Boosting
(source: David Rosenberg)

Yann Chevaleyre

November 4, 2022

Contents

Introduction

Adaptive Basis Function Model

- **Base hypothesis space**: $\mathcal{F}$ of $\hat{\mathcal{Y}}$**-valued functions**
- **Combined hypothesis space:** $\mathcal{F}_M$:

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

- Suppose we're given some data $S = ((x_1, y_1), \ldots, (x_n, y_n))$.
- Learning is choosing $v_1, \ldots, v_M \in \mathbf{R}$ and $h_1, \ldots, h_M \in \mathcal{F}$ to fit S.

Note:

in bagging, we learn h_i, but $v_i = \frac{1}{M}$ for all classifiers. Boosting will learn both!!

Empirical Risk Minimization

- We'll consider learning by **empirical risk minimization**:

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

  for some **loss function** $\ell(y, \hat{y})$.
- 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?

Gradient-Based Methods

- **Suppose** our base hypothesis space is parameterized by $\Theta = \mathbf{R}^d$:

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

- Can we can differentiate J w.r.t. v_m's and θ_m's? Optimize with SGD?
- 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 $\mathcal{F}$ consists of decision trees?
- Can we even parameterize trees with $\Theta = \mathbf{R}^b$?
- Even if we could for some set of trees,
  - predictions would not change continuously w.r.t. $\theta \in \Theta$,
  - and so certainly not differentiable.
- Today we'll discuss **boosting**. It applies whenever
  - we can compute a particular form of the above ERM (FSAM algorithms)
  - our loss function is [sub]differentiable w.r.t. training predictions $f(x_i)$, and we can do regression with the base hypothesis space $\mathcal{F}$ (gradient-boost).

Forward Stagewise Additive Modeling (FSAM)

Forward Stagewise Additive Modeling (FSAM)

- FSAM is an iterative optimization algorithm for fitting adaptive basis function models.
- Start with $f_0 \equiv 0$.
- After $m-1$ stages, we have

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

- In m'th round, we want to find
  - **step direction** $h_m \in \mathcal{F}$ (i.e. a basis function) and
  - **step size** $\nu_i > 0$
- such that

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

improves objective function value by as much as possible.

Forward Stagewise Additive Modeling for ERM

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

Application de FSAM à la Regression: L^2 Boosting

FSAM pour la Regression: L^2 Boosting

- Utilisons la “mean square error”.

$$L(\nu,h)=\frac{1}{n}\sum_{i=1}^{N}\left(y_i-\left[f_{m-1}(x_i)\quad\underbrace{+\nu h(x_i)}_{\text{nouveau classifieur}}\right]\right)^2$$

- Si $\mathcal{F}$ est “fermé par changement d'échelle” alors on peut oublier ν et n'apprendre que h.
- minimiser

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

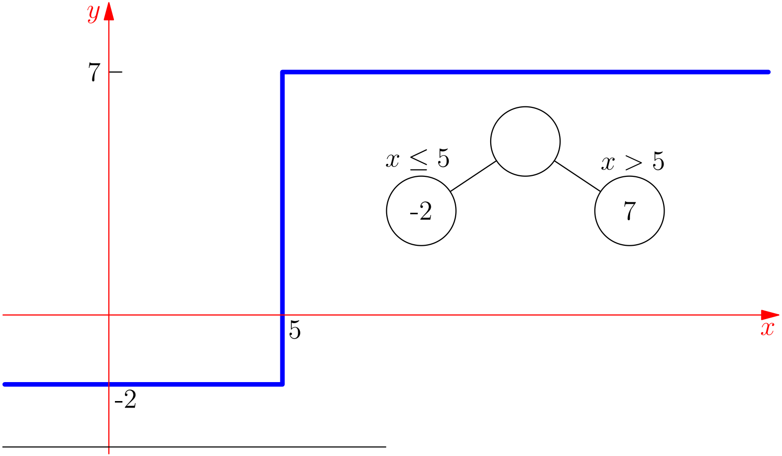
- Ce revient à faire un moindre carré sur les résidus !
- L'algorithme s'appelle parfois “matching pursuit”

L^2 Boosting - interprétation géométrique

(au tableau)

