   - Compute $\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right)$
   - Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot G_m(x_i)]$

3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$
AdaBoost, Visually in More Detail

![Diagram of AdaBoost iterations]

- $D_1$:
  - Points: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$
  - Target Classes: $+$, $-$

- $h_1$:
  - Points: $+$, $+$, $-$, $+$
  - Target Classes: $+$, $-$

This diagram illustrates the first iteration of AdaBoost, showing how the distribution $D$ is weighted and how the hypothesis $h_1$ is constructed.
AdaBoost, Visually in More Detail

$D_1$

$D_2$

$D_3$

$h_1$

$h_2$

$h_3$
AdaBoost, Visually in More Detail

\[ H = \text{sign}(0.42 + 0.65 + 0.92) \]

\[
\begin{array}{c|c|c}
+ & + & - \\
- & + & - \\
+ & - & - \\
\end{array}
\]
1. AdaBoost
   - Forward Stagewise Additive Modeling
   - The Loss Function

2. Selecting a Loss Function

3. Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
What is AdaBoost doing and how can we generalize it to other classification problems and to regression?
What is AdaBoost doing and how can we generalize it to other classification problems and to regression?

It turns out that AdaBoost implements Forward Stagewise Additive Modeling (FSAM) using an exponential loss function. Let’s take these one at a time.
Initialize $f_0(x) = 0$
Forward Stagewise Additive Modeling

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

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

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

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

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

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

At each iteration, we fit the optimal basis function and corresponding coefficient \( \beta_m \) to add to the current expansion, \( f_{m-1}(x) \). We do not update the parameters of previously estimated functions.
1. AdaBoost
   - Forward Stagewise Additive Modeling
   - The Loss Function

2. Selecting a Loss Function

3. Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
AdaBoost minimizes an exponential loss function. But let’s look at the more familiar case of squared-error loss familiar from linear regression

\[
L(y, f(x)) = (y - f(x))^2
\]

\[
L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2
= (r_{im} - \beta b(x_i; \gamma))^2
\]
AdaBoost minimizes an exponential loss function. But let’s look at the more familiar case of squared-error loss familiar from linear regression

\[ L(y, f(x)) = (y - f(x))^2 \]

\[ L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 = (r_{im} - \beta b(x_i; \gamma))^2 \]

So the basis function that best fits the residuals from the last iteration will minimize the loss and be added to the model.
1. AdaBoost

2. Selecting a Loss Function
   - Classification
   - Regression

3. Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
Selecting a Loss Function

- Will depend on the problem at hand.
1. AdaBoost

2. Selecting a Loss Function
   - Classification
   - Regression

3. Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
In the binary classification situation outlined above, the margin, $yf(x)$, plays a role analogous to the residuals $y - f(x)$ in regression. Positive margins are classified correctly; negative margins are misclassified.

We want a loss function that penalizes negative margins more heavily than positive ones.
Loss Functions for Classification

Figure 2: (Hastie et al. 2009:347)
1. AdaBoost

2. Selecting a Loss Function
   - Classification
   - Regression

3. Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
Loss Functions for Regression

- In the regression setting we can look at squared error loss (seen above) or absolute loss, \( L(y, f(x)) = |y - f(x)| \)
- We can also use the Huber loss criterion used in robust regression to address the strengths of each
Figure 3: (Hastie et al. 2009:350)
AdaBoost

Selecting a Loss Function

Boosting Trees
  - Brief Background on CART
  - Boosting Trees

Gradient Boosting

Tuning and Metaparameter Values

Implementation in R
Let’s boost some trees!
1. AdaBoost

2. Selecting a Loss Function

3. Boosting Trees
   - Brief Background on CART
   - Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
What is a Classification/Regression Tree?

