Decision Trees and Boosting

Mengye Ren

NYU

Nov 14, 2023

Review: Decision Trees

- Non-linear, non-metric, and non-parametric.
- Regression or classification.
- Interpretable, up to certain depth.
- Greedy algorithm – maximizing the purity of nodes.
- Can overfit – need to limit the capacity.

Bagging and Random Forests

Recap: Statistics and Point Estimators

- We observe data $\mathcal{D} = (x_1, x_2, \ldots, x_n)$ sampled i.i.d. from a parametric distribution $p(\cdot \mid \theta)$
- A **statistic** $s = s(\mathcal{D})$ is any function of the data:
  - E.g., sample mean, sample variance, histogram, empirical data distribution
- A statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ is a **point estimator** of θ if $\hat{\theta} \approx \theta$

Recap: Bias and Variance of an Estimator

- Statistics are random, so they have probability distributions.
- The distribution of a statistic is called a **sampling distribution**.
- The standard deviation of the sampling distribution is called the **standard error**.
- Some parameters of the sampling distribution we might be interested in:

  Bias $\text{Bias}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}\right] - \theta.$

  Variance $\text{Var}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}^2\right] - \mathbb{E}^2\left[\hat{\theta}\right].$
- Why does variance matter if an estimator is unbiased?
  - $\hat{\theta}(\mathcal{D}) = x_1$ is an unbiased estimator of the mean of a Gaussian, but would be farther away from θ than the sample mean.

Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance σ^2: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\text{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate θ.
- Its standard error is $\sqrt{\text{Var}(\hat{\theta})} = \sigma$
- Consider a new estimator that takes the average of i.i.d. $\hat{\theta}_1, \ldots, \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(\mathcal{D}^i)$.
- The average has the same expected value but smaller standard error (recall that $Var(cX) = c^2 Var(X)$, and that the $\hat{\theta}_i$-s are uncorrelated):

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \theta \qquad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n} \tag{1}$$

Averaging Independent Prediction Functions

- Suppose we have B independent training sets, all drawn from the same distribution ($\mathcal{D} \sim p(\cdot \mid \theta)$).
- Our learning algorithm gives us B prediction functions: $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$
- We will define the average prediction function as:

$$\hat{f}_{\text{avg}} \stackrel{\text{def}}{=} \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b \tag{2}$$

Averaging Reduces Variance of Predictions

- The average prediction for x_0 is

$$\hat{f}_{\text{avg}}(x_0) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x_0).$$

- $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_b(x_0)$ have the same expected value, but
- $\hat{f}_{\text{avg}}(x_0)$ has smaller variance:

$$\text{Var}(\hat{f}_{\text{avg}}(x_0)) = \frac{1}{B}\text{Var}\left(\hat{f}_1(x_0)\right)$$

- Problem: in practice we don't have B independent training sets!

The Bootstrap Sample

How do we simulate multiple samples when we only have one?

- A **bootstrap sample** from $\mathcal{D}_n = (x_1, \ldots, x_n)$ is a sample of size n drawn *with replacement* from $\mathcal{D}_n$
- Some elements of $\mathcal{D}_n$ will show up multiple times, and some won't show up at all
- Each x_i has a probability of $(1-1/n)^n$ of not being included in a given bootstrap sample
- For large n,

$$\left(1-\frac{1}{n}\right)^n \approx \frac{1}{e} \approx .368. \tag{3}$$

- So we expect ~63.2% of elements of $\mathcal{D}_n$ will show up at least once.

The Bootstrap Method

Definition

A **bootstrap method** simulates B independent samples from P by taking B bootstrap samples from the sample $\mathcal{D}_n$.

- Given original data $\mathcal{D}_n$, compute B bootstrap samples $D_n^1, \ldots, D_n^B$.
- For each bootstrap sample, compute some function

$$\phi(D_n^1), \ldots, \phi(D_n^B)$$

- Use these values as though $D_n^1, \ldots, D_n^B$ were i.i.d. samples from P.
- This often ends up being very close to what we'd get with independent samples from P!

