CSCE 633: Machine Learning

Lecture 19: Boosting

Texas A&M University

10-7-19

Last Time

- Decision Trees
- Random Forest

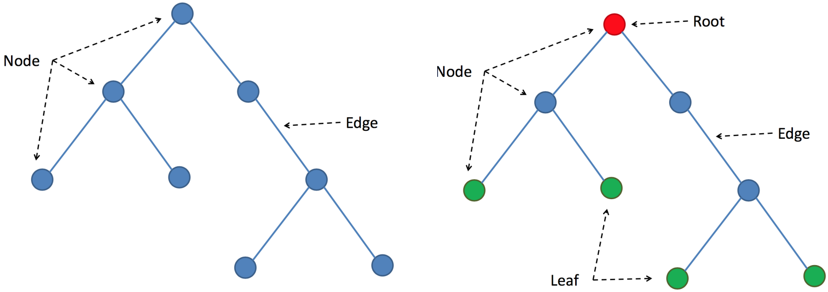
Goals of this lecture

- Reminder: Exam 1 - Monday, October 14 in class
- CLOSED BOOK, CLOSED NOTES - Starts right at 1:50. DO NOT BE LATE! YOU WILL NOT GET EXTRA TIME!
- Boosting

Decision Trees

What is a decision tree

A hierarchical data structure implementing the divide-and-conquer strategy for decision making



Can be used for both classification & regression

Gini Index and Entropy

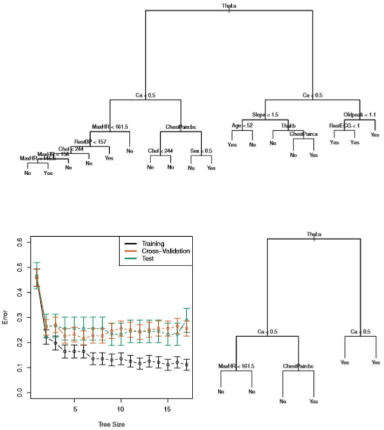
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

, which measures the total variance across K classes. This is a measure of node purity.

$$H = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

, Entropy which takes a value near 0 if all the $\hat{p}$ are near zero or one - smaller value if node is pure

Classification and Pruning



Bagging

- Take B different bootstraps of our one dataset
- $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{*B} \hat{f}^{b}(x)$
- Turns out, you can grow these trees without pruning
- Regression - average the values from each tree
- Classification - majority vote from each tree
- Test error can be plotted as a function of B
- B is not a critical parameter (will see shortly) so large B does not mean we overfit

Out of Bag Error

- If we repeatedly fit bootstrapped subsets (say 2/3 of data)
- Each time we are left with 1/3 of the data we can call out of bag
- We can estimate error for this - called Out of Bag Estimation

- set $m \approx \sqrt{p}$

Random Forest

- set $m \approx \sqrt{p}$
- Each time $\frac{p-m}{p}$ predictors aren't even considered

Random Forest

- set $m \approx \sqrt{p}$
- Each time $\frac{p-m}{p}$ predictors aren't even considered
- other predictors have a chance

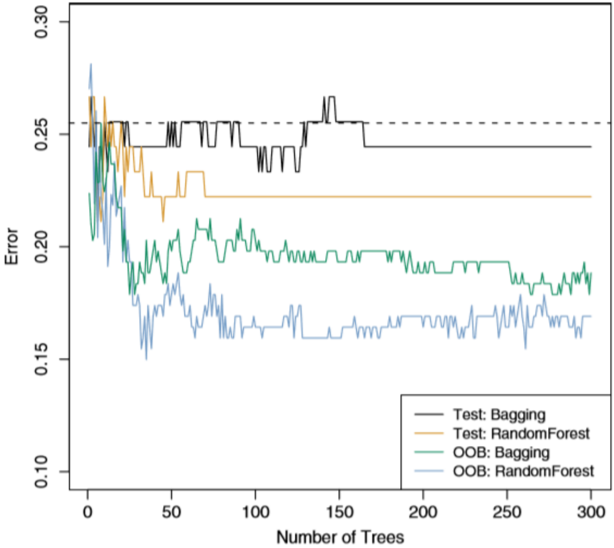
Random Forest

- set $m \approx \sqrt{p}$
- Each time $\frac{p-m}{p}$ predictors aren't even considered
- other predictors have a chance
- Turns out, this process decorrelates trees

