Boosting: from Adaboost to Gradient Boosting

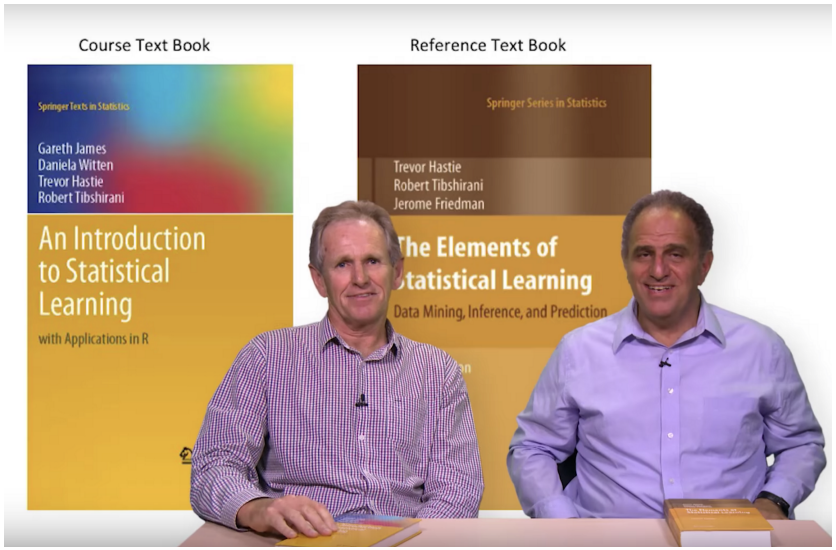
Jean-Michel Poggi
U. Paris Cité & LMO, Orsay, U. Paris-Saclay

Master 2 Course in Statistics
Universidad de la República – Facultad de Ingeniería,
Montevideo, Uruguay

February 2025

Plan

References [1]



- Hastie, T., Tibshirani, R., Friedman, J. *The elements of statistical learning: data mining, inference, and prediction.* Springer (2009)
- James, G., Witten, D., Hastie, T., Tibshirani, R. *An introduction to statistical learning: with applications in R.* Springer (2013)

[1] This document is also based on the course "Introduction au boosting" from Jean-Marc Lasgouttes (INRIA Paris) course of Mastere ESD, INSA Rouen, 2021

The history of Gradient Boosting in 3 references

- Invention of Adaboost, the first algorithm of boosting
  Freund and Schapire, 1997 [2]
- Formulating Adaboost as a gradient descent with a special loss function,
  Breiman 1998 [3]
- Generalization of Adaboost to Gradient Boosting to manage a variety of loss functions,
  Friedman et al., 2000 [4]

---

[2] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1):119-139.

[3] Breiman, L. (1998). Arcing classifiers. Annals of statistics, 26(3):801-849.

[4] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. Annals of statistics, 337-374.

Plan

Framework: the binary classification

$\mathcal{L}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ i.i.d. random variables, with the same distribution as (X, Y).
$X \in \mathcal{X} = \mathbb{R}^p$ (explanatory variables); we can also consider
$X \in \mathcal{X} = \mathbb{R}^{p'} \otimes \mathcal{Q}$ mixed.
$Y \in \mathcal{Y}$ (response) : $\mathcal{Y} = \{-1, 1\}$: binary classification

Goal: build a classifier (predictor) $\widehat{h} : \mathcal{X} \to \{-1, 1\}$

Sets of weak predictors (typically shallow trees) Freund, Schapire (1997)

- family of ensemble methods
- algorithm of adaptive statistical learning, successive models are built to correct the weaknesses of previous models

