Boosting Methods
• Boosting is a procedure to combine the output of many weak classifiers to produce a powerful committee.

• A weak classifier is one whose error rate is only slightly better than random guessing.

• Boosting produces a sequence of weak classifiers $G_m(x)$ for $m = 1, \ldots, M$ whose predictions are then combined through a weighted majority to produce the final prediction:

$$G(x) = \text{sgn} \left( \sum_{m=1}^{M} \alpha_m G_m(x) \right)$$

• Each $\alpha_m > 0$ is computed by the boosting algorithm and reflects how accurately $G_m$ classified the data.
Most popular boosting algorithm: AdaBoost.M1


- Have training data \((x_i, y_i), i = 1, 2, \ldots, n\)
- Introduce a weight \(w_i = 1/n\) for each training example.
- for \(m = 1, \ldots, M\)
  - Let \(G_m\) be the weak classifier with minimum error:
    \[
    err_m = \sum_{i=1}^{n} w_i \text{Ind}(y_i \neq G_m(x_i))
    \]
  - Set \(\alpha_m = \log((1 - err_m)/err_m)\).
  - Set
    \[
    w_i \leftarrow w_i e^{\alpha_m \text{Ind}(y_i \neq G_m(x_i))}
    \]
    for \(i = 1, \ldots, n\)
  - This increases (decreases) \(w_i\) for \(x_i\) misclassified (correctly classified) by \(G_M\)
  - Normalize the \(w_i\)'s so that they sum to one.
What AdaBoost.M1 does

- As iterations proceed, observations difficult to classify correctly receive more influence.

- Each successive classifier is forced to concentrate on training observations missed by previous ones in the sequence.
$\mathcal{H}$ is the set of all possible oriented vertical and horizontal lines.
Round 1

Chosen weak classifier

\( \epsilon_1 = 0.19, \alpha_1 = 1.45 \)

Re-weight training points

\( w_i^{(2)} \)’s

Current strong classifier

\( G'(x) \)
Round 2

Chosen weak classifier

$\epsilon_2 = 0.1512, \alpha_2 = 1.725$

Re-weight training points

$w_i^{(3)}$'s

Current strong classifier

$G(x)$
Round 3

Chosen weak classifier

$\epsilon_3 = 0.2324, \alpha_3 = 1.1946$

Re-weight training points

$w_i^{(4)}$'s

Current strong classifier

$G(x)$
Round 4

Chosen weak classifier
\[ \epsilon_4 = 0.2714, \alpha_4 = 0.9874 \]

Re-weight training points
\[ w_i^{(5)}'s \]

Current strong classifier
\[ G(x) \]
Round 5

Chosen weak classifier

\[ \epsilon_5 = 0.2616, \alpha_5 = 1.0375 \]

Re-weight training points

\[ w_i^{(6)}, s \]

Current strong classifier

\[ G(x) \]
Round 6

Chosen weak classifier
\[ \epsilon_6 = 0.2262, \alpha_6 = 1.2298 \]

Re-weight training points
\[ w_i^{(7)} \text{'s} \]

Current strong classifier
\[ G(x) \]
Round 7

Chosen weak classifier

$\epsilon_7 = 0.2680$, $\alpha_7 = 1.0049$

Re-weight training points

$w_i^{(8)}$, $s$

Current strong classifier

$G(x)$
Round 8

Chosen weak classifier

\[ \epsilon_8 = 0.3282, \alpha_8 = 0.7165 \]

Re-weight training points

\[ w_i^{(9)}, s \]

Current strong classifier

\[ G(x) \]
Round 9

Chosen weak classifier

$\epsilon_9 = 0.3048$, $\alpha_9 = 0.8246$

Re-weight training points

$w_i^{(10)}$, $s$

Current strong classifier

$G(x)$
Round 10

Chosen weak classifier

\( \epsilon_{10} = 0.2943, \alpha_{10} = 0.8744 \)

Re-weight training points

\( w_{i}^{(11)} \cdot s \)

Current strong classifier

\( G(x) \)
Example

Round 11

Chosen weak classifier

$\epsilon_{11} = 0.2876$, $\alpha_{11} = 0.9071$

Re-weight training points

$w_{i}^{(12)}$, $s$

Current strong classifier

$G(x)$
Round 21

Chosen weak classifier

\[ \epsilon_{21} = 0.3491, \alpha_{21} = 0.6232 \]

Re-weight training points

\[ w_i^{(22)}, s \]

Current strong classifier

\[ G(x) \]
AdaBoost can dramatically increases the performance of very weak classifier.
• Show AdaBoost fits an additive model in a base learner, optimizing a novel exponential loss function

• Show the population minimizer of the exponential loss function is the log-odds of the class probabilities

• Present loss functions that are more robust than squared error or exponential loss

• Argue decision trees are an ideal base learner for data mining applications of boosting.

