Notes on Statistical and Machine Learning

Boosting and Additive Trees

Chapter: *14* **Prepared by:** *Chenxi Zhou*

This note is prepared based on

- *Chapter 10, Boosting and Additive Trees* in Hastie, Tibshirani, and Friedman (2009), and
- *Chapter 14, Committee Machines* in Izenman (2009).

The main idea of *boosting* is to combine the outputs of many "weak" learners to produce a powerful "committee".

I. Introduction and AdaBoosting Algorithm

1. **Setup:** Consider a two-class problem with the output variable coded as $Y \in \{-1, +1\}$. Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction belonging to $\{-1, +1\}$.

   Let the training data be $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $y_i \in \{-1, +1\}$.

2. **Error Rate:** The *error rate* of the training sample is

$$\overline{\text{err}} := \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(y_i \neq G(\mathbf{x}_i)),$$

   and the *expected error rate* on future predictions is $\mathbb{E}_{X,Y}[\mathbb{1}(Y \neq G(X))]$.

3. **Weak Classifier:** A *weak classifier* is the one whose error rate is only *slightly better* than random guessing.

4. **Main Idea of Boosting:** Boosting sequentially apply the weak classification algorithm to *repeatedly* modified versions of the data and produces a *sequence of weak classifiers* G_m, for $m = 1, 2, \cdots, M$. The predictions from all of them are then combined through a *weighted* majority vote to produce the final prediction

$$G(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right), \tag{1}$$

   where $\alpha_1, \cdots, \alpha_M$ are computed from the boosting algorithm and provide weights of each weak classifier G_m. The effect of $\{\alpha_m\}_{m=1}^M$ is to give higher weights to the more accurate classifiers in the sequence.

5. **Boosting Algorithm:** Below is the AdaBoost.M1 Algorithm.

Algorithm 1 AdaBoost.M1 Algorithm

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

Remark: AdaBoost in Algorithm 1 is called the *discrete AdaBoost*, since its output is a discrete label $\{-1, 1\}$.

6. **More on the Weights in Boosting:** Note that at each step, we modify the weights of each training observation $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$.

   - At the first step, the weights are all equal to $\frac{1}{n}$;
   - At the m-th step for $m = 2, 3, \cdots, M$, the observations that were misclassified by G_{m-1} have their weights *increased*, and the weights are decreased for those that were classified correctly, by noting that $1 - \mathrm{err}_m > \mathrm{err}_m$ so that $\alpha_m > 0$, since G_m's are weak classifiers and perform better than random guessing.

   As iterations proceed, observations that are difficult to be classified correctly receive gradually *increasing* influence.

II. Boosting Fits an Additive Model

1. **Additive Expansion View of Boosting:** Boosting can be viewed as a way of fitting an additive expansion in a set of elementary "basis" functions, where the basis functions are the individual classifiers G_m that map to $\{-1, +1\}$.

2. **Basis Functions and the Associated Model Fitting:** In general, basis function

expansions are of the form

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m),$$

where β_m, for $m = 1, \cdots, M$, are the expansion coefficients and $b(\,\cdot\,; \gamma)$ are the simple functions of a multivariate argument, characterized by a set of parameters γ.

Models with additive expansions are fit by *minimizing* a loss function L averaged over the training data

$$\underset{\{\beta_m, \gamma_m\}_{m=1}^{M}}{\text{minimize}} \left\{ \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \gamma_m) \right) \right\}. \tag{2}$$

Typical choices of the loss functions are

- the squared-error loss, or
- the likelihood-based loss functions.

3. **Computational Consideration:** It is typically *computationally expensive* to solve the problem (2) *directly*. A simple alternative is to solve the subproblem of fitting just a *single basis function*, and the corresponding optimization problem becomes

$$\underset{\beta, \gamma}{\text{minimize}} \left\{ \sum_{i=1}^{n} L(y_i, \beta b(\mathbf{x}_i; \gamma)) \right\}.$$

III. Forward Stagewise Additive Modeling

1. **Forward Stagewise Additive Modeling:** *Forward stagewise modeling* approximates the solution to (2)

   (a) by *sequentially* adding new basis functions to the expansion, and

   (b) *without* adjusting the parameters and coefficients of those that have already been added.

   The algorithm is outlined as below.

Algorithm 2 Forward Stagewise Additive Modeling

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

At each iteration m, one

(a) solves for the optimal basis function $b(\mathbf{x}; \gamma_m)$ and the corresponding coefficient β_m, and

(b) adds to the current function f_{m-1}.

In particular, notice that previously added terms are *not* modified.

2. Example — Square Error Loss Function: Consider the squared error loss function

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

Then, at the m-step, we have

$$\begin{aligned} L(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma)) &= (y_i - f_{m-1}(\mathbf{x}_i) - \beta b(\mathbf{x}_i; \gamma))^2 \\ &= (r_{i,m-1}(\mathbf{x}_i) - \beta b(\mathbf{x}_i; \gamma))^2, \end{aligned}$$

where

$$r_{i,m-1}(\mathbf{x}_i) := y_i - f_{m-1}(\mathbf{x}_i)$$

is the *residual* of the current model on the i-th observation.

In this setting, we see that the term $\beta_m b(\mathbf{x}_i; \gamma_m)$ that best fits the *current residuals* is added to the expansion at each step.

IV. Exponential Loss and AdaBoost

1. Exponential Loss: Define the *exponential loss function* as

$$L(y, f(\mathbf{x})) := \exp(-y f(\mathbf{x})). \tag{3}$$

2. **Goal:** We show that the AdaBoost outlined in Algorithm 1 is equivalent to the forward stagewise additive modeling (Algorithm 2) using the exponential loss function (3).

3. **Linking Forward Stagewise Additive Modeling and AdaBoost:** For AdaBoost, the basis functions are the individual classifier $G_m(x) \in \{-1, +1\}$. Using the exponential loss, we solve

$$(\beta_m, G_m) := \operatorname*{arg\,min}_{\beta, G} \left\{ \sum_{i=1}^{n} \exp\big[-y_i(f_{m-1}(\mathbf{x}_i) + \beta G(\mathbf{x}_i))\big] \right\}$$

for the classifier G_m and corresponding coefficient β_m to be added at each step.

Letting $w_i^{(m)} = \exp(-y_i f_{m-1}(\mathbf{x}_i))$, we have

$$(\beta_m, G_m) = \operatorname*{arg\,min}_{\beta, G} \left\{ \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i G(\mathbf{x}_i)) \right\}, \tag{4}$$

where $w_i^{(m)}$ does *not* depend on β or G and can be regarded as the weight applied to the i-th observation.

