

Ensemble Methods

Error decomposition, bagging and random forests, boosting,
stacking and gradient boosting

Guillaume Metzler

Université Lumière Lyon 2 – Laboratoire ERIC, UR 3083
guillaume.metzler@univ-lyon2.fr

Master MALIA – 2026/2027

Outline

1. Error decomposition
2. Why combine models?
3. Bagging and random forests
4. Boosting
5. Stacking and the Super Learner
6. Gradient boosting
7. Summary

The question of this lecture. Why can a combination of models be better than each of them, and how do we build good combinations?

Every result is stated on one slide and proved on the next one: **take a few minutes to look for the proof** before moving on. Parts 3, 4 and 6 end with two short exercises (with solutions).

Lecture notes: *Statistical Supervised Machine Learning*, Section “Advanced Supervised Algorithms”.

Notations used in this lecture

- ▶ $(\mathbf{x}, y) \sim \mathcal{D}$: an example (features $\mathbf{x} \in \mathcal{X}$, label $y \in \mathcal{Y}$) drawn from an unknown distribution $\mathcal{D}$.
- ▶ $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \sim \mathcal{D}^m$: the training sample, i.i.d.
- ▶ $h : \mathcal{X} \rightarrow \mathcal{Y}$: a hypothesis (model), chosen in a class $\mathcal{H}$.
- ▶ $\ell(h(\mathbf{x}), y)$: the loss of h on $(\mathbf{x}, y)$, e.g. $(h(\mathbf{x}) - y)^2$ or $\mathbb{1}[h(\mathbf{x}) \neq y]$.
- ▶ True risk: $R(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\ell(h(\mathbf{x}), y)]$ (unknown).
- ▶ Empirical risk:
 $\hat{R}_S(h) = \frac{1}{m} \sum_{i=1}^m \ell(h(\mathbf{x}_i), y_i)$ (computable).
- ▶ Ensemble of T models: $h_1, \dots, h_T$, combined into H_T , e.g. $H_T = \frac{1}{T} \sum_t h_t$ or $H_T = \text{sign} \left(\sum_t \alpha_t h_t \right)$.
- ▶ $\mathbb{1}[A] = 1$ if A is true, 0 otherwise.

How each method is presented: intuition $\rightarrow$ algorithm $\rightarrow$ worked example on a few points $\rightarrow$ theory (statement, then proof on the next slide) $\rightarrow$ practice.

Part 1

Error decomposition

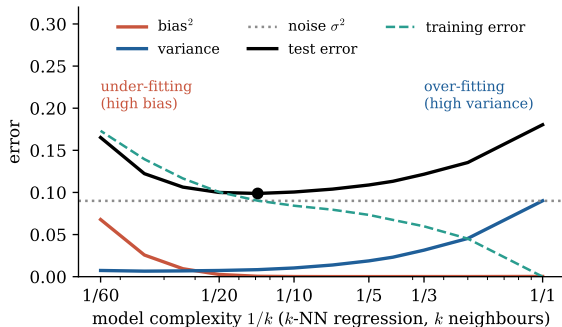
Question of this part. Before combining models, we need to know **which part of the error** a combination can reduce.

In this part we will:

- ▶ split the error of a model into **bias**, **variance** and noise (squared loss);
- ▶ do the same for any loss: approximation error and estimation error;
- ▶ measure the “size” of a class of models with the Rademacher complexity, and derive a generalization bound.

Main idea. The error of a model has components of different nature; each ensemble method targets one of them.

Performance and complexity



- ▶ The training error always decreases with the complexity of the model.
- ▶ The test error first decreases, then increases: the model starts to learn the *noise* of the sample.
- ▶ Here: k -NN regression of $\sin(2\pi x)$ ($m = 120$, noise $\sigma = 0.3$), errors estimated over 300 samples.

What is the smallest error we can hope for? Where do the two branches of the U-shaped curve come from?

We assume that the data are generated by

$$y = f(\mathbf{x}) + \varepsilon, \quad \mathbb{E}[\varepsilon] = 0, \quad \varepsilon \text{ independent of } \mathbf{x},$$

where ε is a noise (measurement errors, everything $\mathbf{x}$ cannot explain).

- ▶ From a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ generated by this model, we learn h , an **estimator** of f .
- ▶ h depends on S : it is a **random** function. Another sample gives another h .
- ▶ We measure the quality of h at a fixed point $\mathbf{x}$ with the mean squared error

$$\mathbb{E}[(y - h(\mathbf{x}))^2],$$

where the expectation is over the draw of S (through h) and over the noise of the new observation y .

Proposition (error decomposition)

With $y = f(\mathbf{x}) + \varepsilon$, $\mathbb{E}[\varepsilon] = 0$, and h learned from S , for every $\mathbf{x}$:

$$\mathbb{E}[(y - h(\mathbf{x}))^2] = \underbrace{(\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2}_{\text{bias}^2} + \underbrace{\mathbb{E}[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2]}_{\text{variance}} + \underbrace{\mathbb{E}[(y - f(\mathbf{x}))^2]}_{\text{noise}}.$$

- ▶ **Bias**: distance between the *average* model (over the samples S) and the target f .
- ▶ **Variance**: how much h moves around its average when the sample changes.
- ▶ Noise (Bayes error): the variance of ε . It does not depend on h .

Your turn. Tools: König–Huygens formula and independence.

Proof of the error decomposition (1/2)

Strategy. expand the square, then compute the three expectations $\mathbb{E}[y^2]$, $\mathbb{E}[y h(\mathbf{x})]$ and $\mathbb{E}[h(\mathbf{x})^2]$ separately. Write $\bar{h} = \mathbb{E}[h(\mathbf{x})]$ (average prediction over the samples S).

Step 1 – expand the square. By linearity of the expectation,

$$\mathbb{E}[(y - h(\mathbf{x}))^2] = \mathbb{E}[y^2] - 2\mathbb{E}[y h(\mathbf{x})] + \mathbb{E}[h(\mathbf{x})^2].$$

Step 2 – first term. König–Huygens, $\mathbb{E}[X^2] = \mathbb{E}[X]^2 + \text{Var}(X)$, applied to $X = y$, with $\mathbb{E}[y] = f(\mathbf{x})$ (since $\mathbb{E}[\varepsilon] = 0$):

$$\mathbb{E}[y^2] = f(\mathbf{x})^2 + \mathbb{E}[(y - f(\mathbf{x}))^2].$$

Step 3 – cross term. The new observation y is independent of the training sample S , hence of $h(\mathbf{x})$, so the expectation of the product is the product of the expectations:

$$\mathbb{E}[y h(\mathbf{x})] = \mathbb{E}[y] \mathbb{E}[h(\mathbf{x})] = f(\mathbf{x}) \bar{h}.$$

Step 4 – last term. König–Huygens applied to $X = h(\mathbf{x})$:

$$\mathbb{E}[h(\mathbf{x})^2] = \bar{h}^2 + \mathbb{E}[(h(\mathbf{x}) - \bar{h})^2].$$

Proof of the error decomposition (2/2)

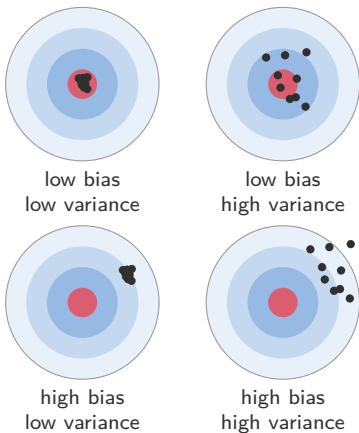
Step 5 – put everything together. Plugging Steps 2, 3 and 4 into Step 1:

$$\begin{aligned}\mathbb{E}[(y - h(\mathbf{x}))^2] &= f(\mathbf{x})^2 + \mathbb{E}[(y - f(\mathbf{x}))^2] - 2f(\mathbf{x})\bar{h} + \bar{h}^2 + \mathbb{E}[(h(\mathbf{x}) - \bar{h})^2] \\ &= \underbrace{f(\mathbf{x})^2 - 2f(\mathbf{x})\bar{h} + \bar{h}^2}_{= (\bar{h} - f(\mathbf{x}))^2} + \mathbb{E}[(h(\mathbf{x}) - \bar{h})^2] + \mathbb{E}[(y - f(\mathbf{x}))^2] \\ &= (\bar{h} - f(\mathbf{x}))^2 + \mathbb{E}[(h(\mathbf{x}) - \bar{h})^2] + \mathbb{E}[(y - f(\mathbf{x}))^2]. \quad \square\end{aligned}$$

Reading the result. Only the first two terms depend on the learning algorithm. What we can act on is the **excess risk** $\mathbb{E}[(y - h(\mathbf{x}))^2] - \mathbb{E}[(y - f(\mathbf{x}))^2] = \text{bias}^2 + \text{variance}$. The noise $\mathbb{E}[(y - f(\mathbf{x}))^2] = \text{Var}(\varepsilon)$ is a floor that no model can go below.

