Boosting Extras

-Introduction to Data Science-

Lecturer: Darren Homrighausen, PhD

Boosting

Summary: In the previous lecture, we made the following observations

1. In low dimensions, we can flexibly fit a local average to estimate the Bayes' rule
2. In higher dimensions, we need to limit the flexibility

   (E.g. with generalized additive models (GAMs))
3. These models are more flexible than linear models, but still encode strong assumptions

   (For instance, no interactions between features)
4. We specified a more general additive model:

$$f(X) = \sum_{b=1}^{B} \beta_b \phi_b(X)$$

but claimed that this can be difficult to fit or it can lead to overfitting

Boosting

Question: Why does boosting work/what does it do?

One answer: Boosting greedily fits the general additive model

$$f(X) = \sum_{b=1}^{B} \beta_b \phi_b(X)$$

The details about

- finding the coefficients β_b
- the functions ϕ_b
- the link function between the additive model and Y

all differ in different boosting algorithms

Boosting approaches

There are many variations on the boosting paradigm

The first major boosting algorithm was called Adaboost

Leveraging the connection to additive models, there were soon many others

(Chronologically speaking, we are talking about the early 2000's)

- Gradient boosting machines (GBM)
- logitBoost
- GentleBoost
- LpBoost
- RobustBoost
- ..

Gradient Boosting Machine (GBM)

This is the idea behind GBM

- GBM seeks to minimize the training error via gradient descent
- However, for additive models, the 'gradient' is with respect to a function
- This leads to several problems

  (The gradient isn't uniquely defined with respect to finite data, can't do predictions at feature values not in training set, ...)
- Hence, the gradient is restricted to simple procedures in a smoothing step

  (Perhaps trees with few splits!)

Gradient Boosting Machine

Let ℓ be a loss function and R be the risk

(Example: $\ell(f(X), Y) = ||f(X) - Y||_2^2$ and $R(f) = \mathbb{E}\ell(f(X), Y)$)

We can seek to minimize R via gradient descent

$$\text{The gradient:} \quad \nabla R(f) = \frac{\partial R(f)}{\partial f} = \mathbb{E}\left[\frac{\partial \ell(f(X), Y)}{\partial f}\right]$$

Gradient Boosting Machine

Functional gradient descent would look like:

For $b = 1, \ldots, B$

$$\hat{f}_b = \hat{f}_{b-1} - \lambda \nabla R(\hat{f}_{b-1})$$

(λ is the learning rate)

The problem: The training data based version of $\nabla R(\hat{f}_{b-1})$:

$$\underbrace{\mathbb{E}\left[\frac{\partial \ell(f(X), Y)}{\partial f}\right]}_{\text{population version}} \longrightarrow \underbrace{\frac{1}{n}\sum_{i=1}^{n} \frac{\partial \ell(f(X_i), Y_i)}{\partial f}}_{\text{training version}}$$

is not well-defined

Gradient Boosting Machine

In practice, we specify a relatively low complexity class of functions $\mathcal{F}$

(Example: $\mathcal{F}$ could be the set of one split decision trees)

The idea is now to find a base learner $\hat{f} \in \mathcal{F}$ that is most similar to ∇R

This means finding the closest procedure in $\mathcal{F}$ via squared error distance over training data

GBM Algorithm

Set

$$\hat{f}_0 \leftarrow \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(f(X_i), Y_i)$$

and $R = Y$

For $b = 1, \ldots, B$:

1. $R_i \leftarrow - \left[\frac{\partial \ell(f(X_i) Y_i)}{\partial f} \right] \Big|_{\hat{f}_{b-1}}$
2. $\hat{f} \leftarrow \operatorname{argmin}_{f \in \mathcal{F}} \sum_{i=1}^{n} (R_i - f(X_i))^2$
3. Update:

$$\hat{f}_b \leftarrow \hat{f}_{b-1} + \lambda \hat{f}$$

Functional Gradient descent

Let's look at step 1. more closely. If using squared error loss

$$\frac{\partial \ell(f(X_i), Y_i)}{\partial f} = \frac{\partial (f(X_i) - Y_i)^2}{\partial f} = 2(f(X_i) - Y_i)$$

Observation: These are (twice) the residuals

(Hence, sometimes the loss is written $(f(X) - Y)^2/2$)