Independent Samples vs. Bootstrap Samples

- Point estimator $\hat{\alpha} = \hat{\alpha}(\mathcal{D}_{100})$ for samples of size 100, for a synthetic case where the data generating distribution is known
- Histograms of $\hat{\alpha}$ based on
  - 1000 independent samples of size 100 (left), vs.
  - 1000 bootstrap samples of size 100 (right)

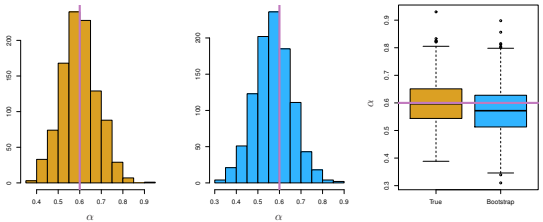


Figure 5.10 from *ISLR* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Ensemble Methods

Key ideas:

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias
- We can use bootstrap to simulate multiple data samples and average them
- Parallel ensemble (e.g., bagging): models are built independently
- Sequential ensemble (e.g., boosting): models are built sequentially
  - We try to find new learners that do well where previous learners fall short

Bagging: Bootstrap Aggregation

- We draw B bootstrap samples $D^1, \ldots, D^B$ from original data $\mathcal{D}$
- Let $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_B$ be the prediction functions resulting from training on $D^1, \ldots, D^B$, respectively
- The **bagged prediction function** is a *combination* of these:

$$\hat{f}_{\text{avg}}(x) = \text{Combine}\left(\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)\right)$$

Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees
- For classification, averaging doesn't make sense; we can take a **majority vote** instead
- Increasing the number of trees we use in bagging does not lead to overfitting
- Is there a downside, compared to having a single decision tree?
- Yes: if we have many trees, the bagged predictor is much less interpretable

Aside: Out-of-Bag Error Estimation

- Recall that each bagged predictor was trained on about 63% of the data.
- The remaining 37% are called **out-of-bag (OOB)** observations.
- For ith training point, let

$$S_i = \left\{ b \mid D^b \text{ does not contain } i\text{th point} \right\}$$

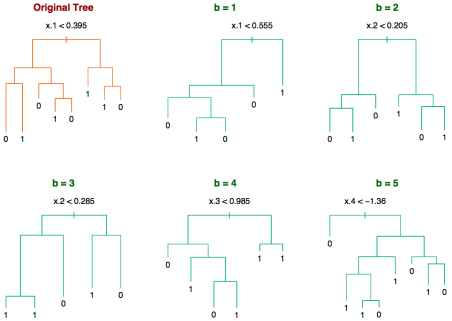
- The **OOB prediction** on x_i is

$$\hat{f}_{\text{OOB}}(x_i) = \frac{1}{|S_i|} \sum_{b \in S_i} \hat{f}_b(x_i)$$

- The OOB error is a good estimate of the test error
- Similar to cross validation error: both are computed on the training set

Applying Bagging to Classification Trees

- Input space $\mathcal{X} = \mathrm{R}^5$ and output space $\mathcal{Y} = \{-1, 1\}$. Sample size $n = 30$.



- Each bootstrap tree is quite different: different splitting variable at the root!
- **High variance**: small perturbations of the training data lead to a high degree of model variability
- Bagging helps most when the base learners are relatively unbiased but have high variance (exactly the case for decision trees)

From HTF Figure 8.9

Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large n, the covariance term dominates, limiting the benefits of averaging
- Bootstrap samples are
  - independent samples from the training set, but
  - not independent samples from $P_{\mathcal{X}\times\mathcal{Y}}$
- Can we reduce the dependence between $\hat{f}_i$'s?

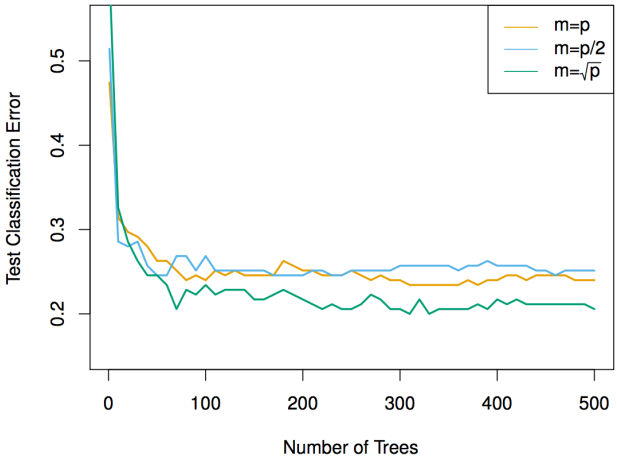
Random Forests

Key idea

Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel), as before
- When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size m
  - This prevents a situation where all trees are dominated by the same small number of strong features (and are therefore too similar to each other)
- We typically choose $m \approx \sqrt{p}$, where p is the number of features (or we can choose m using cross validation)
- If $m = p$, this is just bagging

Random Forests: Effect of m



From *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Review

- The usual approach is to build very deep trees—low bias but high variance
- Ensembling many models reduces variance
  - Motivation: Mean of i.i.d. estimates has smaller variance than single estimate
- Use bootstrap to simulate many data samples from one dataset
  - $\Longrightarrow$ Bagged decision trees
- But bootstrap samples (and the induced models) are correlated
- Ensembling works better when we combine a diverse set of prediction functions
  - $\Longrightarrow$ Random forests: select a random subset of features for each decision tree

Boosting

Boosting: Overview

Bagging Reduce variance of a low bias, high variance estimator by ensembling many estimators trained in parallel (on different datasets obtained through sampling).