Where was the independence between the new y and the training sample used? What would go wrong if we evaluated h on a training point?

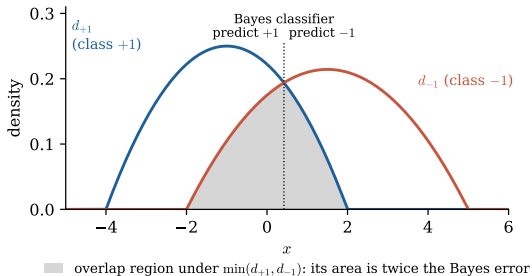
Bias and variance: a picture



- ▶ Center of the target: $f(\mathbf{x})$. Each dot: $h(\mathbf{x})$ learned on a different sample S .
- ▶ **Bias**: the cloud is off-center.
Variance: the cloud is spread.
- ▶ Simple models (linear, shallow trees): high bias, low variance.
- ▶ Flexible models (deep trees, 1-NN, large networks): low bias, high variance.

Ensemble methods mainly act on **one** of the two terms: bagging reduces the **variance**, boosting mainly the **bias**.

The Bayes error



- ▶ Classification: two classes with equal priors and class-conditional densities d_{+1} , d_{-1} .
- ▶ The **Bayes classifier** predicts, at each x , the class with the largest density.
- ▶ Even this optimal classifier errs in the region under both curves: no classifier can tell where such a point comes from.
- ▶ Bayes error: half of the shaded area, here $R^* \simeq 0.224$.

The Bayes error only depends on the distribution of the data.

Exercise – Bayes error

Two classes, $\mathbb{P}(Y = 1) = \mathbb{P}(Y = 2) = \frac{1}{2}$, class-conditional densities on $\mathbb{R}$:

$$d_1(x) = \begin{cases} \frac{3}{2}x^2 + x & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad d_2(x) = \begin{cases} 1 & \frac{3}{4} \leq x \leq \frac{7}{4} \\ 0 & \text{otherwise} \end{cases}$$

1. Show that, for any classifier h , $R(h) = \frac{1}{2} \int_{\{h=2\}} d_1 + \frac{1}{2} \int_{\{h=1\}} d_2$, and deduce that $R^* = \frac{1}{2} \int_{\mathbb{R}} \min(d_1, d_2)$.
2. Check that d_1 and d_2 are densities and give the Bayes classifier.
3. Compute the Bayes error.
4. What happens if $\mathbb{P}(Y = 1) = 0.9$?

Solution

1. $R(h) = \mathbb{P}(Y = 1)\mathbb{P}(h(X) = 2 \mid Y = 1) + \mathbb{P}(Y = 2)\mathbb{P}(h(X) = 1 \mid Y = 2)$. At each x the integrand is $\frac{1}{2}d_1(x)$ or $\frac{1}{2}d_2(x)$, hence $\geq \frac{1}{2} \min(d_1, d_2)$, with equality for $h^*(x) = 1$ iff $d_1(x) \geq d_2(x)$.
2. $\int_0^1 (\frac{3}{2}x^2 + x) dx = \frac{1}{2} + \frac{1}{2} = 1$. On $[\frac{3}{4}, 1]$, $d_1(x) \geq d_1(\frac{3}{4}) = \frac{51}{32} > 1 = d_2(x)$. Hence $h^*(x) = 1$ if $x \leq 1$, 2 if $x > 1$.
3. $\min(d_1, d_2) = 1$ on $[\frac{3}{4}, 1]$, 0 elsewhere: $R^* = \frac{1}{2} \times \frac{1}{4} = 0.125$.
4. $R^* = \int \min(\pi_1 d_1, \pi_2 d_2)$; on $[\frac{3}{4}, 1]$, $0.9 d_1 > 0.1$: same classifier, $R^* = 0.1 \times \frac{1}{4} = 0.025$.

Beyond the squared loss: the Bayes regret

General setting: $(\mathbf{x}, y) \sim \mathcal{D}$, loss ℓ , risk $R(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}}[\ell(h(\mathbf{x}), y)]$, empirical risk $\hat{R}_S(h) = \frac{1}{m} \sum_{i=1}^m \ell(h(\mathbf{x}_i), y_i)$, Bayes risk $R^* = \inf_h R(h)$.

Decomposition of the Bayes regret

For h in a hypothesis class $\mathcal{H}$:

$$R(h) - R^* = \underbrace{\left(R(h) - \inf_{g \in \mathcal{H}} R(g) \right)}_{\text{estimation error}} + \underbrace{\left(\inf_{g \in \mathcal{H}} R(g) - R^* \right)}_{\text{approximation error}}$$

- ▶ **Approximation error**: price of restricting ourselves to $\mathcal{H}$; does not depend on the data. A **bias** term.
- ▶ **Estimation error**: price of learning from a finite sample; grows with the “size” of $\mathcal{H}$. A **variance** term.

Let $h \in \arg \min_{g \in \mathcal{H}} \hat{R}_S(g)$ (empirical risk minimization) and $h_{\mathcal{H}} \in \arg \min_{g \in \mathcal{H}} R(g)$.

Proposition

$$R(h) - R(h_{\mathcal{H}}) \leq 2 \sup_{g \in \mathcal{H}} |R(g) - \hat{R}_S(g)|.$$

- ▶ $h_{\mathcal{H}}$ is fixed: the law of large numbers controls $\hat{R}_S(h_{\mathcal{H}}) - R(h_{\mathcal{H}})$.
- ▶ h depends on S : the $\ell(h(\mathbf{x}_i), y_i)$ are *not* independent draws of $\ell(h(\mathbf{x}), y)$. We need a control that is **uniform** over $\mathcal{H}$.

Your turn. Hint: add and subtract $\hat{R}_S(h)$ and $\hat{R}_S(h_{\mathcal{H}})$.

Proof of the proposition

Strategy. insert the empirical risks of h and $h_{\mathcal{H}}$ to create three differences, and bound each of them.

Step 1 – add and subtract. Adding and subtracting $\widehat{R}_S(h)$ and $\widehat{R}_S(h_{\mathcal{H}})$:

$$R(h) - R(h_{\mathcal{H}}) = \underbrace{(R(h) - \widehat{R}_S(h))}_{(a)} + \underbrace{(\widehat{R}_S(h) - \widehat{R}_S(h_{\mathcal{H}}))}_{(b)} + \underbrace{(\widehat{R}_S(h_{\mathcal{H}}) - R(h_{\mathcal{H}}))}_{(c)}.$$

Step 2 – term (b). h minimizes $\widehat{R}_S$ over $\mathcal{H}$ and $h_{\mathcal{H}} \in \mathcal{H}$, so $\widehat{R}_S(h) \leq \widehat{R}_S(h_{\mathcal{H}})$: $(b) \leq 0$.

Step 3 – terms (a) and (c). Each is the gap $\pm(R(g) - \widehat{R}_S(g))$ for some $g \in \mathcal{H}$ ($g = h$ or $g = h_{\mathcal{H}}$), hence

$$(a) \leq \sup_{g \in \mathcal{H}} |R(g) - \widehat{R}_S(g)|, \quad (c) \leq \sup_{g \in \mathcal{H}} |R(g) - \widehat{R}_S(g)|.$$

Step 4 – conclusion. Summing: $R(h) - R(h_{\mathcal{H}}) \leq 0 + 2 \sup_{g \in \mathcal{H}} |R(g) - \widehat{R}_S(g)|$. □

The estimation error is controlled by the **largest gap** between true and empirical risks over $\mathcal{H}$.
For an infinite $\mathcal{H}$, we need a measure of its size.

Definition – (Empirical) Rademacher complexity

Let $S = \{\mathbf{x}_i\}_{i=1}^m$ be a fixed sample and $\sigma_1, \dots, \sigma_m$ independent Rademacher variables (± 1 with probability $\frac{1}{2}$).

$$\mathfrak{R}_S(\mathcal{H}) = \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i h(\mathbf{x}_i) \right], \quad \mathfrak{R}_m(\mathcal{H}) = \mathbb{E}_{S \sim \mathcal{D}^m} [\mathfrak{R}_S(\mathcal{H})].$$

Intuition. Label each point by a coin toss and look for the $h \in \mathcal{H}$ that best agrees with these meaningless labels: $\mathfrak{R}_S(\mathcal{H})$ is the best correlation with noise reachable in $\mathcal{H}$, on average.