4. **Derivation of the Solution to (4):** For first note that, for any $\beta > 0$, the solution to (4) in G_m is

$$\begin{aligned} G_m &:= \operatorname*{arg\,min}_{G \in \{\pm 1\}} \left\{ e^{-\beta} \sum_{\{i \,|\, y_i = G(\mathbf{x}_i)\}} w_i^{(m)} + e^{\beta} \sum_{\{i \,|\, y_i \neq G(\mathbf{x}_i)\}} w_i^{(m)} \right\} \\ &= \operatorname*{arg\,min}_{G \in \{\pm 1\}} \left\{ (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G(\mathbf{x}_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \right\} \\ &= \operatorname*{arg\,min}_{G \in \{\pm 1\}} \left\{ \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G(\mathbf{x}_i)) \right\}, \end{aligned}$$

which is the classifier that minimizes the weighted error rate in predicting y.

Plugging G_m into the objective function in (4), we have

$$(e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G_m(\mathbf{x}_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}.$$

Viewing it as a function of β and differentiating it with respect to β yield

$$(e^{\beta} + e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G_m(\mathbf{x}_i)) - e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \overset{\text{set}}{=} 0.$$

It follows that the minimizer β_m must satisfy the equation

$$(e^{2\beta_m} + 1) \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G_m(\mathbf{x}_i)) = \sum_{i=1}^{n} w_i^{(m)},$$

that is,

$$e^{2\beta_m} = \frac{\sum_{i=1}^n w_i^{(m)} - \sum_{i=1}^n w_i^{(m)} \mathbb{1}(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i^{(m)} \mathbb{1}(y_i \neq G_m(\mathbf{x}_i))},$$

implying

$$\beta_m = \frac{1}{2} \log\left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}\right),$$

and err_m is the minimized weighted error rate

$$\mathrm{err}_m = \frac{\sum_{i=1}^n w_i^{(m)} \mathbb{1}(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i^{(m)}}.$$

Then, the approximation is updated as

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

and the weights for the next iteration is

$$w_i^{(m+1)} = w_i^{(m)} \cdot \exp(-\beta_m y_i G_m(\mathbf{x}_i)).$$

Since $-y_i G_m(\mathbf{x}_i) = 2 \cdot \mathbb{1}(y_i \neq G_m(\mathbf{x}_i)) - 1$, we can write the preceding equation as

$$w_i^{(m+1)} = w_i^{(m)} \exp(\alpha_m \mathbb{1}(y_i \neq G_m(\mathbf{x}_i))) \cdot \exp(-\beta_m),$$

where $\alpha_m = 2\beta_m$ is the quantity define in Algorithm 1. Since $e^{-\beta_m}$ multiplies all weights by the same amount, it has no effect.

Therefore, we have seen that the AdaBoost (Algorithm 1) minimizes the exponential loss function (3) via a forward stagewise additive modeling approach.

V. Properties of Exponential Loss

1. **Computational Advantages of Exponential Loss:** Using the exponential loss function (3) is computationally appealing — it leads to the simple modular re-weighting AdaBoost algorithm.

2. **Minimizer of Population-version Exponential Loss:** It can be shown that

$$f^*(\mathbf{x}) := \operatorname*{arg\,min}_{f(\mathbf{x})} \mathbb{E}_{Y \mid X = \mathbf{x}}\left[e^{-Yf(\mathbf{x})}\right] = \frac{1}{2} \log\left(\frac{\mathbb{P}(Y = 1 \mid X = \mathbf{x})}{\mathbb{P}(Y = -1 \mid X = \mathbf{x})}\right), \tag{5}$$

or, equivalently,

$$\mathbb{P}(Y = 1 \mid X = \mathbf{x}) = \frac{1}{1 + e^{-2f^*(\mathbf{x})}}.$$

Therefore, the additive expansion produced by AdaBoost is estimating one-half of the log-odds of $\mathbb{P}(Y=1 \mid X=\mathbf{x})$.

To show (5), we first note that

$$\mathbb{E}_{Y \mid X=\mathbf{x}}\big[e^{-Yf(\mathbf{x})}\big] = \mathbb{P}(Y=1 \mid X=\mathbf{x})e^{-f(\mathbf{x})} + \mathbb{P}(Y=-1 \mid X=\mathbf{x})e^{f(\mathbf{x})} =: L(f(\mathbf{x})).$$

Take the derivative of L with respect to $f(\mathbf{x})$, and we have

$$\frac{\mathrm{d}L(f(\mathbf{x}))}{\mathrm{d}f(\mathbf{x})} = -\mathbb{P}(Y=1 \mid X=\mathbf{x})e^{-f(\mathbf{x})} + \mathbb{P}(Y=-1 \mid X=\mathbf{x})e^{f(\mathbf{x})} \overset{\text{set}}{=} 0,$$

yielding

$$e^{2f^*(\mathbf{x})} = \frac{\mathbb{P}(Y=1 \mid X=\mathbf{x})}{\mathbb{P}(Y=-1 \mid X=\mathbf{x})}.$$

Taking the logarithm on both sides and dividing by 2 yield the desired result.

3. **Connection to the Deviance:** Assume

$$p(\mathbf{x}) := \mathbb{P}(Y=1 \mid X=\mathbf{x}) = \frac{e^{f(\mathbf{x})}}{e^{f(\mathbf{x})}+e^{-f(\mathbf{x})}} = \frac{1}{1+e^{-2f(\mathbf{x})}}.$$

Let $Y' := \frac{Y+1}{2} \in \{0,1\}$, and notice that

$$p(\mathbf{x}) = \mathbb{P}(Y=1 \mid X=\mathbf{x}) = \mathbb{P}(Y'=1 \mid X=\mathbf{x}).$$

The binomial log-likelihood function is

$$\begin{aligned}
\ell(Y', f(\mathbf{x})) :=& Y' \log p(\mathbf{x}) + (1-Y')\cdot\log(1-p(\mathbf{x})) \\
=& Y' \log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) + \log(1-p(\mathbf{x})) \\
=& 2Y'f(\mathbf{x}) - 2f(\mathbf{x}) - \log(1+e^{-2f(\mathbf{x})}) \\
=& \begin{cases} -\log(1+e^{-2f(\mathbf{x})}), & \text{if } Y=1, \\ -2f(\mathbf{x}) - \log(1+e^{-2f(\mathbf{x})}) = -\log(1+e^{2f(\mathbf{x})}), & \text{if } Y=-1, \end{cases} \\
=& -\log(1+e^{-2Yf(\mathbf{x})}),
\end{aligned}$$

where $-\ell(Y', f(\mathbf{x}))$ is also called the *binomial deviance*. Then,

$$\begin{aligned}
\mathbb{E}_{Y \mid X=\mathbf{x}}[\ell(Y', f(\mathbf{x}))] =& -\mathbb{P}(Y=1 \mid X=\mathbf{x})\log(1+e^{-2f(\mathbf{x})}) \\
& - (1-\mathbb{P}(Y=1 \mid X=\mathbf{x}))\log(1+e^{2f(\mathbf{x})}).
\end{aligned}$$