Boosting Reduce the error rate of a high bias estimator by ensembling many estimators trained in sequence (without bootstrapping).

- Like bagging, boosting is a general method that is particularly popular with decision trees.
- Main intuition: instead of fitting the data very closely using a large decision tree, train gradually, using a sequence of simpler trees

Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam
- **Key idea**:
  - Each weak learner focuses on different training examples (*reweighted data*)
  - Weak learners make different contributions to the final prediction (*reweighted classifier*)
- A set of smaller, simpler trees may improve interpretability
- We'll focus on a specific implementation, AdaBoost (Freund & Schapire, 1997)

AdaBoost: Setting

- Binary classification: $\mathcal{Y} = \{-1, 1\}$
- Base hypothesis space $\mathcal{H} = \{h : \mathcal{X} \to \{-1, 1\}\}$.
- Typical base hypothesis spaces:
  - **Decision stumps** (tree with a single split)
  - Trees with few terminal nodes
  - Linear decision functions

Weighted Training Set

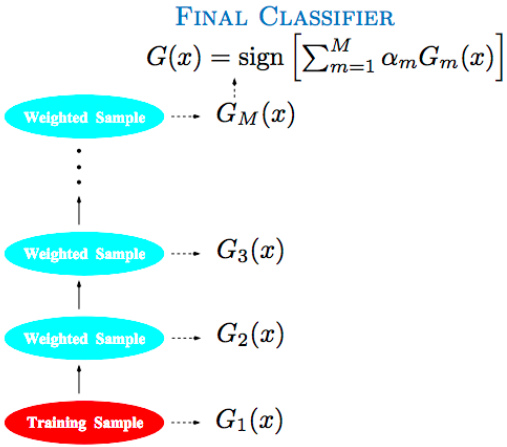
Each base learner is trained on weighted data.

- Training set $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$.
- Weights $(w_1, \ldots, w_n)$ associated with each example.
- **Weighted empirical risk**:

$$\hat{R}_n^w(f) \stackrel{\text{def}}{=} \frac{1}{W} \sum_{i=1}^{n} w_i \ell(f(x_i), y_i) \quad \text{where } W = \sum_{i=1}^{n} w_i$$

- Examples with larger weights affect the loss more.

AdaBoost: Schematic



From ESL Figure 10.1

AdaBoost: Sketch of the Algorithm

- Start with equal weights for all training points: $w_1 = \cdots = w_n = 1$
- Repeat for $m = 1, \ldots, M$ (where M is the number of classifiers we plan to train):
  - Train base classifier $G_m(x)$ on the weighted training data; this classifier may not fit the data well
  - Increase the weight of the points misclassified by $G_m(x)$ (this is the key idea of boosting!)
- Our final prediction is $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

AdaBoost: Classifier Weights

- Our final prediction is $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
- We would like α_m to be:
  - Nonnegative
  - Larger when G_m fits its weighted training data well
- The **weighted 0-1 error** of $G_m(x)$ is

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

- $\text{err}_m \in [0, 1]$

AdaBoost: Classifier Weights

- The weight of classifier $G_m(x)$ is $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$

- Higher weighted error $\Longrightarrow$ lower weight

AdaBoost: Example Reweighting

- We train G_m to minimize weighted error; the resulting error rate is err_m
- Then $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of G_m in the final ensemble

We want the next base learner to focus more on examples misclassified by the previous learner.

- Suppose w_i is the weight of example x_i before training:
  - If G_m classifies x_i correctly, keep w_i as is
  - Otherwise, increase w_i:

$$\begin{aligned} w_i &\leftarrow w_i e^{\alpha_m} \\ &= w_i \left(\frac{1-\text{err}_m}{\text{err}_m}\right) \end{aligned}$$

  - If G_m is a strong classifier overall, then its α_m will be large; this means that if x_i is misclassified, w_i will increase to a greater extent

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*:

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

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

AdaBoost with Decision Stumps

- After 1 round:

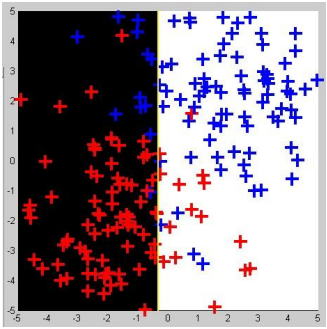


Figure: Size of plus sign represents weight of example. Blackness represents preference for red class; whiteness represents preference for blue class.

