Gradient Boosting

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

CDS, NYU

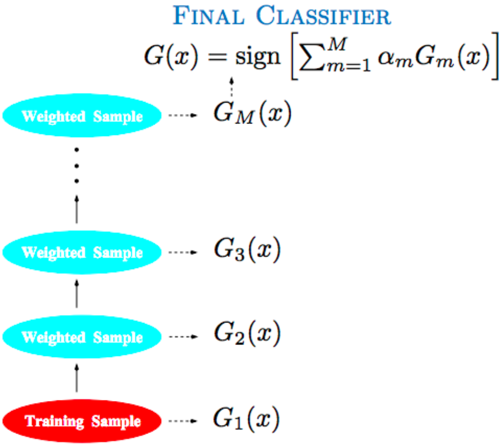
April 12, 2022

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.
    - Adaboost is a special case.

Motivation

Recap: Adaboost



From ESL Figure 10.1

AdaBoost: Algorithm

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

1. Initialize observation weights $w_i = 1,\ i = 1, 2, \ldots, n$.
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*:

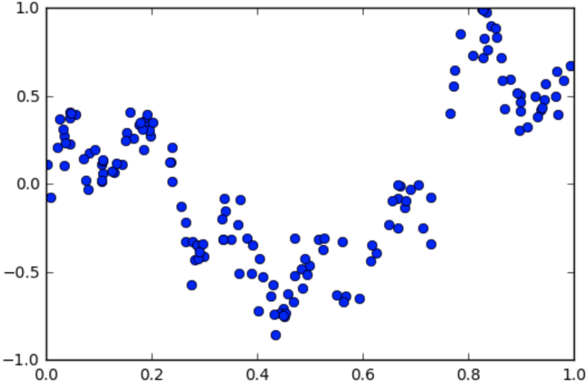
$$\mathrm{err}_m = \frac{1}{W} \sum_{i=1}^{n} w_i 1(y_i \neq G_m(x_i)) \quad \text{where } W = \sum_{i=1}^{n} w_i .$$

    3. Compute *classifier weight*: $\alpha_m = \ln\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right)$.
    4. Update *example weight*: $w_i \leftarrow w_i \cdot \exp\left[\alpha_m 1(y_i \neq G_m(x_i))\right]$
3. Return *voted classifier*: $G(x) = \operatorname{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

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):

$$h_1, \ldots, h_M : \mathcal{X} \to \mathrm{R}$$

- 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 $h : \mathcal{X} \to \mathrm{R}$.
- An **adaptive basis function expansion** over $\mathcal{H}$ is an ensemble model:

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

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

- Combined hypothesis space:

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

- What are the learnable?

Empirical Risk Minimization

- What's our learning objective?

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

for some loss function ℓ.

- Write ERM objective function as

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

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

Gradient-Based Methods

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

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

- Can we optimize it with SGD?
  - Can we differentiate J w.r.t. v_m's and θ_m's?
- For some hypothesis spaces and typical loss functions, yes!
  - Neural networks fall into this category! ($h_1, \ldots, h_M$ 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 $\Theta = \mathrm{R}^b$?
- 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 $\mathcal{H}$.

History

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.

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

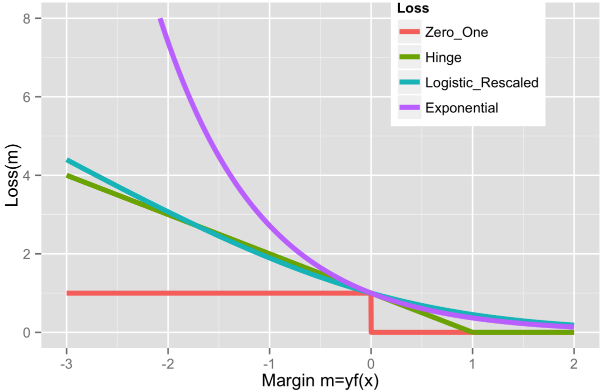
Recap: margin-based classifier

Binary classification

- Outcome space $\mathcal{Y} = \{-1, 1\}$
- Action space $\mathcal{A} = \mathrm{R}$ (model outoput)
- Score function $f : \mathcal{X} \to \mathcal{A}$.
- 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.