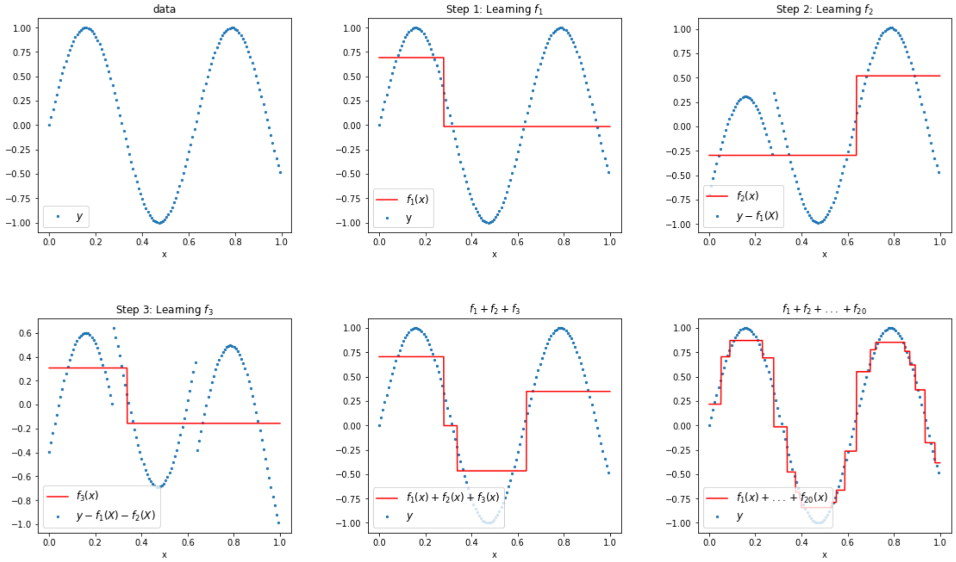
Regression Stumps

- A **regression stump** is a regression tree with a single split.
- A **regression stump** is a function of the form $h(x) = a1(x_i \leqslant c) + b1(x_i > c)$.



Plot courtesy of Brett Bernstein.

FSAM L^2 Boosting with Decision Stumps: Demo



after step 1
residue
$= y - f_1$

Application de FSAM à la Classification: Algorithme AdaBoost

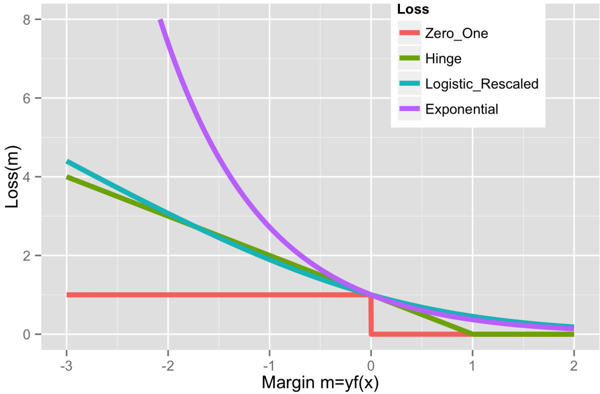
The Classification Problem

- Outcome space $\mathcal{Y} = \{-1, 1\}$
- The set of *base classifiers* is $\mathcal{F} \subset \mathcal{X} \mapsto \{-1, 1\}$ (e.g. decision stumps)
- We want to learn a *scoring function* $f_M \in \mathcal{F}_M$ where

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

- As usual, this scoring function induces a "hard"-classifier $\operatorname{sign}(f_M(x))$
- Can we optimize a scoring loss $f_M = \arg\min_{f \in \mathcal{F}_M} \sum_{i=1}^{N} \ell(f(x_i), y_i)$?

Scoring Losses for Classification



- All these losses are *well calibrated*
- For these functions, a direct computation of FSAM is not easy.
- For the **exp loss**: $\ell(f(x), y) = \exp(-yf(x))$ there is an "indirect" algo: **Adaboost**

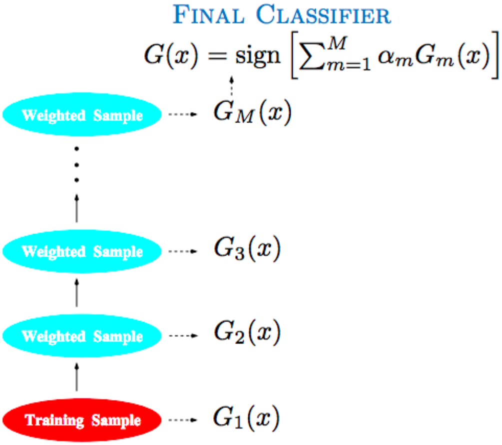
AdaBoost - Rough Sketch

- Training set $S = ((x_1, y_1), \ldots, (x_n, y_n))$.
- Start with equal weight on all training points $w_1 = \cdots = w_n = 1$.
- Repeat for $m = 1, \ldots, M$:
  - Find base classifier $h_m(x)$ that **tries** to fit weighted training data with 0/1 loss

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

  - Increase weight w_i on the points $h_m(x)$ misclassifies
- So far, we've generated M classifiers: $h_1, \ldots, h_M : \mathcal{X} \to \{-1, 1\}$.
- Final scoring function is $f_M(x) = \sum_{m=1}^{M} v_m h_m(x)$, for some weights v_m.