KPM Figure 16.10

AdaBoost with Decision Stumps

- After 3 rounds:

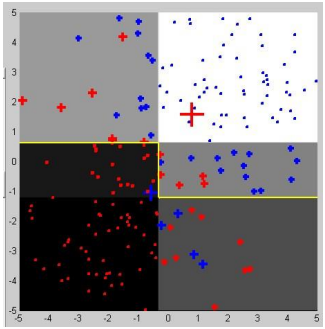


Figure: Size of plus sign represents weight of example. Blackness represents preference for red class; whiteness represents preference for blue class.

KPM Figure 16.10

AdaBoost with Decision Stumps

- After 120 rounds:

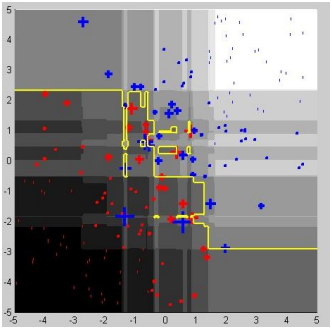
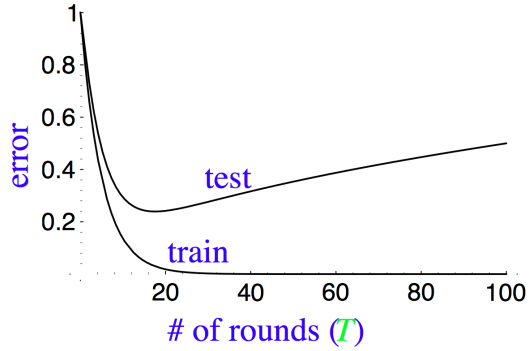


Figure: Size of plus sign represents weight of example. Blackness represents preference for red class; whiteness represents preference for blue class.

KPM Figure 16.10

Does AdaBoost overfit?

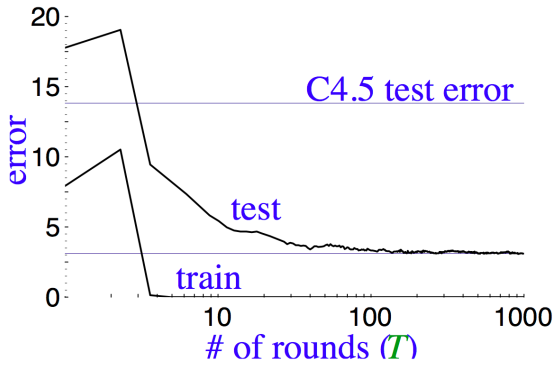
- Does a large number of rounds of boosting lead to overfitting?
- If we were overfitting, the learning curves would look like:



From Rob Schapire's NIPS 2007 Boosting tutorial.

Learning Curves for AdaBoost

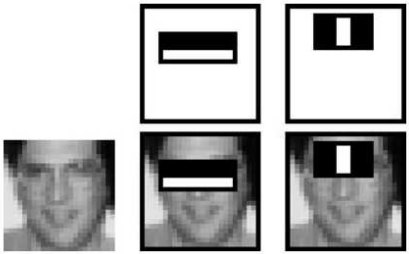
- AdaBoost is usually quite resistant to overfitting
- The test error continues to decrease even after the training error drops to zero!



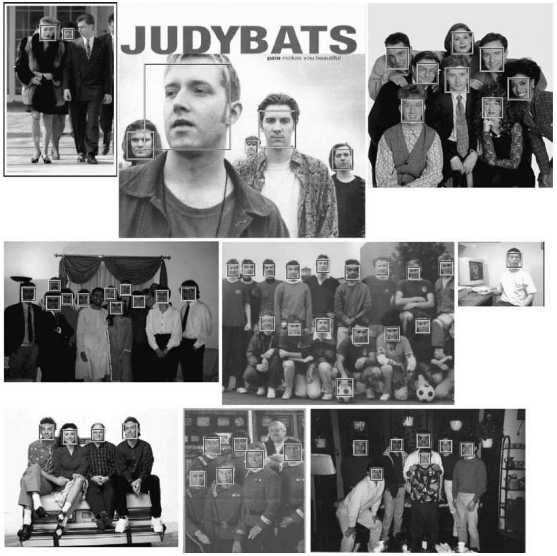
From Rob Schapire's NIPS 2007 Boosting tutorial.

