Gradient Boosting (Continued)

David Rosenberg

New York University

April 4, 2016

Boosting Fits an Additive Model

Adaptive Basis Function Model

- AdaBoost produces a classification score function of the form

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

  - each G_m is a **base classifier**
- The G_m's are like basis functions, but they are learned from the data.
- Let's move beyond classification models...

Adaptive Basis Function Model

- **Base hypothesis space** $\mathcal{H}$
- An **adaptive basis function expansion** over $\mathcal{H}$ is

$$f(x) = \sum_{m=1}^{M} \nu_m h_m(x),$$

  - $h_m \in \mathcal{H}$ chosen in a learning process ("adaptive")
  - $\nu_m \in \mathbf{R}$ are **expansion coefficients.**
- **Note:** We are taking linear combination of outputs of $h_m(x)$.
  - Functions in $h_m \in \mathcal{H}$ must produce values in $\mathbf{R}$ (or a vector space)

How to fit an adaptive basis function model?

- **Loss function**: $\ell(y, \hat{y})$
- **Base hypothesis space**: $\mathcal{H}$ of **real-valued** functions
- The combined hypothesis space is then

$$\mathcal{F}_M = \left\{ f(x) = \sum_{m=1}^{M} \nu_m h_m(x) \right\}.$$

- Find $f \in \mathcal{F}_M$ that minimizes some objective function $J(f)$. e.g. penalized ERM:

$$\hat{f} = \underset{f \in \mathcal{F}_M}{\arg\min} \left[\frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f(x_i)\right) + \Omega(f) \right].$$

- To fit this, we'll proceed in stages, adding a new h_m in every stage.

Forward Stagewise Additive Modeling (FSAM)

- Start with $f_0 \equiv 0$.
- After $m-1$ stages, we have

$$f_{m-1} = \sum_{i=1}^{m-1} \nu_i h_i,$$

  where $h_1, \ldots, h_{m-1} \in \mathcal{H}$.

- Want to find
  - **step direction** $h_m \in \mathcal{H}$ and
  - **step size** $\nu_i > 0$
- so that

$$f_m = f_{m-1} + \nu_i h_m$$

  improves objective function value by as much as possible.

Forward Stagewise Additive Modeling for ERM

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   1. Compute:
   $$(\nu_m, h_m) = \operatorname*{arg\,min}_{\nu \in \mathbf{R}, h \in \mathcal{H}} \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i) \underbrace{+\nu h(x_i)}_{\text{new piece}}\right).$$
   2. Set $f_m = f_{m-1} + \nu_m h$.
3. Return: f_M.

Exponential Loss and AdaBoost

- Take loss function to be
$$\ell(y, f(x)) = \exp(-yf(x)).$$
- Let $\mathcal{H}$ be our base hypothesis space of classifiers $h : \mathcal{X} \to \{-1, 1\}$.
- Then Forward Stagewise Additive Modeling (FSAM) reduces to AdaBoost!
  - Proof on Homework #6 (and see HTF Section 10.4).
- Only difference:
  - AdaBoost gets whichever G_m the base learner returns from $\mathcal{H}$ – no guarantees it's best in $\mathcal{H}$.
  - FSAM explicitly requires getting the best in $\mathcal{H}$

$$G_m = \arg\min_{G \in \mathcal{H}} \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq G(x_i))$$

Gradient Boosting / "Anyboost"

FSAM Looks Like Iterative Optimization

- The FSAM step

$$(\nu_m, h_m) = \underset{\nu \in \mathbf{R}, h \in \mathcal{H}}{\arg\min} \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i) \underbrace{+\nu h(x_i)}_{\text{new piece}}\right).$$

- Hard part: finding the **best step direction** h.
- What if we looked for the **locally best** step direction?
  - like in gradient descent

"Functional" Gradient Descent

- We want to minimize
$$J(f) = \sum_{i=1}^{n} \ell\left(y_i, f(x_i)\right).$$
- Only depends on f at the n training points.
- Define
$$\mathbf{f} = \left(f(x_1), \ldots, f(x_n)\right)^T$$
and write the objective function as
$$J(\mathbf{f}) = \sum_{i=1}^{n} \ell\left(y_i, \mathbf{f}_i\right).$$

Functional Gradient Descent: Unconstrained Step Direction

- Consider gradient descent on

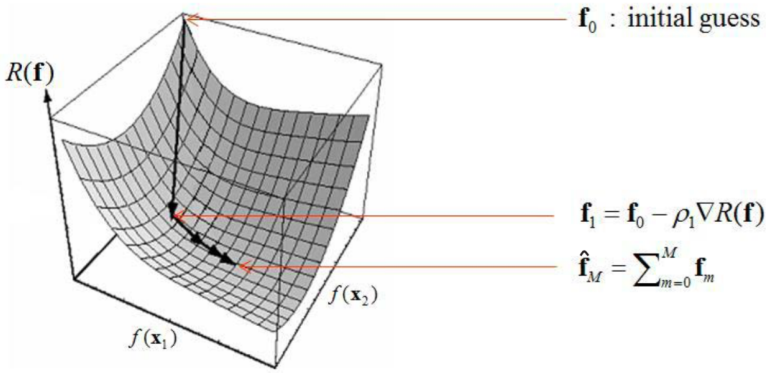
$$J(\mathbf{f}) = \sum_{i=1}^{n} \ell(y_i, \mathbf{f}_i).$$

- The **negative gradient step direction** at $\mathbf{f}$ is

$$-\mathbf{g} = -\nabla_{\boldsymbol{f}} J(\mathbf{f}),$$

  which we can easily calculate.
- Eventually we need more than just $\mathbf{f}$, which is just predictions on training.
- We'll need a full function $f : \mathcal{X} \to \mathbf{R}$.