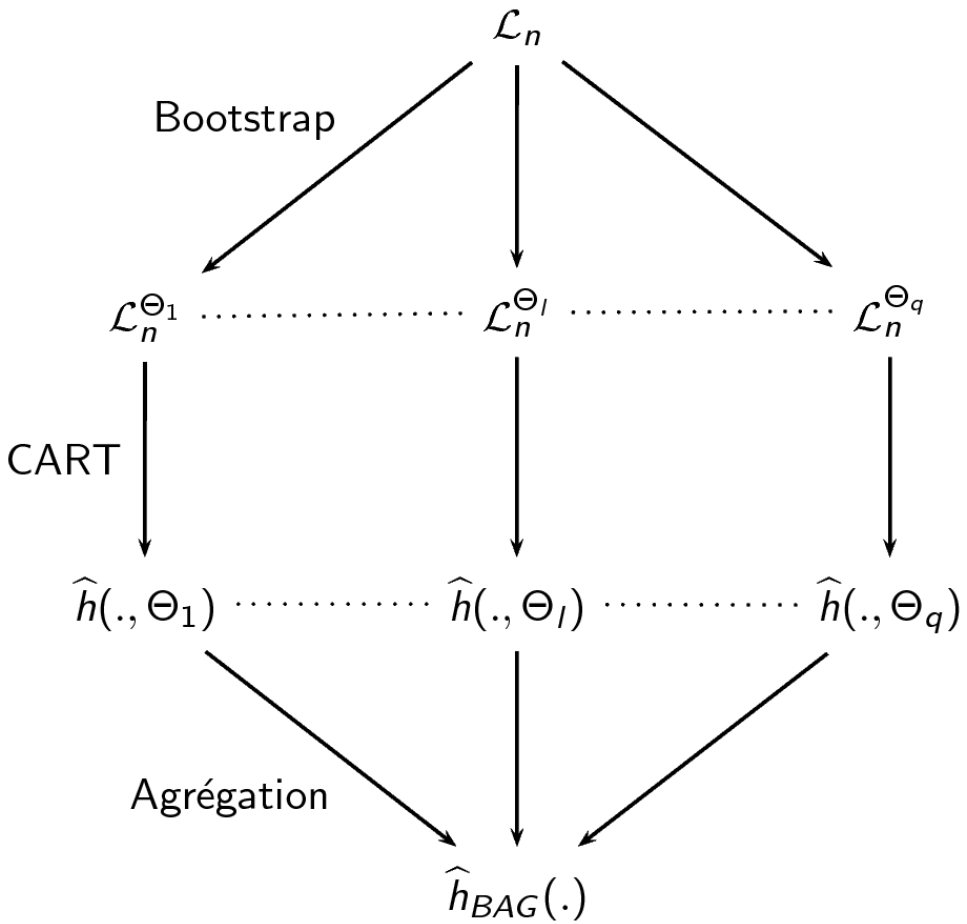
Weak classifiers

- We use a set of $\mathcal{H}$ weak classifiers $h : \mathcal{X} \to \{-1, 1\}$ very simple, which predict y just a little better than chance (i.e. with small γ) :

$$\epsilon = P(h(x) \neq y) \leq \frac{1}{2} - \gamma, \gamma > 0$$

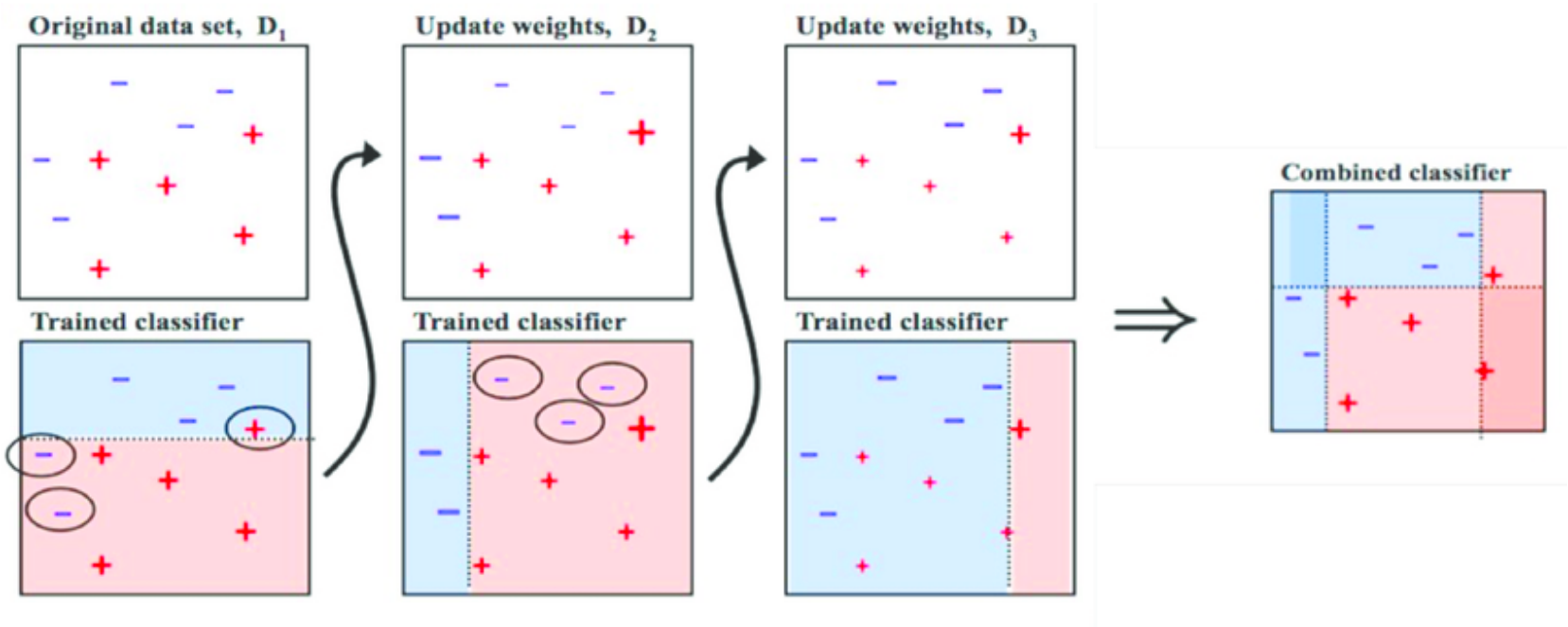
- For example :
  - shallow CART trees;
  - stumps (CART trees of depth 1).
- How can a strong classifier (very low error) be built from weak classifiers (error barely smaller than 0.5) using adaptive sampling, using an iterative (rather than parallel) approach?