AdaBoost: Schematic



From ESL Figure 10.1

AdaBoost: Algorithm

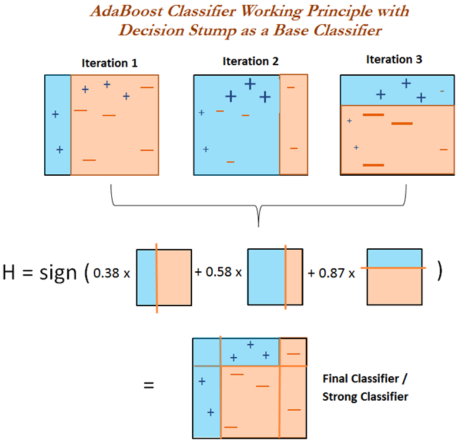
Given training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.

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

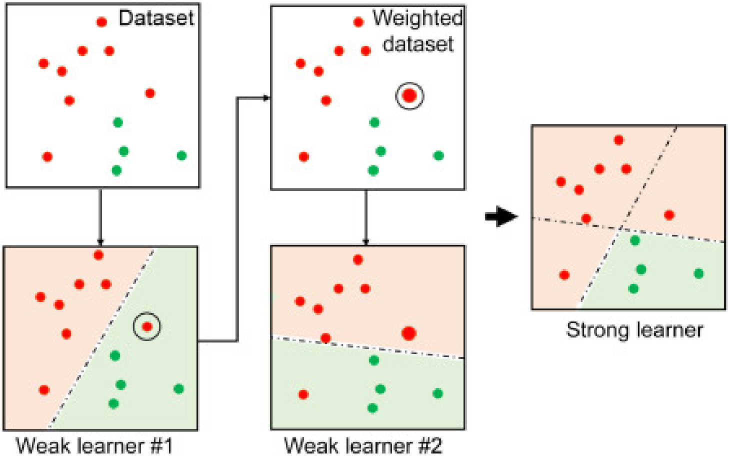
$$\mathrm{err}_m = \frac{1}{W} \sum_{i=1}^{n} w_i 1(y_i \neq h_m(x_i)) \quad \text{where } W = \sum_{i=1}^{n} w_i.$$

   3. Compute $v_m = \ln\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right)$ **[classifier weight]**
   4. Set $w_i \leftarrow w_i \cdot \exp\left[v_m 1(y_i \neq h_m(x_i))\right], \quad i = 1, 2, \ldots, n$ **[example weight adjustment]**
3. Ouput $f_M(x) = \sum_{m=1}^{M} v_m h_m(x)$.

AdaBoost: Pas à pas



AdaBoost: Pas à pas



Adaboost en tant qu'algorithme FSAM

- Soit $\ell(\hat{y}, y) = \exp(-\hat{y}y)$. Montrons qu'à chaque étape d'Adaboost, l'algorithme calcule

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

$$= \arg\min \frac{1}{n} \sum_i \exp(-y_i (f_{m-1}(x_i) + \nu h(x_i)))$$

$$= \arg\min \frac{1}{n} \sum_i \alpha_i \exp(-y_i \nu h(x_i))$$

with $\alpha_i = \exp(-y_i f_{m-1}(x_i))$

Analyser la convergence d'Adaboost: lien avec Hedge

Simplified variant of AdaBoost

- In this section, we will analyse a simplified version of Adaboost, which we will relate to the online learning *Hedge* algorithm.

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

1. Initialize observation weights $w_i = 1$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   1. learner fits weighted training data and returns $h_m(x)$
   2. Compute **weighted empirical 0-1 risk**:
   $$\mathrm{err}_m = \frac{1}{W} \sum_{i=1}^{n} w_i 1(y_i \neq h_m(x_i))$$
   3. Set $w_i \leftarrow w_i \cdot \exp\left[-\beta.1(y_i = h_m(x_i))\right], \quad i = 1, 2, \ldots, n$ **[example weight adjustment]**
3. Ouput $f_M(x) = \sum_{m=1}^{M} h_m(x)$.