Examples (h with values in $\{\pm 1\}$):

- ▶ one function: $\mathfrak{R}_S = 0$;
- ▶ the two constants ± 1 : $\mathfrak{R}_S \leq 1/\sqrt{m}$;
- ▶ all functions: $\mathfrak{R}_S = 1$ (it fits any labeling of distinct points).

A class that can fit pure noise can fit the noise of the data: this is over-fitting. Below, $\mathfrak{R}_m(\mathcal{H})$ is the complexity of the class of losses $\{(\mathbf{x}, y) \mapsto \ell(g(\mathbf{x}), y)\}$.

Example: computing a Rademacher complexity

Class of the two constant classifiers $\mathcal{H}_{\pm} = \{\mathbf{x} \mapsto +1, \mathbf{x} \mapsto -1\}$, with $m = 2$ points. For a given σ , the best constant is the one with the sign of $\sigma_1 + \sigma_2$:

$$\sup_{h \in \mathcal{H}_{\pm}} \frac{1}{2} (\sigma_1 h(\mathbf{x}_1) + \sigma_2 h(\mathbf{x}_2)) = \frac{1}{2} |\sigma_1 + \sigma_2|.$$

(σ_1, σ_2)	prob.	best h	value
$(+1, +1)$	$\frac{1}{4}$	$+1$	1
$(+1, -1)$	$\frac{1}{4}$	any	0
$(-1, +1)$	$\frac{1}{4}$	any	0
$(-1, -1)$	$\frac{1}{4}$	-1	1

$$\text{Average: } \mathfrak{R}_S(\mathcal{H}_{\pm}) = \frac{1}{4}(1 + 0 + 0 + 1) = \frac{1}{2}.$$

Example – general m

$\mathfrak{R}_S(\mathcal{H}_{\pm}) = \frac{1}{m} \mathbb{E} \left| \sum_i \sigma_i \right|$. Jensen ($\mathbb{E}|Z| \leq \sqrt{\mathbb{E}Z^2}$) and independence ($\mathbb{E}[\sigma_i \sigma_j] = 0$ for $i \neq j$, $\sigma_i^2 = 1$) give $\mathbb{E} \left| \sum_i \sigma_i \right| \leq \sqrt{\mathbb{E}[(\sum_i \sigma_i)^2]} = \sqrt{m}$, so $\mathfrak{R}_S(\mathcal{H}_{\pm}) \leq 1/\sqrt{m} \rightarrow 0$.

Example – all functions

Take $h(\mathbf{x}_i) = \sigma_i$: the sum is $\frac{1}{m} \sum_i \sigma_i^2 = 1$ for every σ , so $\mathfrak{R}_S = 1$: the class fits *any* labeling.

Lemma (symmetrization)

$$\mathbb{E}_{S \sim \mathcal{D}^m} \left[\sup_{g \in \mathcal{H}} (R(g) - \hat{R}_S(g)) \right] \leq 2 \mathfrak{R}_m(\mathcal{H}).$$

Consequence: a bias–variance trade-off for any loss. Take the expectation of the decomposition $R(h) - R(h_{\mathcal{H}}) = (a) + (b) + (c)$ of the previous proof: $(b) \leq 0$, $\mathbb{E}_S[(c)] = 0$ because $h_{\mathcal{H}}$ is fixed (it does not depend on S , so $\hat{R}_S(h_{\mathcal{H}})$ is an unbiased estimate of $R(h_{\mathcal{H}})$), and $(a) \leq \sup_g (R(g) - \hat{R}_S(g))$. Hence

$$\mathbb{E}_S[R(h)] - R^* \leq \inf_{g \in \mathcal{H}} R(g) - R^* + 2 \mathfrak{R}_m(\mathcal{H}).$$

Enlarging $\mathcal{H}$ decreases the first term and increases the second one.

Your turn. Idea: introduce a second, “ghost” sample S' to write $R(g)$ as an expectation.

Proof of the symmetrization lemma (1/2)

Strategy. replace the unknown $R(g)$ by the empirical risk on a second, independent “ghost” sample S' ; the difference of two empirical risks can then be symmetrized with random signs.

Notation: $\mathbf{z}_i = (\mathbf{x}_i, y_i)$, $\ell_g(\mathbf{z}) = \ell(g(\mathbf{x}), y)$, and $S' = \{\mathbf{z}'_i\}_{i=1}^m \sim \mathcal{D}^m$ independent of S .

Step 1 – ghost sample. $\mathbb{E}_{S'} [\hat{R}_{S'}(g)] = \frac{1}{m} \sum_i \mathbb{E}[\ell_g(\mathbf{z}'_i)] = R(g)$, so

$$R(g) - \hat{R}_S(g) = \mathbb{E}_{S'} \left[\frac{1}{m} \sum_{i=1}^m (\ell_g(\mathbf{z}'_i) - \ell_g(\mathbf{z}_i)) \right] \quad (\hat{R}_S(g) \text{ does not depend on } S').$$

Step 2 – move $\mathbb{E}_{S'}$ outside the supremum. For every g , $X_g \leq \sup_{g'} X_{g'}$, hence $\mathbb{E}[X_g] \leq \mathbb{E}[\sup_{g'} X_{g'}]$ and $\sup_g \mathbb{E}[X_g] \leq \mathbb{E}[\sup_g X_g]$. Therefore

$$\mathbb{E}_S \left[\sup_g (R(g) - \hat{R}_S(g)) \right] \leq \mathbb{E}_{S,S'} \left[\sup_g \frac{1}{m} \sum_{i=1}^m (\ell_g(\mathbf{z}'_i) - \ell_g(\mathbf{z}_i)) \right].$$

Proof of the symmetrization lemma (2/2)

Step 3 – random signs. Fix $\sigma \in \{-1, +1\}^m$. Multiplying the i -th term by $\sigma_i = -1$ is the same as **exchanging** $\mathbf{z}_i$ and $\mathbf{z}'_i$. As $\mathbf{z}_i$ and $\mathbf{z}'_i$ are i.i.d., this exchange does not change the distribution of (S, S') , so the expectation is unchanged:

$$\mathbb{E}_{S, S'} \left[\sup_g \frac{1}{m} \sum_i (\ell_g(\mathbf{z}'_i) - \ell_g(\mathbf{z}_i)) \right] = \mathbb{E}_{S, S'} \left[\sup_g \frac{1}{m} \sum_i \sigma_i (\ell_g(\mathbf{z}'_i) - \ell_g(\mathbf{z}_i)) \right].$$

True for every σ , hence also on average over Rademacher signs σ .

Step 4 – split. $\sup(u + v) \leq \sup u + \sup v$:

$$\leq \mathbb{E}_{S', \sigma} \left[\sup_g \frac{1}{m} \sum_i \sigma_i \ell_g(\mathbf{z}'_i) \right] + \mathbb{E}_{S, \sigma} \left[\sup_g \frac{1}{m} \sum_i (-\sigma_i) \ell_g(\mathbf{z}_i) \right] = 2 \mathfrak{R}_m(\mathcal{H}),$$

because S' and S have the same distribution, and $-\sigma$ has the same distribution as σ . $\square$

Example of Step 3 ($m = 1$): $\ell_g(\mathbf{z}'_1) - \ell_g(\mathbf{z}_1)$ and $\ell_g(\mathbf{z}_1) - \ell_g(\mathbf{z}'_1)$ have the same distribution, because $(\mathbf{z}_1, \mathbf{z}'_1)$ and $(\mathbf{z}'_1, \mathbf{z}_1)$ have the same distribution.

Theorem (Rademacher generalization bound)

Assume $\ell(h(\mathbf{x}), y) \in [0, 1]$ for all $h \in \mathcal{H}$. For any $\delta \in (0, 1)$, with probability at least $1 - \delta$ over $S \sim \mathcal{D}^m$, for all $h \in \mathcal{H}$:

$$R(h) \leq \hat{R}_S(h) + 2\mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\ln(1/\delta)}{2m}}, \quad R(h) \leq \hat{R}_S(h) + 2\mathfrak{R}_S(\mathcal{H}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}.$$

Tool: McDiarmid's inequality

If $|\Phi(S) - \Phi(S')| \leq c$ whenever S, S' differ in a single point, then with probability at least $1 - \delta$: $\Phi(S) \leq \mathbb{E}[\Phi(S)] + c\sqrt{m \ln(1/\delta)/2}$.

The second bound only involves quantities computable on the sample.

Your turn. Apply McDiarmid to $\Phi(S) = \sup_h (R(h) - \hat{R}_S(h))$.

