

# Forward Stagewise Additive Modeling

He He

Slides based on Lecture 11c from David Rosenberg's course materials  
(<https://github.com/davidrosenberg/mlcourse>)

CDS, NYU

April 13, 2021

## Gradient Boosting / “Anyboost”

## FSAM with squared loss

- Objective function at  $m$ 'th round:

$$J(v, h) = \frac{1}{n} \sum_{i=1}^n \left( y_i - \left[ f_{m-1}(x_i) + \underbrace{vh(x_i)}_{\text{new piece}} \right] \right)^2$$

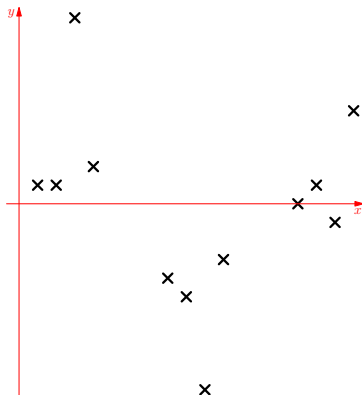
- If  $\mathcal{H}$  is closed under rescaling (i.e. if  $h \in \mathcal{H}$ , then  $vh \in \mathcal{H}$  for all  $h \in \mathcal{H}$ ), then don't need  $v$ .
- Take  $v = 1$  and minimize

$$J(h) = \frac{1}{n} \sum_{i=1}^n \left( \left[ \underbrace{y_i - f_{m-1}(x_i)}_{\text{residual}} \right] - h(x_i) \right)^2$$

- This is just fitting the residuals with least-squares regression!
- Example base hypothesis space: regression stumps.

## $L^2$ Boosting with Decision Stumps: Demo

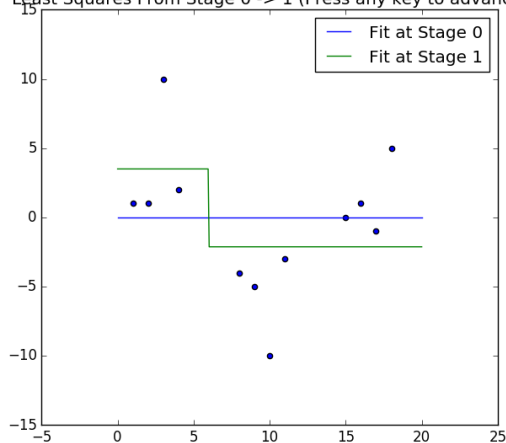
- Consider FSAM with  $L^2$  loss (i.e.  $L^2$  Boosting)
- For base hypothesis space of **regression stumps**



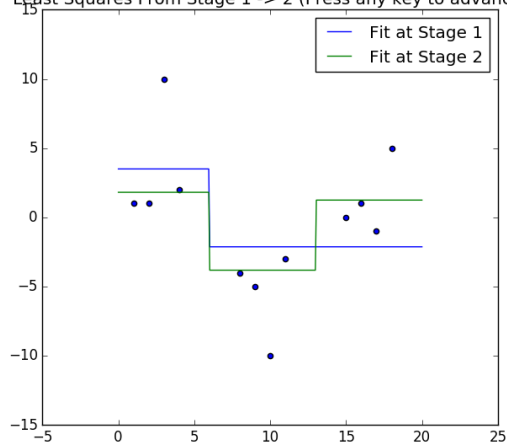
Plot courtesy of Brett Bernstein.

# $L^2$ Boosting with Decision Stumps: Results

Least Squares From Stage 0 -> 1 (Press any key to advance)

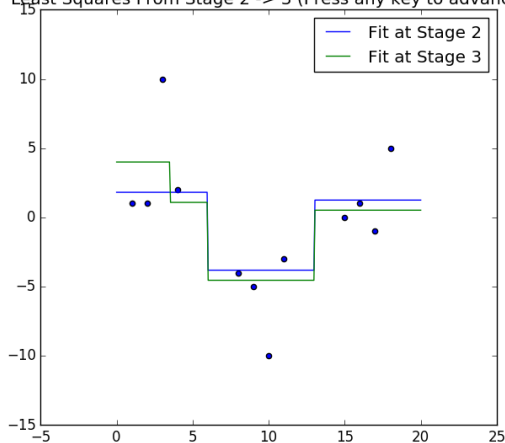


Least Squares From Stage 1 -> 2 (Press any key to advance)