Functional Gradient descent

Reminder: Back to boosting. Fix any b

1. Fit $\hat{f}_b$ with $M+1$ regions to $\tilde{\mathcal{D}} = \{(X_1, R_1), \ldots, (X_n, R_n)\}$
2. Update: $\hat{f} \leftarrow \hat{f} + \lambda \hat{f}_b$
3. Update: $R \leftarrow R - \lambda \hat{f}_b$

Compare: Functional gradient descent:

1. $R_i \leftarrow -\frac{\partial \ell(f(X_i), Y_i)}{\partial f}\Bigg|_{f=\hat{f}_{b-1}} = 2(R_i - \hat{f}_{b-1}(X_i))$
2. $\hat{f} \leftarrow \text{argmin}_{f \in \mathcal{F}} ||R - f||_2^2$

   (Smoothing step, let $\mathcal{F}$ be class of trees with $M+1$ regions)
3. Update: $\hat{f}_b \leftarrow \hat{f}_{b-1} + \lambda \hat{f}$

Boosting

Conclusion: These approaches are the same!

Boosting is an algorithmic way of fitting a general additive model using data

Recap: The following ideas are the same:

- Iteratively refitting a simple procedure to reweighted training data such that observations that have large residual or are misclassified are upweighted
- Minimizing the training error using gradient descent, where the gradient is restricted to being a simple procedure

  (This restricting is done via minimizing squared error)

Boosting for classification

Boosting for classification

We can adapt the GBM paradigm to classification via changing the loss function

This will produce an additive model $\hat{f}(X) = \sum_{b=1}^{B} \lambda \hat{f}_b(X)$

(Again, the $\hat{f}_b$ are the fitted base learner that minimizes the loss)

This gets converted to classifier via $\mathrm{sgn}(\hat{f}(X))$

We can equivalently express this via the product $\hat{f}(X)Y$, where (X, Y) are a feature/supervisor pair

$$\hat{f}(X)Y \text{ is } \begin{cases} > 0 & \text{if correct classification} \\ < 0 & \text{if incorrect classification} \end{cases}$$

$\rightarrow$ loss functions for classification should seek to make $\hat{f}(X)Y$ as large as possible

Boosting for classification

The specifics of the type of boosting depend on

- The loss function ℓ
- The simple functions $\mathcal{F}$
- The learning rate λ

For regression, ℓ is commonly squared error loss

(Least-absolute deviation also exists for robust boosting)

For classification, ℓ is usually either:

- Adaboost
- Bernoulli
- squared error loss

  (This is not advisable, however)

Boosting for classification

More details:

- Adaboost: Iteratively fits a classifier on reweighted training data such that misclassified observations are upweighted.

  It turns out this is equivalent to doing GBM with the exponential loss function:

  $$\ell(f(X), Y) = \exp\{-f(X)Y\}$$

- Logistic: If we assume a Bernoulli distribution with logistic loss, we acquire another GBM:

  $$\ell(f(X), Y) = \log\left(1 + \exp\{-2Yf(X)\}\right)$$

  (Note that the '2' is from us defining $Y \in \{-1, 1\}$ whereas the Bernoulli is commonly written $Y \in \{0, 1\}$ and hence our label is "twice" the usual label)

Boosting for classification

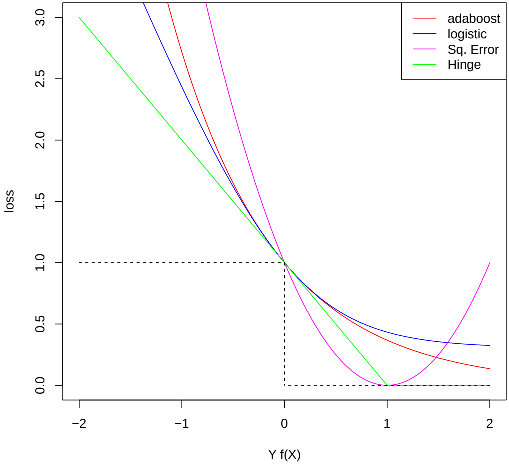

Figure: Here, I've rescaled the logistic loss so that it is easier to compare to Adaboost

Adaboost

AdaBoost outline

We give an overview of 'AdaBoost.M1.'