Proof of the Rademacher bound (1/2)

Strategy. the random variable $\Phi(S) = \sup_{h \in \mathcal{H}} (R(h) - \hat{R}_S(h))$ is concentrated around its mean (McDiarmid), and its mean is bounded by symmetrization.

Step 1 – bounded differences. Let S and S' differ only in the last point, $\mathbf{z}_m$ in S and $\mathbf{z}'_m$ in S' . A difference of suprema is at most the supremum of the differences, and $\hat{R}_{S'}(h) - \hat{R}_S(h)$ only involves the last point:

$$\Phi(S) - \Phi(S') \leq \sup_h (\hat{R}_{S'}(h) - \hat{R}_S(h)) = \sup_h \frac{\ell_h(\mathbf{z}'_m) - \ell_h(\mathbf{z}_m)}{m} \leq \frac{1}{m},$$

because the loss is in $[0, 1]$. Exchanging the roles of S and S' : $|\Phi(S) - \Phi(S')| \leq \frac{1}{m}$.

Step 2 – McDiarmid. With $c = \frac{1}{m}$, $c\sqrt{m \ln(1/\delta)/2} = \sqrt{\ln(1/\delta)/(2m)}$, so with probability at least $1 - \delta$:

$$\Phi(S) \leq \mathbb{E}_S[\Phi(S)] + \sqrt{\frac{\ln(1/\delta)}{2m}}.$$

Step 3 – the expectation. $\mathbb{E}_S[\Phi(S)] = \mathbb{E}_S[\sup_h (R(h) - \hat{R}_S(h))] \leq 2\mathfrak{R}_m(\mathcal{H})$: this is exactly the symmetrization lemma.

As $R(h) - \hat{R}_S(h) \leq \Phi(S)$ for every h , Steps 2 and 3 give the **first bound**.

Step 4 – second bound. Changing one point of S changes $\mathfrak{R}_S(\mathcal{H})$ by at most $\frac{1}{m}$, so McDiarmid gives, with probability $\geq 1 - \frac{\delta}{2}$: $\mathfrak{R}_m(\mathcal{H}) \leq \mathfrak{R}_S(\mathcal{H}) + \sqrt{\ln(2/\delta)/(2m)}$. The first bound with $\frac{\delta}{2}$ holds with probability $\geq 1 - \frac{\delta}{2}$. By a union bound, both hold with probability $\geq 1 - \delta$, and then

$$R(h) \leq \hat{R}_S(h) + 2\mathfrak{R}_S(\mathcal{H}) + 2\sqrt{\frac{\ln(2/\delta)}{2m}} + \sqrt{\frac{\ln(2/\delta)}{2m}}. \quad \square$$

What to remember – error decomposition

- ▶ Squared loss: error = $\text{bias}^2 + \text{variance} + \text{noise}$. The noise is a floor; the learner trades bias against variance.
- ▶ General loss: Bayes regret = estimation error (depends on the sample and on the size of $\mathcal{H}$) + $\text{approximation error}$ (depends on $\mathcal{H}$ and $\mathcal{D}$, not on the sample).
- ▶ The Rademacher complexity measures how well $\mathcal{H}$ fits pure noise; it controls the estimation error: $\mathbb{E}_S[R(h)] - R^* \leq \inf_{\mathcal{H}} R - R^* + 2\mathfrak{R}_m(\mathcal{H})$.

Next. Ensemble methods reduce one of the two terms *without* changing the base learner: bagging reduces the **variance** of complex models, boosting the **bias** of simple ones.

Part 2

Why combine models?

Question of this part. Why can an average or a vote of models be **better than each of them**?

In this part:

- ▶ averaging never increases the error, and can divide it by the number of models;
- ▶ the gain is exactly the **disagreement** (ambiguity) between the models;
- ▶ majority votes of independent voters (Condorcet);
- ▶ an overview of the three ways to build diverse models: bagging, boosting, stacking.

Main idea. An ensemble needs models that are both accurate and different.

Why use only one model?

- ▶ So far: *one* model per problem (a linear regression, a tree, an SVM...).
- ▶ Combinations of models are among the best-performing methods in practice (random forests, XGBoost, LightGBM, winning solutions of most tabular competitions).

A naive rule. Learn $h_1, \dots, h_T$ on the same sample S and average them:

$$H_T(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}) \quad (\text{regression}), \quad H_T(\mathbf{x}) = \text{sign} \left(\frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}) \right) \quad (\text{classification}).$$

With $h_t \in \{-1, +1\}$ this is a **majority vote**; with real outputs, a vote weighted by the confidences $|h_t(\mathbf{x})|$.

Learned on the same S (and with a convex problem), all the h_t are **identical**: nothing is gained. Varying the hyper-parameters gives poor models and conflicts with cross-validation.

Averaging reduces the error

Regression with a true function f : write $h_t(\mathbf{x}) = f(\mathbf{x}) + \varepsilon_t(\mathbf{x})$, so that the error of h_t is $\mathbb{E}_{\mathbf{x}}[\varepsilon_t(\mathbf{x})^2]$, and let $H_T = \frac{1}{T} \sum_t h_t$.

Proposition (averaging)

1. If the errors are centered and uncorrelated ($\mathbb{E}_{\mathbf{x}}[\varepsilon_t \varepsilon_s] = 0$ for $t \neq s$):

$$\mathbb{E}_{\mathbf{x}}[(H_T(\mathbf{x}) - f(\mathbf{x}))^2] = \frac{1}{T} \times \underbrace{\frac{1}{T} \sum_{t=1}^T \mathbb{E}_{\mathbf{x}}[\varepsilon_t(\mathbf{x})^2]}_{\text{average error of the } h_t}.$$

2. In general: $\mathbb{E}_{\mathbf{x}}[(H_T(\mathbf{x}) - f(\mathbf{x}))^2] \leq \frac{1}{T} \sum_{t=1}^T \mathbb{E}_{\mathbf{x}}[\varepsilon_t(\mathbf{x})^2]$.

Your turn. 1: expand the square. 2: which inequality involves a convex function?

Proof of the averaging proposition

Strategy. write the error of H_T with the errors ε_t of the models, then expand (1) or use convexity (2).

Step 1 – error of the average. $H_T(\mathbf{x}) - f(\mathbf{x}) = \frac{1}{T} \sum_t h_t(\mathbf{x}) - f(\mathbf{x}) = \frac{1}{T} \sum_t (h_t(\mathbf{x}) - f(\mathbf{x})) = \frac{1}{T} \sum_t \varepsilon_t(\mathbf{x})$.

Step 2 – expand the square (point 1). A square of a sum is a double sum: $(\sum_t a_t)^2 = \sum_t \sum_s a_t a_s$. Hence

$$\begin{aligned}\mathbb{E}_{\mathbf{x}}[(H_T - f)^2] &= \frac{1}{T^2} \sum_{t=1}^T \sum_{s=1}^T \mathbb{E}_{\mathbf{x}}[\varepsilon_t \varepsilon_s] \\ &= \frac{1}{T^2} \left(\underbrace{\sum_t \mathbb{E}_{\mathbf{x}}[\varepsilon_t^2]}_{t=s} + \underbrace{\sum_{t \neq s} \mathbb{E}_{\mathbf{x}}[\varepsilon_t \varepsilon_s]}_{=0 \text{ (uncorrelated)}} \right) = \frac{1}{T} \left(\frac{1}{T} \sum_t \mathbb{E}_{\mathbf{x}}[\varepsilon_t^2] \right).\end{aligned}$$

Step 3 – convexity (point 2). Jensen for the convex function $u \mapsto u^2$ and the uniform weights $\frac{1}{T}$: $(\frac{1}{T} \sum_t u_t)^2 \leq \frac{1}{T} \sum_t u_t^2$. With $u_t = \varepsilon_t(\mathbf{x})$, for every $\mathbf{x}$, $(H_T(\mathbf{x}) - f(\mathbf{x}))^2 \leq \frac{1}{T} \sum_t \varepsilon_t(\mathbf{x})^2$; take the expectation over $\mathbf{x}$. $\square$

Example: averaging two models

Two models whose errors have mean 0 and variance 1: $\mathbb{E}[\varepsilon_1^2] = \mathbb{E}[\varepsilon_2^2] = 1$. The error of the average is

$$\mathbb{E}\left[\left(\frac{\varepsilon_1 + \varepsilon_2}{2}\right)^2\right] = \frac{1}{4}\left(\mathbb{E}[\varepsilon_1^2] + 2\mathbb{E}[\varepsilon_1\varepsilon_2] + \mathbb{E}[\varepsilon_2^2]\right) = \frac{1 + \mathbb{E}[\varepsilon_1\varepsilon_2]}{2}.$$

situation	$\mathbb{E}[\varepsilon_1\varepsilon_2]$	error of the average	gain
identical models ($\varepsilon_2 = \varepsilon_1$)	1	1	none
uncorrelated errors	0	$\frac{1}{2}$	error divided by $T = 2$
opposite errors ($\varepsilon_2 = -\varepsilon_1$)	-1	0	errors cancel out

The whole game of ensemble methods: make the errors of the models as **little correlated** as possible, without making each model much worse.

