

# Gradient Boosting

He He  
(adapted from David Rosenberg's slides)

CDS, NYU

April 21, 2019

# Contents

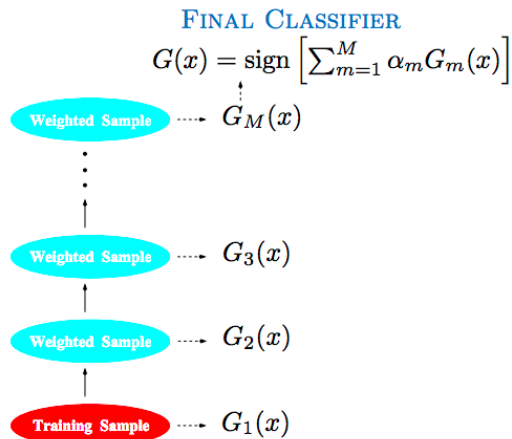
- 1 Motivation
- 2 Forward Stagewise Additive Modeling
  - Example: AdaBoost
- 3 Gradient Boosting / “Anyboost”
  - Motivating example: L2 Boosting
  - Gradient Descent in the Function Space
  - Example: BinomialBoost
  - Gradient Tree Boosting
- 4 Gradient Boosting in Practice

# Today's lecture

- Another way to get non-linear models in a linear form—adaptive basis function models.
- A general algorithm for greedy function approximation—gradient boosting machine.

# Motivation

## Recap: Adaboost



From ESL Figure 10.1

# AdaBoost: Algorithm

Given training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ .

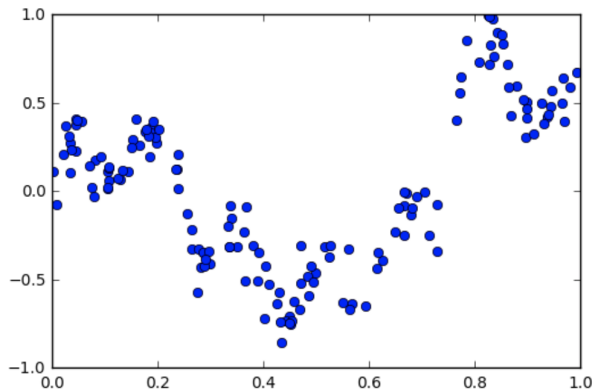
- ① Initialize observation weights  $w_i = 1, i = 1, 2, \dots, n$ .
- ② For  $m = 1$  to  $M$ :
  - ① Base learner fits weighted training data and returns  $G_m(x)$
  - ② Compute **weighted empirical 0-1 risk**:

$$\text{err}_m = \frac{1}{W} \sum_{i=1}^n w_i \mathbf{1}(y_i \neq G_m(x_i)) \quad \text{where } W = \sum_{i=1}^n w_i.$$

- ③ Compute **classifier weight**:  $\alpha_m = \ln \left( \frac{1 - \text{err}_m}{\text{err}_m} \right)$ .
  - ④ Update **example weight**:  $w_i \leftarrow w_i \cdot \exp[\alpha_m \mathbf{1}(y_i \neq G_m(x_i))]$
- ③ Return **voted classifier**:  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ . Why not learn  $G(x)$  directly?

# Nonlinear Regression

- How do we fit the following data?



# Linear Model with Basis Functions

- Fit a linear combination of transformations of the input:

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

where  $h_m$ 's are called **basis functions** (or feature functions in ML):

$$h_1, \dots, h_M : \mathcal{X} \rightarrow \mathbb{R}$$

- Example: polynomial regression where  $h_m(x) = x^m$ .
- Can we use this model for classification?
- Can fit this using standard methods for linear models (e.g. least squares, lasso, ridge, etc.)
  - Note that  $h_m$ 's are fixed and known, i.e. chosen ahead of time.