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

FSAM with Exponential Loss: objective function

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

$$
\begin{aligned}
J(v,h) &= \sum_{i=1}^{n} \exp\left[-y_i\left(f_{m-1}(x_i) + vh(x_i)\right)\right] && (2)\\
&= \sum_{i=1}^{n} w_i^m \exp\left[-y_i vh(x_i)\right] && w_i^m \stackrel{\text{def}}{=} \exp\left[-y_i f_{m-1}(x_i)\right] \quad (3)\\
&= \sum_{i=1}^{n} w_i^m \left[\mathbb{I}\left(y_i = h(x_i)\right) e^{-v} + \mathbb{I}\left(y_i \neq h(x_i)\right) e^{v}\right] && h(x_i) \in \{1, -1\} \quad (4)\\
&= \sum_{i=1}^{n} w_i^m \left[\left(e^{v} - e^{-v}\right)\mathbb{I}\left(y_i \neq h(x_i)\right) + e^{-v}\right] && \mathbb{I}\left(y_i = h(x_i)\right) = 1 - \mathbb{I}\left(y_i \neq h(x_i)\right)\\
& && (5)
\end{aligned}
$$

FSAM with Exponential Loss: basis function

- Objective function in the m'th round:

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

- If $v>0$, then

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

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

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

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:

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

- Exercise: show that the optimal v is:

$$v_m = \frac{1}{2} \log \frac{1 - \operatorname{err}_m}{\operatorname{err}_m} \tag{11}$$

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

FSAM with Exponential Loss: example weights

- Weights in the next round:

$$
\begin{aligned}
w_i^{m+1} &\stackrel{\text{def}}{=} \exp\left[-y_i f_m(x_i)\right] && (12)\\
&= w_i^m \exp\left[-y_i v_m h_m(x_i)\right] \qquad f_m(x_i) = f_{m-1}(x_i) + v_m h_m(x_i) && (13)\\
&= w_i^m \exp\left[-v_m \mathbb{I}\left(y_i = h_m(x_i)\right) + v_m \mathbb{I}\left(y_i \neq h_m(x_i)\right)\right] && (14)\\
&= w_i^m \exp\left[2v_m \mathbb{I}\left(y_i \neq h_m(x_i)\right)\right] \underbrace{\exp^{-v_m}}_{\text{scaler}} && (15)
\end{aligned}
$$

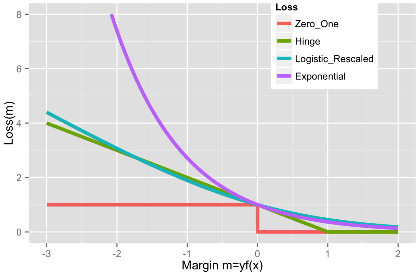
- The constant scaler will cancel out during normalization.
- $2v_m = \alpha_m$ in Adaboost.

Why Exponential Loss

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

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

- 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.

Review

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:

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

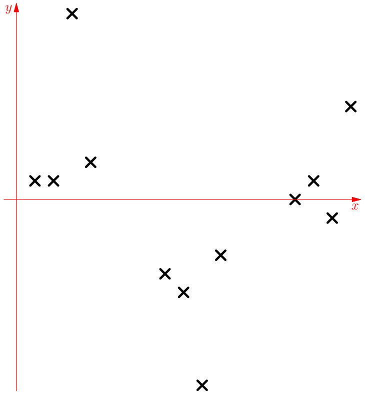
- If $\mathcal{H}$ is closed under rescaling (i.e. if $h\in\mathcal{H}$, then $vh\in\mathcal{H}$ for all $h\in\mathrm{R}$), then don't need v.
- Take $v=1$ and minimize