AdaBoost for Face Detection

- Famous application of boosting: detecting faces in images (Viola & Jones, 2001)
- A few twists on standard algorithm
  - Pre-define weak classifiers, so optimization=selection
  - Smart way to do inference in real-time (in 2001 hardware)



AdaBoost Face Detection Results

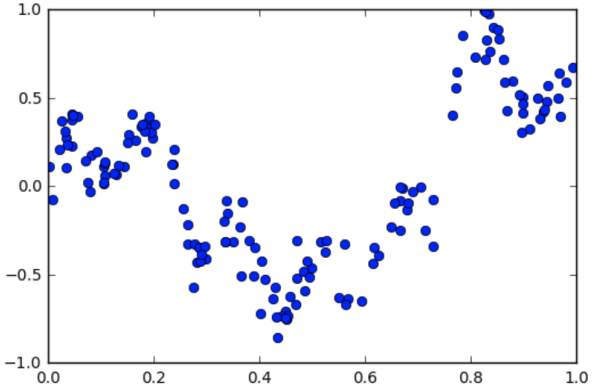


Interim Summary

- Boosting is used to reduce bias from shallow decision trees
- Each classifier is trained to reduce errors of its previous ensemble.
- AdaBoost is a very powerful off-the-shelf classifier.
- Next
  - What is the objective function of AdaBoost?
  - Generalizations to other loss functions
  - Gradient Boosting

Nonlinear Regression

- How do we fit the following data?
- Another way to get non-linear models in a linear form—adaptive basis function models.



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:

$$f(x) = \sum_{m=1}^{M} v_m h_m(x), \tag{4}$$

  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 $\Theta = \mathrm{R}^b$:

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

- 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?

Gradient Boosting

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 $\mathcal{H}$.

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.

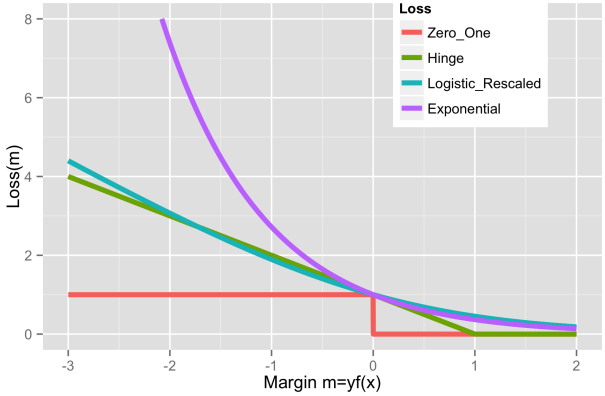
Forward Stagewise Additive Modeling for ERM

Let's plug in our objective function.

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

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:
   $$(v_m, h_m) = \underset{v \in \mathbb{R}, h \in \mathcal{H}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell_{\exp} \left(y_i, f_{m-1}(x_i) \underbrace{+ v h(x_i)}_{\text{new piece}} \right).$$
   2. Set $f_m = f_{m-1} + v_m h_m$.
3. Return: f_M.

FSAM with Exponential Loss: objective function

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

$$J(v, h) = \sum_{i=1}^{n} \exp\left[-y_i \left(f_{m-1}(x_i) + vh(x_i)\right)\right] \tag{5}$$

$$= \sum_{i=1}^{n} w_i^m \exp\left[-y_i vh(x_i)\right] \qquad w_i^m \stackrel{\text{def}}{=} \exp\left[-y_i f_{m-1}(x_i)\right] \tag{6}$$

$$= \sum_{i=1}^{n} w_i^m \left[\mathbb{I}(y_i = h(x_i))\, e^{-v} + \mathbb{I}(y_i \neq h(x_i))\, e^{v}\right] \qquad h(x_i) \in \{1, -1\} \tag{7}$$

$$= \sum_{i=1}^{n} w_i^m \left[(e^{v} - e^{-v})\mathbb{I}(y_i \neq h(x_i)) + e^{-v}\right] \qquad \mathbb{I}(y_i = h(x_i)) = 1 - \mathbb{I}(y_i \neq h(x_i)) \tag{8}$$

FSAM with Exponential Loss: basis function

- Objective function in the m'th round:

$$J(v, h) = \sum_{i=1}^{n} w_i^m \left[(e^v - e^{-v}) \mathbb{I}(y_i \neq h(x_i)) + e^{-v} \right]. \tag{9}$$

- If $v > 0$, then

$$\arg\min_{h \in \mathcal{H}} J(v, h) = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} w_i^m \mathbb{I}(y_i \neq h(x_i)) \tag{10}$$

$$h_m = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} w_i^m \mathbb{I}(y_i \neq h(x_i)) \tag{11}$$

$$= \arg\min_{h \in \mathcal{H}} \frac{1}{\sum_{i=1}^{n} w_i^m} \sum_{i=1}^{n} w_i^m \mathbb{I}(y_i \neq h(x_i)) \quad \text{multiply by a positive constant} \tag{12}$$