# Adaptive Basis Function Model

- What if we want to learn the basis functions? (hence **adaptive**)
- Base hypothesis space  $\mathcal{H}$  consisting of functions  $h: \mathcal{X} \rightarrow \mathbb{R}$ .
- An **adaptive basis function expansion** over  $\mathcal{H}$  is an ensemble model:

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

where  $v_m \in \mathbb{R}$  and  $h_m \in \mathcal{H}$ .

- Combined hypothesis space:

$$\mathcal{F}_M = \left\{ \sum_{m=1}^M v_m h_m(x) \mid v_m \in \mathbb{R}, h_m \in \mathcal{H}, m = 1, \dots, M \right\}$$

- What are the learnable?

# Empirical Risk Minimization

- What's our learning objective?

$$\hat{f} = \arg \min_{f \in \mathcal{F}_M} \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i)),$$

for some loss function  $\ell$ .

- Write ERM objective function as

$$J(v_1, \dots, v_M, h_1, \dots, h_M) = \frac{1}{n} \sum_{i=1}^n \ell \left( y_i, \sum_{m=1}^M v_m h_m(x) \right).$$

- How to optimize  $J$ ? i.e. how to learn?

- Suppose our base hypothesis space is parameterized by  $\Theta = \mathbb{R}^b$ :

$$J(v_1, \dots, v_M, \theta_1, \dots, \theta_M) = \frac{1}{n} \sum_{i=1}^n \ell \left( y_i, \sum_{m=1}^M v_m h(x; \theta_m) \right).$$

- Can we optimize it with SGD?
  - Can we differentiate  $J$  w.r.t.  $v_m$ 's and  $\theta_m$ 's?
- For some hypothesis spaces and typical loss functions, yes!
  - Neural networks fall into this category! ( $h_1, \dots, h_M$  are neurons of last hidden layer.)

# What if Gradient Based Methods Don't Apply?

What if base hypothesis space  $\mathcal{H}$  consists of decision trees?

- Can we even parameterize trees with  $\Theta = \mathbb{R}^b$ ?
- Even if we could, predictions would not change continuously w.r.t.  $\theta \in \Theta$ , so certainly not differentiable.

What about a greedy algorithm similar to Adaboost?

- Applies to non-parametric or non-differentiable basis functions.
- But is it optimizing our objective using some loss function?

Today we'll discuss **gradient boosting**.

- Gradient descent in the **function space**.
- It applies whenever
  - our loss function is [sub]differentiable w.r.t. training predictions  $f(x_i)$ , and
  - we can do regression with the base hypothesis space  $\mathcal{H}$ .

Kearns, Valiant (1989): Can weak learners (e.g., 51% accuracy) be transformed to strong learners (e.g., 99.9% accuracy)?

Schapire (1990) & Freund (1995): Yes, weak learners can be iteratively improved to a strong learner.

Freund, Schapire (1996): And here is a practical algorithm—Adaboost.

Breiman (1996 & 1998): Yes, it works! Boosting is the best off-the-shelf classifier in the world.

(Attempts to explain why Adaboost works and improvements)

Friedman, Hastie, Tibshirani (2000): Actually, boosting fits an additive model.

Friedman (2001): Furthermore, it can be considered as gradient descent in the function space.

## Forward Stagewise Additive Modeling

# Forward Stagewise Additive Modeling (FSAM)

**Goal** fit model  $f(x) = \sum_{m=1}^M v_m h_m(x)$  given some loss function.

**Approach** Greedily fit one function at a time without adjusting previous functions, hence “forward stagewise”.

- After  $m-1$  stages, we have

$$f_{m-1} = \sum_{i=1}^{m-1} v_i h_i.$$

- In  $m$ 'th round, we want to find  $h_m \in \mathcal{H}$  (i.e. a basis function) and  $v_m > 0$  such that

$$f_m = \underbrace{f_{m-1}}_{\text{fixed}} + v_m h_m$$

improves objective function value by as much as possible.