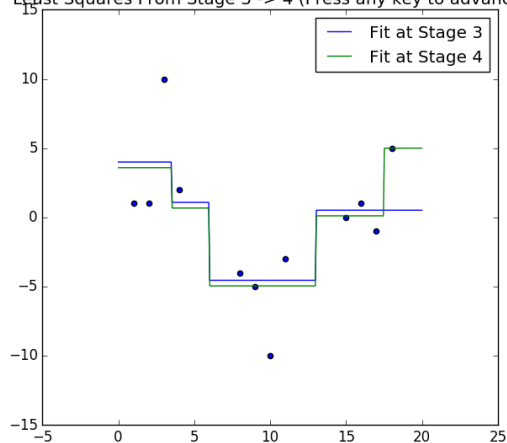


# $L^2$ Boosting with Decision Stumps: Results

Least Squares From Stage 2 -> 3 (Press any key to advance)

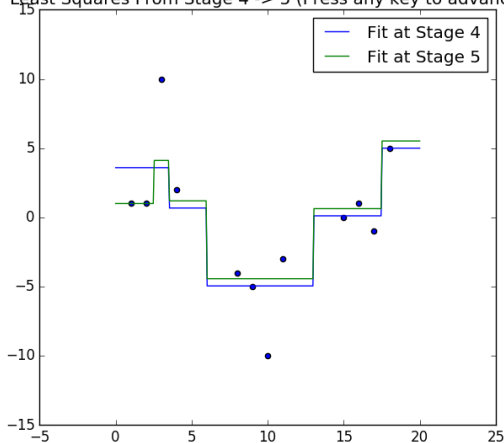


Least Squares From Stage 3 -> 4 (Press any key to advance)

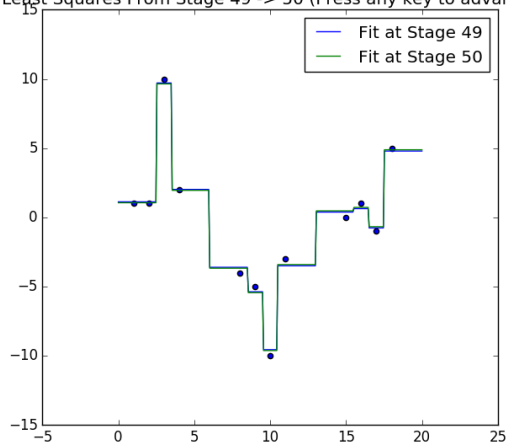


# $L^2$ Boosting with Decision Stumps: Results

Least Squares From Stage 4 -> 5 (Press any key to advance)



Least Squares From Stage 49 -> 50 (Press any key to advance)



## Interpret the residual

- Objective:  $J(f) = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2$ .
- What is the residual at  $x = x_i$ ?

$$\frac{\partial}{\partial f(x_i)} J(f) = -2(y_i - f(x_i)) \quad (1)$$

- Gradient w.r.t.  $f$ : how should the output of  $f$  change to minimize the squared loss.
- *Residual is the negative gradient* (differ by some constant).
- At each boosting round, we learn a function  $h \in \mathcal{H}$  to fit the residual.

$$f \leftarrow f + \nu h \quad \text{FSAM / boosting} \quad (2)$$

$$f \leftarrow f - \alpha \nabla_f J(f) \quad \text{gradient descent} \quad (3)$$

- $h$  approximates the gradient (step direction).

$\alpha$  is the step size



# “Functional” Gradient Descent

- We want to minimize

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

- In some sense, we want to take the gradient w.r.t.  $f$ .
- $J(f)$  only depends on  $f$  at the  $n$  training points.
- Define “parameters”

$$\mathbf{f} = (f(x_1), \dots, f(x_n))^T$$

and write the objective function as

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

# 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

$$\begin{aligned} -\mathbf{g} &= -\nabla_{\mathbf{f}} J(\mathbf{f}) \\ &= -(\partial_{\mathbf{f}_1} \ell(y_1, \mathbf{f}_1), \dots, \partial_{\mathbf{f}_n} \ell(y_n, \mathbf{f}_n)) \end{aligned}$$

which we can easily calculate.

- $-\mathbf{g} \in \mathbf{R}^n$  is the direction we want to change each of our  $n$  predictions on training data.
- With gradient descent, our final predictor will be an additive model:  $f_0 + \sum_{m=1}^M v_t(-\mathbf{g}_t)$ .