Differentiating $\mathbb{E}_{Y' \mid X=\mathbf{x}}[\ell(Y', f(\mathbf{x}))]$ with respect to $f(\mathbf{x})$ and setting the derivation to 0 yield

$$\begin{aligned}
f^*(\mathbf{x}) =& \operatorname*{arg\,max}_{f(\mathbf{x})} \mathbb{E}_{Y' \mid X=\mathbf{x}}[\ell(Y', f(\mathbf{x}))] \\
=& \frac{1}{2}\log\left(\frac{\mathbb{P}(Y=1 \mid X=\mathbf{x})}{\mathbb{P}(Y=-1 \mid X=\mathbf{x})}\right).
\end{aligned}$$

Therefore, the *population version of the maximizer* of the binomial log-likelihood function $\mathbb{E}_{Y' \mid X=\mathbf{x}}[\ell(Y', f(\mathbf{x}))]$ and the *population version of the minimizer* of the exponential loss function $\mathbb{E}_{Y \mid X=\mathbf{x}}[e^{-Yf(\mathbf{x})}]$ are identical.

Remark. This result only refers to the *population level*, or if one has *infinitely many* data. If one works finite samples, then the minimizers of $\mathbb{E}_{Y' \mid X=\mathbf{x}}[-\ell(Y', f(\mathbf{x}))]$ and $\mathbb{E}_{Y \mid X=\mathbf{x}}[e^{-Yf(\mathbf{x})}]$ are *different*.

VI. Loss Functions and Robustness

VI.1 Robust Loss Functions for Classification

1. **Margin $yf(\mathbf{x})$:** Let $y \in \{-1, +1\}$. Both the exponential loss function and the binomial deviance are monotonically decreasing functions of the "margin" $yf(\mathbf{x})$: for the classification rule $G(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x}))$,

   (a) if the margin $y_i f(\mathbf{x}_i) > 0$, the observation i is classified correctly, and

   (b) if the margin $y_i f(\mathbf{x}_i) < 0$, the observation i is classified incorrectly.

   The classification boundary is defined by $f(\mathbf{x}) = 0$. 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.

2. **Misclassification Loss and Its Continuous Approximations:** The misclassification loss function is defined as

$$L(y, f(\mathbf{x})) := \mathbb{1}(yf(\mathbf{x}) < 0),$$

   which takes on the value of 1 if the observation i is misclassified and 0 otherwise. In other words, the penalty for a negative margin (a misclassification) is 1 and no penalty is imposed for a positive margin.

   Remark. The exponential loss and deviance loss can be viewed as monotone continuous approximations to the misclassification loss.

3. **Comparisons and Contrasts between Exponential Loss and Deviance Loss:**

   (a) *Similarity:* Both continuously penalize *increasingly* negative margin values more *heavily* than they reward increasingly positive ones.

   (b) *Difference:*

      i. *Degree of Penalization on Negative Margin:*
         - For large negative margins, the deviance loss increases their influence *linearly*, and
         - the exponential loss increases their influence *exponentially*.

As a consequence, the exponential loss puts much more influence on observations with large negative margins, and the deviance concentrates less on these observations, and puts influence more evenly among all of the data.

ii. *Robustness:* The binomial deviance is *more robust* in noisy settings where the Bayes error rate is not close to zero, especially in situations where there is misspecification of the class labels in the training data.

4. A Discussion of Squared-Error Loss Function Used in Classification:

- *Population Version of Minimizer:* The minimizer of the squared-error loss is

$$\begin{aligned} f^*(\mathbf{x}) := & \operatorname*{arg\,min}_{f(\mathbf{x})} \mathbb{E}_{Y \mid X=\mathbf{x}}\big[(Y - f(\mathbf{x}))^2\big] \\ = & \,\mathbb{E}[Y \mid X = \mathbf{x}] \\ = & \,2\mathbb{P}(Y = 1 \mid X = \mathbf{x}) - 1. \end{aligned}$$

- *Discussion:* Squared-error loss is *not* a good surrogate for misclassification error as it is not a monotonically decreasing function in terms of the margin $yf(\mathbf{x})$. It is increasing when $yf(\mathbf{x}) > 1$. Therefore, if $y_i f(\mathbf{x}_i) > 1$, it increases *quadratically* and increases the influence (error) on observations that are *correctly* classified with increasing certainty, thereby reducing the relative influence of those incorrectly classified $y_i f(\mathbf{x}_i) < 0$.
- *Conclusion:* If class assignment is the goal, a **monotone decreasing** criterion in the margin serves as a better surrogate loss function.

5. W-class Classification Problem: Supposing that we have W classes ($W > 2$) and the response variable Y takes values in the set $\mathcal{W} := \{1, 2, \cdots, W\}$. We seek a classifier G mapping to $\mathcal{W}$.

It is sufficient to know the class conditional probabilities $p_w(\mathbf{x}) := \mathbb{P}(Y = w \mid \mathbf{x})$ for $w = 1, \cdots, W$, and the Bayes classifier is

$$G(\mathbf{x}) = \operatorname*{arg\,max}_{w \in \mathcal{W}} p_w(\mathbf{x}).$$

We don't need to know $p_w(\mathbf{x})$ for all $w = 1, 2, \cdots, W$, but just the largest one.

6. W-class Logistic Regression: The logistic model generalized to W classes is

$$p_w(\mathbf{x}) = \frac{e^{f_w(\mathbf{x})}}{\sum_{\ell=1}^{W} e^{f_\ell(\mathbf{x})}}, \qquad \text{for each } w = 1, \cdots, W,$$

which ensures that $0 \le p_w(\mathbf{x}) \le 1$, for all $w = 1, 2, \cdots, W$, and that they sum to 1.

Note that the functions f_w's are identifiable only up to an arbitrary common function h. To avoid redundancy, one can set $f_W(\mathbf{x}) = 0$ or $\sum_{w=1}^{W} f_w(\mathbf{x}) = 0$.