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

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

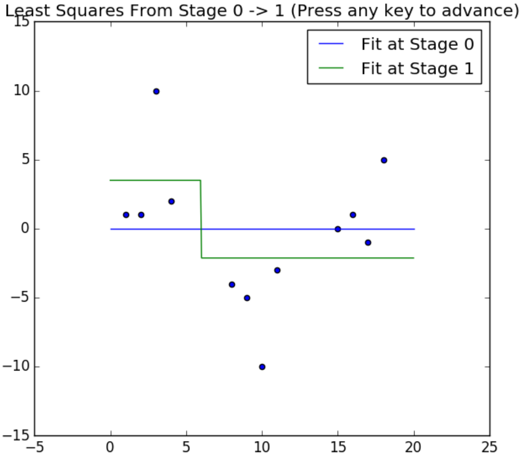
L^2 Boosting with Decision Stumps: Demo

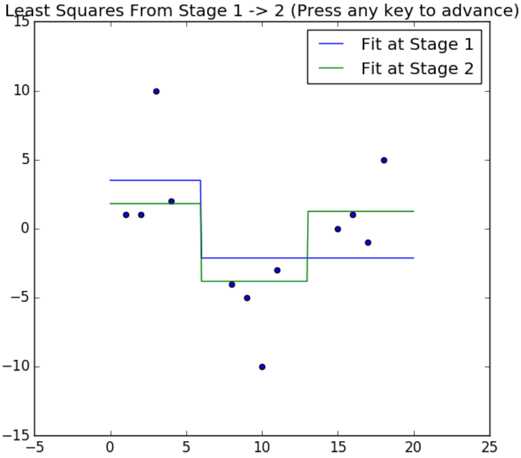
- Consider FSAM with L^2 loss (i.e. L^2 Boosting)
- For base hypothesis space of **regression stumps**



Plot courtesy of Brett Bernstein.

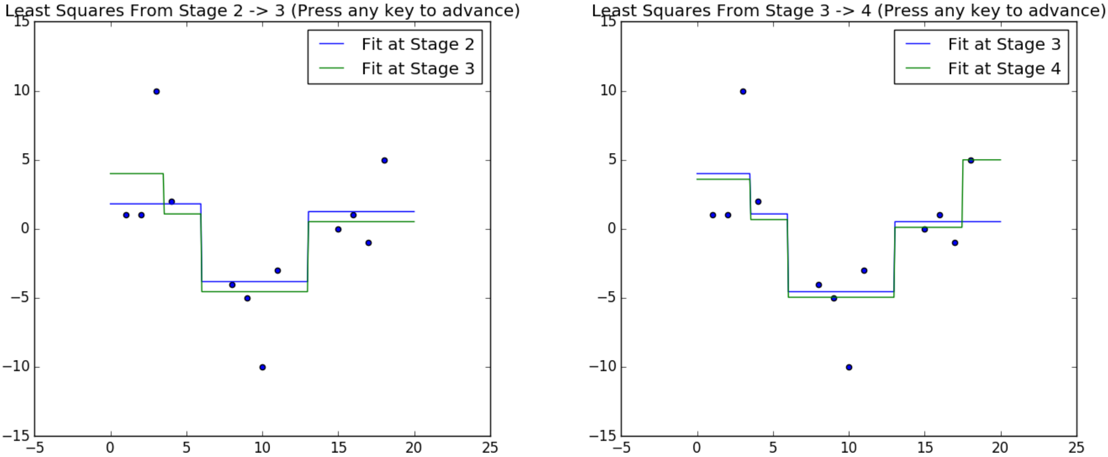
L^2 Boosting with Decision Stumps: Results