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:

$$\text{err}_m = \frac{\sum_{i=1}^{n} w_i^m \mathbb{I}(y_i \neq h(x_i))}{\sum_{i=1}^{n} w_i^m}. \tag{13}$$

- Exercise: show that the optimal v is:

$$v_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m} \tag{14}$$

  - Same as the classifier weights in Adaboost (differ by 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:

$$
\begin{aligned}
w_i^{m+1} &\stackrel{\text{def}}{=} \exp\left[-y_i f_m(x_i)\right] && (15)\\
&= w_i^m \exp\left[-y_i v_m h_m(x_i)\right] \qquad f_m(x_i) = f_{m-1}(x_i) + v_m h_m(x_i) && (16)\\
&= w_i^m \exp\left[-v_m \mathbb{I}(y_i = h_m(x_i)) + v_m \mathbb{I}(y_i \neq h_m(x_i))\right] && (17)\\
&= w_i^m \exp\left[2 v_m \mathbb{I}(y_i \neq h_m(x_i))\right] \underbrace{\exp^{-v_m}}_{\text{scaler}} && (18)
\end{aligned}
$$

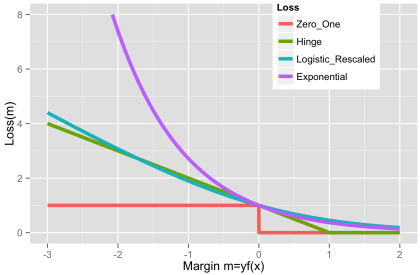
- The constant scaler will cancel out during normalization.
- $2v_m = \alpha_m$ in Adaboost.

Why Exponential Loss

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

$$f^*(x) = \frac{1}{2} \log \frac{p(y=1 \mid x)}{p(y=0 \mid x)}. \tag{19}$$

- How is it different from other losses?



AdaBoost / Exponential Loss: Robustness Issues

- Exponential loss puts a high penalty on misclassified examples.
  - $\Longrightarrow$ 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:

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

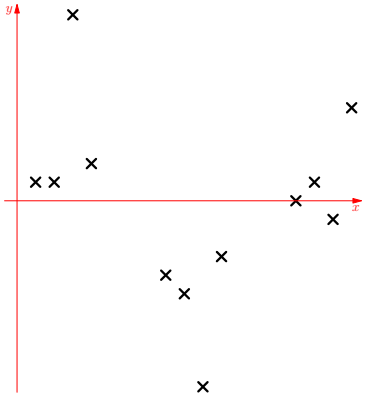
- If $\mathcal{H}$ is closed under rescaling (i.e. if $h \in \mathcal{H}$, then $vh \in \mathcal{H}$ for all $h \in \mathrm{R}$), then don't need v.
- Take $v = 1$ and minimize

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

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

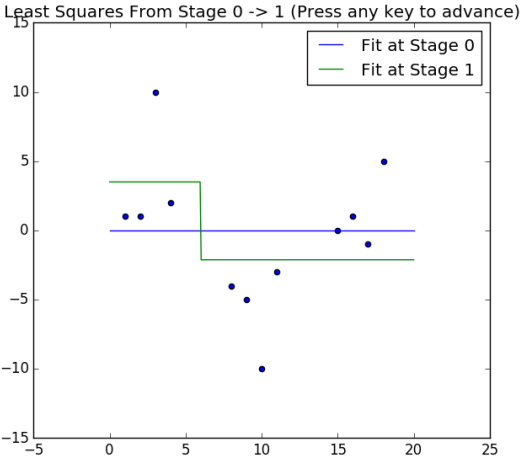
L^2 Boosting with Decision Stumps: Demo

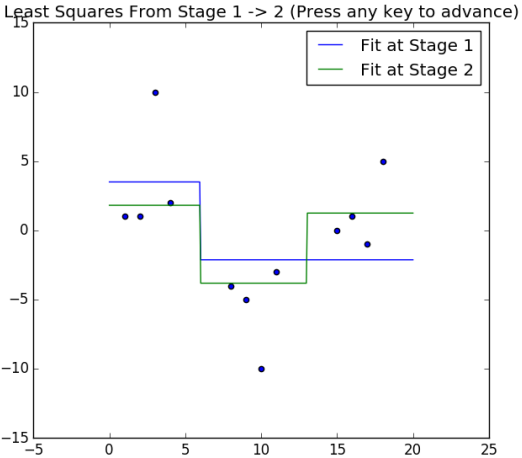
- Consider FSAM with L^2 loss (i.e. L^2 Boosting)
- For base hypothesis space of **regression stumps**



Plot courtesy of Brett Bernstein.

