CSCE 633: Machine Learning

Lecture 19: Boosting

Texas A&M University

Bobak Mortazavi

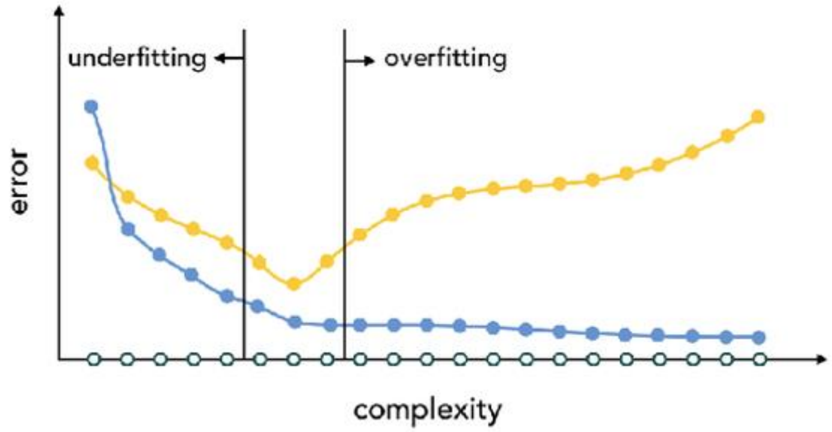
Review

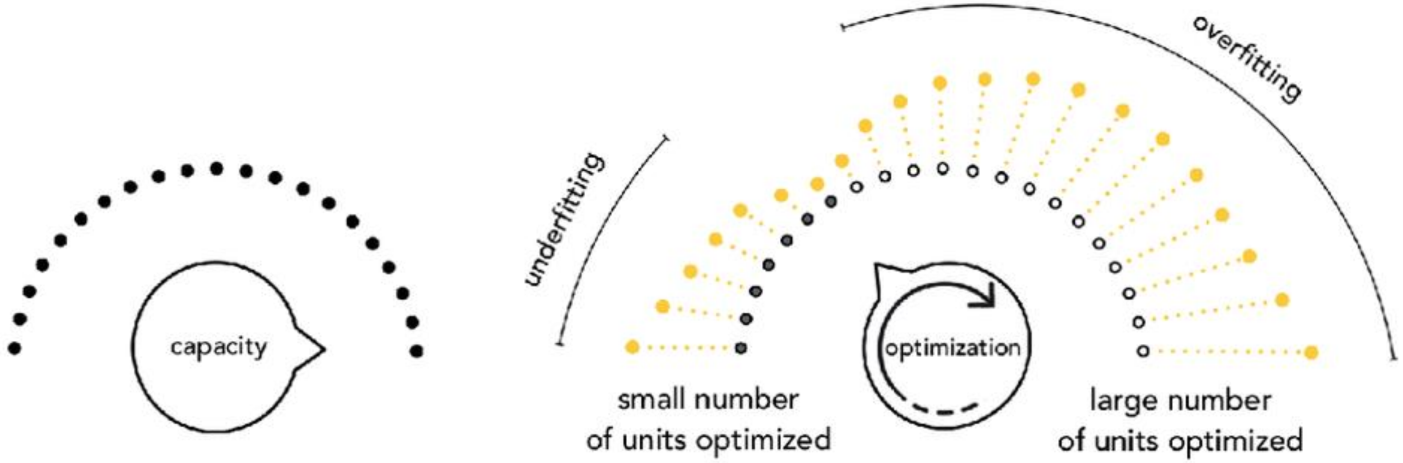
1. What is bagging?
2. What is Random Forest?
3. What is boosting?

Goals

- Review Boosting
- Introduce Boosting for Classification
- Introduce Gradient Boosting

Boosting Capacity





Initial Round (Round 0)

- Start with model:

$$model_0(x, \theta) = w_0$$

- Whose weight set $\theta_0 = \{w_0\}$, which contains a single bias weight which minimizes least squares (notation from Watt, Borhani, and Kastaggelos – P is people):

$$\frac{1}{P}\sum_{p=1}^{P}\left(model_0\left(x_p, \theta_0\right) - y_p\right)^2 = \frac{1}{P}\sum_{p=1}^{P}\left(w_0 - y_p\right)^2$$