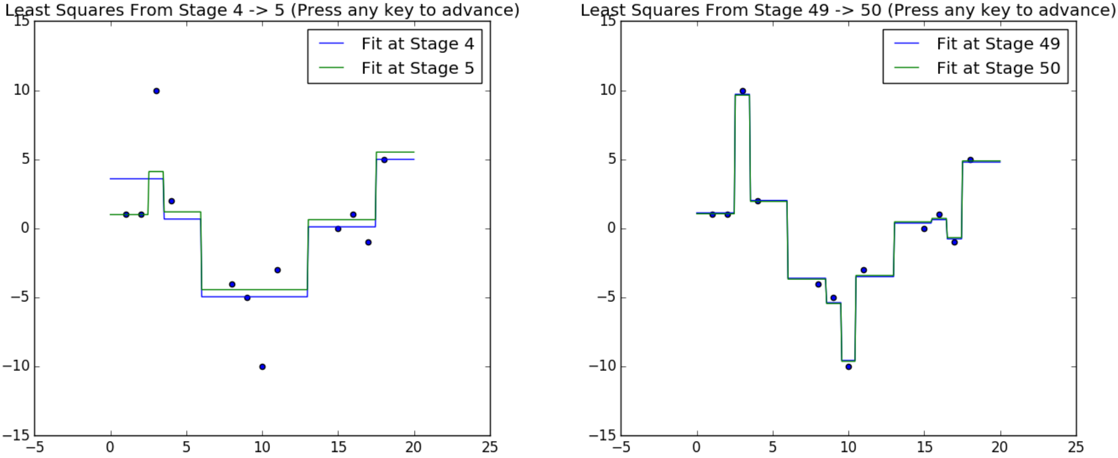
Plots and code courtesy of Brett Bernstein.

L^2 Boosting with Decision Stumps: Results



Plots and code courtesy of Brett Bernstein.

L^2 Boosting with Decision Stumps: Results



Plots and code courtesy of Brett Bernstein.

Interpret the residual

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

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

  - 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.

$$f \leftarrow f + vh \qquad \text{FSAM / boosting} \tag{18}$$

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

  - h approximates the gradient (step direction).
  - v is the step size.

"Functional" Gradient Descent

- We want to minimize
$$J(f) = \sum_{i=1}^{n} \ell\left(y_i, f(x_i)\right).$$
- 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"
$$\mathrm{f} = \left(f(x_1), \ldots, f(x_n)\right)^T$$
and write the objective function as
$$J(\mathrm{f}) = \sum_{i=1}^{n} \ell\left(y_{i,} \mathrm{f}_i\right).$$

Functional Gradient Descent: Unconstrained Step Direction

- Consider gradient descent on

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

- The negative gradient step direction at f is

$$\begin{aligned}-\mathrm{g} &= -\nabla_{\boldsymbol{f}} J(\mathrm{f}) \\ &= -\left(\partial_{\mathrm{f}_1}\ell\left(y_1,\mathrm{f}_1\right),\ldots,\partial_{\mathrm{f}_n}\ell\left(y_n,\mathrm{f}_n\right)\right)\end{aligned}$$

  which we can easily calculate.

- $-\mathrm{g}\in\mathrm{R}^n$ 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: $f_0+\sum_{m=1}^{M}v_t\left(-\mathrm{g}_t\right)$.

Functional Gradient Descent: Projection Step

- Unconstrained step direction is

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

  - 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 ℓ^2 sense):

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \left(-\mathbf{g}_i - h(x_i)\right)^2. \qquad \text{least square regression} \qquad (20)$$

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

Explain by figure

Recap

- Objective function:

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

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

$$\mathrm{g} = \nabla_{\boldsymbol{f}} J(\mathrm{f}) = \left(\partial_{\mathrm{f}_1} \ell(y_1, \mathrm{f}_1), \ldots, \partial_{\mathrm{f}_n} \ell(y_n, \mathrm{f}_n)\right). \tag{22}$$

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

$$h = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \left(-\mathrm{g}_i - h(x_i)\right)^2. \tag{23}$$