# Forward Stagewise Additive Modeling for ERM

Let's plug in our objective function.

❶ Initialize  $f_0(x) = 0$ .

❷ For  $m = 1$  to  $M$ :

❶ Compute:

$$(v_m, h_m) = \arg \min_{v \in \mathbb{R}, h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \ell \left( y_i, f_{m-1}(x_i) + \underbrace{vh(x_i)}_{\text{new piece}} \right).$$

❷ Set  $f_m = f_{m-1} + v_m h_m$ .

❸ Return:  $f_M$ .



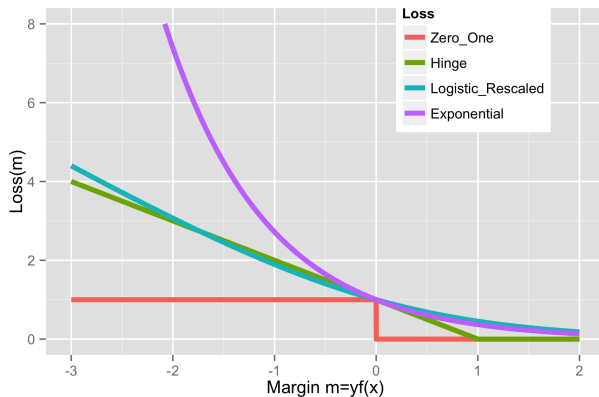
## Recap: margin-based classifier

### Binary classification

- Outcome space  $\mathcal{Y} = \{-1, 1\}$
- Action space  $\mathcal{A} = \mathbb{R}$  (model output)
- Score function  $f : \mathcal{X} \rightarrow \mathcal{A}$ .
- Margin for example  $(x, y)$  is  $m = yf(x)$ .
  - $m > 0 \iff$  classification correct
  - Larger  $m$  is better.
- **Concept check:** What are margin-based loss functions we've seen?

# Exponential Loss

- Introduce the **exponential loss**:  $\ell(y, f(x)) = \exp\left(-\underbrace{yf(x)}_{\text{margin}}\right)$ .



# Forward Stagewise Additive Modeling with exponential loss

Recall that we want to do FSAM with exponential loss.

❶ Initialize  $f_0(x) = 0$ .

❷ For  $m = 1$  to  $M$ :

❶ Compute:

$$(v_m, h_m) = \arg \min_{v \in \mathbb{R}, h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \ell_{\text{exp}} \left( y_i, f_{m-1}(x_i) + \underbrace{vh(x_i)}_{\text{new piece}} \right).$$

❷ Set  $f_m = f_{m-1} + v_m h_m$ .

❸ Return:  $f_M$ .

## FSAM with Exponential Loss: objective function

- Base hypothesis:  $\mathcal{H} = \{h: \mathcal{X} \rightarrow \{-1, 1\}\}$ .
- Objective function in the  $m$ 'th round:

$$J(v, h) = \sum_{i=1}^n \exp[-y_i (f_{m-1}(x_i) + v h(x_i))] \quad (2)$$

$$= \sum_{i=1}^n w_i^m \exp[-y_i v h(x_i)] \quad w_i^m \stackrel{\text{def}}{=} \exp[-y_i f_{m-1}(x_i)] \quad (3)$$

$$= \sum_{i=1}^n w_i^m [\mathbb{I}(y_i = h(x_i)) e^{-v} + \mathbb{I}(y_i \neq h(x_i)) e^v] \quad h(x_i) \in \{1, -1\} \quad (4)$$

$$= \sum_{i=1}^n w_i^m [(e^v - e^{-v}) \mathbb{I}(y_i \neq h(x_i)) + e^{-v}] \quad \mathbb{I}(y_i = h(x_i)) = 1 - \mathbb{I}(y_i \neq h(x_i)) \quad (5)$$