- This optimal w_0 remains fixed forever forward

Round 1 of Boosting

- Having tuned the only parameter, we now boost its complexity by adding weighted unit $f_{s_1}(x)\, w_1$:

$$model_1(x, \theta_1) = model_0(x, \theta_0) + f_{s_1}(x)\, w_1$$

- To determine which unit in our set F best lowers the training error, we pick the $f_s \in F$ that minimizes the cost:

$$\frac{1}{P}\sum_{p=1}^{P}\left(model_0(x_p, \theta_0) + f_{s_1}(x_p)\, w_1 - y_p\right)^2 =$$

$$\frac{1}{P}\sum_{p=1}^{P}\left(w_0 + f_{s_1}(x_p)\, w_1 - y_p\right)^2$$

Round m > 1 of Boosting

$$model_{m-1}(x, \theta_{m-1}) = w_0 + f_{s_1}(x) * w_1 + f_{s_2}(x) * w_2 + \ldots + f_{s_{m-1}}(x) * w_{m-1}$$

- We then seek out the best next unit to add

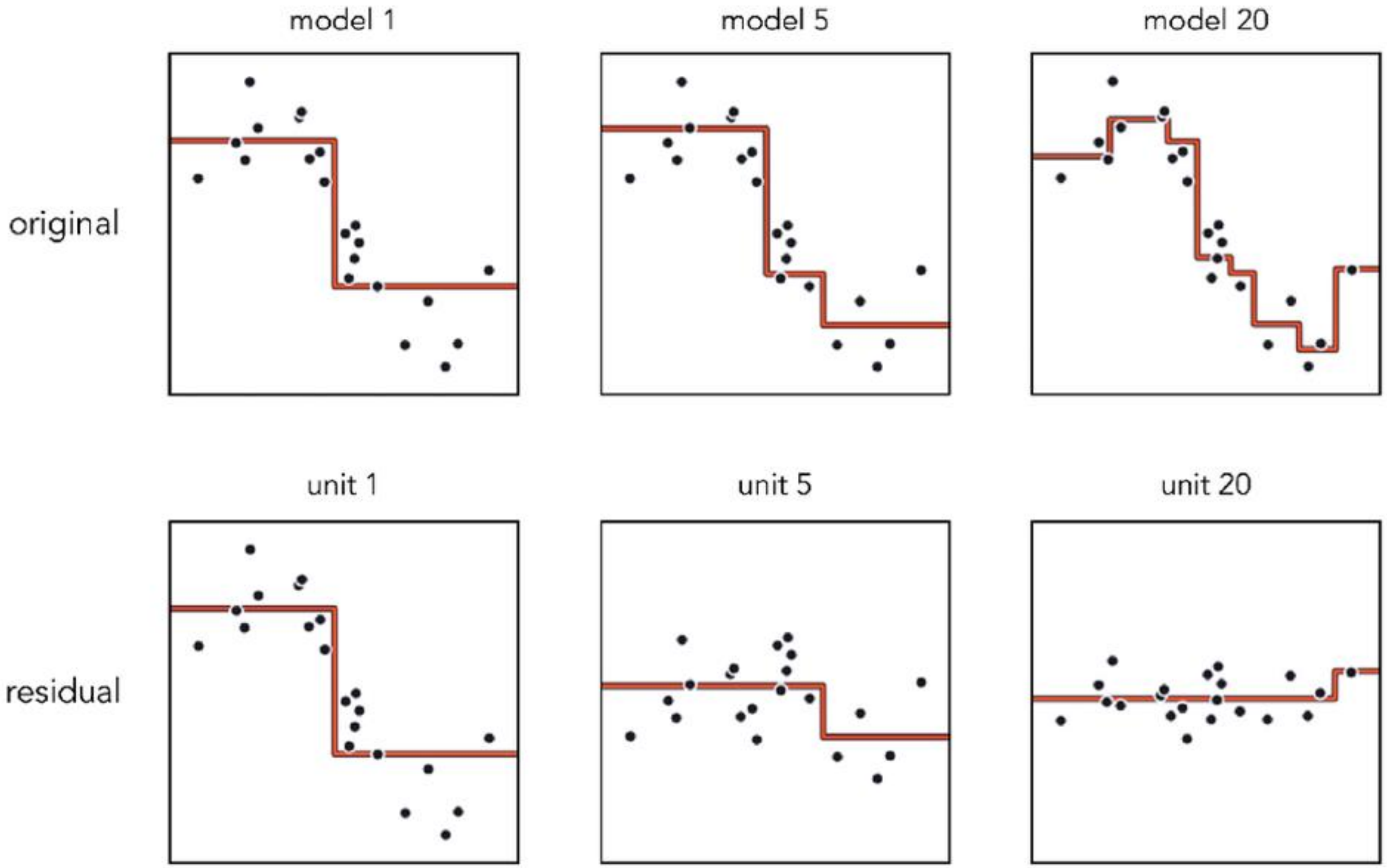
$$model_m(x, \theta_m) = model_{m-1}(x, \theta_{m-1}) + f_{s_m}(x)\, w_m$$