- Gradient descent:

$$f \leftarrow f + v h \tag{24}$$

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**:

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

  - 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):

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

    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 $v_m = \arg\min_v \sum_{i=1}^n \ell\left(yi, f_{m-1}(x_i) + vh_m(x_i)\right)$.
    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 $\mathcal{Y} = \{-1, 1\}$:

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

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

$$
\begin{aligned}
r_i &= -\frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)) && (27)\\
&= -\frac{\partial}{\partial f(x_i)} \left[\log\left(1 + e^{-y_i f(x_i)}\right)\right] && (28)\\
&= \frac{y_i e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} && (29)\\
&= \frac{y_i}{1 + e^{y_i f(x_i)}} && (30)
\end{aligned}
$$

BinomialBoost: Gradient Boosting with Logistic Loss

- Pseudoresidual for ith 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

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

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

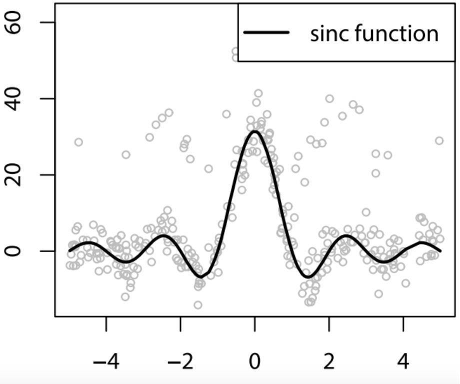
Gradient Tree Boosting

- One common form of gradient boosting machine takes

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

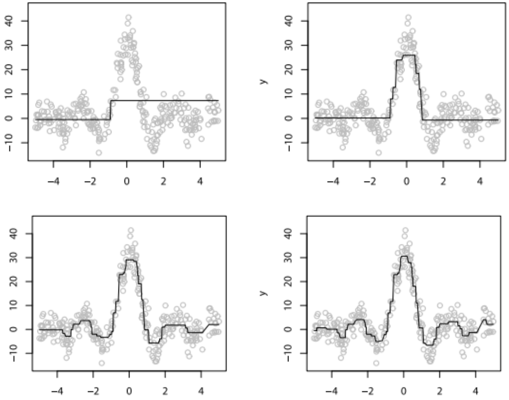
  where S is the number of terminal nodes.
- $S = 2$ gives decision stumps
- HTF recommends $4 \leqslant S \leqslant 8$ (but more recent results use much larger trees)
- Software packages:
  - Gradient tree boosting is implemented by the gbm package for R
  - as `GradientBoostingClassifier` and `GradientBoostingRegressor` 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 $\lambda = 1$.

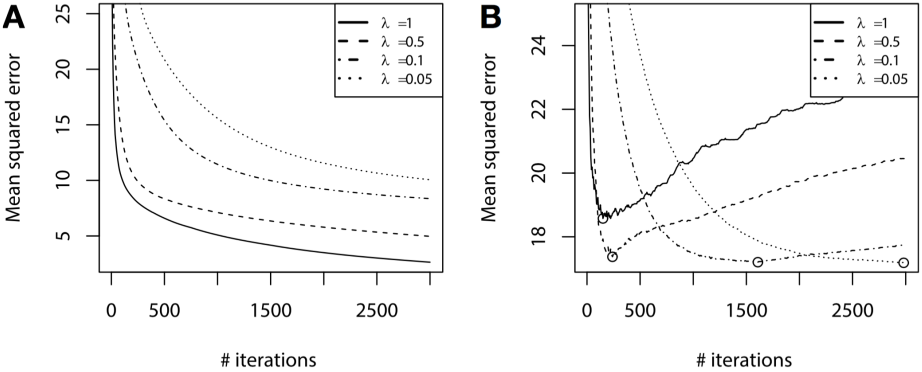
Figure 3 from Natekin and Knoll's "Gradient boosting machines, a tutorial"

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