Proposition (ambiguity decomposition, Krogh & Vedelsby 1995)

For every point $(\mathbf{x}, y)$ and $H_T = \frac{1}{T} \sum_t h_t$:

$$\underbrace{\frac{1}{T} \sum_{t=1}^T (h_t(\mathbf{x}) - y)^2}_{\text{average error of the models}} = \underbrace{(H_T(\mathbf{x}) - y)^2}_{\text{error of the ensemble}} + \underbrace{\frac{1}{T} \sum_{t=1}^T (h_t(\mathbf{x}) - H_T(\mathbf{x}))^2}_{\text{ambiguity (diversity)}}.$$

- ▶ The ensemble is better than its average member by exactly the **ambiguity**: the variance of the predictions at $\mathbf{x}$.
- ▶ Its error is small when the members are **accurate** and **disagree**.

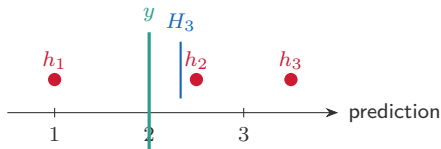
Your turn. Write $h_t - y = (h_t - H_T) + (H_T - y)$.

Write $h_t - y = (h_t - H_T) + (H_T - y)$ and expand:

$$\frac{1}{T} \sum_t (h_t - y)^2 = \frac{1}{T} \sum_t (h_t - H_T)^2 + \frac{2}{T} (H_T - y) \underbrace{\sum_t (h_t - H_T)}_{= T H_T - T H_T = 0} + (H_T - y)^2. \quad \square$$

Recipe for a good ensemble: accurate models (low bias) that make different predictions (high variance). Deep **decision trees** are exactly such models. We still need a way to make them **different** with a single sample S .

Example: ambiguity at one point



Example – three models, true value $y = 2$

Predictions $h_1 = 1$, $h_2 = 2.5$, $h_3 = 3.5$; average $H_3 = \frac{7}{3} \simeq 2.33$.

- ▶ Average error:
 $\frac{1}{3}(1^2 + 0.5^2 + 1.5^2) = \frac{3.5}{3} \simeq 1.167$.
- ▶ Ensemble error: $(2.33 - 2)^2 \simeq 0.111$.
- ▶ Ambiguity:
 $\frac{1}{3}(1.33^2 + 0.17^2 + 1.17^2) \simeq 1.056$.

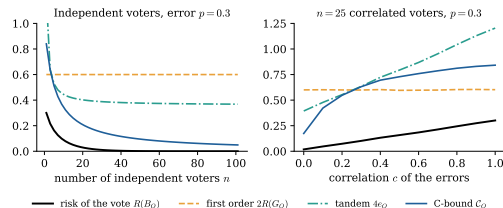
Check: $0.111 + 1.056 = 1.167$. The ensemble is 10 times better than its average member, because the models **disagree** on both sides of y .

Proposition (Condorcet, 1785)

Let $h_1, \dots, h_n$ be voters with uniform weights that err *independently*, each with probability $p < \frac{1}{2}$. The majority vote errs with probability at most

$$e^{-2n(\frac{1}{2}-p)^2} \xrightarrow{n \rightarrow \infty} 0.$$

Example: $n = 3$, $p = 0.3$: the vote errs with probability $3p^2(1-p) + p^3 = 0.216 < 0.3$.
With $n = 101$: less than $4 \cdot 10^{-4}$.



Voters of error $p = 0.3$. Left: independent errors, vote risk in black. Right: 25 voters whose errors are correlated (each copies a common leader with probability c).

Your turn. Hoeffding: $\mathbb{P}(\bar{W} - p \geq \varepsilon) \leq e^{-2n\varepsilon^2}$ for a mean $\bar{W}$ of n independent Bernoulli(p).

Proof of Condorcet's theorem

Strategy. the number of voters that err is a binomial variable; the vote errs when this number exceeds half of n , a “large deviation” controlled by Hoeffding's inequality.

Step 1 – fraction of errors. On a random example, let $W = \frac{1}{n} \sum_{t=1}^n \mathbb{1}[h_t(\mathbf{x}) \neq y]$. The indicators are independent Bernoulli variables of parameter p , so $nW \sim \mathcal{B}(n, p)$ and $\mathbb{E}[W] = p$.

Step 2 – when does the vote err?. The majority vote errs only if at least half of the voters err, i.e. $W \geq \frac{1}{2}$ (“if and only if” when n is odd).

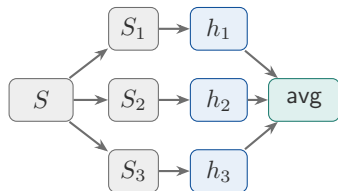
Step 3 – Hoeffding. For a mean of n independent variables in $[0, 1]$: $\mathbb{P}(W - \mathbb{E}W \geq \varepsilon) \leq e^{-2n\varepsilon^2}$. With $\varepsilon = \frac{1}{2} - p > 0$:

$$\mathbb{P}(\text{vote errs}) \leq \mathbb{P}(W - p \geq \tfrac{1}{2} - p) \leq e^{-2n(\frac{1}{2}-p)^2}. \quad \square$$

Voters learned on the same data are **not** independent: they err on the same hard examples. What matters is how often they **err together** (PAC-Bayes lecture).

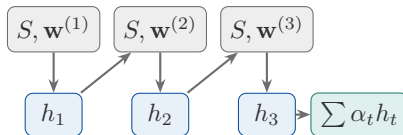
Three ways to build diverse models

Bagging



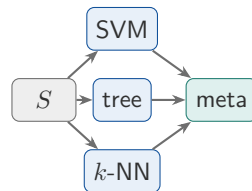
bootstrap samples,
models in **parallel**

Boosting



reweighted data,
models in **sequence**

Stacking



different algorithms,
learned combination

- ▶ **Bagging** combines **strong**, unstable learners to reduce their **variance**.
- ▶ **Boosting** combines **weak** learners to reduce their **bias**.
- ▶ **Stacking** learns how to combine **heterogeneous** models.

What to remember – why combine

- ▶ The error of an average of models is **never larger** than the average error of the models (Jensen).
- ▶ The gain is exactly the **ambiguity**: the more the models disagree, the larger the gain.
- ▶ Uncorrelated errors: the error is divided by T . Independent voters better than chance: the vote becomes almost perfect (Condorcet). Identical models: no gain at all.

The challenge: build **accurate** models that **disagree**, from a **single** training sample.

Part 3

Bagging and random forests

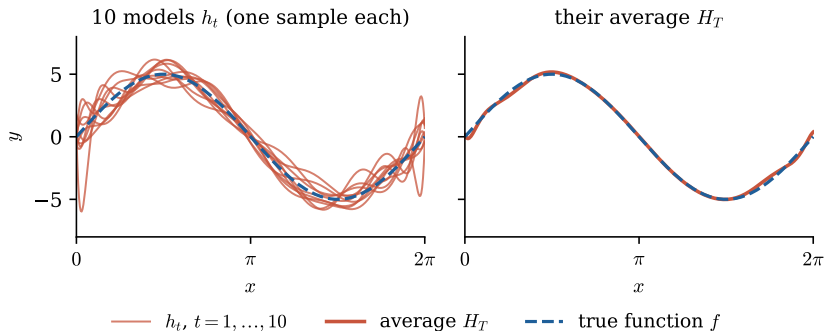
Question of this part. How do we obtain **different** models from a **single** training sample, to reduce the variance?

In this part we will:

- ▶ study the bootstrap and bagging, on a worked example with six points;
- ▶ see how random forests decorrelate the trees by sampling the features;
- ▶ compute out-of-bag errors and feature importances; two exercises at the end.

Main idea. Bagging averages unstable models (deep trees) learned on resampled data.

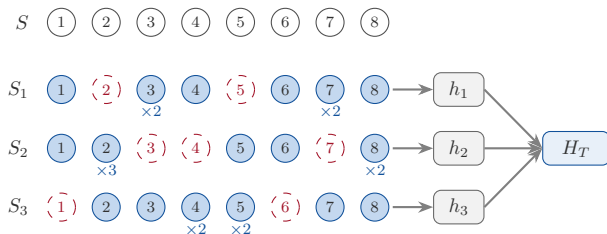
Combining models reduces the variance



Polynomials of degree 15 fitted to $f(x) = 5\sin(x) + \text{Gaussian noise}$. Left: 10 models learned on 10 datasets of size 30 drawn from the same distribution: for the same x , very different predictions (high variance). Right: their average varies much less around f , and remains accurate.