- By minimizing

$$\frac{1}{P}\sum_{p=1}^{P}\left(model_{m-1}(x_p, \theta_{m-1}) + f_{s_m}(x_p)\, w_m - y_p\right)^2 =$$

$$\frac{1}{P}\sum_{p=1}^{P}\left(w_0 + f_{s_1}(x_p)\, w_1 + \ldots + f_{s_m}(x_p)\, w_m - y_p\right)^2$$

Visualization using trees



Boosting for Regression Trees: Algorithm

1. Set $\hat{f}(x) = 0$ and error $r_i = y_i$
2. For b = 1, 2, … , B repeat:
   a. Fit a tree $\widehat{f^b}$ with d splits (d + 1 terminal nodes) to the training data (X, r)
   b. Update $\hat{f}$ by adding in a shrunken version of the new tree

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   c. Update the residuals

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Boosted Decision Trees

- Learn slowly from shallow trees
- Given a current model – calculate residuals
- Build next tree to improve on the remaining residuals
- Slowly improve where the model does not currently perform well
- Boosted classification becomes a bit trickier in how it updates
- Some key notation reminders from last time:
  - Boosting learns slowly from a bunch of weak classifiers
  - Boosting learns a strong classifier, this model can often be denoted as G, f, or H in common texts. Similarly, weights may vary in notation (w or α for example)

Boosting for Classification

- Consider a dataset $D = \{(x_p, y_p)\}_{p=1}^{P}$, where $y \in \{-1, +1\}$
- Would like to learn:

$$F(x) = w_o + \sum_{m=1}^{M} w_m f_m(x)$$

- So that we may classify based upon

$$F(x) = sign(w_o + \sum_{m=1}^{M} w_m f_m(x))$$

Boosting for Classification: Learning the Boundary

- Consider a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, where $y \in \{-1, +1\}$
- Let's revisit Misclassification, just like with Logistic Regression:

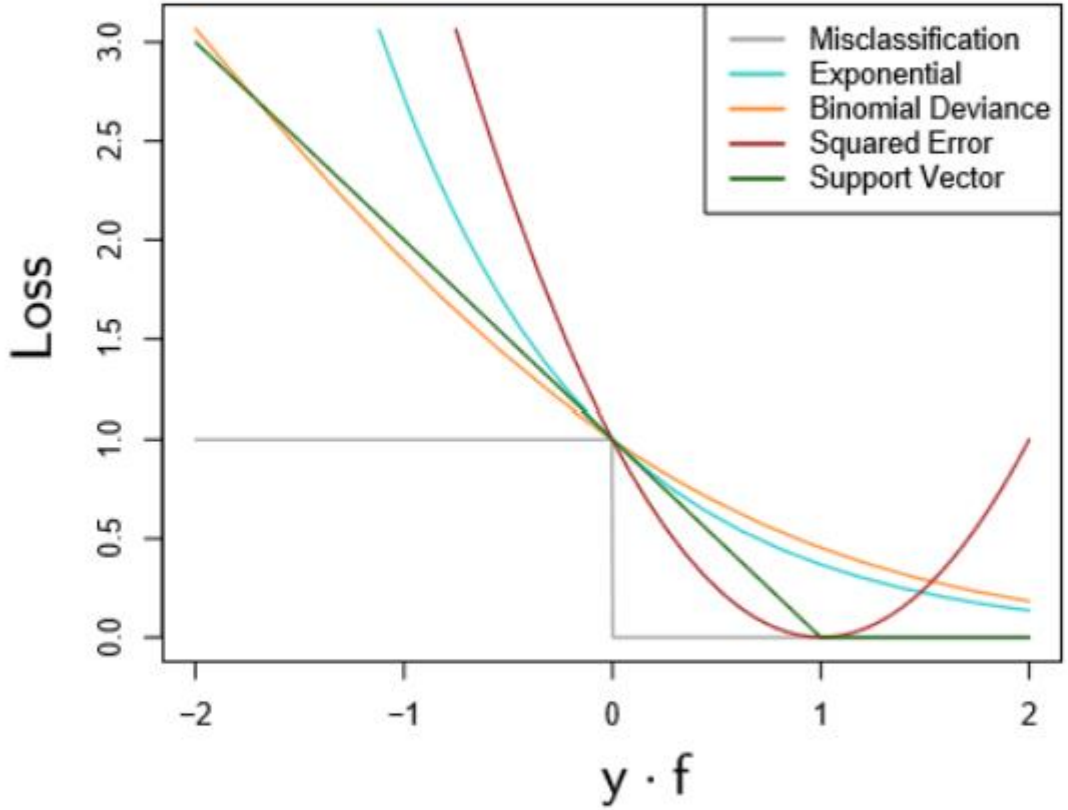
$$L\big(y, f(x)\big) = \overline{Err} = \frac{1}{N}\sum_{i=1}^{N} I\,(y_i \neq \hat{y}_i)$$