(Freund and Schapire (1997))

Select a base classifier that you want to boost

(E.g. a tree with a very small number of terminal nodes)

First, train the base classifier as usual on the training data $\mathcal{D}$

Then start iterating $b = 1, 2, \ldots B$,

At each step b,

1. the observations are re-weighted to increase the weights of misclassified observations

   (Implicitly, this lowers the weight on correctly classified observations)

2. A new classifier is trained on the re-weighted training data

(Discrete) AdaBoost algorithm

Assume $Y \in \{-1, 1\}$

1. Initialize $w_i \equiv 1/n$ for $i = 1, \ldots, n$
2. For $b = 1, \ldots, B$
   2.1 Fit the base classifier on $\mathcal{D}$, weighted by $w_i \to \hat{g}_b$
   2.2 Compute

$$\hat{R}_b = \frac{\sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq \hat{g}_b(X_i))}{\sum_{i=1}^{n} w_i}$$

   2.3 Find $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$
   2.4 Set $w_i \leftarrow w_i \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$
3. Output: $\hat{g}(X) = \mathrm{sgn}\left(\sum_{b=1}^{B} \hat{\beta}_b \hat{g}_b(X)\right)$

(Discrete) AdaBoost interpretation

Forward stepwise additive modeling:

(Using a general loss ℓ)

1. $\hat{\beta}_b, \hat{\theta}_b = \text{argmin}_{\beta,\theta} \sum_{i=1}^{n} \ell(Y_i, \hat{f}(X_i) + \beta\phi(X_i, \theta))$
2. Set $\hat{f}(X) \leftarrow \hat{f}(X) + \hat{\beta}_b \phi(X; \hat{\theta}_b)$

AdaBoost implicitly does this by use of the exponential loss function

$$\ell(Y, f) = \exp\{-Yf(X)\}$$

and basis functions $\phi(X, \theta) = g_b(X)$

AdaBoost intuition

Suppose we minimize exponential loss in a forward stepwise manner

Doing the forward selection for this loss, we get

$$(\hat{\beta}_b, \hat{g}_b) = \underset{\beta, g}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\{-Y_i(\hat{f}(X_i) + \beta g(X_i))\}$$

AdaBoost intuition

Rewriting:

$$(\hat{\beta}_b, \hat{g}_b) = \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\{-Y_i(\hat{f}(X_i) + \beta g(X_i))\}$$

$$= \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\{-Y_i \hat{f}(X_i)\} \exp\{-Y_i \beta g(X_i))\}$$

$$= \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \exp\{-Y_i \beta g(X_i)\}$$

Where

- Define $w_i = \exp\{-Y_i \hat{f}(X_i)\}$
  (This is independent of β, g)
- $\sum_{i=1}^{n} w_i \exp\{-Y_i \beta g_b(X_i))\}$ needs to be optimized

AdaBoost intuition

Note that

$$\sum_{i=1}^{n} w_i \exp\{-\beta Y_i g(X_i))\} = e^{-\beta} \sum_{i:Y_i=g(X_i)} w_i + e^{\beta} \sum_{i:Y_i \neq g(X_i)} w_i$$

$$= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq g(X_i)) +$$

$$+ e^{-\beta} \sum_{i=1}^{n} w_i$$

As long as $(e^{\beta} - e^{-\beta}) \geq 0$, we can find

$$\hat{g}_b = \underset{g}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq g(X_i))$$

(Note: If $(e^{\beta} - e^{-\beta}) < 0$, then $\beta < 0$. However, as $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$, this implies $\hat{R} > 1/2$. Hence, we would flip the labels and get $\hat{R} \leq 1/2$.)

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $b = 1, \ldots, B$
   2.1 Fit $g_b(x)$ on $\mathcal{D}$, weighted by w_i

   (In this case, we would be growing the (heavily pruned) tree via minimizing misclassifications)

   2.2 Compute

   $$\hat{R}_b = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq \hat{g}_b(X_i))}{\sum_{i=1}^n w_i}$$

   2.3 Find $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$

   2.4 Set $w_i \leftarrow w_i \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$
3. Output: $\hat{g}(X) = \mathrm{sgn}\left(\sum_{b=1}^B \beta_b \hat{g}_b(X)\right)$

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $b = 1, \ldots, B$
    2.1 Fit $g_b(x)$ on $\mathcal{D}$, weighted by w_i