# Functional Gradient Descent: Projection Step

- Unconstrained step direction is

$$-\mathbf{g} = -\nabla_{\mathbf{f}} J(\mathbf{f}) = -(\partial_{\mathbf{f}_1} \ell(y_1, \mathbf{f}_1), \dots, \partial_{\mathbf{f}_n} \ell(y_n, \mathbf{f}_n)).$$

- Also called the “**pseudo-residuals**”. (For squared loss, they’re exactly the residuals.)
- **Problem**: only know how to update at  $n$  points. How do we take a gradient step in  $\mathcal{H}$ ?
- **Solution**: approximate by the closest base hypothesis  $h \in \mathcal{H}$  (in the  $\ell^2$  sense):

$$\min_{h \in \mathcal{H}} \sum_{i=1}^n (-\mathbf{g}_i - h(x_i))^2. \quad \text{least square regression} \quad (4)$$

- Take the  $h \in \mathcal{H}$  that best approximates  $-\mathbf{g}$  as our step direction.

## Explain by figure

# Recap

- Objective function:

$$J(f) = \sum_{i=1}^n \ell(y_i, f(x_i)). \quad (5)$$

- Unconstrained gradient  $\mathbf{g} \in \mathbf{R}^n$  w.r.t.  $\mathbf{f} = (f(x_1), \dots, f(x_n))^T$ :

$$\mathbf{g} = \nabla_{\mathbf{f}} J(\mathbf{f}) = (\partial_{\mathbf{f}_1} \ell(y_1, \mathbf{f}_1), \dots, \partial_{\mathbf{f}_n} \ell(y_n, \mathbf{f}_n)). \quad (6)$$

- Projected negative gradient  $h \in \mathcal{H}$ :

$$h = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^n (-\mathbf{g}_i - h(x_i))^2. \quad (7)$$

- Gradient descent:

$$f \leftarrow f + \textcolor{red}{v} h \quad (8)$$

# Functional Gradient Descent: hyperparameters

- Choose a step size by **line search**.

$$v_m = \arg \min_v \sum_{i=1}^n \ell\{y_i, f_{m-1}(x_i) + v h_m(x_i)\}.$$

- Not necessary. Can also choose a fixed hyperparameter  $v$ .
- Regularization through **shrinkage**:

$$f_m \leftarrow f_{m-1} + \lambda v_m h_m \quad \text{where } \lambda \in [0, 1]. \quad (9)$$

- Typically choose  $\lambda = 0.1$ .
- Choose  $M$ , i.e. when to stop.
  - Tune on validation set.

# Gradient boosting algorithm

- 1 Initialize  $f$  to a constant:  $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^n \ell(y_i, \gamma)$ .
- 2 For  $m$  from 1 to  $M$ :
  - 1 Compute the pseudo-residuals (negative gradient):

$$r_{im} = - \left[ \frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)) \right]_{f(x_i) = f_{m-1}(x_i)} \quad (10)$$

- 2 Fit a base learner  $h_m$  with squared loss using the dataset  $\{(x_i, r_{im})\}_{i=1}^n$ .
  - 3 [Optional] Find the best step size  $v_m = \arg \min_v \sum_{i=1}^n \ell(y_i, f_{m-1}(x_i) + v h_m(x_i))$ .
  - 4 Update  $f_m = f_{m-1} + \lambda v_m h_m$
- 3 Return  $f_M(x)$ .

# The Gradient Boosting Machine Ingredients (Recap)

- Take any loss function [sub]differentiable w.r.t. the prediction  $f(x_i)$
- 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!



# BinomialBoost: Gradient Boosting with Logistic Loss

- Recall the logistic loss for classification, with  $\mathcal{Y} = \{-1, 1\}$ :

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

- Pseudoresidual for  $i$ 'th example is negative derivative of loss w.r.t. prediction:

$$r_i = -\frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)) \quad (11)$$

$$= -\frac{\partial}{\partial f(x_i)} \left[ \log(1 + e^{-y_i f(x_i)}) \right] \quad (12)$$

$$= \frac{y_i e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} \quad (13)$$

$$= \frac{y_i}{1 + e^{y_i f(x_i)}} \quad (14)$$

# BinomialBoost: Gradient Boosting with Logistic Loss

- Pseudoresidual for  $i$ th example:

$$r_i = -\frac{\partial}{\partial f(x_i)} \left[ \log \left( 1 + e^{-y_i f(x_i)} \right) \right] = \frac{y_i}{1 + e^{y_i f(x_i)}}$$

- So if  $f_{m-1}(x)$  is prediction after  $m-1$  rounds, step direction for  $m$ 'th round is

$$h_m = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^n \left[ \left( \frac{y_i}{1 + e^{y_i f_{m-1}(x_i)}} \right) - h(x_i) \right]^2.$$

- And  $f_m(x) = f_{m-1}(x) + \eta h_m(x)$ .