Unconstrained Functional Gradient Stepping



$R(\mathbf{f})$ is the empirical risk, where $\mathbf{f} = (f(x_1), f(x_2))$ are predictions on training set.

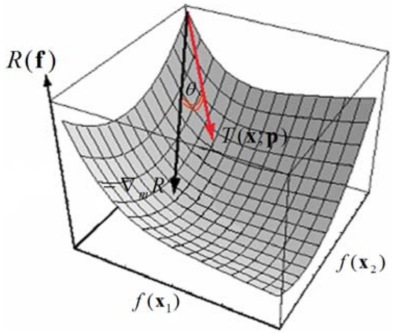
Issue: $\hat{\mathbf{f}}_M$ only defined at training points.

From Seni and Elder's *Ensemble Methods in Data Mining*, Fig B.1.

Functional Gradient Descent: Projection Step

- Unconstrained step direction is
$$-\mathbf{g} = -\nabla_{\mathbf{f}} J(\mathbf{f}).$$
- Suppose $\mathcal{H}$ is our base hypothesis space.
- Find $h \in \mathcal{H}$ that is closest to $-\mathbf{g}$ at the training points, in the ℓ^2 sense:
$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n} (-\mathbf{g}_i - h(x_i))^2.$$
- This is a least squares regression problem.
- $\mathcal{H}$ should have **real-valued** functions.
- So the h that best approximates $-\mathbf{g}$ is our step direction.

"Projected" Functional Gradient Stepping



$T(x;p) \in \mathcal{H}$ is our actual step direction – like the projection of -$\mathbf{g}$=-$\nabla R(\mathbf{f})$ onto $\mathcal{H}$.

From Seni and Elder's *Ensemble Methods in Data Mining*, Fig B.2.

Functional Gradient Descent: Step Size

- Finally, we choose a stepsize.
- Option 1 (Line search – *not sure this is actually used in practice*):

$$\nu_m = \arg\min_{\nu>0} \sum_{i=1}^{n} \ell\{y_i, f_{m-1}(x_i) + \nu h_m(x_i)\}.$$

- Option 2: (Shrinkage parameter)
  - We consider $\nu = 1$ to be the full gradient step.
  - Choose a fixed $\nu \in (0,1)$ – called a **shrinkage parameter.**
  - A value of $\nu = 0.1$ is typical – optimize as a hyperparameter .

The Gradient Boosting Machine Ingredients (Recap)

- Take any [sub]differentiable loss function.
- Choose a base hypothesis space for regression.
- Choose number of steps (or a stopping criterion).
- Choose step size methodology.
- Then you're good to go!

Gradient Tree Boosting

Gradient Tree Boosting

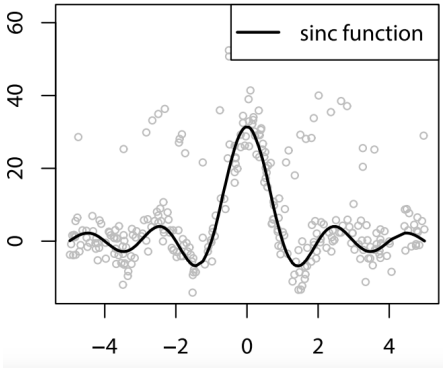
- Common form of gradient boosting machine takes

$$\mathcal{H} = \{\text{regression trees of size } J\},$$

  where J is the number of terminal nodes.
- $J = 2$ gives decision stumps
- HTF recommends $4 \leqslant J \leqslant 8$ (but some recent results use much larger trees)
- Software packages:
  - Gradient tree boosting is implemented by the **gbm package** for R
  - as `GradientBoostingClassifier` and `GradientBoostingRegressor` in **sklearn**
  - **xgboost** and **lightGBM** are state of the art for speed and performance

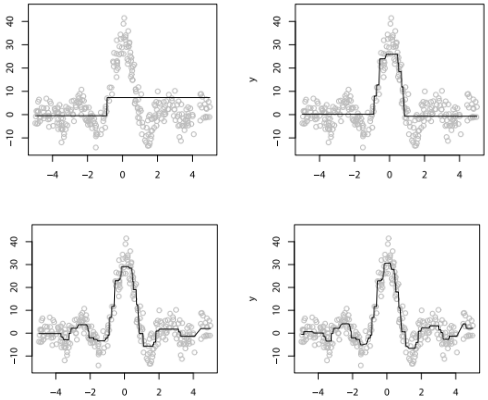
GBM Regression with Stumps

Sinc Function: Our Dataset



From Natekin and Knoll's "Gradient boosting machines, a tutorial"