Adaboost: from Bagging to Boosting



- Bagging: classifiers are built in parallel
- Boosting: the approach is iterative

Adaboost: a simple illustration with stumps



- weak rules: stump on the abscissa or ordinate;
- the weight of misclassified individuals is increased at each iteration;
- the final classifier is a linear combination of the classifiers built iteratively.

Adaboost: the algorithm

Input	$\mathcal{L}_n$, $\mathcal{H}$ of weak classifiers, M iterations
Initialization	$w_i = 1/n$, i from 1 to n
Iteration	`for` m from 1 to M,
	- `Draw` a bootstrap sample from $\mathcal{L}_n$ according to weights w_i
	- `Fit` a weak classifier $\widehat{h}_m(x)$ using the sample $\mathcal{L}_n^m$
	- `Compute` the error rate $\epsilon_m = \sum_{i=1}^{n} w_i \mathbb{1}_{\{y_i \neq \widehat{h}_m(x_i)\}}$
	- `Compute` the coefficient of $\widehat{h}_m$ in the output $\alpha_m = \log \sqrt{(1-\epsilon_m)/\epsilon_m}$
	- `Reweight` the observations (increasing the weight of misclassified obs. and renormalising) $w_i \leftarrow w_i \exp(-\alpha_m y_i \widehat{h}_m(x_i))/C$
Output	it is the sign of $\sum_{i=1}^{M} \alpha_m \widehat{h}_m(x)$, i.e. $\widehat{H}_M(x) = sign(\sum_{i=1}^{M} \alpha_m \widehat{h}_m(x))$

Table: Algorithm Adaboost

Adaboost: the algorithm (weight-based classifier version)

Input	$\mathcal{L}_n$, $\mathcal{H}$ of weak classifiers, M iterations
Initialization	$w_i = 1/n$, i from 1 to n
Iteration	`for` m from 1 to M,
	- `Weight` the sample $\mathcal{L}_n$ with the w_i: $\mathcal{L}_n^m$
	- `Fit` a weak classifier $\widehat{h}_m(x)$ using the sample $\mathcal{L}_n^m$
	- `Compute` the error rate $\epsilon_m = \sum_{i=1}^{n} w_i \mathbb{1}_{\{y_i \neq \widehat{h}_m(x_i)\}}$
	- `Compute` the coefficient of $\widehat{h}_m$ in the output $\alpha_m = \log\sqrt{(1-\epsilon_m)/\epsilon_m}$
	- `Reweight` the observations (increasing the weight of misclassified obs. and renormalising) $w_i \leftarrow w_i \exp(-\alpha_m y_i \widehat{h}_m(x_i))/C$
Output	it is the sign of $\sum_{i=1}^{M} \alpha_m \widehat{h}_m(x)$, i.e. $\widehat{H}_M(x) = sign(\sum_{i=1}^{M} \alpha_m \widehat{h}_m(x))$