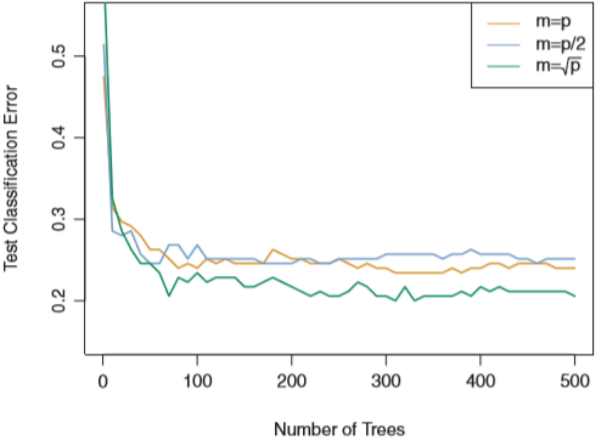
Random Forest

- set $m \approx \sqrt{p}$
- Each time $\frac{p-m}{p}$ predictors aren't even considered
- other predictors have a chance
- Turns out, this process decorrelates trees
- The average tree becomes less variable and thus more reliable

Example: Heart Dataset



RF with different m



Boosting

- Can improve prediction of decision trees even further
- develop a method that can work on any classifier - applied here to regression trees
- Bagging - build each tree randomly
- Random Forest - build each tree randomly, with random variations in predictors allowed
- What if we built trees sequentially?

Boosting

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat: for outliers it's easy to overfit when there are too many trees
   (a) Fit a tree $\hat{f}^b$ 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:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

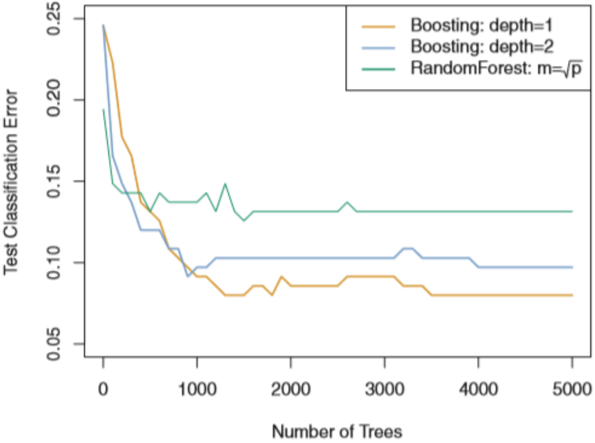
Boosting

- Learn slowly on shallow trees
- Given a current model (number of trees) - calculate residuals
- Build next tree to improve, iteratively, on the residuals
- Slowly improve $\hat{f}$ where it does not perform well!
- Boosted classification is a bit trickier - next time

Boosting: Hyperparameters

- Number of trees - if B is too large, this model DOES overfit
- λ is small, but greater than 0. Typically $0.001 < \lambda < 0.01$
- Depth d of trees is often small, often $d = 1$ decision stumps (very interpretable)
- Boosts a bunch of weak classifiers into a strong classifier.

Boosting



Boosting: depth=1
Boosting: depth=2
RandomForest: m=√p
Test Classification Error
0.05
0.10
0.15
0.20
0.25
0
1000
2000
3000
4000
5000
Number of Trees

Boosting:Formulation

$$f(x) = \beta_0 + \sum_{b=1}^{B} \beta_m \phi_m(x)$$

Or - in other notation

$$f(x) = w_0 + \sum_{m=1}^{M} w_m \phi_m(x)$$

Boosting:Formulation

Consider $y \in \{-1, +1\}$ instead of $\{0, 1\}$
Can calculate mean error in a classifier f as:

$$\bar{err} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(y_i \neq f(x_i))$$

Some definitions:

- A weak classifier is one that is just slightly better than random guessing.
- we define $m = 1, 2, \cdots, M$ as a sequence of weak classifier models
- You may see formulations of a strong classifier as G f or H - usually capital to denote the strong classifier from weak
- You may see α weights instead of w weights

Boosting:Formulation

$$f(x) = \beta_0 + \sum_{b=1}^{B} \beta_m \phi_m(x)$$

Or - in other notation

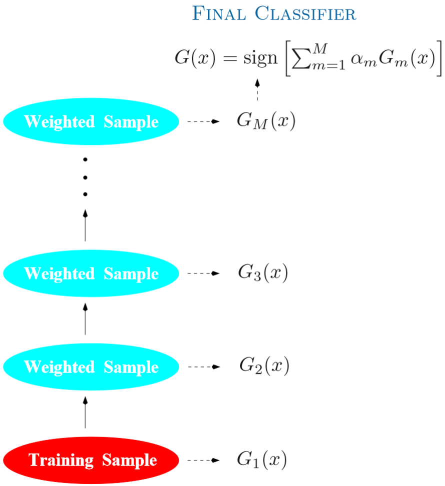
$$f(x) = w_0 + \sum_{m=1}^{M} w_m \phi_m(x)$$