• Develop class of gradient boosted models (GBMs), for boosting trees with any loss function.

• Emphasize the importance of “slow learning”.
Boosting Fits an Additive Model
• Boosting fits an additive expansion in a set of elementary *basis functions*.

\[ G(x) = \text{sgn} \left( \sum_{m=1}^{M} \alpha_m G_m(x) \right) \]

• The basis functions are the weak classifiers \( G_m(x) \in \{-1, 1\} \).

• More generally, basis function expansions take the form

\[ f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \]

where \( \beta_m \)'s are the expansion coefficients and \( b(x; \gamma) \in \mathbb{R} \) are simple functions of the input \( x \) parameterized by \( \gamma \).
Examples of additive models

- **Single-hidden-layer neural networks** where

  \[ b(x; \gamma) = \frac{1}{1 + \exp(-\gamma_0 - \gamma_1^t x)} \]

- **Multivariate adaptive regression splines (MARS)**
  Use truncated-power spline basis functions where \( \gamma \) parameterizes the variables and values for the knots.

- **Trees**
  \( \gamma \) parameterizes the split variables and split points at the internal nodes, and the predictions at the terminal nodes.
Fitting these additive models

• Typically fit model by minimizing a loss function averaged over the training data:

$$\min_{\beta_1, \gamma_1, \ldots, \beta_M, \gamma_M} \sum_{i=1}^{n} L \left( y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m) \right)$$

• For many loss functions $L(x, f(x))$ and/or basis functions $b(x; \gamma)$ this is too hard....
Forward Stagewise Additive Modeling
Approximate the global solution to fitting additive model

- More simply can greedily add one basis function at a time in the following fashion.

- Set $f_0(x) = 0$

- for $m = 1, \ldots, M$
  - Compute
    
    $$
    (\hat{\beta}_m, \hat{\gamma}_m) = \arg\min_{\beta_m, \gamma_m} \sum_{i=1}^{n} L \left( y_i, \sum_{m=1}^{M} f_{m-1}(x_i) + \beta_m b(x_i; \gamma_m) \right)
    $$

  - Set
    
    $$
    f_m(x) = f_{m-1}(x) + \hat{\beta}_m b(x; \hat{\gamma}_m)
    $$

  - Note: Previously added terms are not modified.
Exponential Loss and AdaBoost
Interpretation of the Adaboost algorithm

• **Interpretation of AdaBoost.M1**

  AdaBoost.M1 \equiv \text{forward stagewise additive modelling with an exponential loss function.}

• **Definition of exponential loss**

  \[
  L(y, f(x)) = \exp\{-y f(x)\}
  \]

• Will now go through the derivation of this result....
Interpretation of the Adaboost algorithm

- Interpretation of AdaBoost.M1

  \[ \text{AdaBoost.M1} \equiv \text{forward stagewise additive modelling with an exponential loss function.} \]

- Definition of exponential loss

  \[ L(y, f(x)) = \exp\left\{-y f(x)\right\} \]

- Will now go through the derivation of this result....
Specifics of forward stagewise additive modelling (fsam)

- At each iteration of **forward stagewise additive modelling** must solve this optimization problem

\[
(\beta_m, G_m) = \arg \min_{\beta, G} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + \beta G(x_i))
\]

\[
= \arg \min_{\beta, G} \sum_{i=1}^{n} \exp\{ -y_i (f_{m-1}(x_i) + \beta G(x_i)) \}
\]

where we assume an exponential loss and \( G(x) \in \{-1, 1\} \).

- Can re-write

\[
\sum_{i=1}^{n} \exp\{ -y_i (f_{m-1}(x_i) + \beta G(x_i)) \} = \sum_{i=1}^{n} \exp\{ -y_i f_{m-1}(x_i) \} \exp\{ -y_i \beta G(x_i) \}
\]

\[
= \sum_{i=1}^{n} w_i^{(m)} \exp\{ -y_i \beta G(x_i) \}
\]
Optimization of the fsam cost function

- The optimization problem becomes

\[
\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\} = \min_{\beta} \left( \min_{G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\} \right)
\]

- Note

\[
y_i \ G(x_i) = \begin{cases} 
1 & \text{if } y_i = G(x_i) \\
-1 & \text{if } y_i \neq G(x_i)
\end{cases}
\]

and this implies \( \exp\{-y_i \beta G(x_i)\} \) is equal to

\[
e^\beta \ \text{Ind}(y_i \neq G(x_i)) + e^{-\beta} \ (1 - \text{Ind}(y_i \neq G(x_i)))
\]

- The above implies \( \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\} \) can be written as:

\[
(e^\beta - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}
\]
The optimization problem becomes

$$\arg \min_G \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\}$$

$$= \arg \min_G \left( (e^\beta - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \right)$$

$$= \arg \min_G \left( \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) \right)$$

Therefore

$$G_m = \arg \min_G \left( \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) \right)$$

$G_m$ minimizes the weighted error in the AdaBoost algorithm. (if the $w_i^{(m)}$'s have the same definition....)
Optimization of the fsam cost function: $G_m$?