The W-class multinomial deviance loss function is

$$\begin{aligned} L(y, p(\mathbf{x})) &= -\sum_{w=1}^{W} \mathbb{1}(y = w) \log p_w(\mathbf{x}) \\ &= -\sum_{w=1}^{W} \mathbb{1}(y = w) f_w(\mathbf{x}) + \log\left(\sum_{\ell=1}^{W} e^{f_\ell(\mathbf{x})}\right). \end{aligned}$$

This criterion penalizes incorrect predictions only *linearly* in their degree of incorrectness.

VI.2 Robust Loss Functions for Regression

1. **Two Types of Loss Functions:** Analogous to the exponential loss function (*not* robust) and the binomial deviance function (robust), the loss functions considered in the regression setting include

   (a) the *squared error loss function*, $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$, and

   (b) the *absolute error loss function*, $L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$.

2. **Solutions to the Two Loss Functions:** The population minimizer of the squared error loss function is

$$f^*(\mathbf{x}) = \operatorname*{arg\,min}_{f(\mathbf{x})} \mathbb{E}_{Y \mid X = \mathbf{x}}\big[(Y - f(\mathbf{x}))^2\big] = \mathbb{E}[Y \mid X = \mathbf{x}],$$

   and that of the absolute error loss function is

$$f^*(\mathbf{x}) = \operatorname*{arg\,min}_{f(\mathbf{x})} \mathbb{E}_{Y \mid X = \mathbf{x}}\big[|Y - f(\mathbf{x})|\big] = \operatorname{median}(Y \mid X = \mathbf{x}).$$

   Remark. If the distribution of the error is *symmetric*, the two solutions coincide.

3. **Robustness Comparison of Two Loss Functions:** In the finite sample case, the squared error loss function places much more emphasis on observations with large absolute residuals $|y_i - f(\mathbf{x}_i)|$, and is *less robust.* The performance of the squared-error loss severely degrades for long-tailed error distributions and for outliers.

4. **Huber Loss Function:** A robust loss function used for regression that is *insensitive* to outliers while is nearly as efficient as the least squares for Gaussian errors is the Huber's loss criterion,

$$L_\delta(y, f(\mathbf{x})) := \begin{cases} (y - f(\mathbf{x}))^2, & \text{if } |y - f(\mathbf{x})| \le \delta, \\ 2\delta|y - f(\mathbf{x})| - \delta^2, & \text{otherwise,} \end{cases} \tag{6}$$

   where $\delta > 0$.

VII. "Off-the-Shelf" Procedures for Data Mining

1. **Typical Characteristics of Datasets in the Real World:**
   (a) Datasets are often very large in terms of the number of observations and number of variables;
   (b) Datasets are usually *messy* in the sense that the inputs tend to be mixtures of quantitative, binary and categorical variables;
   (c) Datasets may contain missing values;
   (d) Distributions of numeric predictors and response variables are often long-tailed and highly skewed;
   (e) The predictor variables are generally measured on different scales.
2. **Difficulties in Data Mining:**
   (a) Due to the large size nature of the datasets, computational consideration play an important role;
   (b) Only a *small fraction* of the large number of predictors are actually relevant to prediction. One needs to determine which variables to be include into the model;
   (c) Data mining applications require *interpretable models* and producing a sole predictive model is *not* enough. It is also desirable to have information providing *qualitative* understanding between joint values of the input variables and the resulting predicted response value. Thus, *black box* methods such as neural networks are far *less* useful for data mining.
3. **"Off-the-Shelf" Method:** An *"off-the-shelf" method* is the one that can be directly applied to the data *without* requiring a great deal of time-consuming data preprocessing or careful tuning of the learning procedure. One example is the *decision tree.*
4. **Advantages of Decision Trees:** The following are some advantages of decision trees from the perspective of the "off-the-shelf" method:
   (a) They are relatively fast to construct and produce interpretable models;
   (b) They naturally incorporate mixtures of numeric and categorical predictors and missing values;
   (c) They are invariant under (strictly monotone) transformations of the individual predictors;
   (d) They are immune to the effects of predictor outliers;
   (e) They perform internal feature selection as an integral part of the procedure and are resistant to the inclusion of many irrelevant predictors;
   (f) The disadvantage of decision trees is that it may not provide predictive accuracy comparable to some other methods.

VIII. Boosting Trees

1. **Review of Classification and Regression Trees (CART):** Classification and regression trees partition the space of all joint predictor variable values into disjoint regions R_j for $j = 1, \cdots, J$, as represented by the terminal nodes of the tree. A constant γ_j is assigned to each such region and the predictive rule is
$$\mathbf{x} \in R_j \qquad \Longrightarrow \qquad f(\mathbf{x}) = \gamma_j.$$
Then, a tree can be expressed as
$$T(\mathbf{x}; \Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}(\mathbf{x} \in R_j),$$
with parameters $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$. Here, J is treated as a meta-parameter.

2. **Parameter Estimation in CART:** The parameters in Θ are estimated by minimizing the empirical risk
$$\widehat{\Theta} := \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_j} L(y_i, \gamma_j). \tag{7}$$
The estimation takes on two steps:

   Step 1: Estimating γ_j given R_j.

   - For <u>regression</u> problems, given the partitions of the regions R_j, one can estimate γ_j trivially by the mean of the y_i's falling in R_j;
   - For <u>classification</u> problems, given the partitions of the regions R_j, $\hat{\gamma}_j$ is the modal class of observations falling in R_j.

   Step 2: Estimating R_j. This is the hard part and one can take a *greedy, top-down recursive partitioning algorithm* to find R_j. Also, it is sometimes necessary to approximate (7) by a smoother and more convenient criterion to optimize R_j, i.e.,
$$\widehat{\Theta} = \arg\min_{\Theta} \left\{ \sum_{i=1}^{n} \widetilde{L}(y_i, T(\mathbf{x}_i, \Theta)) \right\}.$$

3. **Boosted Trees:** The *boosted tree model* is a sum of trees
$$f_M(\mathbf{x}) = \sum_{m=1}^{M} T(\mathbf{x}; \Theta_m) \tag{8}$$
induced in a *forward stagewise* manner (Algorithm 2).

At each step in the forward stagewise procedure, one solves the problem

$$\widehat{\Theta}_m = \arg\min_{\Theta_m} \left\{ \sum_{i=1}^{n} L\big(y_i, f_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \Theta_m)\big) \right\} \tag{9}$$

for the region set and constants $\Theta_m = \{R_{m,j}, \gamma_{m,j}\}_{j=1}^{J_m}$ of the next tree, given the current model f_{m-1}.

4. **Parameter Estimation in Boosted Tree:** We proceed by a two-step procedure:

   Step 1: Given the regions $R_{m,j}$, estimate the optimal constants $\gamma_{m,j}$ in each region by

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

   Step 2: Estimating the regions is harder.