## FSAM with Exponential Loss: basis function

- Objective function in the  $m$ 'th round:

$$J(v, h) = \sum_{i=1}^n w_i^m [(e^v - e^{-v})\mathbb{I}(y_i \neq h(x_i)) + e^{-v}]. \quad (6)$$

- If  $v > 0$ , then

$$\arg \min_{h \in \mathcal{H}} J(v, h) = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^n w_i^m \mathbb{I}(y_i \neq h(x_i)) \quad (7)$$

$$h_m = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^n w_i^m \mathbb{I}(y_i \neq h(x_i)) \quad (8)$$

$$= \arg \min_{h \in \mathcal{H}} \frac{1}{\sum_{i=1}^n w_i^m} \sum_{i=1}^n w_i^m \mathbb{I}(y_i \neq h(x_i)) \quad \text{multiply by a positive constant} \quad (9)$$

i.e.  $h_m$  is the minimizer of the weighted zero-one loss.

## FSAM with Exponential Loss: classifier weights

- Define the weighted zero-one error:

$$\text{err}_m = \frac{\sum_{i=1}^n w_i^m \mathbb{I}(y_i \neq h(x_i))}{\sum_{i=1}^n w_i^m}. \quad (10)$$

- **Exercise:** show that the optimal  $v$  is:

$$v_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m} \quad (11)$$

- Same as the classifier weights in Adaboost modulo a constant.
- If  $\text{err}_m < 0.5$  (better than chance), then  $v_m > 0$ .

## FSAM with Exponential Loss: example weights

- Weights in the next round:

$$w_i^{m+1} \stackrel{\text{def}}{=} \exp[-y_i f_m(x_i)] \quad (12)$$

$$= w_i^m \exp[-y_i v_m h_m(x_i)] \quad f_m(x_i) = f_{m-1}(x_i) + v_m h_m(x_i) \quad (13)$$

$$= w_i^m \exp[-v_m \mathbb{I}(y_i = h_m(x_i)) + v_m \mathbb{I}(y_i \neq h_m(x_i))] \quad (14)$$

$$= w_i^m \exp[2v_m \mathbb{I}(y_i \neq h_m(x_i))] \underbrace{\exp^{-v_m}}_{\text{scaler}} \quad (15)$$

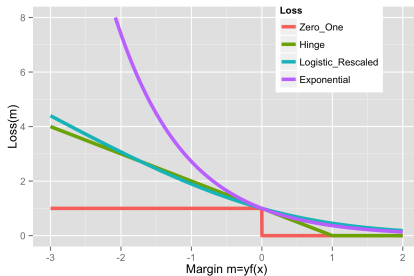
- The constant scaler will cancel out during normalization.
- $2v_m = \alpha_m$  in Adaboost.

# Why Exponential Loss

- $\ell_{\text{exp}}(y, f(x)) = \exp(-yf(x))$ .
- **Exercise:** show that the optimal estimate is

$$f^*(x) = \frac{1}{2} \log \frac{p(y = 1 | x)}{p(y = 0 | x)}. \quad (16)$$

- How is it different from other losses?





## AdaBoost / Exponential Loss: Robustness Issues

- Exponential loss puts a high penalty on misclassified examples.
  - $\implies$  not robust to outliers / noise.
- Empirically, AdaBoost has degraded performance in situations with
  - high Bayes error rate (intrinsic randomness in the label)
- Logistic/Log loss performs better in settings with high Bayes error.
- Exponential loss has some computational advantages over log loss though.

We've seen

- Use basis function to obtain **nonlinear** models:  $f(x) = \sum_{i=1}^M v_m h_m(x)$  with known  $h_m$ 's.
- **Adaptive** basis function models:  $f(x) = \sum_{i=1}^M v_m h_m(x)$  with unknown  $h_m$ 's.
- Forward stagewise additive modeling: greedily fit  $h_m$ 's to minimize the average loss.