- The optimization problem becomes

$$\arg \min_G \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\}$$

$$= \arg \min_G \left( (e^\beta - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \right)$$

$$= \arg \min_G \left( \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) \right)$$

- Therefore

$$G_m = \arg \min_G \left( \sum_{i=1}^{n} w_i^{(m)} \text{Ind}(y_i \neq G(x_i)) \right)$$

$G_m$ minimizes the **weighted error** in the AdaBoost algorithm. (if the $w_i^{(m)}$’s have the same definition....)
Optimization of the fsam cost function: $\beta_m$?

- Plugging $G_m$ into the original optimization problem

$$
\min_{\beta} \left( \min_G \sum_{i=1}^{n} w^{(m)}_i \exp\{-y_i \beta G(x_i)\} \right)
$$

and using the previous result, it becomes

$$
\arg \min_{\beta} \left( (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w^{(m)}_i \text{Ind}(y_i \neq G_m(x_i)) + e^{-\beta} \sum_{i=1}^{n} w^{(m)}_i \right)
$$

- This quantity is minimized when

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

where

$$
\text{err}_m = \frac{\sum_{i=1}^{n} w^{(m)}_i \text{Ind}(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w^{(m)}_i}
$$

- Expression for $\beta_m$ same (upto a multiplicative constant) as for $\alpha_m$ in AdaBoost.M1 (if the $w^{(m)}_i$'s have same definition....)
Optimization of the fsam cost function: $\beta_m$?

- Plugging $G_m$ into the original optimization problem

$$\min_\beta \left( \min_G \sum_{i=1}^n w_i^{(m)} \exp\{ -y_i \beta G(x_i) \} \right)$$

and using the previous result, it becomes

$$\arg\min_\beta \left( (e^\beta - e^{-\beta}) \sum_{i=1}^n w_i^{(m)} \text{Ind}(y_i \neq G_m(x_i)) + e^{-\beta} \sum_{i=1}^n w_i^{(m)} \right)$$

- This quantity is minimized when

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

where

$$\text{err}_m = \frac{\sum_{i=1}^n w_i^{(m)} \text{Ind}(y_i \neq G_m(x_i))}{\sum_{i=1}^n w_i^{(m)}}$$

- Expression for $\beta_m$ same (upto a multiplicative constant) as for $\alpha_m$ in AdaBoost.M1 (if the $w_i^{(m)}$’s have same definition....)
Optimization of the fsam cost function: $w_i^{(m)}$?

Update of the weights: Expression for $w_i^{(m+1)}$

- Need the following result

\[-y_i G_m(x_i) = -\text{Ind}(y_i = G(x_i)) + \text{Ind}(y_i \neq G(x_i))
\]
\[= -(1 - \text{Ind}(y_i \neq G(x_i))) + \text{Ind}(y_i \neq G(x_i))
\]
\[= -1 + 2\text{Ind}(y_i \neq G(x_i))
\]

- The updated weights can then be written as

\[w_i^{(m+1)} = e^{-y_if_m(x_i)} = e^{-y_i(f_{m-1}(x) + \beta_m G_m(x))}
\]
\[= w_i^{(m)} e^{-y_i \beta_m G_m(x)}
\]
\[= w_i^{(m)} e^{2\beta_m \text{Ind}(y_i \neq G_m(x))} e^{-\beta_m}
\]

- As factor $e^{-\beta_m}$ is the same for all weights it has no effect.

- Expression for weight update for each example is the same as for AdaBoost.M1 as $\alpha_m = 2\beta_m$. 

Optimization of the fsam cost function: $w_i^{(m)}$?

**Update of the weights:** Expression for $w_i^{(m+1)}$

- Need the following result

$$-y_i G_m(x_i) = -\text{Ind}(y_i = G(x_i)) + \text{Ind}(y_i \neq G(x_i))$$

$$= -(1 - \text{Ind}(y_i \neq G(x_i))) + \text{Ind}(y_i \neq G(x_i))$$

$$= -1 + 2 \text{Ind}(y_i \neq G(x_i))$$

- The updated weights can then be written as

$$w_i^{(m+1)} = e^{-y_if_m(x_i)} = e^{-y_i(f_m-1(x) + \beta_m G_m(x))}$$

$$= w_i^{(m)} e^{-y_i \beta_m G_m(x)}$$

$$= w_i^{(m)} e^{2\beta_m \text{Ind}(y_i \neq G_m(x))} e^{-\beta_m}$$

- As factor $e^{-\beta_m}$ is the same for all weights it has no effect.