   - In the regression problem, for the squared-error loss, the solution is simply the regression tree that best predicts the current residuals $y_i - f_{m-1}(\mathbf{x}_i)$, and $\hat{\gamma}_{m,j}$ is the mean of these residuals in each corresponding region;
   - In the binary classification problem, for the exponential loss, the stagewise approach gives rise to the AdaBoost method. That is, if the trees $T(\mathbf{x}; \Theta_m)$ are restricted to be *scaled* classification trees[1], the solution to (9) is the tree that minimizes the weighted error rate $\sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq T(\mathbf{x}_i; \Theta_m))$ with $w_i^{(m)} = e^{-y_i f_{m-1}(\mathbf{x}_i)}$.
   - Still in the two-class classification problem with the exponential loss, without the scaled classification tree restriction, one can simplifies (9) to a weighted exponential criterion for the new tree

     $$\widehat{\Theta}_m = \arg\min_{\Theta_m} \left\{ \sum_{i=1}^{n} w_i^{(m)} \exp(-y_i T(\mathbf{x}_i; \Theta_m)) \right\}.$$

     Then, one can implement a greedy recursive-partitioning algorithm using this weighted exponential loss as a splitting criterion. Given $R_{m,j}$, the solution is the weighted log-odds in each corresponding region

     $$\hat{\gamma}_{m,j} = \frac{1}{2} \log \left(\frac{\sum_{\mathbf{x}_i \in R_{m,j}} w_i^{(m)} \mathbb{1}(y_i = 1)}{\sum_{\mathbf{x}_i \in R_{m,j}} w_i^{(m)} \mathbb{1}(y_i = -1)} \right). \tag{10}$$

5. **Derivation of (10):** Given the $R_{m,j}$ for $j = 1, \cdots, J$, by the definition of a tree, we have

$$\begin{aligned}
\widehat{\boldsymbol{\gamma}}_m :&= \arg\min_{\boldsymbol{\gamma}_m \in \mathbb{R}^J} \left\{ \sum_{i=1}^{n} w_i^{(m)} \exp\big[-y_i T(x_i; R_{m,j}, \boldsymbol{\gamma}_{m,j})\big] \right\} \\
&= \arg\min_{\boldsymbol{\gamma}_m \in \mathbb{R}^J} \left\{ \sum_{i=1}^{n} w_i^{(m)} \exp\left[-y_i \sum_{j=1}^{J} \gamma_{m,j} \mathbb{1}(\mathbf{x}_i \in R_{m,j})\right] \right\},
\end{aligned}$$

[1] A tree is said to be a *scaled classification tree* if the tree is of the form $\beta_m \cdot T(\mathbf{x}; \Theta_m)$, with the restriction that $\gamma_{m,j} \in \{-1, +1\}$.

where $\boldsymbol{\gamma}_m := (\gamma_{1,m}, \cdots, \gamma_{J,m})^\top \in \mathbb{R}^J$.

Let $f : \mathbb{R}^J \to \mathbb{R}$ so that

$$f(\boldsymbol{\gamma}_m) := \sum_{i=1}^{n} w_i^{(m)} \exp\left[-y_i \sum_{j=1}^{J} \gamma_{m,j} \mathbb{1}(\mathbf{x}_i \in R_{m,j})\right].$$

Taking the partial derivative of f with respect to $\gamma_{m,j'}$ for some $j' \in \{1, \cdots, J\}$ yields

$$\begin{aligned}\frac{\partial f(\boldsymbol{\gamma}_m)}{\partial \gamma_{m,j'}} &= \sum_{i=1}^{n} w_i^{(m)} \exp\left[-y_i \sum_{j=1}^{J} \gamma_{m,j} \mathbb{1}(\mathbf{x}_i \in R_{m,j})\right] \big(-y_i \mathbb{1}(\mathbf{x}_i \in R_{m,j'})\big) \\ &= - \sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} y_i w_i^{(m)} \exp\big(-y_i \gamma_{m,j'}\big) \\ &= - \sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} \Big(\mathbb{1}(y_i = 1) w_i^{(m)} \exp\big(-\gamma_{m,j'}\big) - \mathbb{1}(y_i = -1) w_i^{(m)} \exp\big(\gamma_{m,j'}\big)\Big).\end{aligned}$$

We must have $\frac{\partial f(\boldsymbol{\gamma}_m)}{\partial \gamma_{m,j'}}\Big|_{\gamma_{m,j'} = \widehat{\gamma}_{m,j'}} = 0$, which is equivalent to solving the equation

$$\sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} \Big(\mathbb{1}(y_i = 1) w_i^{(m)} \exp\big(-\widehat{\gamma}_{m,j'}\big) - \mathbb{1}(y_i = -1) w_i^{(m)} \exp\big(\widehat{\gamma}_{m,j'}\big)\Big) = 0,$$

that is,

$$\sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} \mathbb{1}(y_i = 1) w_i^{(m)} \exp\big(-\widehat{\gamma}_{m,j'}\big) = \sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'm}\}} \mathbb{1}(y_i = -1) w_i^{(m)} \exp\big(\widehat{\gamma}_{m,j'}\big),$$

or, equivalently,

$$\exp(2\widehat{\gamma}_{m,j'}) \sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} \mathbb{1}(y_i = -1) w_i^{(m)} = \sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} \mathbb{1}(y_i = 1) w_i^{(m)},$$

or, equivalently,

$$\widehat{\gamma}_{m,j'} = \frac{1}{2} \log\left(\frac{\sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} w_i^{(m)} \mathbb{1}(y_i = 1)}{\sum_{\{i \,|\, \mathbf{x}_i \in R_{m,j'}\}} w_i^{(m)} \mathbb{1}(y_i = -1)}\right),$$

which is desired result.