In practice we only have **one** training set S . How can we create several datasets S_t from it?

The bootstrap (Efron, 1979)



- ▶ A **bootstrap sample** S_t of size m : m draws from S , *uniformly and with replacement* (probability $1/m$ for each example at each draw).
- ▶ Some examples are drawn several times ($\times k$): the model pays more attention to them.
- ▶ Others are never drawn: they are **out-of-bag** (OOB) for h_t (dashed).
- ▶ Different $S_t \Rightarrow$ different h_t , focusing on different regions: **diversity**.

Algorithm 1: Bagging

Input : $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, number of models T

Output: H_T

```
1 for  $t = 1$  to  $T$  do
2   | draw a bootstrap sample  $S_t$  of size
   |    $m$  from  $S$ ;
3   | learn  $h_t$  on  $S_t$ ;
4  $H_T = \frac{1}{T} \sum_{t=1}^T h_t$  (or majority vote);
```

- ▶ **Bootstrap aggregating:** generate samples, then aggregate the predictions.
- ▶ The h_t are learned **independently**: trivially parallel.
- ▶ Any base learner can be bagged; it pays off for **unstable** learners: those that change a lot when the data change slightly.

Bagging is mainly a **variance reduction** technique.

Example: bagging regression stumps (1/2)

Six points $x = (1, 2, 3, 4, 5, 6)$, $y = (1, 2.5, 2, 4, 3, 5.5)$. Base learner: a **regression stump** (one threshold θ , each leaf predicts the mean of its targets, θ chosen to minimize the sum of squared errors, SSE). We draw $T = 3$ bootstrap samples (indices of the drawn examples):

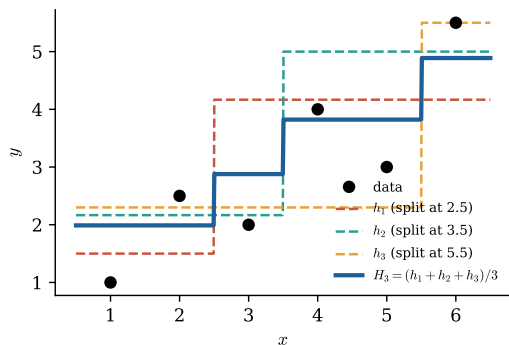
sample	drawn examples	OOB	best threshold	left leaf	right leaf
S_1	1, 1, 2, 4, 5, 6	3	$\theta_1 = 2.5^{(*)}$	$\frac{1+1+2.5}{3} = 1.5$	$\frac{4+3+5.5}{3} \simeq 4.17$
S_2	2, 3, 3, 4, 6, 6	1, 5	$\theta_2 = 3.5$	$\frac{2.5+2+2}{3} \simeq 2.17$	$\frac{4+5.5+5.5}{3} = 5$
S_3	1, 2, 3, 5, 5, 6	4	$\theta_3 = 5.5$	$\frac{1+2.5+2+3+3}{5} = 2.3$	5.5

- ▶ On the full sample, the best stump splits at 3.5. On the bootstrap samples, the best split moves (2.5, 3.5, 5.5): **the stump is unstable**, it depends on which points are drawn.

(*) S_1 has no point between 2 and 4: thresholds 2.5 and 3.5 give the same split; convention: we take the smallest one.

- ▶ Example: on S_1 , point 1 is drawn twice, so it counts twice in the left mean.
- ▶ Examples 1, 3, 4, 5 are out-of-bag for one stump (column OOB); examples 2 and 6 are out-of-bag for none.

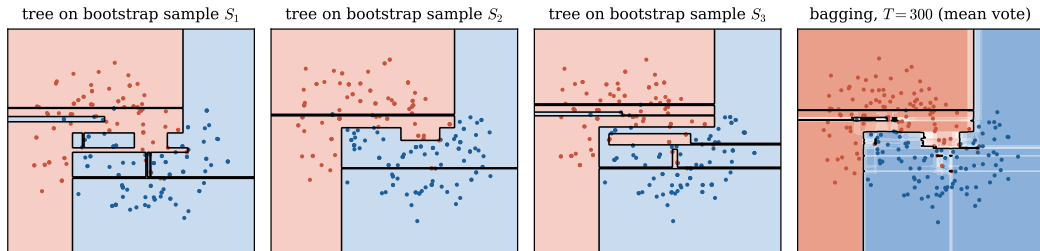
Example: bagging regression stumps (2/2)



x	1	2	3	4	5	6
h_1	1.5	1.5	4.17	4.17	4.17	4.17
h_2	2.17	2.17	2.17	5	5	5
h_3	2.3	2.3	2.3	2.3	2.3	5.5
H_3	1.99	1.99	2.88	3.82	3.82	4.89
y	1	2.5	2	4	3	5.5

- ▶ $H_3(x) = \frac{1}{3}(h_1(x) + h_2(x) + h_3(x))$,
e.g. $H_3(3) = \frac{4.17+2.17+2.3}{3} \simeq 2.88$.
- ▶ Each stump has 2 levels; their average has 4 levels: a **smoother** model, less sensitive to one particular sample.

Illustration: decision trees are unstable



Two moons ($m = 200$). Three fully grown trees learned on three bootstrap samples of the same data have very different decision regions: deep trees have a **low bias** (they fit the data) and a **high variance** (they fit *the* sample). The bagged ensemble (right, shading = proportion of votes) has a smoother and more stable boundary.

Why does bagging work?

Proposition (the bagged model is never worse than its average member)

For the squared loss and $H_T = \frac{1}{T} \sum_t h_t$:

$$\mathbb{E}_{(\mathbf{x}, y)} [(y - H_T(\mathbf{x}))^2] = \mathbb{E}_{(\mathbf{x}, y)} \left[\frac{1}{T} \sum_{t=1}^T (h_t(\mathbf{x}) - y)^2 \right] - \underbrace{\mathbb{E}_{\mathbf{x}} \left[\frac{1}{T} \sum_{t=1}^T (h_t(\mathbf{x}) - H_T(\mathbf{x}))^2 \right]}_{\text{ambiguity} \geq 0}.$$

The error of the bagged model is at most the average error of the bagged models.

- ▶ The gain is the **variance of the predictions** of the h_t : large when the base models are unstable.
- ▶ The average error of the h_t is small when the base models have a low bias.

Your turn. Jensen, or the ambiguity decomposition.

Proof of the bagging proposition

Strategy. this is the ambiguity decomposition, taken in expectation over the data.

Step 1 – expand the average error. At a fixed $(\mathbf{x}, y)$, write $a_t = h_t(\mathbf{x})$ and $\bar{a} = H_T(\mathbf{x}) = \frac{1}{T} \sum_t a_t$:

$$\frac{1}{T} \sum_t (a_t - y)^2 = y^2 - 2y\bar{a} + \frac{1}{T} \sum_t a_t^2.$$

Step 2 – empirical variance. The T numbers a_t have mean $\bar{a}$; König–Huygens for this finite set gives $\frac{1}{T} \sum_t a_t^2 = \bar{a}^2 + \frac{1}{T} \sum_t (a_t - \bar{a})^2$.

Step 3 – regroup. $y^2 - 2y\bar{a} + \bar{a}^2 = (y - \bar{a})^2$, hence

$$\frac{1}{T} \sum_t (h_t(\mathbf{x}) - y)^2 = (y - H_T(\mathbf{x}))^2 + \frac{1}{T} \sum_t (h_t(\mathbf{x}) - H_T(\mathbf{x}))^2.$$

Step 4 – expectation. Take $\mathbb{E}_{(\mathbf{x}, y)}$ on both sides; the ambiguity term is non-negative. $\square$

Example – on the six points

Training MSE of the three stumps: 1.52, 1.13, 0.87 (average 1.17); training MSE of H_3 : 0.52; average ambiguity: 0.65. Check: $1.17 = 0.52 + 0.65$.

Which learners to bag? A *stable*, high-bias learner (linear model, k -NN with large k) gives almost identical bootstrap models: no ambiguity, no gain. Bag **deep trees**.