- Expression for weight update for each example is the same as for AdaBoost.M1 as $\alpha_m = 2/\beta_m$. 
Adaboost minimizes the exponential loss

- Hence can view **AdaBoost.M1** as a method that approximates minimizing

\[
\arg \min_{\beta_1, G_1, \ldots, \beta_M, G_M} \sum_{i=1}^{n} \exp \left( -y_i \sum_{m=1}^{M} \beta_m G_m(x_i) \right)
\]

via a **forward-stagewise additive modeling approach**.
Adaboost minimizes the exponential loss

For a simulated problem the training-set mis-classification error and average exponential loss:

While the mis-classification error decreases to zero $\approx 250$ iterations, the exponential loss keeps decreasing.
Loss Functions and Robustness
Loss functions for classification

- **Exponential Loss**

  \[ L(y, f(x)) = \exp\{-y f(x)\} \]

- **Binomial deviance loss**

  \[ L(y, f(x)) = -\log (1 + \exp\{-2y f(x)\}) \]

  where

  \[
p(x) = P(Y = 1 \mid x) = \frac{1}{1 + \exp\{-2f(x)\}}\]

- **Misclassification loss**

  \[ L(y, f(x)) = \text{Ind}(y f(x) < 0) \]
• These loss functions are functions of the “margin”: \( y f(x) \)

• Classification rule

\[
G(x) = \text{sign}\{f(x)\}
\]

\( \mapsto \) training examples with

- **positive margin** \( y_i f(x_i) > 0 \) are correctly classified and
- **negative margin** \( y_i f(x_i) < 0 \) are misclassified

• Decision boundary defined by \( f(x) = 0 \)

• Classification algorithms attempt to produce positive margins for each training data point.

• Loss criterion for classification should penalize negative margins more heavily than positive margins.
Loss functions for two-class classification. The response is \( y = \pm 1 \); the prediction is \( f \), with class prediction 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 \((0, 1)\). The goal of the classification algorithm is to produce positive margins as frequently as possible. Any loss criterion used for classification should penalize negative margins more heavily than positive ones since positive margin observations are already correctly classified.

Figure 10.4 shows both the exponential (10.8) and binomial deviance criteria as a function of the margin \( y \cdot f \). Also shown is misclassification loss \( L(y, f(x)) = I(y \cdot f(x) < 0) \), which gives unit penalty for negative margin values, and no penalty at all for positive ones. Both the exponential and deviance loss can be viewed as monotone continuous approximations to misclassification loss. They continuously penalize increasingly negative margin values more heavily than they reward increasingly positive ones. The difference between them is in degree. The penalty associated with binomial deviance increases linearly for large increasingly negative margin, whereas the exponential criterion increases the influence of such observations exponentially. At any point in the training process the exponential criterion concentrates much more influence on observations with large negative margins. Binomial deviance concentrates relatively less influence on such observations.
Loss functions for classification

- *Exponential* and *deviance* loss continuous approx. to *mis-classification* loss.

- They increasingly penalize negative margin values more heavily than they reward positive ones.

- Binomial deviance penalty increases linearly with negative margin.

- Exponential loss penalty increases exponentially with negative margin.

  \[ \Rightarrow \] in training the exponential criterion concentrates more of its efforts on large negative margin examples than the binomial criterion.

- Thus binomial criterion is far more robust than the exponential criterion in noisy settings - mislabels, overlapping classes.
Robust loss functions for regression

- **Squared error loss**

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

  Population optimum for this loss function: \( f(x) = E[Y | x] \)

- **Absolute loss**

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

  Population optimum for this loss function: \( f(x) = \text{median}(Y | x) \)

- On finite samples **squared error loss** puts far more emphasis on observations with large \( |y_i - f(x_i)| \) than **absolute loss**.

- Thus **squared error loss** is less robust and performance degrades for long-tailed error distributions and mis-labellings.
Robust loss functions for regression

- **Squared error loss**

  $$L(y, f(x)) = (y - f(x))^2$$

  Population optimum for this loss function: $$f(x) = E[Y | x]$$

- **Absolute loss**

  $$L(y, f(x)) = |y - f(x)|$$

  Population optimum for this loss function: $$f(x) = \text{median}(Y | x)$$

- On finite samples **squared error loss** puts far more emphasis on observations with large $$|y_i - f(x_i)|$$ than **absolute loss**.