Or - in other notation

$$G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$$

where α_m are the weights on each weak classifier G_m and the binary classification is just the sum (regression would be mapped to a $[-1, 1]$ interval in the same setting (called Real AdaBoost)

AdaBoost



FINAL CLASSIFIER
$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
Weighted Sample
$G_M(x)$
Weighted Sample
$G_3(x)$
Weighted Sample
$G_2(x)$
Training Sample
$G_1(x)$

Boosting:Minimizing Loss

Consider $y \in \{-1, +1\}$ instead of $\{0, 1\}$
Can calculate mean error in a classifier f as:

$$\bar{err} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i \neq f(x_i))$$

Boosting aims to minimize loss as:

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

Where $L(y_i, \hat{(}y_i))$ is some loss function. What does this look like visually for binary classification?

Consider $y \in \{-1, +1\}$ instead of $\{0, 1\}$
Binary $0 - 1$ loss is not differentiable - so we need convex approximations.

Squared Error Loss

$$f^*(x) = argmin_{f(x)} \mathbb{E}_{y|x}[(Y - f(x))^2] = \mathbb{E}[Y|x]$$

Cannot compute this because it requires $p(y|x)$ to be known. This is commonly known as the population minimizer. Other loss functions with boosting, then, try to approximate this probability.

Consider $y \in \{-1, +1\}$ instead of $\{0, 1\}$
Binary $0 - 1$ loss is not differentiable - so we need convex approximations.

Squared Error Loss

$$f^*(x) = argmin_{f(x)} \mathbb{E}_{y|x}[(Y - f(x))^2] = \mathbb{E}[Y|x]$$

Log Loss:

$$f^*(x) = \frac{1}{2} \log \frac{p(\tilde{y} = 1|x)}{p(\tilde{y} = -1|x)}$$

Exponential Loss:

$$L(\tilde{y}, f) = \exp(-\tilde{y} f)$$

Which has the same optimal estimate as log-loss it turns out!

Initially:

$$f_0(x) = argmin_{\gamma)} \sum_{i=1}^{n} L(y_i, f(x_i; \gamma))$$

For squared error, can start with $f_0(x) = \bar{y}$ Then:

$$f_0(x) = \frac{1}{2} \log \frac{\hat{\pi}}{1 - \hat{\pi}}$$

where:

$$\hat{\pi} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(y_i = 1)$$

Initially:

$$f_0(x) = argmin_{\gamma)} \sum_{i=1}^{n} L(y_i, f(x_i; \gamma))$$

For squared error, can start with $f_0(x) = \bar{y}$ Then:

$$f_0(x) = \frac{1}{2} \log \frac{\hat{\pi}}{1 - \hat{\pi}}$$

Iterating:

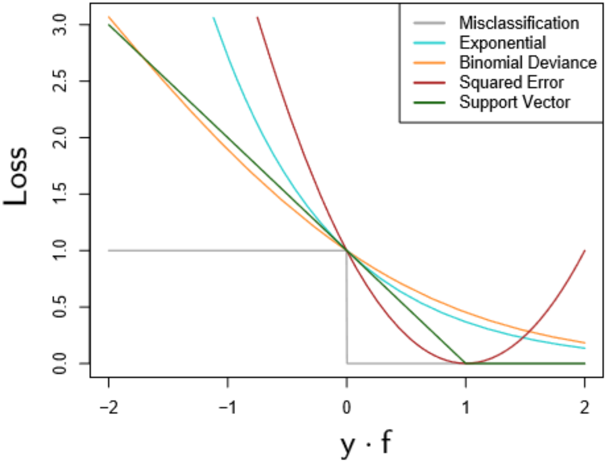
$$f_m(x) = f_{m-1}(x) + \nu \beta_m \phi(x; \gamma_m)$$

Where $0 < \nu \leq 1$ is a shrinkage parameter to make sure you learn slowly slowly!
Early Stopping: If accuracy (loss) does not improve on a validation set, or AIC, BIC, etc.