Table: Algorithm Adaboost

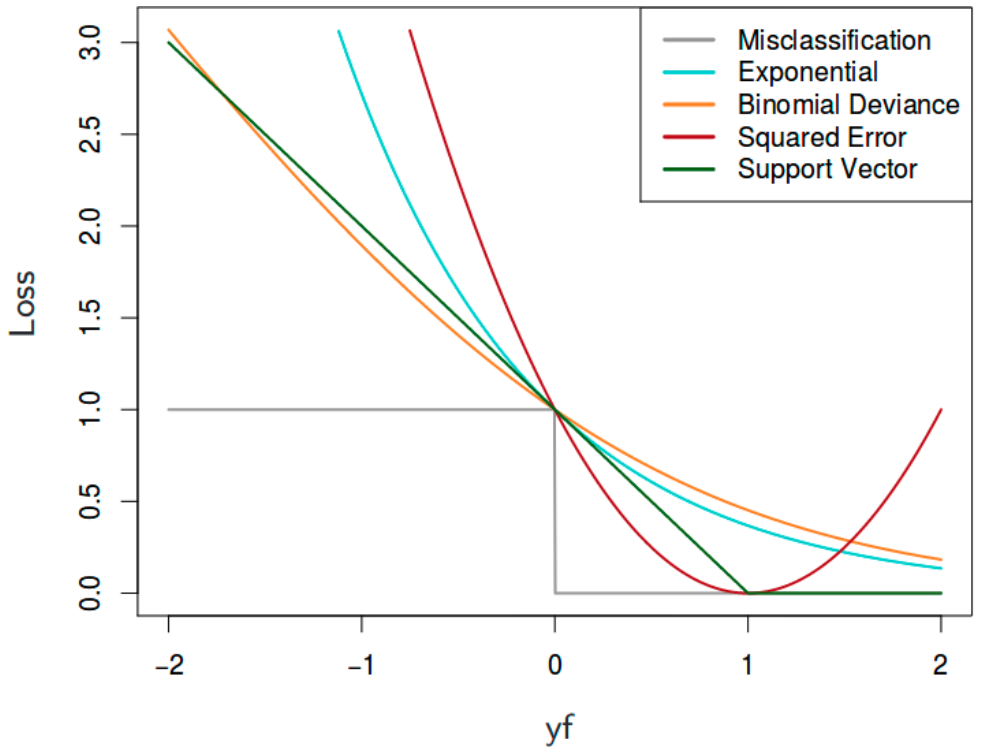
Plan

Additive linear modeling

- The framework is the same as that of Adaboost but there are two differences:
- The 1st difference is that to obtain h_m, instead of minimizing the misclassification rate ϵ at each iteration, we will minimize a more general loss function (also called deviance) noted $L(y, h)$, often assumed to be convex and differentiable. The new framework is therefore more general.
- The 2nd difference is in the algorithm: at each iteration, instead of changing the data weights, we model the residuals from the previous step, so that the model is built additively:

$$\widehat{H}_M(x) = \sum_{i=1}^{M} \beta_m \widehat{h}_m(x) = \widehat{H}_{M-1}(x) + \beta_M \widehat{h}_M(x)$$

Examples of loss functions



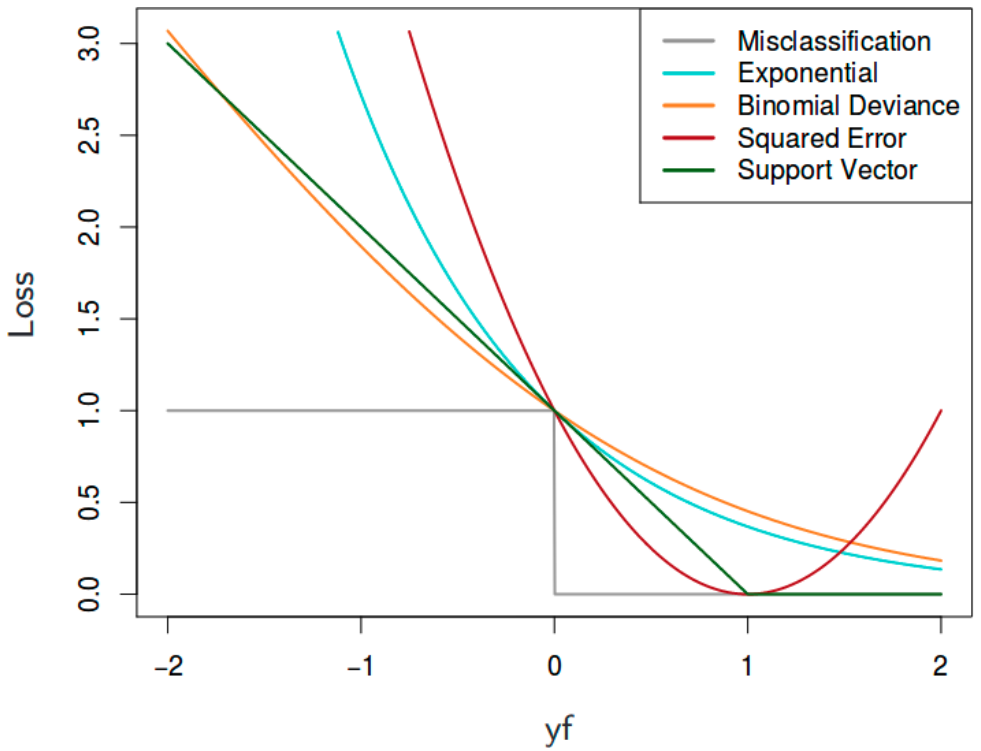
- Exponential
  $L(y, h) = \exp(-yh)$
- Logistic
  $L(y, h) = \log(1 + \exp(-2yh))$
- Quadratic
  $L(y, h) = (y - h)^2$

Additive linear modeling: the algorithm

Input	$\mathcal{L}_n$, $\mathcal{H}$ of weak classifiers, M iterations
Initialization	the initial predictor is $\widehat{H}_0(x) = 0$
Iteration	`for` m from 1 to M, – `Fit` a weak classifier $\widehat{h}_m(x)$ and a coefficient β_m which minimize in $\mathcal{H}$ $\sum_{i=1}^{n} L(y_i, \widehat{H}_{m-1}(x_i) + \beta_m \widehat{h}_m(x_i))$ – `Add` the resulting new term: $\widehat{H}_m(x) = \widehat{H}_{m-1}(x) + \beta_m \widehat{h}_m(x)$
Output	it is sign of $\widehat{H}_M(x)$