Algorithm 2: Random forest

Input : S , number of trees T , sample size $m' \leq m$, number of features p'

Output: H_T

- 1 **for** $t = 1$ **to** T **do**
 - 2 draw a bootstrap sample S_t of size m' ;
 - 3 grow a deep tree h_t on S_t ; at each node, only p' **randomly drawn features** are candidates for the split;
 - 4 $H_T = \frac{1}{T} \sum_t h_t$ (regression) or majority vote (classification);
-

- ▶ Bagging of deep trees + **double sampling**: examples *and* features.
- ▶ Usual choices: $p' \simeq \sqrt{d}$ (classification), $p' \simeq d/3$ (regression; scikit-learn uses all the features by default).
- ▶ Both samplings also make each tree faster to build.

The bootstrap already makes the trees different. Why also sample the features?

Example: sampling the features at a node

At the root of a tree, $d = 4$ features; the decrease of impurity of the best split on each feature is:

feature	X^1	X^2	X^3	X^4
best impurity decrease	0.20	0.15	0.12	0.03

Bagging ($p' = d = 4$): every tree compares the 4 features and splits on X^1 . All the trees start in the same way: they are **strongly correlated**.

Random forest ($p' = 2$): 2 features drawn among 4, i.e. $\binom{4}{2} = 6$ equally likely pairs:

pair	split on	pair	split on
$\{1, 2\}$	X^1	$\{2, 3\}$	X^2
$\{1, 3\}$	X^1	$\{2, 4\}$	X^2
$\{1, 4\}$	X^1	$\{3, 4\}$	X^3

Root split on X^1 w.p. $\frac{1}{2}$, X^2 w.p. $\frac{1}{3}$, X^3 w.p. $\frac{1}{6}$.

The trees of a forest start differently: they are **less correlated**, at the price of sometimes using a less informative split (slightly worse trees).

Why sample the features? Correlated trees

In Part 2, uncorrelated errors divided the error by T . For bagged trees this is far too optimistic: if one feature is very informative, every bagged tree splits on it first, and all the trees look alike.

Proposition (variance of an average of correlated models)

Let $Z_1, \dots, Z_T$ (the predictions of the trees at $\mathbf{x}$) have the same variance σ^2 and the same pairwise correlation ρ . Then

$$\text{Var} \left(\frac{1}{T} \sum_{t=1}^T Z_t \right) = \rho \sigma^2 + \frac{1-\rho}{T} \sigma^2.$$

- ▶ Adding trees ($T \rightarrow \infty$) only removes the second term: $\rho \sigma^2$ **remains**.
- ▶ To go further we must **decorrelate** the trees, i.e. decrease ρ .

Your turn. Bilinearity of the covariance.

Proof of the variance formula

Strategy. the variance of a sum is the sum of all the entries of the covariance matrix.

Step 1 – bilinearity. The covariance is bilinear and $\text{Var}(X) = \text{Cov}(X, X)$:

$$\text{Var}\left(\sum_t Z_t\right) = \text{Cov}\left(\sum_t Z_t, \sum_s Z_s\right) = \sum_t \sum_s \text{Cov}(Z_t, Z_s).$$

Step 2 – count the entries. The covariance matrix has T diagonal entries $\text{Var}(Z_t) = \sigma^2$ and $T(T-1)$ off-diagonal entries $\text{Cov}(Z_t, Z_s) = \rho\sigma^2$ (definition of the correlation). So

$$\text{Var}\left(\sum_t Z_t\right) = T\sigma^2 + T(T-1)\rho\sigma^2.$$

Step 3 – divide by T^2 . $\text{Var}\left(\frac{1}{T}X\right) = \frac{1}{T^2} \text{Var}(X)$:

$$\text{Var}\left(\frac{1}{T} \sum_t Z_t\right) = \frac{\sigma^2}{T} + \frac{T-1}{T} \rho\sigma^2 = \rho\sigma^2 + \frac{1-\rho}{T} \sigma^2. \quad \square$$

$$\begin{array}{l} T = 3: \\ \begin{pmatrix} \sigma^2 & \rho\sigma^2 & \rho\sigma^2 \\ \rho\sigma^2 & \sigma^2 & \rho\sigma^2 \\ \rho\sigma^2 & \rho\sigma^2 & \sigma^2 \end{pmatrix} \\ \text{sum} = 3\sigma^2 + 6\rho\sigma^2 \end{array}$$

Example: bagged trees versus random forest

Example – numerical illustration

Bagged trees, $\sigma^2 = 1$, $\rho = 0.3$: variance $0.3 + 0.7/10 = 0.37$ for $T = 10$, 0.30 for $T = \infty$.
Random forest: $\rho = 0.1$ but worse trees, $\sigma^2 = 1.2$: $0.12 + 0.108 = 0.228$ ($T = 10$), 0.12 ($T = \infty$).

	σ^2	ρ	variance, $T = 10$	variance, $T = \infty$
single tree	1	–	1	1
bagged trees	1	0.3	0.37	0.30
random forest	1.2	0.1	0.228	0.12

Random forests accept slightly worse trees ($\sigma^2 \nearrow$) to get much less correlated ones ($\rho \searrow$): the floor $\rho\sigma^2$ is divided by 2.5.

Proposition

The probability that a given example is *not* in a bootstrap sample of size m is

$$\left(1 - \frac{1}{m}\right)^m \xrightarrow{m \rightarrow \infty} e^{-1} \simeq 0.368.$$

- ▶ $m = 10$: 0.349; $m = 100$: 0.366; $m = 1000$: 0.368.
- ▶ Each tree uses only $\sim 63\%$ of the distinct examples; each example is OOB for $\sim 37\%$ of the trees (about 37 trees out of 100).

Your turn. Each draw misses the example with probability $1 - 1/m$.

Step 1 – one draw. A draw picks a given example with probability $\frac{1}{m}$, so misses it with probability $1 - \frac{1}{m}$.

Step 2 – m independent draws. The example is never drawn with probability $(1 - \frac{1}{m})^m$.

Step 3 – limit. $(1 - \frac{1}{m})^m = \exp(m \ln(1 - \frac{1}{m}))$ and $\ln(1 - u) = -u + O(u^2)$, so $m \ln(1 - \frac{1}{m}) = -1 + O(\frac{1}{m}) \rightarrow -1$. $\square$

Definition – Out-of-bag (OOB) error

For each training example $\mathbf{x}_i$, average (or vote) the predictions of the models for which $\mathbf{x}_i$ is OOB only, and compare with y_i . The OOB error is the average of these errors over the examples that have OOB predictions.

- ▶ Each OOB prediction is made by models that have **not seen** the example: an almost free estimate of the generalization error, with no validation set. Useful to tune p' .
- ▶ Requires i.i.d. examples (beware of repeated measurements and time series).

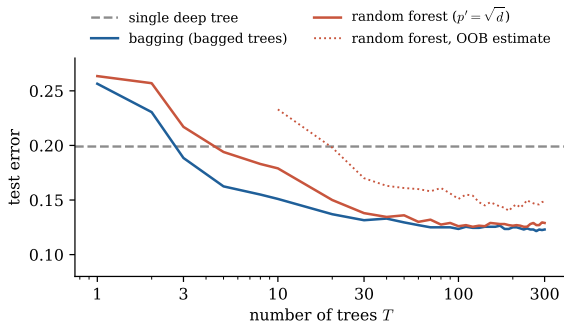
Example: the OOB error of the six-point ensemble

Back to the three bagged stumps. OOB examples: 3 for h_1 ; 1, 5 for h_2 ; 4 for h_3 .

example	y_i	OOB for	OOB prediction	squared error	prediction of H_3
1	1	h_2	2.17	1.36	1.99
2	2.5	–	none	–	1.99
3	2	h_1	4.17	4.69	2.88
4	4	h_3	2.30	2.89	3.82
5	3	h_2	5.00	4.00	3.82
6	5.5	–	none	–	4.89

- ▶ OOB error = $\frac{1}{4}(1.36 + 4.69 + 2.89 + 4.00) \simeq 3.24$, whereas the training MSE of H_3 is 0.52: the training error is **optimistic**, because H_3 has seen the points.
- ▶ With $T = 3$, examples 2 and 6 have no OOB prediction, and each OOB prediction uses one stump only. With hundreds of trees, every example has OOB predictions made by $\sim 37\%$ of the trees.

Bagging and forests in practice



- ▶ Simulated problem, $d = 20$, $m = 1000$. A single deep tree over-fits.
- ▶ Bagging and forests improve quickly with T , then reach a plateau.
- ▶ **Adding trees does not over-fit:** T only reduces the Monte Carlo variability. Take T large enough, no need to tune it.
- ▶ The OOB estimate (dotted) follows the test error, slightly pessimistic: each OOB prediction only uses $\sim \frac{1}{3}$ of the trees.