- Thus **squared error loss** is less robust and performance degrades for long-tailed error distributions and mis-labellings.
Robust loss functions for regression

- **Huber loss**

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

- strong resistance to gross outliers while
- being nearly as efficient as least squares for Gaussian errors

- Combines the good properties of squared-error loss near zero and absolute error loss when \(|y - f|\) is large.
Robust loss functions for regression

10. Boosting and Additive Trees

FIGURE 10.5. A comparison of three loss functions for regression, plotted as a function of the margin \( y - f \). The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when \( |y - f| \) is large.

Exponential loss one performs a weighted fit of the base learner to the output values \( y_i \), with weights \( w_i = \exp(-y_i f_i / m - 1(x_i)) \). Using other more robust criteria directly in their place does not give rise to such simple feasible boosting algorithms. However, in Section 10.10.2 we show how one can derive simple elegant boosting algorithms based on any differentiable loss criterion, thereby producing highly robust boosting procedures for data mining.

10.7 "Off-the-Shelf" Procedures for Data Mining

Predictive learning is an important aspect of data mining. As can be seen from this book, a wide variety of methods have been developed for predictive learning from data. For each particular method there are situations for which it is particularly well suited, and others where it performs badly compared to the best that can be done with that data. We have attempted to characterize appropriate situations in our discussions of each of the respective methods. However, it is seldom known in advance which procedure will perform best or even well for any given problem. Table 10.1 summarizes some of the characteristics of a number of learning methods.

Industrial and commercial data mining applications tend to be especially challenging in terms of the requirements placed on learning procedures. Data sets are often very large in terms of number of observations and number of variables measured on each of them. Thus, computational con-
• When robustness is an issue
  - squared-error loss for regression and
  - exponential loss for classification
are not the best criterion to be optimizing.

• But, both these loss functions lead to elegant modular
  boosting algorithms in the context of forward stagewise
  additive modelling.

• For classification: perform a weighted fit of the base learner to
  the outputs $y_i$ with weights $w_i = \exp\{-y_i f(x_i)\}$

• More robust criteria in their place do not give rise to such
  simple feasible boosting algorithms

• Later derive simple boosting algorithms based on any
  differentiable loss criterion.
• When robustness is an issue
  - **squared-error loss** for regression and
  - **exponential loss** for classification
  are **not** the best criterion to be optimizing.

• But, both these loss functions lead to elegant modular
  boosting algorithms in the context of *forward stagewise
  additive modelling*.

• For classification: perform a weighted fit of the base learner to
  the outputs $y_i$ with weights $w_i = \exp\{-y_i f(x_i)\}$

• More robust criteria in their place do not give rise to such
  simple feasible boosting algorithms

• Later derive simple boosting algorithms based on any
  differentiable loss criterion.
“Off-the-Shelf” Procedures for Data Mining
**TABLE 10.1. Some characteristics of different learning methods. Key: ▲ = good, ◆ = fair, and ▼ = poor.**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Neural Nets</th>
<th>SVM</th>
<th>Trees</th>
<th>MARS</th>
<th>k-NN, Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural handling of data of “mixed” type</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
</tr>
<tr>
<td>Handling of missing values</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
</tr>
<tr>
<td>Robustness to outliers in input space</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▲</td>
</tr>
<tr>
<td>Insensitive to monotone transformations of inputs</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Computational scalability (large N)</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Ability to deal with irrelevant inputs</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Ability to extract linear combinations of features</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Interpretability</td>
<td>▼</td>
<td>▼</td>
<td>▲</td>
<td>▼</td>
<td>▲</td>
</tr>
<tr>
<td>Predictive power</td>
<td>▲</td>
<td>▲</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
</tbody>
</table>
- Trees are great except....
  - they are inaccurate at making predictions.

- Boosting decision trees improve their accuracy but at the cost of
  - speed
  - interpretability and
  - for AdaBoost, robustness against overlapping class distributions and especially mislabeling of the training data.

- A gradient boosted model (GBM) is a generalization of tree boosting that attempts to mitigate these problems.

- It aims to produce an accurate and effective off-the-shelf procedure for data mining.
Boosting Trees
• Tree partitions the input space into $\mathcal{R}_j, j = 1, \ldots, J$.

• Terminal/leaf nodes of tree represent the regions $\mathcal{R}_j$.

• Constant $\gamma_j$ assigned to each leaf.

• The predictive rule is

$$x \in \mathcal{R}_j \implies f(x) = \gamma_j$$

• A tree with parameters $\Theta = \{\mathcal{R}_j, \gamma_j\}_{j=1}^J$ is expressed as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j \text{Ind}(x \in \mathcal{R}_j)$$

($J$ is usually treated as a meta-parameter)
Learning a Regression Tree

- Ideally parameters found by minimizing the empirical risk

\[ \hat{\Theta} = \arg \min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \]

- Very hard optimization problem, instead settle for approximate suboptimal solutions

- **Typical approach:** Divide optimization into two parts
  
  - **Find** \( \gamma_j \) **given** \( R_j \)
    
    Typically trivial - \( \hat{\gamma}_j \) the mean of the training \( y \)'s falling in \( R_j \).