But,

- We only know how to do FSAM for certain loss functions.
- Need to derive new algorithms for different loss functions.

Next, how to do FSAM in general.

## Gradient Boosting / “Anyboost”

## FSAM with squared loss

- Objective function at  $m$ 'th round:

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

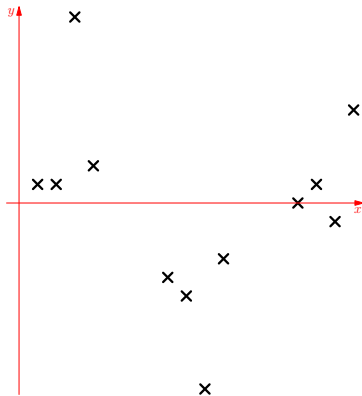
- If  $\mathcal{H}$  is closed under rescaling (i.e. if  $h \in \mathcal{H}$ , then  $vh \in \mathcal{H}$  for all  $h \in \mathcal{H}$ ), then don't need  $v$ .
- Take  $v = 1$  and minimize

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

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

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

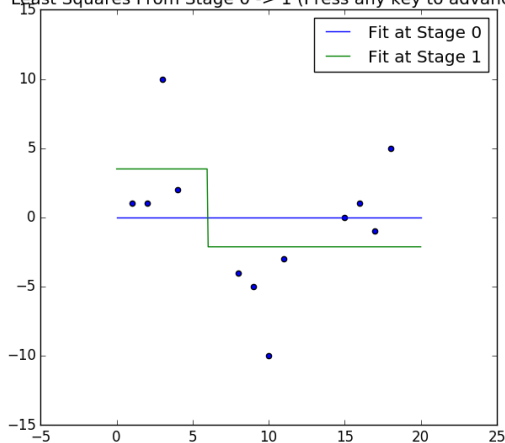
- Consider FSAM with  $L^2$  loss (i.e.  $L^2$  Boosting)
- For base hypothesis space of **regression stumps**



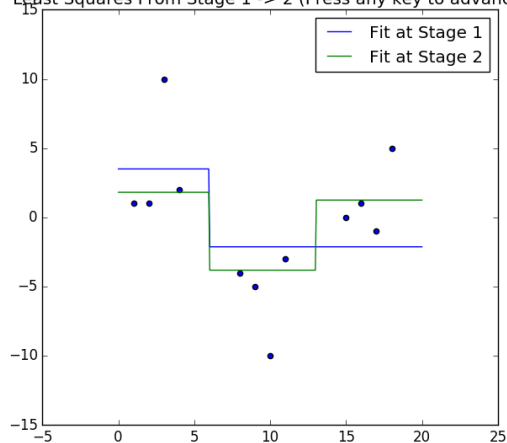
Plot courtesy of Brett Bernstein.

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

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

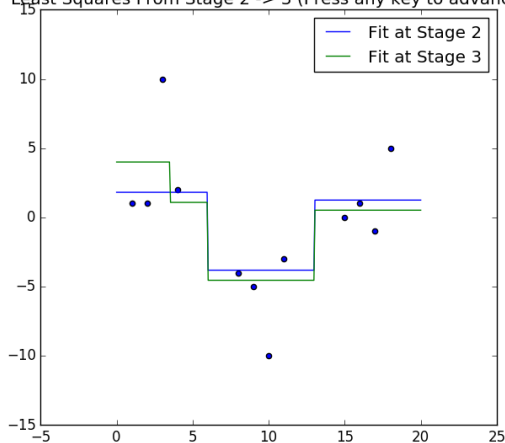


Least Squares From Stage 1 -> 2 (Press any key to advance)