Table: Additive linear modeling: the algorithm

Loss functions for classification



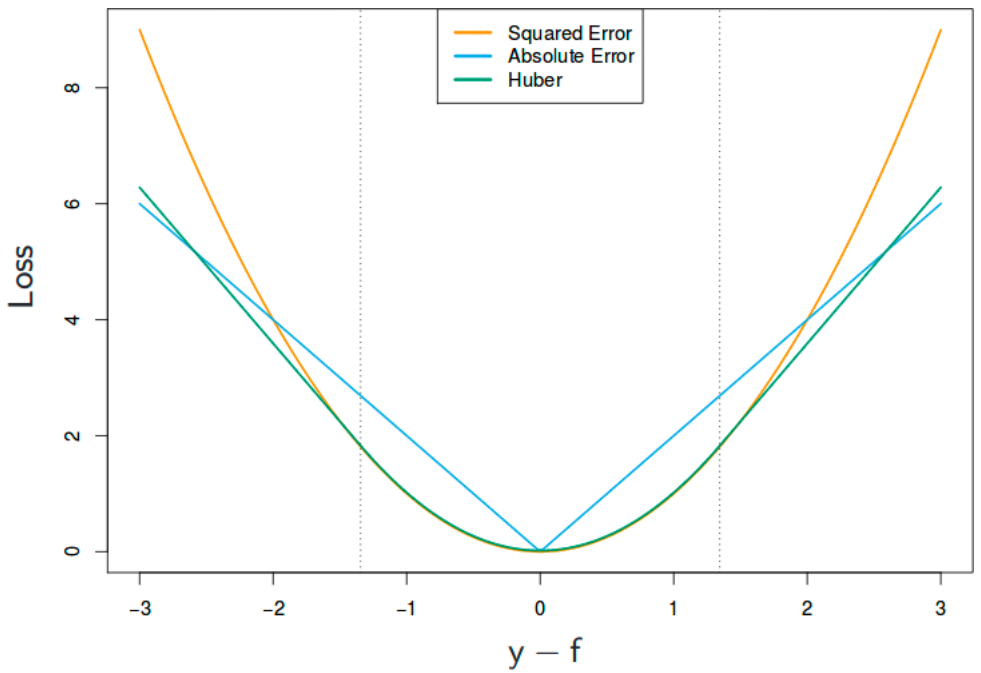
- Exponential
  $L(y, h) = \exp(-yh)$
  it can be shown that it is equivalent to Adaboost (nothing obvious at first sight).
- Logistic
  $L(y, h) = \log(1+\exp(-2yh))$
  Similar to AdaBoost, but less sensitive to misclassified observations.
- Quadratic
  $L(y, h) = (y - h)^2$. Not suitable here, the cost function is not decreasing.

Additive linear modeling: the algorithm

Input	$\mathcal{L}_n$, $\mathcal{H}$ of weak predictors, M iterations
Initialization	the initial predictor is $\widehat{H}_0(x) = 0$
Iteration	`for` m from 1 to M, – `Fit` a weak predictor $\widehat{h}_m(x)$ and a coefficient β_m which minimize (for L adapted for regression) $\sum_{i=1}^{n} L(y_i, \widehat{H}_{m-1}(x_i) + \beta_m \widehat{h}_m(x_i))$ – `Add` the resulting new term: $\widehat{H}_m(x) = \widehat{H}_{m-1}(x) + \beta_m \widehat{h}_m(x)$
Output	it is the value of $\widehat{H}_M(x)$

Table: Additive linear modeling for regression

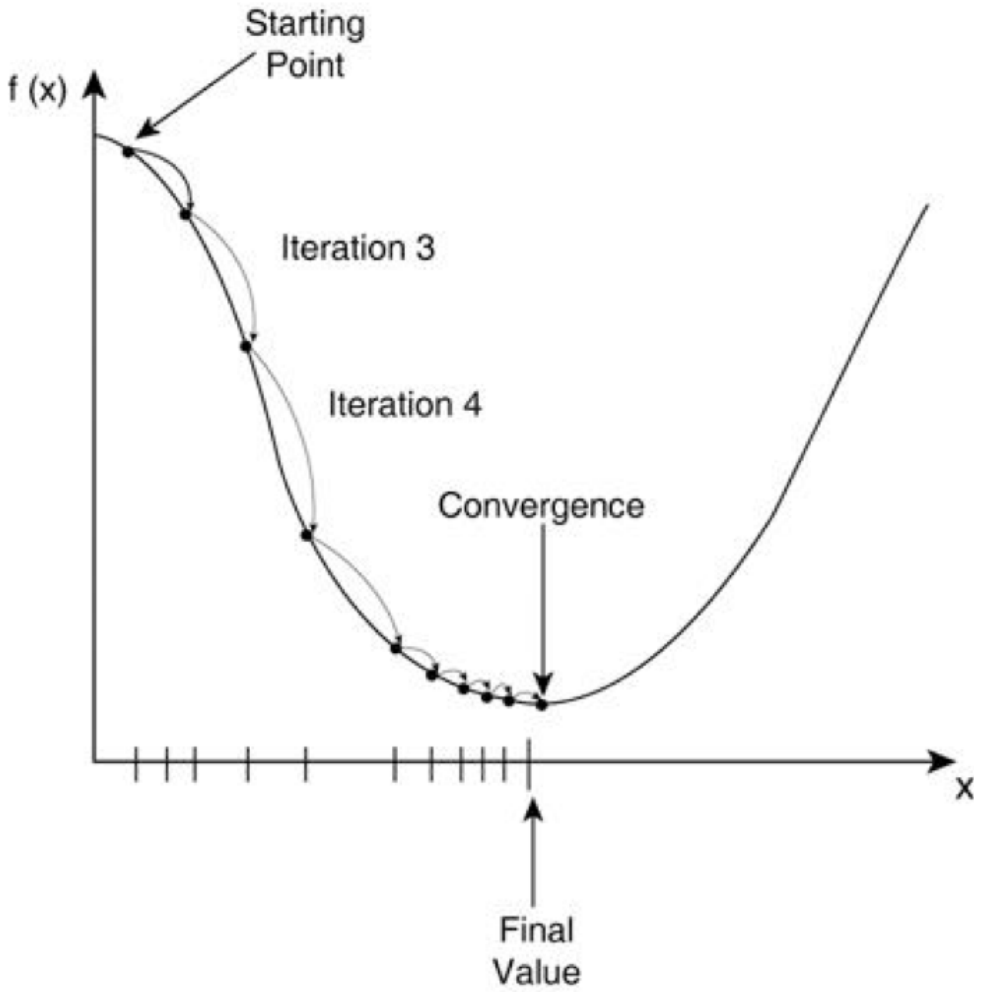
Loss functions for regression ($y \in \mathbb{R}$)