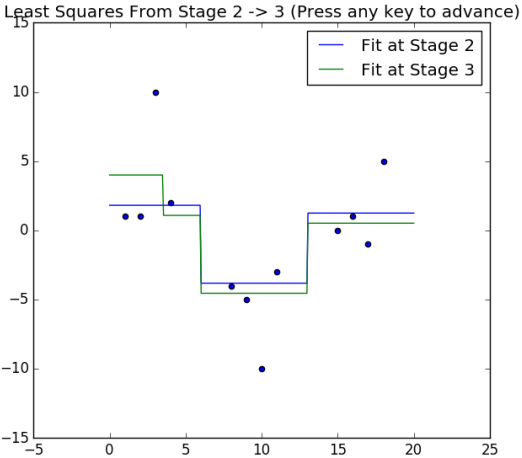
L^2 Boosting with Decision Stumps: Results

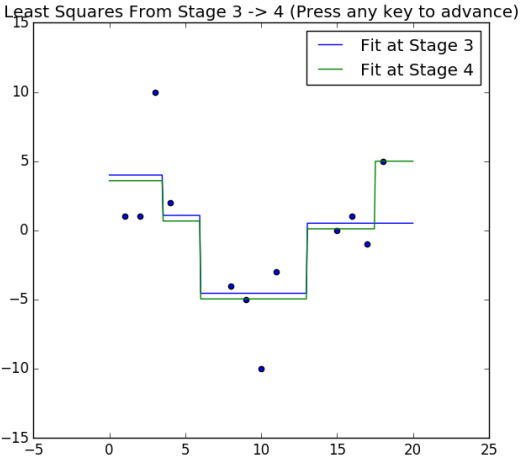




Plots and code courtesy of Brett Bernstein

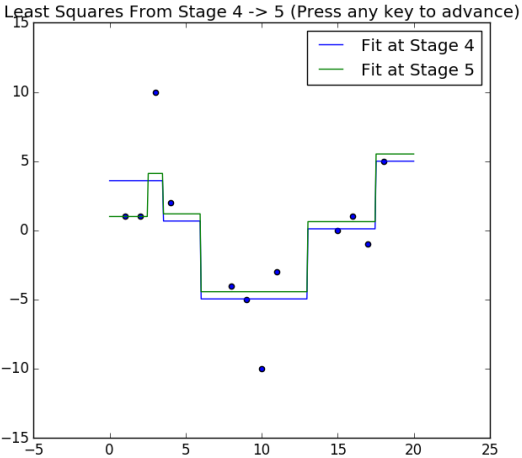
L^2 Boosting with Decision Stumps: Results

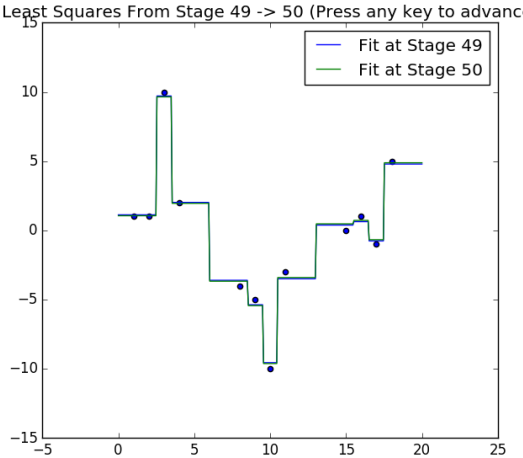




Plots and code courtesy of Brett Bernstein

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 $x = x_i$?

$$\frac{\partial}{\partial f(x_i)} J(f) = -2(y_i - f(x_i)) \tag{20}$$

  - 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.

$$f \leftarrow f + \nu h \qquad \text{FSAM / boosting} \tag{21}$$

$$f \leftarrow f - \alpha \nabla_f J(f) \qquad \text{gradient descent} \tag{22}$$

  - h approximates the gradient (step direction), ν 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"
$$\mathsf{f} = \left(f(x_1), \ldots, f(x_n)\right)^T$$
and write the objective function as
$$J(\mathsf{f}) = \sum_{i=1}^{n} \ell\left(y_i, \mathsf{f}_i\right).$$

Functional Gradient Descent: Unconstrained Step Direction

- Consider gradient descent on

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

- The negative gradient step direction at f is

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

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

Functional Gradient Descent: Projection Step

- Unconstrained step direction is

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

  - 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):

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \left(-\mathsf{g}_i - h(x_i)\right)^2. \qquad \text{least square regression} \qquad (23)$$