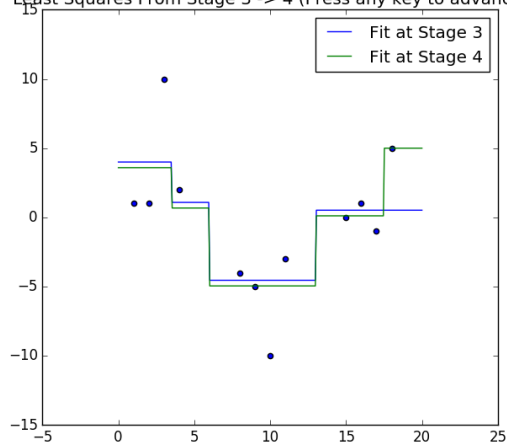


# $L^2$ Boosting with Decision Stumps: Results

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

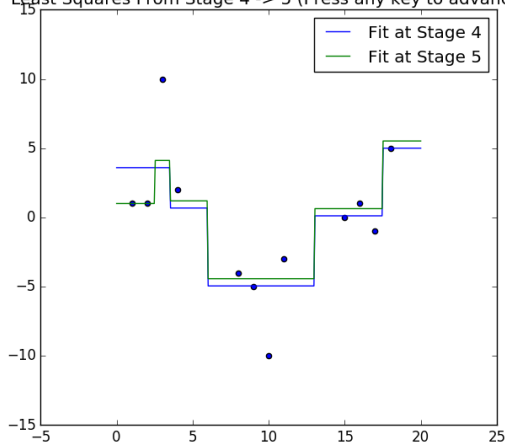


Least Squares From Stage 3 -> 4 (Press any key to advance)

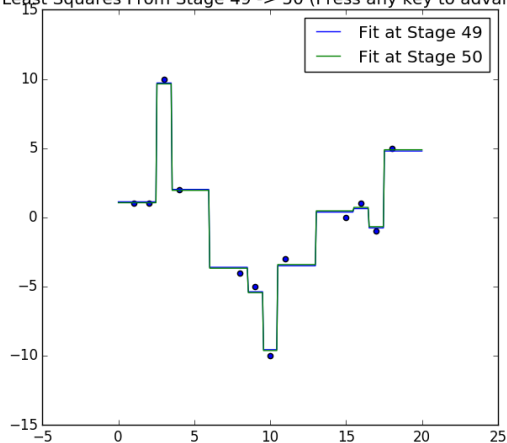


# $L^2$ Boosting with Decision Stumps: Results

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  $x = x_i$ ?

$$\frac{\partial}{\partial f(x_i)} J(f) = -2(y_i - f(x_i)) \quad (17)$$

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

$$f \leftarrow f + \nu h \quad \text{FSAM / boosting} \quad (18)$$

$$f \leftarrow f - \alpha \nabla_f J(f) \quad \text{gradient descent} \quad (19)$$

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

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

- The negative gradient step direction at  $f$  is

$$\begin{aligned} -g &= -\nabla_f J(f) \\ &= -(\partial_{f_1} \ell(y_1, f_1), \dots, \partial_{f_n} \ell(y_n, f_n)) \end{aligned}$$

which we can easily calculate.

- $-g \in \mathbb{R}^n$  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:  $f_0 + \sum_{m=1}^M v_t(-g_t)$ .

# Functional Gradient Descent: Projection Step

- Unconstrained step direction is

$$-g = -\nabla_{\mathbf{f}} J(\mathbf{f}) = -(\partial_{f_1} \ell(y_1, f_1), \dots, \partial_{f_n} \ell(y_n, f_n)).$$

- 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):

$$\min_{h \in \mathcal{H}} \sum_{i=1}^n (-g_i - h(x_i))^2. \quad \text{least square regression} \quad (20)$$

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

# Explain by figure

# Recap

- Objective function:

$$J(f) = \sum_{i=1}^n \ell(y_i, f(x_i)). \quad (21)$$

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

$$g = \nabla_{\mathbf{f}} J(\mathbf{f}) = (\partial_{f_1} \ell(y_1, f_1), \dots, \partial_{f_n} \ell(y_n, f_n)). \quad (22)$$