# Gradient Tree Boosting

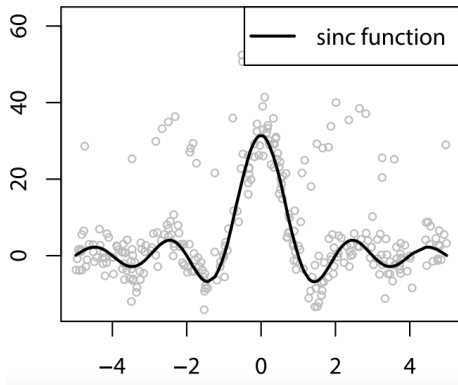
- One common form of gradient boosting machine takes

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

where  $S$  is the number of terminal nodes.

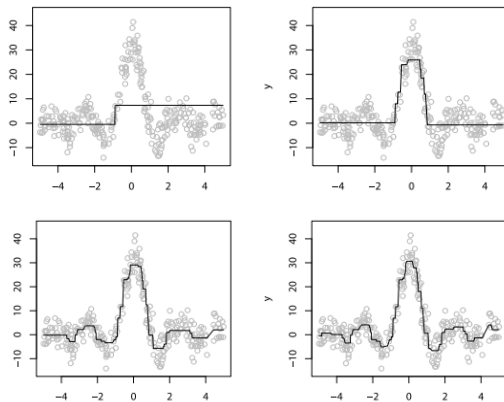
- $S = 2$  gives decision stumps
- HTF recommends  $4 \leq S \leq 8$  (but more recent results use much larger trees)
- Software packages:
  - Gradient tree boosting is implemented by the `gbm` package for R
  - as `GradientBoostingClassifier` and `GradientBoostingRegressor` in `sklearn`
  - `xgboost`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

# 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, shrinkage  $\lambda = 1$ .

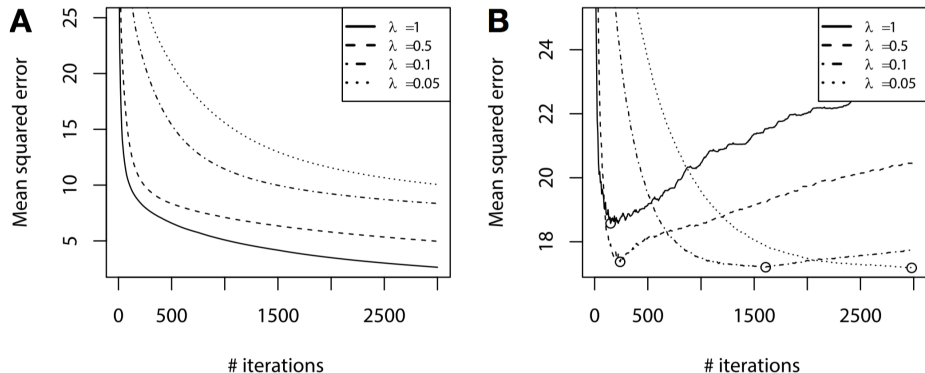
Figure 3 from Natekin and Knoll's "Gradient boosting machines, a tutorial"

## Gradient Boosting in Practice

# Prevent overfitting

- Boosting is resistant to overfitting. Some explanations:
  - Implicit feature selection: greedily selects the best feature (weak learner)
  - As training goes on, impact of change is localized.
- But it can of course overfit. Common regularization methods:
  - Shrinkage (small learning rate)
  - Stochastic gradient boosting (row subsampling)
  - Feature subsampling (column subsampling)

# Step Size as Regularization



- (continued) sinc function regression
- Performance vs rounds of boosting and shrinkage. (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 set your step size as small as you have patience for.

# Stochastic Gradient Boosting

- For each stage,
  - choose random *subset of data* for computing projected gradient step.
- Why do this?
  - Introduce randomization thus may help overfitting.
  - Faster; often better than gradient descent given the same computation resource.
- We can view this is a **minibatch method**.
  - Estimate the “true” step direction using a subset of data.

---

Introduced by Friedman (1999) in [Stochastic Gradient Boosting](#).

## Column / Feature Subsampling

- 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.”
- Speeds up computation.

# Summary

- Motivating idea of boosting: combine weak learners to produce a strong learner.
- The statistical view: boosting is fitting an additive model (greedily).
- The numerical optimization view: boosting makes local improvement iteratively—gradient descent in the function space.
- Gradient boosting is a generic framework
  - Any differentiable loss function
  - Classification, regression, ranking, multiclass etc.
  - Scalable, e.g., XGBoost