  - **Find** \( R_j \)
    
    Difficult part! Approximate solutions found. One strategy is to use a greedy, top-down recursive partitioning algorithm.
A boosted tree is a sum of regression/classification trees

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

learned in a forward stagewise manner.

At each step solve

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

for the parameters \( \Theta_m = \{R_{jm}, \gamma_{jm}\}_{j=1}^{J_m} \) of the next tree.

How do we solve this optimization problem?
Learning a boosted tree model

- **Find** $\gamma_{jm}$ **given** $\mathcal{R}_{jm}$ - easy

$$\hat{\gamma}_{jm} = \arg \min_{\gamma_{jm}} \sum_{x_i \in \mathcal{R}_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

- **Find** $\mathcal{R}_{jm}$’s - not so easy....

**A few exceptions**

- **Squared-error loss**
  
  At each stage fit a regression tree to residuals $y_i - f_{m-1}(x_i)$

- **Two-class classification and exponential loss**
  
  Gives rise to an AdaBoost method for boosting classification trees...
• If the trees are restricted to type where

\[ \beta_m T(x_i; \Theta_m) \text{ and each } \gamma_{jm} \in \{-1, 1\} \]

• The solution to

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

is the tree that minimizes the

\[ \sum_{i=1}^{N} w_{i}^{(m)} \text{Ind}(y_i \neq T(x_i; \Theta_m)) \]

with weights

\[ w_{i}^{(m)} = \exp\{-y_i f_{m-1}(x_i)\} \]

• Straightforward to implement a greedy recursive-partitioning algorithm with this loss as a splitting criterion.
If there is no restriction on the type of tree, then the solution to

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

is the tree that minimizes the

$$\sum_{i=1}^{N} w_i^{(m)} \exp\{-y_i T(x_i; \Theta_m)\}$$

with weights

$$w_i^{(m)} = \exp\{-y_i f_{m-1}(x_i)\}$$
Numerical Optimization via Gradient Boosting
If the loss, $L(\cdot, \cdot)$, is differentiable, can

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

be approximately solved with numerical optimization.

Consider this...

The loss associated with using any $f(x)$ to predict $y$ is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

Goal: Find $f$ which minimizes $L(f)$.

Re-interpret this optimization problem as find

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

where $f = \{f(x_1), \ldots, f(x_N)\}$. 
• If the loss, $L(\cdot, \cdot)$, is differentiable, can

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

be approximately solved with numerical optimization.

• Consider this...

• The loss associated with using any $f(x)$ to predict $y$ is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

• Goal: Find $f$ which minimizes $L(f)$.

• Re-interpret this optimization problem as find

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

where $f = \{f(x_1), \ldots, f(x_N)\}$.
Numerical optimization approximates

\[ \hat{f} = \arg \min_f L(f) \]

as a sum of vectors

\[ f_M = \sum_{m=0}^{M} h_m, \quad h_m \in \mathbb{R}^N \]

where \( f_0 = h_0 \) is an initial guess and each \( f_m \) is estimated from \( f_{m-1} \).
Steepest Descent

- Steepest descent chooses

\[ h_m = -\rho_m g_m \]

where

- \( \rho_m \) is a scalar and

- \( g_m \in \mathbb{R}^N \) is the gradient of \( L(f) \) evaluated at \( f = f_{m-1} \).

- Components of \( g_m \) are

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

- Step length is the solution to

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

- Solution is updated: \( f_m = f_{m-1} - \rho_m g_m \)
Forward stagewise Tree Boosting & Gradient Boosting

- Forward stagewise boosting is also a very greedy strategy:
  \[ \hat{\Theta}_m = \arg \min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \]

- Tree predictions \( T(x_i; \Theta_m) \) are analogous to the negative gradients \(-g_1, \ldots, -g_Nm\).

- But \( t_m = \{T(x_1; \Theta_m), \ldots, T(x_N; \Theta_m)\} \) are constrained to be predictions of a \( J_m \)-terminal node decision tree.

- Whereas \(-g_m\) is the unconstrained maximal descent direction.

- Also analogous
  \[ \rho_m = \arg \min_{\rho} L(f_{m-1} - \rho g_m) \quad \text{to} \quad \hat{\gamma}_{jm} = \arg \min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \]
  but perform a line search for each terminal node.
Forward stagewise Tree Boosting & Gradient Boosting

- Forward stagewise boosting is also a very greedy strategy:
  \[
  \hat{\Theta}_m = \arg \min_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))
  \]

- Tree predictions \(T(x_i; \Theta_m)\) are analogous to the negative gradients \(-g_1m, \ldots, -g_Nm\).