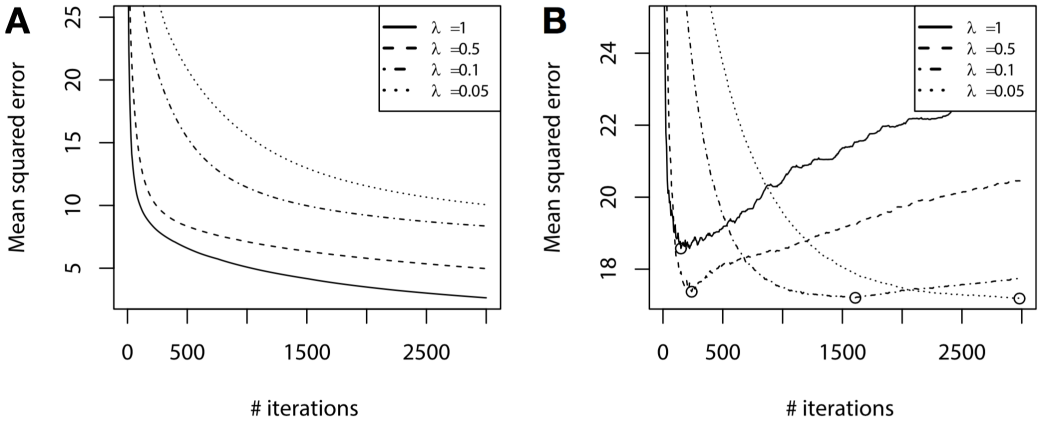
Minimizing Square Loss with Ensemble of Decision Stumps



Decision stumps with $1, 10, 50,$ and 100 steps, step size $\lambda = 1$.

Figure 3 from Natekin and Knoll's "Gradient boosting machines, a tutorial"

Step Size as Regularization



Performance vs rounds of boosting and step size. (Left is training set, right is validation set)

Figure 5 from Natekin and Knoll's "Gradient boosting machines, a tutorial"

Rule of Thumb

- The smaller the step size, the more steps you'll need.
- But never seems to make results worse, and often better.
- So make your step size as small as you have patience for.

Variations on Gradient Boosting

Stochastic Gradient Boosting

- For each stage,
  - choose random subset of data for computing projected gradient step.
  - "Typically, about 50% of the dataset size, can be much smaller for large training set."
  - Fraction is called the **bag fraction**.
- Why do this?
  - Subsample percentage is additional regularization parameter – may help overfitting.
  - Faster.
- We can view this is a **minibatch method**.
  - we're estimating the "true" step direction (the projected gradient) using a subset of data

Introduced by Friedman (1999) in Stochastic Gradient Boosting.

Some Comments on Bag Size

- Justification for 50% of dataset size:
  - In bagging, sampling 50% without replacement gives very similar results to full bootstrap sample
  - See Buja and Stuetzle's Observations on Bagging.
  - So if we're subsampling because we're inspired by bagging, this makes sense.
- But if we think of stochastic gradient boosting as a minibatch method,
  - then makes little sense to choose batch size as a fixed percent of dataset size,
  - especially for large datasets.

Bag as Minibatch

- Just as we argued for minibatch SGD,
  - sample size needed for a good estimate of step direction is independent of training set size
- Minibatch size should depend on
  - the complexity of base hypothesis space
  - the complexity of the target function (Bayes decision function)
- Seems like an interesting area for both practical and theoretical pursuit.

Column / Feature Subsampling for Regularization

- Similar to random forest, randomly choose a subset of features for each round.
- XGBoost paper says: "According to user feedback, using column sub-sampling prevents overfitting even more so than the traditional row sub-sampling."
- Zhao Xing (top Kaggle competitor) finds optimal percentage to be 20%-100%

Newton Step Direction

- For GBM, we find the closest $h \in \mathcal{F}$ to the negative gradient

$$-\mathbf{g} = -\nabla_{\boldsymbol{f}} J(\mathbf{f}).$$

- This is a "first order" method.
- Newton's method is a "second order method":
  - Find 2nd order (quadratic) approximation to J at $\boldsymbol{f}$.
    - Requires computing gradient and Hessian of J.
  - Newton step direction points towards minimizer of the quadratic.
  - Minimizer of quadratic is easy to find in closed form
- Boosting methods with projected Newton step direction:
  - LogitBoost (logistic loss function)
  - XGBoost (any loss – uses regression trees for base classifier)

XGBoost

- Adds explicit penalty term on tree complexity to the empirical risk:

$$\Omega(h) = \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{T} w_j^2,$$

  where $h \in \mathcal{H}$ is a regression tree from our base hypothesis space and
  - T is the number of leaf nodes and
  - w_j is the prediction in the j'th node
- Objective function at step m:

$$J(h) = \text{2nd Order Approximation to empirical risk}(h) + \Omega(h)$$

- In XGBoost, they also use this objective to decide on tree splits
- See XGBoost Introduction for a nice introduction.