IX. Numerical Optimization via Gradient Boosting

1. **Setup:** Assume that the loss criterion differentiable L is differentiable. The loss in using $f(\mathbf{x})$ to predict y on the training data is

$$J(f) = \sum_{i=1}^{n} L\big(y_i, f(\mathbf{x}_i)\big).$$

The goal is to minimize J with respect to f, where $f(\mathbf{x})$ is constrained to be a sum of trees, i.e.,

$$f(\mathbf{x}) = \sum_{m=1}^{M} T(\mathbf{x}; \Theta_m) = \sum_{m=1}^{M} \sum_{j=1}^{J} \gamma_{m,j} \mathbb{1}(\mathbf{x} \in R_{m,j}).$$

2. **Steepest Descent Review:** Ignore the assumption that f takes on the form of a sum of trees for the moment. Minimizing the loss function J can be viewed as a numerical optimization problem

$$\hat{\mathbf{f}} := \arg\min_{\mathbf{f}\in\mathbb{R}^n} J(\mathbf{f}), \tag{11}$$

where the "parameters" $\mathbf{f} \in \mathbb{R}^n$ are the values of the approximating function $f(\mathbf{x}_i)$ at each of the n data points $\mathbf{x}_i$

$$\mathbf{f} := \big(f(\mathbf{x}_1), f(\mathbf{x}_2), \cdots, f(\mathbf{x}_n)\big)^\top \in \mathbb{R}^n.$$

Numerical optimization procedures solve (11) as a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \qquad \text{where } \mathbf{h}_m \in \mathbb{R}^n,$$

where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess, and each successive $\mathbf{f}_m$ is induced based on the current parameter vector $\mathbf{f}_{m-1}$, the sum of the previously induced updates.

The *steepest descent* chooses $\mathbf{h}_m = -\rho_m \mathbf{g}_m$, where $\rho_m > 0$ is a scalar step size and $\mathbf{g}_m \in \mathbb{R}^n$ is the *gradient* of J evaluated at $\mathbf{f}_{m-1}$. The i-th component of the gradient $\mathbf{g}_m$ is given by

$$g_{m,i} = \left[\frac{\partial J(\mathbf{f})}{\partial \mathbf{f}} \bigg|_{\mathbf{f}=\mathbf{f}_{m-1}} \right]_i, \qquad \text{for all } i = 1, 2, \cdots, n.$$

The *step length* ρ_m is given by

$$\rho_m := \arg\min_{\rho>0} L(\mathbf{f}_{m-1} - \rho\, \mathbf{g}_m).$$

The current solution is then updated as

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m\, \mathbf{g}_m,$$

and the process repeats at the next iteration. Steepest descent can be viewed as a very *greedy* strategy, since $-\mathbf{g}_m$ is the local direction in $\mathbb{R}^n$ for which J is most rapidly decreasing at $\mathbf{f} = \mathbf{f}_{m-1}$.

3. **Forward Stagewise Boosting as a Greedy Algorithm:** Forward stagewise boosting (Algorithm 2) is also a *greedy* strategy. At each step, the solution tree is the one that maximally reduces (9), given the current model f_{m-1} and its fits $f_{m-1}(\mathbf{x}_i)$. Thus, the tree predictions $T(\mathbf{x}_i; \Theta_m)$ are analogous to the components of the negative gradient $\mathbf{g}$.

4. A Dilemma:

(a) *Dilemma:* In the *steepest descent* method, the gradient is defined only at the training data points $\mathbf{x}_i$, whereas the ultimate goal is to generalize to new data *not* present in the training set.

(b) *Possible Solution:* A possible solution is to induce a tree $T(\,\cdot\,;\Theta_m)$ at the m-th iteration whose predictions are as close as possible to the negative gradient.

(c) *Example:* If one use the squared error loss function, one can approximate as follows

$$\widetilde{\Theta} = \arg\min \sum_{i=1}^{n} (-g_{m,i} - T(\mathbf{x}_i; \Theta))^2,$$

i.e., one fits the tree T to the negative gradient values by least squares.

5. Examples of Negative Gradients:

- *Least Squared Error Loss in Regression:* The loss function is $L(y_i, f(\mathbf{x}_i)) = \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$, and

$$-\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} = y_i - f(\mathbf{x}_i),$$

the ordinary residual;

- *Absolute Error Loss in Regression:* The loss function is $L(y_i, f(\mathbf{x}_i)) = |y_i - f(\mathbf{x}_i)|$, and

$$-\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} = \operatorname{sign}(y_i - f(\mathbf{x}_i)),$$

the sign of the ordinary residual;

- *Huber Loss in Regression:* The loss function is

$$L(y_i, f(\mathbf{x}_i)) = \begin{cases} \frac{1}{2}(y_i - f(\mathbf{x}_i))^2, & \text{if } |y_i - f(\mathbf{x}_i)| \le \delta, \\ \delta|y_i - f(\mathbf{x}_i)| - \frac{1}{2}\delta^2, & \text{otherwise,} \end{cases}$$

and

$$-\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} = \begin{cases} y_i - f(\mathbf{x}_i), & \text{for } |y_i - f(\mathbf{x}_i)| \le \delta, \\ \delta \operatorname{sign}(y_i - f(\mathbf{x}_i)), & \text{for } |y_i - f(\mathbf{x}_i)| > \delta, \end{cases}$$

where $\delta > 0$;

- *Multinomial Deviance Loss in W-class Classification:* The loss function is the multinomial deviance

$$L(y_i, f_1(\mathbf{x}_i), f_2(\mathbf{x}_i), \cdots, f_K(\mathbf{x}_i)) = -\sum_{w=1}^{W} \mathbb{1}(y_i = w) f_w(\mathbf{x}_i) + \log\left(\sum_{\ell=1}^{W} e^{f_\ell(\mathbf{x}_i)}\right),$$

and

$$-\frac{\partial L(y_i, f_1(\mathbf{x}_i), f_2(\mathbf{x}_i), \cdots, f_W(\mathbf{x}_i))}{\partial f_w(\mathbf{x}_i)} = \mathbb{1}(y_i = w) - p_w(\mathbf{x}_i),$$

where $p_w(\mathbf{x}_i) = \frac{e^{f_w(\mathbf{x}_i)}}{\sum_{\ell=1}^{W} e^{f_\ell(\mathbf{x}_i)}}$ for all $w = 1, 2, \cdots, W$. Note that W least squares trees are constructed at each iteration.

6. **Gradient Boosting Algorithm for Regression:** The gradient boosted algorithm is outlined below.

Algorithm 3 Gradient Tree Boosting Algorithm

```
1: Initialize $f_0 = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$;
2: For $m = 1$ to $M$:
    (a) For $i = 1, \cdots, n$, compute
            $r_{m,i} = -\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\Big|_{f=f_{m-1}}$;
    (b) Fit a regression tree to the targets $r_{m,i}$ giving terminal regions $R_{m,j}$ for $j = 1, 2, \cdots, J_m$;
    (c) For $j = 1, 2, \cdots, J_m$, compute
            $\gamma_{m,j} = \arg\min_{\gamma} \left\{ \sum_{\mathbf{x}_i \in R_{m,j}} L(y_i, f_{m-1}(\mathbf{x}_i) + \gamma) \right\}$;
    (d) Update
            $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \sum_{j=1}^{J_m} \gamma_{m,j} \mathbb{1}(\mathbf{x} \in R_{m,j})$;
3: Output $f_M(\mathbf{x})$.
```

Remark. In the algorithm above, two basic *tuning parameters* are

(a) the number of iterations M, and

(b) the sizes of each of the constituent trees J_m for $m = 1, 2, \cdots, M$.