    (In this case, we would be growing the (heavily pruned) tree via minimizing misclassifications)

    2.2 Compute

    $$\hat{R}_b = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq \hat{g}_b(X_i))}{\sum_{i=1}^n w_i}$$

    2.3 Find $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$

    2.4 Set $w_i \leftarrow w_i \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$
3. Output: $\hat{g}(X) = \mathrm{sgn}\left(\sum_{b=1}^B \beta_b \hat{g}_b(X)\right)$

AdaBoost intuition

Goal: Minimize

$$\sum_{i=1}^{n} w_i \exp\{-\beta Y_i \hat{g}_b(X_i))\}$$

We showed this can be written

$$\sum_{i=1}^{n} w_i \exp\{-\beta Y_i \hat{g}_b(X_i))\} = (e^{\beta} - e^{-\beta})\hat{R}_b W + e^{-\beta} W \quad (W = \sum w_i)$$

Take derivative with respect to β

$$(e^{\beta} + e^{-\beta})\hat{R}_b W - e^{-\beta} W \stackrel{set}{=} 0 \stackrel{set}{=} e^{\beta}\hat{R}_b + e^{-\beta}(\hat{R}_b - 1)$$

Solve for β to find $\hat{\beta}_b = 1/2 \log[(1 - \hat{R}_b)/\hat{R}_b]$

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $b = 1, \ldots, B$
   2.1 Fit $g_b(x)$ on $\mathcal{D}$, weighted by w_i
   (This step is finding the next best version of the classifier, trained on weighted data and added to the previous classifiers)
   2.2 Compute
   $$\hat{R}_b = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq \hat{g}_b(X_i))}{\sum_{i=1}^n w_i}$$
   2.3 Find $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$
   2.4 Set $w_i \leftarrow w_i \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$
3. Output: $\hat{g}(x) = \mathrm{sgn}\left(\sum_{b=1}^B \hat{\beta}_b \hat{g}_b(x)\right)$

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $b = 1, \ldots, B$
   2.1 Fit $g_b(x)$ on $\mathcal{D}$, weighted by w_i
   (This step is finding the next best version of the classifier, trained on weighted data and added to the previous classifiers)
   2.2 Compute
   $$\hat{R}_b = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq \hat{g}_b(X_i))}{\sum_{i=1}^n w_i}$$
   2.3 Find $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$
   2.4 Set $w_i \leftarrow w_i \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$
3. Output: $\hat{g}(x) = \mathrm{sgn}\left(\sum_{b=1}^B \hat{\beta}_b \hat{g}_b(x)\right)$

AdaBoost intuition

The approximation is updated

$$\hat{f}(X) \leftarrow \hat{f}(X) + \hat{\beta}_b \hat{g}_b(X)$$

This causes the weights

$$w_i^{(b+1)} = \exp\{-Y_i \hat{f}(X_i)\} = w_i^{(b)} \exp\{-\hat{\beta}_b Y_i \hat{g}_b(X_i)\}$$

Using $-Y_i \hat{g}_b(X_i) = 2\mathbf{1}(Y_i \neq \hat{g}_b(X_i)) - 1$, this becomes

$$w_i^{(b+1)} \propto w_i^{(b)} \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$$

where $\hat{\beta}_b \leftarrow 2\hat{\beta}_b$, giving the last step of the algorithm

Why exponential loss?

AdaBoost was originally the result of a constructive proof

Friedman et al. (2000) showed that it was equivalent to greedily fitting/learning a basis expansion with exponential loss

Note that this is compared to fitting:

$$\min_{(\beta_b),(g_b)} \sum_{i=1}^{n} \ell(Y_i, f(X_i))$$

(Which can be difficult or overfit, e.g. minimizing training error)

versus:

$$\min_{\beta,g} \sum_{i=1}^{n} \ell(Y_i, \hat{f}(X_i) + \beta g(X_i))$$

Additional topics

Boosting: The controversy

Claim: Boosting is another version of bagging

The early versions of Boosting involved (weighted) resampling

Therefore, it was initially speculated that a connection with bagging explained its performance

However, boosting continues to work well when

- The algorithm is trained on weighted data rather than on sampling with weights

  (This removes the randomization component that is essential to bagging)

- Weak learners are used that have high bias and low variance

  (This is the opposite of what is prescribed for bagging)