- But \(t_m = \{T(x_1; \Theta_m), \ldots, T(x_N; \Theta_m)\}\) are constrained to be predictions of a \(J_m\)-terminal node decision tree.

- Whereas \(-g_m\) is the unconstrained maximal descent direction.

- Also analogous
  \[
  \rho_m = \arg \min_{\rho} L(f_{m-1} - \rho g_m) \quad \text{to} \quad \hat{\gamma}_{jm} = \arg \min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})
  \]
  but perform a \textit{line search} for each terminal node.
• If only goal is minimizing

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

then perform steepest descent.

• However, the ultimate goal is to generalize $f_{M}(x)$ to new unseen data.

• A possible solution is as follows....
Gradient Tree Boosting

- Fit a tree $T(x; \Theta_m)$ at $m$th iteration whose predictions $t_m$ are as close as possible to the negative gradient

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2$$

- From the solution regions $\tilde{\mathcal{R}}_{jm}$ set

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in \tilde{\mathcal{R}}_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

- The regions $\tilde{\mathcal{R}}_{jm}$ will not be identical to the regions $\mathcal{R}_{jm}$ that solve the original problem, but they are similar enough.
Gradients for common loss functions

<table>
<thead>
<tr>
<th>Setting</th>
<th>Loss function</th>
<th>$-\partial L(y_i, f(x_i))/\partial f(x_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>$\frac{1}{2} (y_i - f(x_i))^2$</td>
<td>$y_i - f(x_i)$</td>
</tr>
<tr>
<td>Regression</td>
<td>$</td>
<td>y_i - f(x_i)</td>
</tr>
</tbody>
</table>
| Regression  | Huber               | \[
\begin{cases}
  y_i - f(x_i) & \text{if } |y_i - f(x_i)| \leq \delta_m \\
  \delta_m \text{sign}\{y_i - f(x_i)\} & \text{if } |y_i - f(x_i)| > \delta_m
\end{cases}
\] |
| Classification | Deviance               | $k$th component: $\text{Ind}(y_i = G_k) - p_k(x_i)$ |

where the $K$-class deviance loss function is

$$L(y, p(x)) = - \sum_{k=1}^{K} \text{Ind}(y = G_k) \log p_k(x) = - \sum_{k=1}^{K} \text{Ind}(y = G_k) + \log \left( \sum_{l=1}^{K} \exp\{f_l(x)\} \right)$$

if $p_k(x) = \exp\{f_k(x)\}/ \sum_{l=1}^{K} \exp\{f_l(x)\}$
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
   \[
   \gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).
   \]

   (d) Update \( f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}). \)

3. Output \( \hat{f}(x) = f_M(x). \)
Right-Sized Trees for Boosting
• Learning a large pruned tree at each round performs poorly.

• Better if
  - Restrict all trees to be same size $J_m = J \forall m$
  - Perform cross-validation to choose an optimal $J$

• Interaction level of tree-based approximations is limited by $J$:
  - if $J = 2$ then $f_M(x)$ can only be of the form
    \[
    \sum_k \eta_k(X_k)
    \]
  - if $J = 3$ then $f_M(x)$ can be of the form
    \[
    \sum_k \eta_k(X_k) + \sum_{jk} \eta_{jk}(X_j, X_k)
    \]
  - if $J = 4$ then $f_M(x)$ can be of the form
    \[
    \sum_k \eta_k(X_k) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l)
    \]
  - etc...
• Learning a large pruned tree at each round performs poorly.

• Better if
  - Restrict all trees to be same size $J_m = J \forall m$
  - Perform cross-validation to choose an optimal $J$

• Interaction level of tree-based approximations is limited by $J$:
  - if $J = 2$ then $f_M(x)$ can only be of the form
    $$\sum_k \eta_k(X_k)$$
  - if $J = 3$ then $f_M(x)$ can be of the form
    $$\sum_k \eta_k(X_k) + \sum_{jk} \eta_{jk}(X_j, X_k)$$
  - if $J = 4$ then $f_M(x)$ can be of the form
    $$\sum_k \eta_k(X_k) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l)$$
  - etc...
Size of trees in a boosted tree

- Learning a large pruned tree at each round performs poorly.

- Better if
  - Restrict all trees to be same size $J_m = J \forall m$
  - Perform cross-validation to choose an optimal $J$

- Interaction level of tree-based approximations is limited by $J$:
  - if $J = 2$ then $f_M(x)$ can only be of the form
    $\sum_k \eta_k(X_k)$
  - if $J = 3$ then $f_M(x)$ can be of the form
    $\sum_k \eta_k(X_k) + \sum_{jk} \eta_{jk}(X_j, X_k)$
  - if $J = 4$ then $f_M(x)$ can be of the form
    $\sum_k \eta_k(X_k) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l)$
  - etc...
• For many practical problems low-order interactions dominate.