Rappel: apprentissage en-ligne en 0/1 loss

Protocole

Pour $t = 1$ à T

- L'environnement choisit x_t, y_t, et révèle x_t à l'apprenant
- **L'apprenant prédit $\hat{y}_t$ (en général, il choisit h_t et prédit $\hat{y}_t = h_t(x_t)$)**
- L'environnement révèle y_t
- L'apprenant reçoit le cout de $\ell^{0/1}(\hat{y}_t, y_t)$

Objectif: produire la séquence de classifieurs $h_1 \dots h_T$ tels que la perte cumulée $\sum_{t=1}^{T} \ell^{0/1}(h_t(x_t), y_t)$ soit minimisée

Rappel: Hedge

- Je choisis $P_t(h) = \frac{1}{\Omega_t} w_{h,t}$ avec
  - $w_{h,1} = 1$
  - $w_{h,t+1} = w_{h,t} e^{-\beta \ell^{0/1}(h(x_t), y_t)}$ pour une constante $\beta > 0$

Algorithme Hedge (voir cours d'apprentissage en ligne)

$\mathcal{F}_1 = \mathcal{F}$
Pour $t = 1$ à T

- je reçois x_t
- **je tire** $h_t \sim P_t$
- je reçois le vrai label y_t, et ma prédiction me coute $\ell(h_t(x_t), y_t)$
- **je mets à jour** P_{t+1}

Thm: in expectation, $Regret_T = \sum_{t=1}^{T} \ell_t(h_t) - \min_{h \in \mathcal{F}} \sum_{t=1}^{T} \ell_t(h) \leqslant \sqrt{2T \ln |\mathcal{F}|}$

Problème dual de l'apprentissage en-ligne

- (on inverse la place de l'apprenant et de l'environnement)

Protocole dual

Pour $t = 1$ à T

- L'environnement choisit sans le révéler un classifieur h_t
- **L'apprenant choisit x_t, y_t de l'ensemble S**
- L'environnement révèle h_t
- L'apprenant reçoit le **gain** de $\ell(h_t(x_t), y_t)$

Objectif: *Trouver les exemples $(x_t, y_t) \in S$ qui maximisent la perte cumulée* $\sum_{t=1}^{T} \ell^{0/1}(h_t(x_t), y_t)$, ou qui minimisent $\sum_{t=1}^{T} \left(1 - \ell^{0/1}(h_t(x_t), y_t)\right)$

Dual Hedge

- Je choisis $P_t(i) = \frac{1}{\Omega_t} w_{i,t}$, distribution discrète sur $S = \{(x_1, y_1) \dots (x_N, y_N)\}$ avec:
  - $w_{i,1} = 1$, et $w_{i,t+1} = w_{i,t} e^{-\beta(1-\ell(h_t(x_i),y_i))}$ pour une constante $\beta > 0$

Algorithme Dual-Hedge

Pour $t = 1$ à T

- l'environnement choisit h_t sans le révéler
- **je tire** $(x_t, y_t) \sim P_t$
- je reçois h_t et mon choix me coute $1 - \ell(h_t(x_t), y_t)$
- **je mets à jour** P_{t+1} en calculant $w_{i,t+1} = w_{i,t} e^{-\beta(1-\ell(h_t(x_i),y_i))}$ pour tout i

Thm: we have in expectation,
$Regret_T = \sum_{t=1}^{T} (1 - \ell(h_t(x_t), y_t)) - \min_{i \in \{1..N\}} \sum_{t=1}^{T} (1 - \ell(h_t(x_t), y_t)) \leqslant \sqrt{2T \ln |S|}$

Dual Hedge

- La borne de regret est vraie pour TOUTE stratégie de l'environnement. On peut donc imposer une condition sur l'environnement sans changer la borne.

Algorithme Dual-Hedge

Pour $t = 1$ à T

- l'environnement choisit h_t tel que $err_t = \frac{1}{W}\sum_{i=1}^{N} w_i 1(y_i \neq h_t(x_i)) \leqslant \frac{1}{2} - \gamma$
- **...**
- **je mets à jour** P_{t+1} en calculant $w_{i,t+1} = w_{i,t} e^{-\beta(1-\ell(h_t(x_i),y_i))}$ pour tout i

Thm:

$$Regret_T = \mathbb{E}\left[\sum_{t=1}^{T}(1-\ell(h_t(x_t),y_t)) - \min_{i\in\{1..N\}}\sum_{t=1}^{T}(1-\ell(h_t(x_t),y_t))\right]$$

$$= \sum_{t=1}^{T}\mathbb{E}_{x,y\sim P_t}\left[(1-\ell(h_t(x),y)) - \min_{i\in\{1..N\}}\sum_{t=1}^{T}(1-\ell(h_t(x),y))\right] \leqslant \sqrt{2T\ln|S|}$$