- Where we seek to minimize:

$$\min_f \sum_{p=1}^{P} L(y_p, f\big(x_p\big))$$

- Where $L\big(y, f(x)\big)$ is a loss function set up above as the 0-1 binary loss

Types of Loss: A Review



Convex Optimizations of Binary Loss

- Reminder – the Binary 0-1 loss is not differentiable
- Might consider the squared error loss

$$f^*(x) = \underset{f}{\operatorname{argmin}}\, E_{y\,|\,x}\left[(Y - f(x))^2\right] = E[Y \mid X]$$

Convex Optimizations of Binary Loss

- Reminder – the Binary 0-1 loss is not differentiable
- Might consider the squared error loss

$$f^*(x) = \underset{f}{\operatorname{argmin}} E_{y\,|\,x}\left[(Y - f(x))^2\right] = E[Y \mid X]$$

- Log Loss (Binomial Deviance):

$$f(x) = \log(1 + e^{-2yf(x)})$$

Convex Optimizations of Binary Loss

- Reminder – the Binary 0-1 loss is not differentiable
- Might consider the squared error loss

$$f^*(x) = \underset{f}{\operatorname{argmin}} E_{y\,|\,x}\left[(Y - f(x))^2\right] = E[Y \mid X]$$

- Log Loss:

$$f(x) = \log(1 + e^{-yf(x)})$$

- Exponential Loss:

$$L(y, f(x)) = \exp(-y * f(x))$$

Convex Optimizations of Binary Loss

- Reminder – the Binary 0-1 loss is not differentiable
- Might consider the squared error loss

$$f^*(x) = \underset{f}{\operatorname{argmin}} E_{y\,|\,x}\left[(Y - f(x))^2\right] = E[Y \mid X]$$

- Log Loss:

$$f(x) = \log(1 + e^{-yf(x)})$$

- Exponential Loss:

$$L(y, f(x)) = \exp(-y * f(x))$$

Turns out, these two have same optimal solution

Learning from Exponential Loss

- Exponential Loss:

$$L(y, f(x)) = \exp(-y * f(x))$$

- Means we would like to solve:

$$(\beta_m, F_m) = \underset{\beta, F}{\operatorname{argmin}} \sum_{i=1}^{N} exp[-y_i(f_m(x_i) + \beta F(x_i))]$$

(this is called forward stagewise additive modeling)

Learning from Exponential Loss

- Exponential Loss:

$$L(y, f(x)) = \exp(-y * f(x))$$

- Means we would like to solve:

$$(\beta_m, F_m) = \underset{\beta, F}{\operatorname{argmin}} \sum_{i=1}^{N} exp[-y_i(f_m(x_i) + \beta F(x_i))]$$

- Which can be re-written as:

$$(\beta_m, F_m) = \underset{\beta, F}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} exp[-\beta y_i F(x_i)]$$

- Where:

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

Learning from Exponential Loss

- Exponential Loss Optimization:

$$(\beta_m, F_m) = \underset{\beta, F}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} exp[-\beta y_i F(x_i)]$$

- For $\beta > 0$, to minimize misclassifications:

$$F_m = \underset{F}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} I\big(y_i \neq F(x_i)\big)$$

$$= e^{-\beta} \sum_{y_i = F(x_p)} w_p^{(m)} + e^{\beta} \sum_{y_p \neq F(x_p)} w_p^{(m)}$$

$$= \left(e^{\beta} - e^{-\beta}\right) \sum_{p=1}^{P} w_p^{(m)} I\left(y_p \neq F(x_p)\right) + e^{\beta} \sum_{p=1}^{P} w_p^{(m)}$$