- Trees partition the space of all joint predictor variable values into disjoint regions $R_j$ as represented by the terminal nodes of the tree. A constant $\gamma_j$ is assigned to each such region and the predictive rule is $x \in R_j \Rightarrow f(x) = \gamma_j$
Trees partition the space of all joint predictor variable values into disjoint regions $R_j$ as represented by the terminal nodes of the tree. A constant $\gamma_j$ is assigned to each such region and the predictive rule is $x \in R_j \Rightarrow f(x) = \gamma_j$

So a tree can be formally expressed as:

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

with $\Theta = \{R_j, \gamma_j\}_{1}^{J}$
What is a Classification/Regression Tree

Figure 4: (Murphy 2012:545)
1. AdaBoost

2. Selecting a Loss Function

3. Boosting Trees
   - Brief Background on CART
   - Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
The boosted tree model is a sum of trees

\[ f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \]
The boosted tree model is a sum of trees

\[ f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \]

We will grow our trees in a FSAM. At each step we must solve:

\[ \hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \]
The boosted tree model is a sum of trees

\[ f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \]

We will grow our trees in a FSAM. At each step we must solve:

\[ \hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \]

Finding the optimal constants \( \gamma_{jm} \) is easy if we know the regions \( R_{jm} \). But finding the regions is difficult.
Two cases where the problem simplifies

1. For **squared-error loss** the solution will be to grow a tree that best predicts the current residuals, and $\hat{\gamma}_{jm}$ is the mean of the residuals in each corresponding region.

But recall that neither exponential nor squared-error loss are robust. Choosing other loss criteria, though, make the solution to the problem more difficult.
Two cases where the problem simplifies

1. For **squared-error loss** the solution will be to grow a tree that best predicts the current residuals, and $\hat{\gamma}_{jm}$ is the mean of the residuals in each corresponding region.

2. For **two-class classification with exponential loss** then (under certain conditions) this gives rise to the AdaBoost method for boosting classification trees. In general, the $\hat{\gamma}_{jm}$ will be the weighted log-odds in each corresponding region.

But recall that neither exponential nor squared-error loss are robust. Choosing other loss criteria, though, make the solution to the problem more difficult.
1. AdaBoost

2. Selecting a Loss Function

3. Boosting Trees

4. Gradient Boosting
   - Steepest Descent
   - Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
Gradient Boosting

In order to solve this complex problem (for any differentiable loss function) we will implement a solution known as gradient boosting.
1 AdaBoost

2 Selecting a Loss Function

3 Boosting Trees

4 Gradient Boosting
   - Steepest Descent
   - Gradient Boosting

5 Tuning and Metaparameter Values

6 Implementation in R
The solution we will implement is analogous to the steepest descent numerical optimization procedure.

- At any point in the procedure we evaluate the gradient \( (g_m) \) of the function \( L(f) \) at the last update:

\[
g_{im} = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i) = f_{m-1}(x_i)}
\]
Steepest Descent

The solution we will implement is analogous to the steepest descent numerical optimization procedure.

- At any point in the procedure we evaluate the gradient \((g_m)\) of the function \(L(f)\) at the last update:
  \[
g_{im} = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i) = f_{m-1}(x_i)}
\]

- and take a step of length \(\rho_m\), which is the solution to
  \[
  \rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m)
  \]
Steepest Descent

The solution we will implement is analogous to the steepest descent numerical optimization procedure.

- At any point in the procedure we evaluate the gradient \((g_m)\) of the function \(L(f)\) at the last update:

\[
g_{im} = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i) = f_{m-1}(x_i)}
\]

- and take a step of length \(\rho_m\), which is the solution to

\[
\rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m)
\]

- We then update by subtracting \(\rho_m g_m\) from the previous update.
1. AdaBoost

2. Selecting a Loss Function

3. Boosting Trees

4. Gradient Boosting
   - Steepest Descent
   - Gradient Boosting

5. Tuning and Metaparameter Values

6. Implementation in R
Steepest descent chooses the direction in which the function is most rapidly decreasing
Steepest descent chooses the direction in which the function is most rapidly decreasing.

We would like to do the same thing but here our solution must be a tree. Also importantly, we don’t want to simply minimize loss on the training set but generalize to new data.
Gradient Boosting

- Steepest descent chooses the direction in which the function is most rapidly decreasing.
- We would like to do the same thing but here our solution must be a tree. Also importantly, we don’t want to simply minimize loss on the training set but generalize to new data.
- A potential solution is to induce a tree at the $m$th iteration whose predictions $t_m$ are as close as possible to the negative gradient

$$
\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2
$$
1 Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
Gradient Boosting

1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

2. For $m = 1$ to $M$:

   - Compute $r_{im} = -\left[ \frac{\partial L(y_i, f_m(x_i))}{\partial f(x_i)} \right]$
   - Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, ..., J_m$
   - For $j = 1, 2, ..., J_m$ compute $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(Y_i - f_{m-1}(x_i) + \gamma)$
   - Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} 1_{x \in R_{jm}}$
Gradient Boosting