Types of Loss

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	$\lvert y_i - f(\mathbf{x}_i) \rvert$	$\mathrm{sgn}(y_i - f(\mathbf{x}_i))$	$\mathrm{median}(y \mid \mathbf{x}_i)$	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

Types of Loss



AdaBoost M1

Algorithm 16.2: Adaboost.MI, for binary classification with exponential loss

$w_i = 1/N$;
for $m = 1 : M$ **do**
  Fit a classifier $\phi_m(\mathbf{x})$ to the training set using weights $\mathbf{w}$;
  Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^{N} w_{i,m})}$;
  Compute $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$;
  Set $w_i \leftarrow w_i \exp[\alpha_m \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))]$;

Return $f(\mathbf{x}) = \text{sgn}\left[\sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x})\right]$;

Logit Boost

- Adaboost with exponential loss puts a lot of weight on misclassified examples.
- Additionally - it is hard to interpret probabilities from f(x)
- If we use log-loss instead - mistakes are only punished linearly
- Can also generalize to multiple classes

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

$$L_m(\phi) = \sum_{i=1}^{n} log(1 + \exp(-2\tilde{y}_i(f_{m-1}(x) + \phi(x_i)))$$

Logit Boost: Algorithm

Algorithm 16.3: LogitBoost, for binary classification with log-loss

$w_i = 1/N$, $\pi_i = 1/2$;
for $m = 1 : M$ **do**
  Compute the working response $z_i = \frac{y_i^* - \pi_i}{\pi_i(1-\pi_i)}$;
  Compute the weights $w_i = \pi_i(1 - \pi_i)$;
  $\phi_m = \operatorname{argmin}_\phi \sum_{i=1}^N w_i(z_i - \phi(\mathbf{x}_i))^2$;
  Update $f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \frac{1}{2}\phi_m(\mathbf{x})$;
  Compute $\pi_i = 1/(1 + \exp(-2f(\mathbf{x}_i)))$;
Return $f(\mathbf{x}) = \operatorname{sgn}\left[\sum_{m=1}^M \phi_m(\mathbf{x})\right]$;

Gradient Descent Boosting

- Rather than rebuild the method per loss function, can we generalize?
- Imagine we want to minimize $\hat{f} = argmin_f L(f)$ where the f are the parameters of a model
- Then at step m let g_m be the gradient of $L(f)$ at step $f = f_{m-1}$

$$g_{im} = [\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]$$

Types of Loss

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y\vert\mathbf{x}_i]$	L2Boosting
Absolute error	$\vert y_i - f(\mathbf{x}_i)\vert$	$\text{sgn}(y_i - f(\mathbf{x}_i))$	$\text{median}(y\vert\mathbf{x}_i)$	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

$$g_{im} = [\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]$$

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

where ρ_m is the step length and

$$\rho = argmin_{\rho} L(f_{m-1} - \rho g_m)$$

But this does not generalize, only optimizes f for a fixed n. so we have to fit weak learners to approximate the negative gradient signal

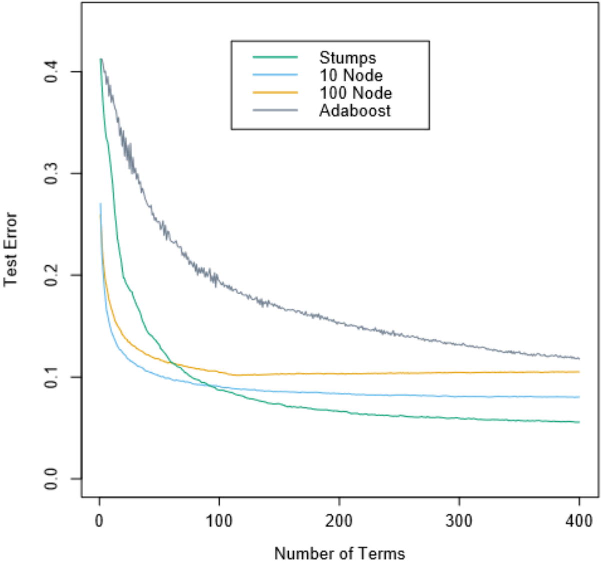
$$\gamma_m = argmin_{\gamma} \sum_{i=1}^{n} (-g_{im} - \phi(x_i; \gamma))^2$$

Gradient Descent Boosting: Algorithm

Algorithm 16.4: Gradient boosting

Initialize $f_0(\mathbf{x}) = \mathrm{argmin}_{\boldsymbol{\gamma}} \sum_{i=1}^{N} L(y_i, \phi(\mathbf{x}_i; \boldsymbol{\gamma}))$;
for $m = 1 : M$ **do**
  Compute the gradient residual using $r_{im} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i)}$;
  Use the weak learner to compute $\boldsymbol{\gamma}_m$ which minimizes $\sum_{i=1}^{N}(r_{im} - \phi(\mathbf{x}_i; \boldsymbol{\gamma}_m))^2$;
  Update $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu\phi(\mathbf{x}; \boldsymbol{\gamma}_m)$;
Return $f(\mathbf{x}) = f_M(\mathbf{x})$

Boosting Comparisons



Takeaways and Next Time

- Boosting
- Next Time: More Boosting
- Reminder: Exam 1 - Monday, October 14