Exponential Loss = Log Loss

- Plugging back to solve for β_m yields:

$$\beta_m = \frac{1}{2}\log\frac{1 - err_m}{err_m}$$

- Where

$$err_m = \frac{\sum_{p=1}^{P} w_p^{(m)} I\left(y_p \neq F(x_p)\right)}{\sum_{p=1}^{P} w_p^{(m)}}$$

Scribe Notes: Use the equation from the previous slides to derive the optimal value for β_m

Creating Update Rules

- Update our approximate f as

$$f_m(x) = f_{m-1}(x) + \beta_m F_m(x)$$

- Which updates weights as

$$w_p^{(m+1)} = w_p^{(m)} e^{\alpha_m I\left(y_p \neq F_m(x_p)\right)} e^{\beta_m}$$

- Where

$$\alpha_m = 2\beta_m$$

Convex Optimizations of Binary Loss

- Reminder – the Binary 0-1 loss is not differentiable
- Might consider the squared error loss

$$f^*(x) = \underset{f}{\operatorname{argmin}} E_{y\,|\,x}\left[(Y - f(x))^2\right] = E[Y \mid X]$$

- Log Loss:

$$f(x) = \log(1 + e^{-yf(x)})$$

- Exponential Loss:

$$L(y, f(x)) = \exp(-y * f(x))$$

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y \mid \mathbf{x}_i]$	L2Boosting
Absolute error	$\lvert y_i - f(\mathbf{x}_i) \rvert$	$\operatorname{sgn}(y_i - f(\mathbf{x}_i))$	$\operatorname{median}(y \mid \mathbf{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

AdaBoost M1

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

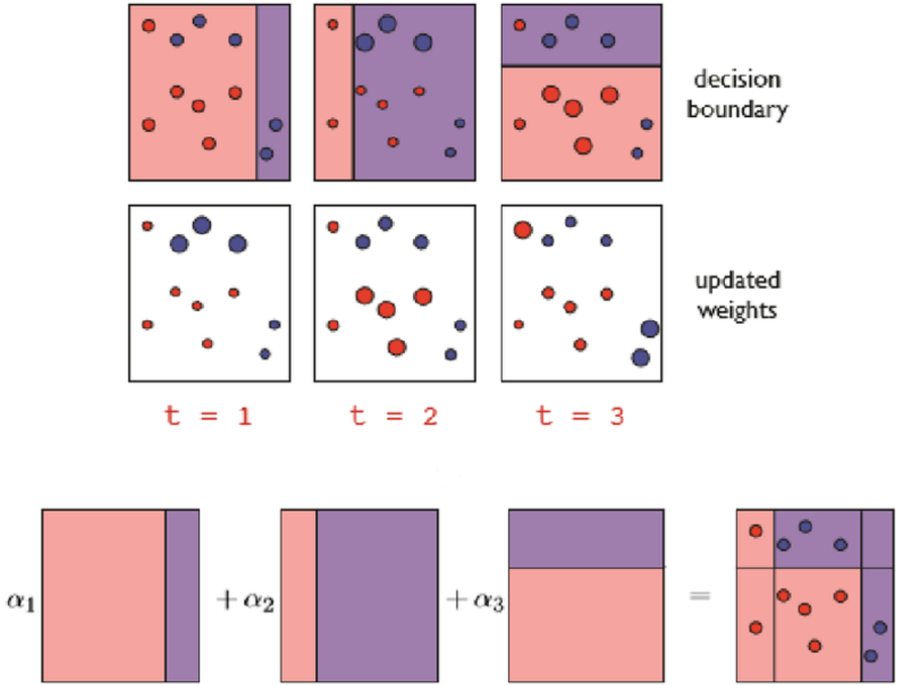
AdaBoost M1

1. Initialize observation weights to $w_i = \frac{1}{N}, i = 1, 2, \ldots, N$
2. For m = 1 to M:
   a. Fit a classifier $f_m(x)$ to the training data using weights w_i
   b. Compute error as:
   $$err_m = \frac{\sum_{i=1}^{N} w_i \ I\ (y_i \neq f_m(x_i))}{\sum_{p=1}^{P} w_p}$$
   c. Compute classifier weight as:
   $$\alpha_m = \log \frac{1 - err_m}{err_m}$$
   d. Re-weigh observations as:
   $$w_i \leftarrow w_i * \exp\left[\alpha_m * I\left(y_i \neq f_m(x_i)\right)\right] \ \forall i$$