7. **Gradient Boosting Algorithm for Classification:** Lines 2(a)-(d) are repeated for W classes at each iteration m. The result at Line 3 is W different (coupled) tree expansions $f_{w,M}(\mathbf{x})$ for all $w = 1, 2, \cdots, W$.

The complete algorithm is shown below.

Algorithm 4 Gradient Tree Boosting Algorithm for W-class Classification

```
1: Initialize $f_{w,0}(\mathbf{x}) = 0$ for $w = 1, 2, \cdots, W$;
2: For $m = 1$ to $M$:
    (a) Set
            $$p_w(\mathbf{x}) = \frac{e^{f_w(\mathbf{x})}}{\sum_{\ell=1}^{W} e^{f_\ell(\mathbf{x})}};$$
    (b) For $w = 1$ to $W$:
        i. Compute $r_{i,w,m} = y_{i,w} - p_w(\mathbf{x}_i)$, for $i = 1, \cdots, n$;
        ii. Fit a regression tree to the targets $r_{i,w,m}$ for all $i = 1, \cdots, n$, given the
            terminal regions $R_{j,w,m}$ for all $j = 1, 2, \cdots, J_m$;
        iii. Compute
            $$\gamma_{j,w,m} = \frac{W-1}{W} \frac{\sum_{\{i \,|\, \mathbf{x}_i \in R_{j,w,m}\}} r_{i,w,m}}{\sum_{\{i \,|\, \mathbf{x}_i \in R_{j,w,m}\}} |r_{i,w,m}|(1 - |r_{i,w,m}|)},$$
            for $j = 1, 2, \cdots, J_m$;
        iv. Update
            $$f_{w,m}(\mathbf{x}) = f_{w,m-1}(\mathbf{x}) + \sum_{j=1}^{J_m} \gamma_{j,w,m} \mathbb{1}(\mathbf{x} \in R_{j,w,m});$$
3: Output $\hat{f}_w(\mathbf{x}) = f_{w,M}(\mathbf{x})$ for all $w = 1, \cdots, W$.
```

X. Right-Sized Trees for Boosting

1. **Review of Tree Model Pruning:** When constructing a tree-based model, we first build a very large (oversized) tree and then a bottom-up procedure is utilized to prune the oversized tree.

2. **Naive Approach to Choose Tree Size in Gradient Boosting:** For each tree we build, we use the approach for building a tree model as above. This approach is *bad* for the following reasons:

   (a) This approach assumes that each tree is the last tree in (8), this is a poor assumption.

   (b) Earlier trees tend to be very large, especially during the early iterations.

(c) This approach can substantially degrades performance and increases computation.

3. **Tree Size in Boosting:** Restrict all trees to be the same size, i.e., $J_m = J$ for all $m = 1, \cdots, M$. At each iteration, a J-terminal node regression tree is induced.

   In this approach, J is a meta-parameter of the entire boosting procedure and needs to be adjusted to maximize the estimated performance for the data.

4. **ANOVA Expansion:** Let

$$\eta := \arg\min_{f} \mathbb{E}_{X,Y}[L(Y, f(X))].$$

   We consider the degree to which the coordinate variables $\mathbf{x} := (x_1, \cdots, x_p)^\top$ interact with one another and look at the ANOVA expansion

$$\eta(\mathbf{x}) = \sum_{j} \eta_j(x_j) + \sum_{j,k} \eta_{j,k}(x_j, x_k) + \sum_{j,k,\ell} \eta_{j,k,\ell}(x_j, x_k, x_\ell) + \cdots .$$

   Here, each η_j is the main effect of X_j, and each $\eta_{j,k}$ is the second-order interaction between X_j and X_k, and so on. For many problems encountered in practice, lower-order interaction effects dominate.

   The interaction level of tree-based approximations is limited by the tree size J and, consequently, no interaction effects of level greater than $J - 1$ are possible:

   (a) $J = 2$ corresponds to single split decision stump and produces boosted models with only *main effects* with *no interaction* permitted;

   (b) $J = 3$ corresponds to the case where only main effects and two-variable interaction effects are allowed, but no more;

   (c) $\cdots$

   This suggests the value chosen for J should reflect the level of dominant interactions of η.

   Remark. Typically, choose $4 \leq J \leq 8$ in the context of boosting.

XI. Regularization

1. **Controlling the Meta-parameter M:**

   (a) *Effects of the Choice of M:* Each iteration reduces the training risk and the training risk can be arbitrarily small for large values of M. Fitting too well to the training datasets can lead to overfitting and degrades the risk in the future predictions.

   (b) *How to Choose the Best M:* The choice of the optimal number of iterations M^* minimizing future risk is application dependent. One can estimate M^* by monitoring prediction risk as a function of M on a validation sample, i.e., early stopping.

2. **Regularization Approach:** One can adopt the *shrinkage technique* to scale the contribution of each tree by a factor $0 < \nu < 1$ when it is added to the current approximation. Line 2(d) in Algorithm 3 is replaced by

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

The parameter ν here can be viewed as controlling the learning rate of the boosting procedure. Smaller values of ν lead to larger values of M for the same training risk, so there is a trade-off between the two.

Empirical Rule in Choosing ν and M: Set ν to be very small, say $\nu < 0.1$, and then choose M by early stopping.

3. **Subsampling:** Use *stochastic gradient boosting.* At each iteration, we sample a fraction $\eta \le \frac{1}{2}$ of the training observations *without* replacement, and grow the next tree using these subsamples.

The *advantages* of using subsampling are two-folds:

(a) the sampling reduces the computing time by the same fraction η;

(b) the sampling, in some circumstances, can produce a more accurate model.

XII. Interpretation

XII.1 Relative Importance of Predictor Variables

1. **Goal:** We want to learn the relative importance or contribution of each input variable in predicting the response.

2. **Importance of a Variable in a Single Decision Tree T for Regression:** For a single decision tree T, use

$$\mathcal{I}_\ell^2 = \sum_{t=1}^{J-1} \hat{\imath}_t^2 \cdot \mathbb{1}(v(t) = \ell) \tag{12}$$

as a measurement of relevance for each predictor variable X_ℓ. The sum is taken over the $J-1$ internal nodes of the tree. At each node t, one of the input variables $X_{v(t)}$ is used to partition the region associated with that node into two sub-regions.