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

$$h = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^n (-g_i - h(x_i))^2. \quad (23)$$

- Gradient descent:

$$f \leftarrow f + \textcolor{red}{\alpha} h \quad (24)$$

# 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**:

$$f_m \leftarrow f_{m-1} + \lambda v_m h_m \quad \text{where } \lambda \in [0, 1]. \quad (25)$$

- 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):

$$r_{im} = - \left[ \frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)) \right]_{f(x_i) = f_{m-1}(x_i)} \quad (26)$$

- 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:

$$r_i = -\frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)) \quad (27)$$

$$= -\frac{\partial}{\partial f(x_i)} \left[ \log(1 + e^{-y_i f(x_i)}) \right] \quad (28)$$

$$= \frac{y_i e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} \quad (29)$$

$$= \frac{y_i}{1 + e^{y_i f(x_i)}} \quad (30)$$

# 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

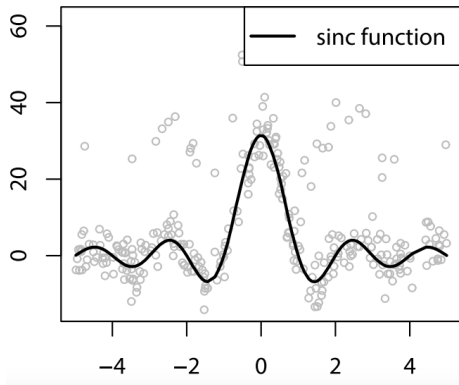
- One common form of gradient boosting machine takes

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

where  $S$  is the number of terminal nodes.

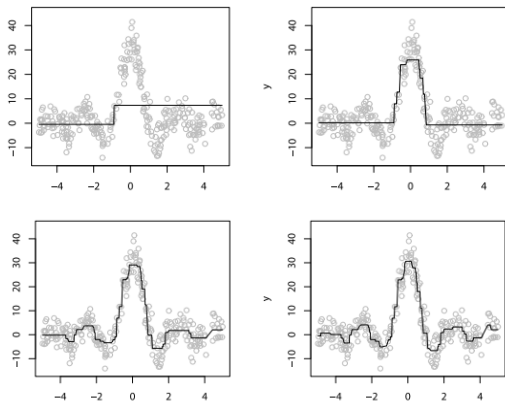
- $S = 2$  gives decision stumps
- HTF recommends  $4 \leq S \leq 8$  (but more recent results use much larger trees)
- Software packages:
  - Gradient tree boosting is implemented by the `gbm` package for R
  - as `GradientBoostingClassifier` and `GradientBoostingRegressor` in `sklearn`
  - `xgboost`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

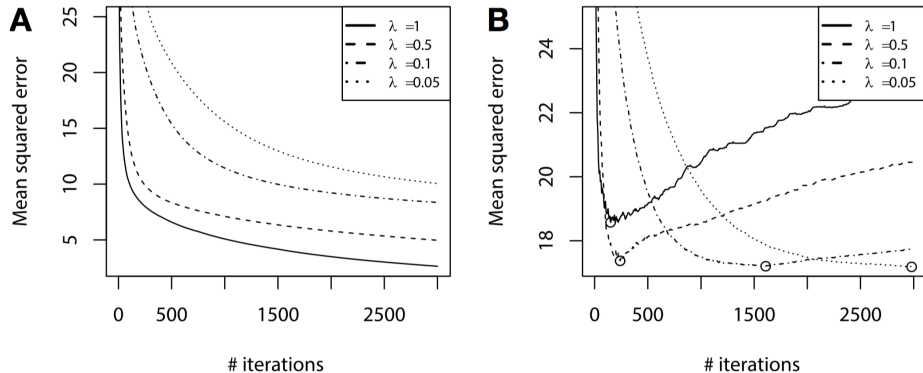
## Gradient Boosting in Practice

# 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.

## 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.

- 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