AdaBoost M1

1. Initialize observation weights to $w_i = \frac{1}{N}, i = 1, 2, \dots, N$
2. For m = 1 to M:
   a. Fit a classifier $f_m(x)$ to the training data using weights w_i
   b. Compute error as:
   $$err_m = \frac{\sum_{i=1}^{N} w_i \; I\,(y_i \neq f_m(x_i))}{\sum_{p=1}^{P} w_p}$$
   c. Compute classifier weight as:
   $$\alpha_m = \log\frac{1 - err_m}{err_m}$$
   d. Re-weigh observations as:
   $$w_i \leftarrow w_i * \exp\left[\alpha_m * I\left(y_i \neq f_m(x_i)\right)\right] \; \forall\, i$$
3. Output:
$$F(x) = sign(\sum_{m=1}^{M} \alpha_m f_m(x))$$

Visualization of AdaBoost



Other Boosting: Logit Boost

- AdaBoost with exponential loss puts a lot of weight on misclassified examples
- Hard to interpret probabilities from f(x)
- If we use log-loss instead of exponential – mistakes are only punished linearly
- And this generalizes to multiple classes

$$p(y = 1\,|x) = \frac{e^{f(x)}}{e^{-f(x)} + e^{f(x)}} = \frac{1}{1 + e^{-2f(x)}}$$

With loss

$$L_m(\phi) = \sum_{p=1}^{P} \log(1 + \exp(-2y_p\left(f_{m-1}(x_p) + \phi(x_p)\right))$$

Logit Boost

1. Initialize observation weights to $w_p = \frac{1}{P}, p = 1, 2, \ldots, P$, $\pi_p = \frac{1}{2}$
2. For m = 1 to M:
   a. Compute working response $z_p = \frac{y_p - \pi_p}{\pi_p\,(\,1 - \pi_p)}$
   b. Compute weights $w_p = \pi_p$ (1 - π_p)
   c. Update

$$\phi_m = \underset{\varphi}{\operatorname{argmin}} \sum_{p=1}^{P} w_p (z_p - \phi(x_p))^2$$

$$f(x) \leftarrow f(x) + \frac{1}{2}\phi_m(x)$$

   d. Compute

$$\pi_p = \frac{1}{1 + e^{-2f(x_p)}}$$

3. Output:

$$F(x) = sign(\sum_{m=1}^{M} \phi_m(x))$$

Can we generalize further?

- Rather than rebuilding the algorithm per loss function – can we create a generic boosting algorithm across all loss functions?
- Imagine we want

$$\hat{f} = \underset{f}{\operatorname{argmin}} L(f)$$

Where f are the parameters of a model

- At step m let g_m be the gradient of L(f) at step f_{m-1}

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

Functional Gradient Descent

$$g_{pm} = \left[\frac{\partial L(y_p, f(x_p))}{\partial f(x_p)}\right]$$

$$f_m = f_{m-1} - \rho_m g_m$$

Where ρ_m is a step length set by

$$\rho_m = \underset{\rho}{\operatorname{argmin}} L(f_{m-1} - \rho_m g_m)$$

This will not generalize, but optimize f for only a fixed size P. So we have to fit weak learners to approximate the negative gradient signal as

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} (-g_{im} - \phi(x_i; \gamma))^2$$

Gradient Boosting

1. Initialize $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \varphi(x_i))$
2. For m = 1 to M:
   a. Compute the residual gradient using
   $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]$$
   b. Fit a weak learner, $\phi(x_i)$, to $\{(\mathrm{x_i}, \mathrm{r_{im}})\}_{\mathrm{i=1}}^{\mathrm{N}}$
   c. Use a weak learner to compute γ_m
   $$\gamma_m = \underset{\gamma}{argmin} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma\varphi(x_i))$$
   c. Update
   $$F_m(x) = F_{m-1}(x) + \upsilon\phi(x; \gamma_m)$$
3. Output:
$$F(x) = F_m(x)$$

Takeaways

- Reviewed a variety of boosting algorithms for classification
- Discussed why functional gradient boosting generalizes across all kinds of loss functions
- Boosting of very weak learners creates a stronger learner over number of terms/iterations