- Quadratic $L(y, h) = (y - h)^2$
  Perfectly suitable, but penalizes error outliers.
- Linear (absolute value)
  $L(y, h) = |y - h|$
  More robust against error outliers, but less accurate for small values.
- Huber's function (see green curve) makes it possible to reconcile the two previous functions and combine their qualities.

Plan

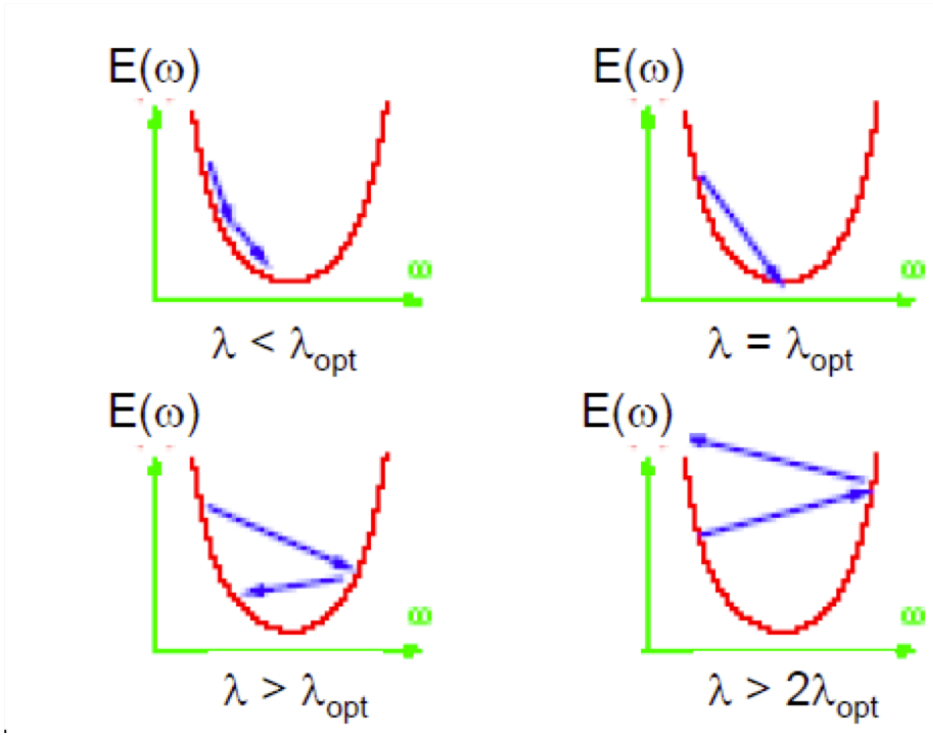
Gradient descent algorithm



- Problem: minimize the function $f : \mathbb{R} \to \mathbb{R}$ assumed to be convex and differentiable
- We fix $\lambda > 0$ and use the iterative algorithm