Analyse de Dual-Hedge / Lien avec (Simplified) Adaboost

- Nous allons montrer que si, à chaque étape, h_t satisfait

$$err_t = \frac{1}{W}\sum_{i=1}^{N} w_i 1(y_i \neq h_t(x_i)) \leqslant \frac{1}{2} - \gamma \ \ \text{(weak learning hypothesis)}$$

  pour $\gamma \in]0, \frac{1}{2}[$, alors au bout de quelques étapes, le classifieur final aura une erreur empirique nulle.
- Notons que $err_t = \mathbb{E}_{x,y \sim P_t}\left(\ell^{0/1}(h_t(x), y)\right)$.
- Plus précisément:

Thm de convergence d'Adaboost: Sous l'hypothèse de weak learning, après $T = \frac{2}{\gamma^2} \ln N$ pas de temps, le classifieur majoritaire a un taux d'erreur de classification nulle sur l'échantillon S. (preuve au tableau)

Gradient Boosting / "Anyboost"

FSAM Is Iterative Optimization

- The FSAM step

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

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

"Functional" Gradient Descent

- We want to minimize

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

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

$$\mathbf{f}=\left(f(x_1),\ldots,f(x_n)\right)^T$$

and write the objective function as

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

Functional Gradient Descent: Unconstrained Step Direction

- Consider gradient descent on

$f_i = f(x_i)$

$J(f) = \sum \ell(y_i, f_i)$

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

$f = \begin{pmatrix} f_1 \\ \vdots \\ f_N \end{pmatrix} \in \mathbb{R}^N$

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

$g \in \mathbb{R}^N$

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

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.
- Eventually we need more than just $\mathbf{f}$, we'll need the function f.

Functional Gradient Descent: Projection Step

- Unconstrained step direction is

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

- Also called the "**pseudo-residuals**"
  - (for square loss, they're exactly the residuals)

$\sum \ell(y_i, f_i - g_i) \lesssim \sum \ell(y_i, f_i)$

- Find the closest base hypothesis $h \in \mathcal{F}$ (in the ℓ^2 sense):

$$\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left(-\mathbf{g}_i - h(x_i)\right)^2.$$

$\ell(y_i, f_{m-1}(x_i) + h(x_i))$

- This is a least squares regression problem over hypothesis space $\mathcal{F}$.
- Take the $h \in \mathcal{F}$ that best approximates $-\mathbf{g}$ as our step direction.

Functional Gradient Descent: Step Size

- Finally, we choose a stepsize.
- Option 1 (Line search):

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

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

The Gradient Boosting Machine Ingredients (Recap)

- Take any loss function [sub]differentiable w.r.t. the prediction
- 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!

Example: BinomialBoost

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:

$$\begin{aligned} r_i &= -\partial_{f(x_i)}\left[\log\left(1 + e^{-y_i f(x_i)}\right)\right] \\ &= \frac{y_i e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} \\ &= \frac{y_i}{1 + e^{y_i f(x_i)}} \end{aligned}$$

BinomialBoost: Gradient Boosting with Logistic Loss

- Pseudoresidual for ith example:

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

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

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

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

Gradient Tree Boosting

Gradient Tree Boosting

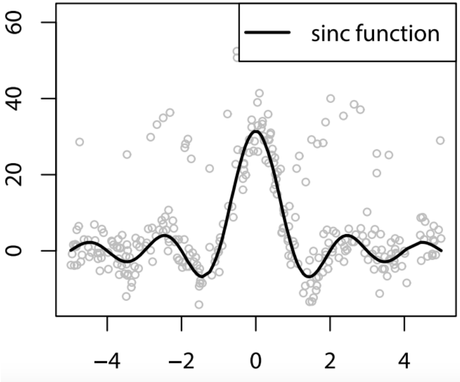
- One common form of gradient boosting machine takes

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

  where J is the number of terminal nodes.
- $J = 2$ gives decision stumps
- HTF recommends $4 \leqslant J \leqslant 8$ (but 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

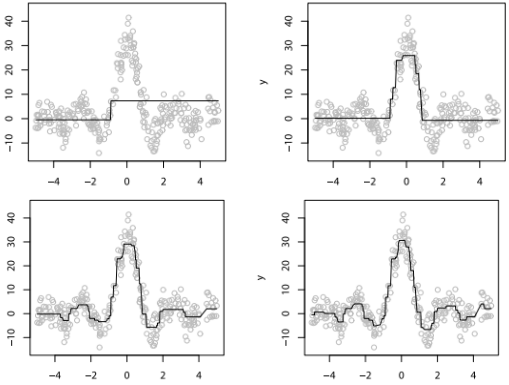
GBM Regression with Stumps

Sinc Function: Our Dataset



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

Minimizing Square Loss with Ensemble of Decision Stumps



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

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

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.