• Therefore models that produce strong higher-order interaction effects suffer in accuracy.

• Authors claim that $4 \leq J \leq 8$ works well in the context of boosting.
Fig. 10.9. Boosting with different sized trees, applied to the example (10.2) used in Figure 10.2. Since the generative model is additive, stumps perform the best. The boosting algorithm used the binomial deviance loss in Algorithm 10.3; shown for comparison is the AdaBoost Algorithm 10.1.

This suggests that the value chosen for $J$ should reflect the level of dominant interactions of $\eta(x)$. This is of course generally unknown, but in most situations it will tend to be low. Figure 10.9 illustrates the effect of interaction order (choice of $J$) on the simulation example (10.2). The generative function is additive (sum of quadratic monomials), so boosting models with $J > 2$ incur unnecessary variance and hence the higher test error. Figure 10.10 compares the coordinate functions found by boosted stumps with the true functions.

Although in many applications $J = 2$ will be insufficient, it is unlikely that $J > 10$ will be required. 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. One can fine-tune the value for $J$ by trying several different values and choosing the one that produces the lowest risk on a validation sample. However, this seldom provides significant improvement over using $J \approx 6$.

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

where $X_1, \ldots, X_{10}$ are standard indpt Gaussian and $\chi^2_{10}(0.5) = 9.34$. 

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

where $X_1, \ldots, X_{10}$ are standard indpt Gaussian and $\chi^2_{10}(0.5) = 9.34$. 
Regularization
Options for regularization

- **Control number of boosting rounds**
  - Too large $M$ $\Rightarrow$ danger of over-fitting
    - There is a $M^*$ that minimizes future risk

- **Shrinkage**
  - Scale the contribution of each tree by factor $0 < \nu < 1$
    \[
    f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} \text{Ind}(x \in \mathcal{R}_{jm})
    \]
  - Smaller $\nu$ $\Rightarrow$ larger $M$ to obtain low training error
  - **Empirical finding**: small $\nu < .1$ and sufficiently large $M$ $\Rightarrow$ better result than no shrinkage. Especially for regression problems.
Options for regularization

- **Control number of boosting rounds**
  - Too large $M \implies$ danger of over-fitting

  $\therefore$ There is a $M^*$ that minimizes future risk

- **Shrinkage**
  - Scale the contribution of each tree by factor $0 < \nu < 1$

  $$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} \text{Ind}(x \in \mathcal{R}_{jm})$$

  - Smaller $\nu \implies$ larger $M$ to obtain low training error

  - **Empirical finding**: small $\nu < .1$ and sufficiently large $M$

    $\implies$ better result than no shrinkage. Especially for regression problems
Shrinkage example

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

where \( X_1, \ldots, X_{10} \) are standard indpt Gaussian and \( \chi^2_{10}(0.5) = 9.34 \).

\textbf{Deviance: } \(-2 \log \hat{p}_G(X)\)
Options for regularization

- **Subsampling**
  - *Stochastic gradient boosting* - each iteration sample a fraction $\eta$ of the training observations (without replacement).

    - A typical value is $\eta = .5$

    - Empirically subsampling without shrinkage works poorly

    - But subsampling with shrinkage works well

    - Now have 4 parameters to estimate $J, M, \nu$, and $\eta$
FIGURE 10.12. Test-error curves for the simulated example (10.2), showing the effect of stochasticity. For the curves labeled "Sample = 0.5", a different 50% subsample of the training data was used each time a tree was grown. In the left panel the models were fit by gbm using a binomial deviance loss function; in the right-hand panel using square-error loss.

The downside is that we now have four parameters to set: $J$, $M$, $\nu$ and $\eta$. Typically some early explorations determine suitable values for $J$, $\nu$ and $\eta$, leaving $M$ as the primary parameter.

10.13 Interpretation

Single decision trees are highly interpretable. The entire model can be completely represented by a simple two-dimensional graphic (binary tree) that is easily visualized. Linear combinations of trees (10.28) lose this important feature, and must therefore be interpreted in a different way.

10.13.1 Relative Importance of Predictor Variables

In data mining applications the input predictor variables are seldom equally relevant. Often only a few of them have substantial influence on the response; the vast majority are irrelevant and could just as well have not been included. It is often useful to learn the relative importance or contribution of each input variable in predicting the response.

$$Y = \begin{cases} 
1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(.5) \\
-1 & \text{otherwise}
\end{cases}$$

where $X_1, \ldots, X_{10}$ are standard indpt Gaussian and $\chi^2_{10}(.5) = 9.34$. 