The particular variable chosen is the one that gives *maximal* estimated improvement $\hat{\imath}_t^2$ in squared error risk over that for a constant fit over the entire region.

The squared relative importance of variable X_ℓ is the *sum* of such squared improvements over all internal nodes for which it was chosen as splitting variable.

3. **Importance of a Variable in Additive Tree Expansions:** Over the additive tree expansions, the importance of the variable X_ℓ is measured by the average

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_m). \tag{13}$$

   Remark. The two importances presented above are referred to as *squared relevance*. The actual relevances are their respective square roots.

4. **Importance of a Variable in W-class Classification Problems:** For W-class classification, W separate models f_w, $w = 1, 2, \cdots, W$ are induced, each consisting of a sum of trees

$$f_w(\mathbf{x}) = \sum_{m=1}^{M} T_{m,w}(\mathbf{x}),$$

   and the *importance* (13) generalizes to

$$\mathcal{I}_{\ell,w}^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_{m,w}). \tag{14}$$

   Here, $\mathcal{I}_{\ell,w}$ is the relevance of X_ℓ in separating the observations in Class w from the other classes.

   The *overall relevance* of X_ℓ is obtained by averaging over all classes

$$\mathcal{I}_\ell^2 = \frac{1}{W} \sum_{w=1}^{W} \mathcal{I}_{\ell,w}^2.$$

XII.2 Partial Dependence Plots

1. **An Introduction to the Partial Dependence Plots:** After identifying the most relevant variables, we plot f as a function of its arguments, which is a comprehensive summary of the dependence of the function on the joint values of the input variables.

   Functions of higher dimensions can be plotted by conditioning on particular sets of values of all but one or two of the arguments, producing a *trellis* of plots.

2. **Partial Dependence Function:** Consider the sub-vector $X_\mathcal{S}$ of $\ell < p$ of the input predictors $X = (X_1, X_2, \cdots, X_p)^\top$, indexed by $\mathcal{S} \subseteq \{1, 2, \cdots, p\}$ and $|\mathcal{S}| = \ell$. Let $\mathcal{C}$ be the complement of $\mathcal{S}$ such that $\mathcal{S} \cup \mathcal{C} = \{1, 2, \cdots, p\}$. Therefore, we can write $f(X) = f(X_\mathcal{S}, X_\mathcal{C})$.

   - *Population Version of Partial Dependence Function:* One way to define the *average or partial dependence* of f on $X_\mathcal{S}$ is

$$f_\mathcal{S}(X_\mathcal{S}) := \mathbb{E}_{X_\mathcal{C}}[f(X_\mathcal{S}, X_\mathcal{C})], \tag{15}$$

     which is a *marginal average* of f. This is particularly a useful description of the effect of the chosen subset on f when the variables in $X_\mathcal{S}$ do *not* have strong interactions with those in $X_\mathcal{C}$.

- *Sample Version of Partial Dependence Function:* Partial dependence function can be estimated by

$$\bar{f}_{\mathcal{S}}(X_{\mathcal{S}}) := \frac{1}{n}\sum_{i=1}^{n} f(X_{\mathcal{S}}, \mathbf{x}_{i,\mathcal{C}}), \tag{16}$$

  where $\{\mathbf{x}_{1,\mathcal{C}}, \mathbf{x}_{2,\mathcal{C}}, \cdots, \mathbf{x}_{n,\mathcal{C}}\}$ are the values of $X_{\mathcal{C}}$ occurring in the training data.

3. **An Important Note:** The partial dependence function represents the effect of $X_{\mathcal{S}}$ on f *after* accounting for the effects of the other variables $X_{\mathcal{C}}$ on f, and they are *not* the effect of $X_{\mathcal{S}}$ on f ignoring the effects of $X_{\mathcal{C}}$, which is given by

$$\tilde{f}_{\mathcal{S}}(X_{\mathcal{S}}) = \mathbb{E}\big[f(X_{\mathcal{S}}, X_{\mathcal{C}}) \,|\, X_{\mathcal{S}}\big],$$

   and is the best least squares approximation to f by a function of $X_{\mathcal{S}}$ alone.

   Remarks.

   (a) The quantities $f_{\mathcal{S}}$ and $\tilde{f}_{\mathcal{S}}$ are the same only when $X_{\mathcal{S}}$ and $X_{\mathcal{C}}$ are independent.

   (b) Viewing plots of the partial dependence of the boosted tree approximations on selected variable subsets can help to provide a qualitative description of its properties.

4. **Partial Dependence Function for W-class Classification Problem:** For W-class classification, there are W separate models, one for each class. Each one is related to the respective probabilities through

$$f_w(X) = \log p_w(X) - \frac{1}{W}\sum_{\ell=1}^{W} \log p_\ell(X).$$

   Each f_w is a *monotone increasing function* of its respective probability on a logarithmic scale.

   Partial dependence plots of each respective f_w on its most relevant predictors (14) can help reveal how the log-odds of realizing that class depend on the respective input variables.

XIII. Variants of Gradient Boosting

1. **XGBoost:** XGBoost, standing for *extreme gradient boosting*, is an optimized distributed gradient boosting algorithm designed to be highly efficient, flexible and portable.

   Main features of XGBoost include

   (a) Base Learner: Base learners in XGBoost can be either trees or linear functions;

   (b) Regularization Term: XGBoost can add either L_1 or L_2 regularization or both to the loss function;

(c) Derivative: XGBoost uses the second derivative to speed up the optimization;

(d) Column Subsampling: Similar to random forest, XGBoost can sample a subset of variables to determine at which variable to split and at which value to split. This can speed up the computation and avoid overfitting.

(e) Efficient Node Splitting Algorithm: XGBoost uses an approximate node splitting algorithm to construct the boosting trees. This algorithm first proposes candidate splitting points based on the percentiles of feature distribution, and then uses the second derivative information to determine a splitting point.

(f) Parallelization: XGBoost stores data into different blocks with each column sorted. Different blocks can be distributed to across the machine. When scanning each variable and determining the splitting variable and value, one can collect statistics for each column from each block, which can be done in parallel.

2. **Light GBM:** Light GBM is another optimized implementation of the gradient boosting algorithm. Two main features of it are:

   (a) *Gradient-based One-side Sampling (GOSS):* Exclude a significant proportion of data with small gradients, and only use the remaining data to estimate the information gain.

   (b) *Exclusive Feature Bundling (EFB):* Bundle mutually exclusive features (those rarely take nonzero values simultaneously) to reduce the number of features.

   These can speed up computation dramatically.

References

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. Vol. 1. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Izenman, Alan J (Mar. 2009). *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. en. Springer Science & Business Media.