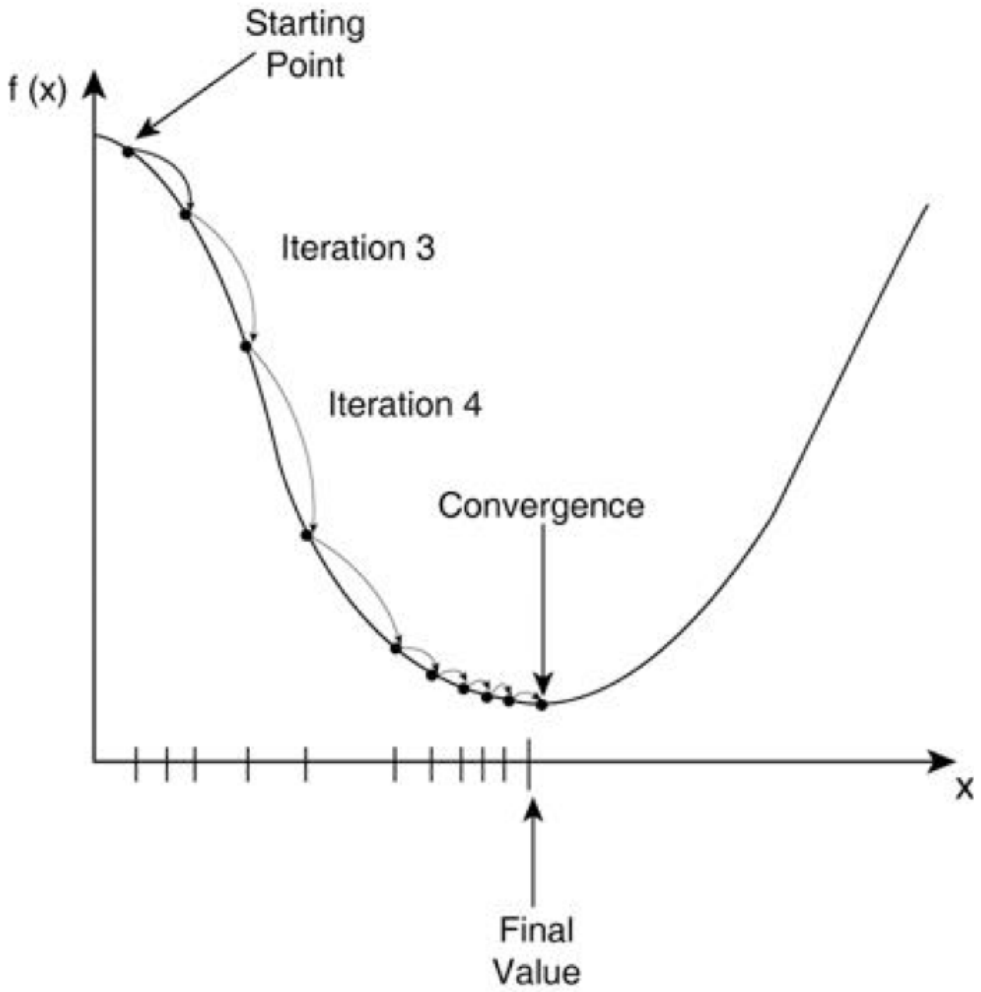
$$x_m = x_{m-1} - \lambda f'(x_{m-1})$$

Gradient descent algorithm



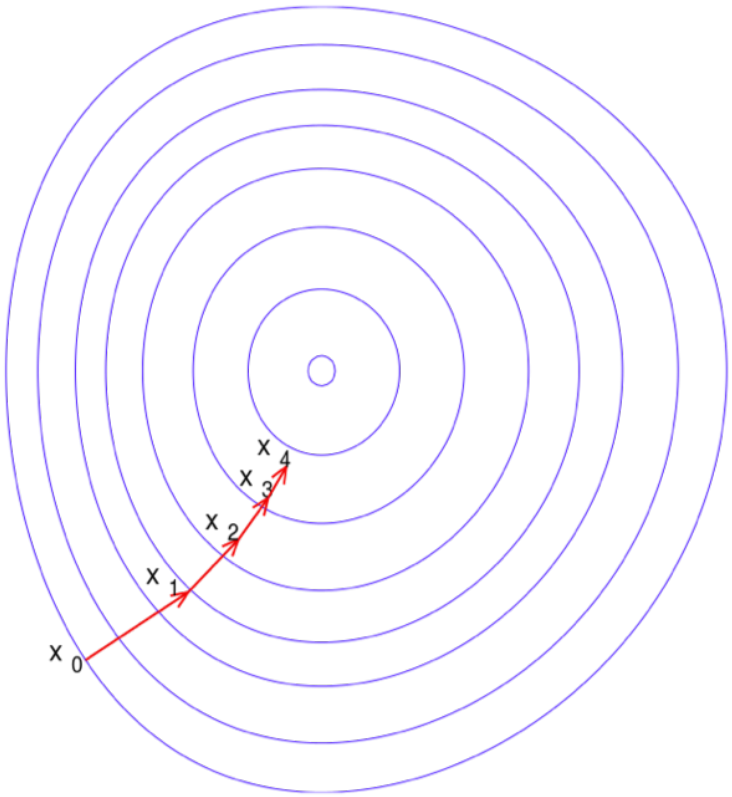
- Minimize the function $E : \mathbb{R} \to \mathbb{R}$
- We fix $\lambda > 0$ and use $x_m = x_{m-1} - \lambda E'(x_{m-1})$
- λ determines the speed of convergence (or even convergence)
- you can also make λ evolve over the iterations (large at the beginning to speed up convergence, small at the end to improve accuracy).

Gradient descent algorithm



- Problem: minimize the function $f : \mathbb{R} \to \mathbb{R}$ assumed to be convex and differentiable
- We fix $\lambda > 0$ and use the iterative algorithm $x_m = x_{m-1} - \lambda f'(x_{m-1})$
- Adaptation to the problem (functional gradient, calculate the gradient at any point): we use a weak rule to model the gradient.