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

Recap

- Objective function:

$$J(f) = \sum_{i=1}^{n} \ell(y_i, f(x_i)). \tag{24}$$

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

$$\mathrm{g} = \nabla_{\boldsymbol{f}} J(\mathrm{f}) = \left(\partial_{\mathrm{f}_1} \ell(y_1, \mathrm{f}_1), \ldots, \partial_{\mathrm{f}_n} \ell(y_n, \mathrm{f}_n)\right). \tag{25}$$

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

$$h = \operatorname*{arg\,min}_{h \in \mathcal{H}} \sum_{i=1}^{n} \left(-\mathrm{g}_i - h(x_i)\right)^2. \tag{26}$$

- Gradient descent:

$$f \leftarrow f + \nu h \tag{27}$$

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**:

$$f_m \leftarrow f_{m-1} + \lambda v_m h_m \quad \text{where } \lambda \in [0, 1]. \tag{28}$$

  - 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):

$$r_{im} = -\left[\frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)\right]_{f(x_i)=f_{m-1}(x_i)} \tag{29}$$

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

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

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

$$r_i = -\frac{\partial}{\partial f(x_i)} \ell(y_i, f(x_i)) \tag{30}$$

$$= -\frac{\partial}{\partial f(x_i)} \left[\log\left(1 + e^{-y_i f(x_i)}\right)\right] \tag{31}$$

$$= \frac{y_i e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} \tag{32}$$

$$= \frac{y_i}{1 + e^{y_i f(x_i)}} \tag{33}$$

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

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

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

Gradient Tree Boosting

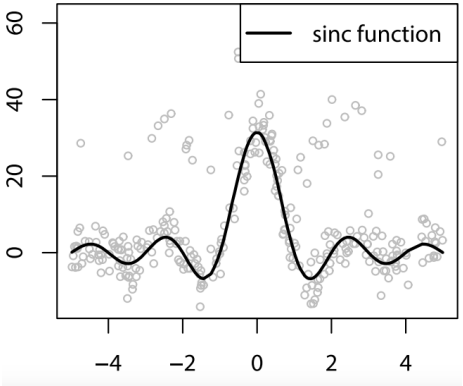
- One common form of gradient boosting machine takes

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

  where S is the number of terminal nodes.

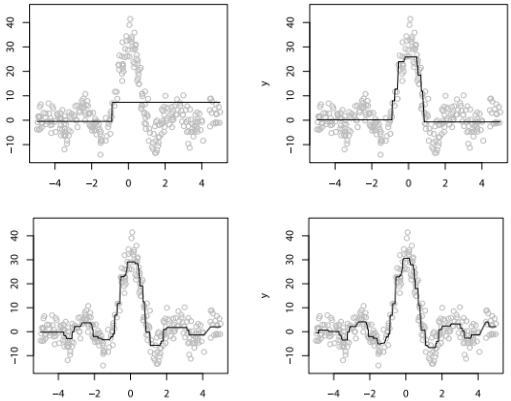
- $S = 2$ gives decision stumps
- HTF recommends $4 \leqslant S \leqslant 8$ (but more recent results use much larger trees)
- Software packages:
  - Gradient tree boosting is implemented by the gbm package for R
  - as `GradientBoostingClassifier` and `GradientBoostingRegressor` in sklearn
  - xgboost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

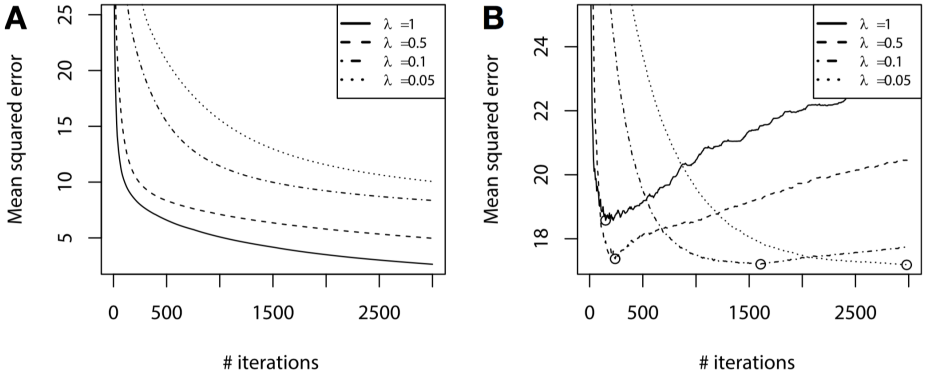
Figure 3 from Natekin and Knoll's "Gradient boosting machines, a tutorial"

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