1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

2. For $m = 1$ to $M$:
   - For $i = 1, 2, ..., N$ compute
     
     $$r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

   - Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, ..., J_m$
   - For $j = 1, 2, ..., J_m$ compute
     
     $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(Y - y_i, f_{m-1}(x_i) + \gamma)$$

   - Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} 1_{x \in R_{jm}}$
Gradient Boosting

1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

2. For $m = 1$ to $M$:
   - For $i = 1, 2, ..., N$ compute
     \[
     r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}
     \]
   - Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}, j = 1, 2, ..., J_m$
Gradient Boosting

1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

2. For $m = 1$ to $M$:
   - For $i = 1, 2, \ldots, N$ compute
     \[
     r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}
     \]
   - Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$
   - For $j = 1, 2, \ldots, J_m$ compute
     \[
     \gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(Y - i, f_{m-1}(x_i) + \gamma)
     \]
Gradient Boosting

1. Initialize \( f_0(x) = \arg \min \gamma \sum_{i=1}^{N} L(y_i, \gamma) \)

2. For \( m = 1 \) to \( M \):
   - For \( i = 1, 2, \ldots, N \) compute
     \[
     r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}
     \]
   - Fit a regression tree to the targets \( r_{im} \) giving terminal regions \( R_{jm}, j = 1, 2, \ldots, J_m \)
   - For \( j = 1, 2, \ldots, J_m \) compute
     \[
     \gamma_{jm} = \arg \min \gamma \sum_{x_i \in R_{jm}} L(Y - i, f_{m-1}(x_i) + \gamma)
     \]
   - Update \( f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm}) \)

Output \( \hat{f}(x) = f_M(x) \)
Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

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

Output $\hat{f}(x) = f_M(x)$
AdaBoost

Selecting a Loss Function

Boosting Trees

Gradient Boosting

Tuning and Metaparameter Values
- Tree Size
- Regularization

Implementation in R
Tuning and Metaparameters

There are a number of parameters for the algorithm that we might be concerned about setting

- $J_m$, the number of terminal nodes in each tree
- $M$, the number of boosting iterations
- $\nu$, a shrinkage parameter
- $\eta$, fraction of training observations to select at each iteration (stochastic gradient boosting)
1. AdaBoost

2. Selecting a Loss Function

3. Boosting Trees

4. Gradient Boosting

5. Tuning and Metaparameter Values
   - Tree Size
   - Regularization

6. Implementation in R
Tree Size, $J$

- Set $J_m = J \forall m$
- Selection of $J$ will affect the number of interactions you allow in your model as the interaction order for any tree is given by $J - 1$.
- HT&F recommend $J \simeq 6$
1. AdaBoost
2. Selecting a Loss Function
3. Boosting Trees
4. Gradient Boosting
5. Tuning and Metaparameter Values
   - Tree Size
   - Regularization
6. Implementation in R
In setting the number of iterations we want to run enough to maximally reduce error on the test sample but not so much that we overfit to the training sample.

This implies some optimal $M = M^*$ that HT&F recommend finding using an early stopping strategy.
Regularization: Shrinkage, $\nu$

- The simplest implementation of shrinkage is to scale the contributions of each tree by a factor $0 < \nu < 1$
- Because smaller values of $\nu$ imply a slower learning, there is a tradeoff between $M$ and $\nu$
- HT&F suggest that the best results are found with $\nu < 0.1$
Regularization: Shrinkage, $\nu$

Figure 5: Hastie et al. 2009:366
Regularization: Subsampling, $\eta$

- Bootstrap averaging (bagging) can improve the performance of a noisy classifier. We can apply a similar logic here.
- At each iteration we sample without replacement some fraction $\eta$ of the training observations. A typical value is $\eta = 0.5$. This reduces computational effort while also (often) improving accuracy.
Figure 6: Hastie et al. 2009:367
1. AdaBoost
2. Selecting a Loss Function
3. Boosting Trees
4. Gradient Boosting
5. Tuning and Metaparameter Values
6. Implementation in R
- `xgboost` implements gradient boosting in R
- The `xgboost` command allows you to control the depth of trees, regularization, subsampling, and the number of rounds of boosting.
- It also creates visualizations of variable importance plots.