Gradient descent algorithm



Source: `http://en.wikipedia.org/wiki/Gradient_descent`

- The basic principle is therefore the same as for AdaBoost, build a sequence of models so that each step, each model added to the combination, appears as a step towards a better solution.
- The first idea is that here this step is taken in the direction of the gradient of the loss function, to improve the convergence properties.
- The second idea is to approach the gradient using a regression tree to avoid overfitting.

Gradient Boosting in Least Square Regression

Input	$\mathcal{L}_n$, $\mathcal{H}$ of weak predictors, M iterations, $\lambda > 0$
Initialization	the initial predictor $\widehat{H}_0(x) = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - h(x_i))^2$
Iteration	`for` m from 1 to M, - `Compute` the residuals $u_i = y_i - \widehat{H}_{m-1}(x_i)$ - `Fit` a weak predictor $\widehat{h}_m(x)$ minimizing in $\mathcal{H}$ the $\sum_{i=1}^{n} (u_i - h(x_i))^2$ - `Add` the new term obtained: $\widehat{H}_m(x) = \widehat{H}_{m-1}(x) + \lambda \widehat{h}_m(x)$
Output	it is simply $\widehat{H}_M(x)$

Table: Gradient Boosting in Least Square Regression

Gradient Boosting in Least Square Regression: interpret the residuals as the opposite of the gradient

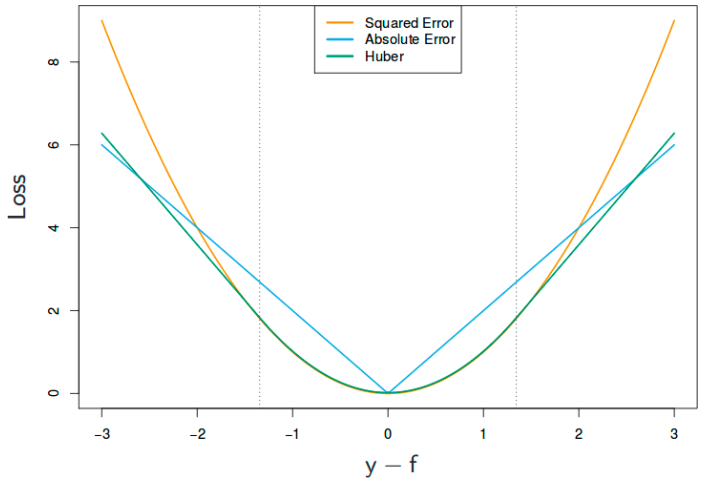
- How does this relate to gradient descent?
  - residual $\longleftrightarrow$ the opposite of the gradient;
    in fact, the gradient is the derivative of $L(y,h) = (y-h)^2/2$
    with respect to h : $\left.\frac{\partial L(y,h)}{\partial h}\right|_h = h - y = -(y-h)$
  - fit h to residual $\longleftrightarrow$ Fit h the opposite of the gradient
  - update H as a function of the residual $\longleftrightarrow$ update H as a function of the opposite of the gradient
- So we're actually updating our model using gradient descent!
- But the concept of gradient is better suited and more useful than the concept of residue for generalizing

Gradient Boosting for Regression

Input	$\mathcal{L}_n$, $\mathcal{H}$ of weak predictors, M iterations, the loss function L, $\lambda > 0$
Initialization	the initial predictor $\widehat{H}_0(x) = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, h(x_i))$
Iteration	`for` m from 1 to M, - `Compute` the opposite of the gradient (w.r.t. H) at the observation points $u_i = -\left.\frac{\partial L(y_i, H)}{\partial H}\right\vert_{H=\widehat{H}_{m-1}(\mathbf{x}_i)}$ - `Fit` a weak predictor $\widehat{h}_m(x)$ minimizing in $\mathcal{H}$ the $\sum_{i=1}^{n} L(u_i, \widehat{h}_m(x_i))$ - `Add` the new term obtained: $\widehat{H}_m(x) = \widehat{H}_{m-1}(x) + \lambda \widehat{h}_m(x)$
Output	it is simply $\widehat{H}_M(x)$

Table: Gradient Boosting for regression

Gradients of loss functions for regression



- Quadratic $L(y,h) = (y-h)^2/2$

  The gradient is the derivative of $L(y,h)$ w.r.t. h, $\left.\frac{\partial L(y,h)}{\partial h}\right|_h = -(y-h)$

- Linear (absolute value)
  $L(y,h) = |y-h|$

  $\left.\frac{\partial L(y,h)}{\partial h}\right|_h = sign(y-h)$

Settings

- The cost function L: see above;
- The set $\mathcal{H}$ of weak predictors:
  - We restrict ourselves to trees;
  - We then choose the depth of CART decision trees (without pruning) or even a stump which is very quick to calculate (but often too imprecise).
- The coefficient $\lambda > 0$ which sets the importance of the most recent predictor in the final predictor:
  - λ must (depending on the situation) be < 1 or more or less close to 1 if $\lambda > 1$ otherwise there is a risk of divergence;
  - when λ is small (relative to 1), the algorithm is slower but limits the risk of overfitting;
  - for Adaboost, we have $\lambda = 1$.
- Number of iterations M.