Measuring the importance of the features

Mean decrease in impurity (MDI). When a node N is split on X^j , its impurity decreases by

$$\Delta \text{Ent}_N(X^j) = \text{Ent}(\mathbf{p}_N) - \frac{|N_L|}{|N|} \text{Ent}(\mathbf{p}_{N_L}) - \frac{|N_R|}{|N|} \text{Ent}(\mathbf{p}_{N_R}), \quad \text{Ent}(\mathbf{p}) = - \sum_k p_k \log_2 p_k.$$

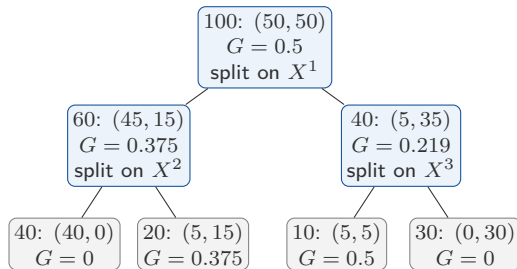
The importance of X^j sums these decreases over the nodes split on X^j , weighted by the fraction of examples reaching the node, and averages over the trees:

$$\text{Imp}^j = \frac{1}{T} \sum_{t=1}^T \sum_{N \in \mathcal{N}_t} \frac{|N|}{m_t} \Delta \text{Ent}_N(X^j) \mathbb{1}[X^j \text{ splits } N].$$

Permutation importance. Increase of the (OOB or test) error when the values of X^j are randomly permuted, which breaks the link between X^j and y .

MDI is free (computed during training) but measured on the training data; the Gini index can replace the entropy (default of `scikit-learn`).

Example: computing the MDI of a tree



node size: (class +, class -), Gini $G = 1 - p_+^2 - p_-^2$

Weighted decrease

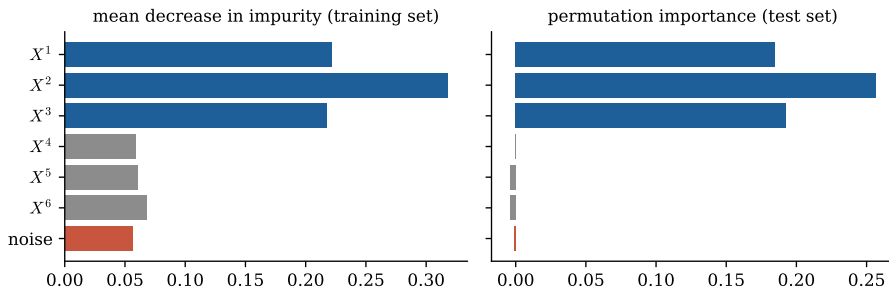
$$\frac{|N|}{m} \left[G_N - \frac{|N_L|}{|N|} G_{N_L} - \frac{|N_R|}{|N|} G_{N_R} \right] \text{ for each split:}$$

- ▶ root (X^1):
 $\frac{100}{100} [0.5 - 0.6 \times 0.375 - 0.4 \times 0.219] = 0.1875;$
- ▶ left (X^2):
 $\frac{60}{100} [0.375 - \frac{40}{60} \times 0 - \frac{20}{60} \times 0.375] = 0.15;$
- ▶ right (X^3):
 $\frac{40}{100} [0.219 - \frac{10}{40} \times 0.5 - \frac{30}{40} \times 0] = 0.0375.$

Example – result

Normalized importances: X^1 : 0.50, X^2 : 0.40, X^3 : 0.10, X^4 : 0 (never used). In a forest, average these numbers over the trees.

Feature importance: an illustration



Random forest on a simulated problem: only X^1, X^2, X^3 are informative (blue); X^4, X^5, X^6 (grey) and the last column (red) are **pure noise**. MDI gives each useless feature an importance of about 0.06, because deep trees also split on them to fit the training data. The permutation importance, computed on test data, is close to 0 for all of them: it correctly detects that they are useless.

Correlated features share their importance (with both methods): an importance is not a causal effect.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance

rf = RandomForestClassifier(
    n_estimators=500, # T: large enough
    max_features="sqrt", # p' (decorrelation)
    min_samples_leaf=1, # deep trees
    oob_score=True, # free validation
    n_jobs=-1, random_state=0)
rf.fit(X_train, y_train)
print("OOB error :", 1 - rf.oob_score_)
print("test error:", 1 - rf.score(X_test, y_test))
mdi = rf.feature_importances_
perm = permutation_importance(rf, X_test, y_test)
```

Main hyper-parameters

- ▶ $n_estimators$ (T): not critical, larger is better (and slower).
- ▶ $max_features$ (p'): the most influential one; tune it with the OOB error.
- ▶ $min_samples_leaf$, max_depth : control the depth of the trees.
- ▶ $max_samples$ (m'): size of the bootstrap samples.

Random forests work well *out of the box*: a strong baseline on tabular data.

What to remember – bagging and random forests

- ▶ Bagging: learn the same unstable learner (deep trees) on bootstrap samples and average. It reduces the **variance**; the gain is the ambiguity of the models.
- ▶ Random forests: also draw p' features at each node, to **decorrelate** the trees: the variance of the average is $\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$.
- ▶ About 37% of the examples are out-of-bag for each tree: the **OOB error** is a free validation.
- ▶ Adding trees never over-fits; the main hyper-parameter is p' . Feature importances: MDI (free, biased) and permutation (on test or OOB data).

Exercise 1: bootstrap, vote and OOB error

Training set of $m = 5$ examples. Three trees are learned on the bootstrap samples (indices of the drawn examples)

$$S_1 = (1, 1, 3, 4, 4), \quad S_2 = (2, 3, 3, 5, 5),$$

$$S_3 = (1, 2, 4, 5, 5),$$

and predict on the training examples:

	x_1	x_2	x_3	x_4	x_5
y	+	+	−	−	+
h_1	+	+	−	−	−
h_2	+	−	−	+	+
h_3	+	+	+	−	+

Exercise – quick computations

1. For $m = 5$, compute the probability that a given example is out-of-bag for one bootstrap sample, and the expected number of distinct examples in a bootstrap sample.
2. Give the OOB examples of each tree.
3. Compute the majority vote H_3 and its training error. Compare with the training errors of the trees.
4. Compute the OOB error of the ensemble. Comment.

Solution

1. $\mathbb{P}(\text{OOB}) = (1 - \frac{1}{5})^5 = 0.8^5 \simeq 0.328$ (already close to $e^{-1} \simeq 0.368$). Distinct examples:
 $m(1 - (1 - \frac{1}{m})^m) = 5 \times 0.672 \simeq 3.36$ on average.
2. S_1 misses $\{2, 5\}$, S_2 misses $\{1, 4\}$, S_3 misses $\{3\}$.
3. Votes: x_1 : +, +, +; x_2 : +, -, +; x_3 : -, -, +; x_4 : -, +, -; x_5 : -, +, +.
 $H_3 = (+, +, -, -, +) = y$: training error 0, whereas the trees have errors $\frac{1}{5}, \frac{2}{5}, \frac{1}{5}$.
4. Each example is predicted by the trees that did not see it:

	x_1	x_2	x_3	x_4	x_5
OOB tree(s)	h_2	h_1	h_3	h_2	h_1
OOB prediction	+	+	+	+	-
correct?	yes	yes	no	no	no

OOB error = $\frac{3}{5} = 0.6$, versus a training error of 0: the training error of the vote is **optimistic** (each point is predicted by trees that have seen it). With only 3 trees each OOB prediction uses a single tree; with hundreds of trees the OOB error becomes a reliable estimate.

Exercise 2: how many trees, and why decorrelate?

Exercise – quick computations

The predictions of bagged trees at a point $\mathbf{x}$ have variance $\sigma^2 = 1$ and pairwise correlation $\rho = 0.5$. Recall $\text{Var} \left(\frac{1}{T} \sum_t Z_t \right) = \rho \sigma^2 + \frac{1-\rho}{T} \sigma^2$.

1. Compute the variance of the bagged prediction for $T = 5$ trees and for $T \rightarrow \infty$.
2. What is the smallest T that guarantees a variance at most 0.55? Can we reach 0.45 with more trees?
3. A random forest decreases the correlation to $\rho = 0.2$, but each tree is worse: $\sigma^2 = 1.3$. Compute the variance for $T = 5$ and $T \rightarrow \infty$.
4. A single tree of the forest has a larger variance than a single bagged tree. Is the forest worse? Conclude.

Your turn. Each question is a one-line computation.

Solution

1. $T = 5$: $0.5 + \frac{0.5}{5} = 0.6$. $T \rightarrow \infty$: 0.5.
2. $0.5 + \frac{0.5}{T} \leq 0.55 \iff \frac{0.5}{T} \leq 0.05 \iff T \geq 10$. The variance never goes below $\rho\sigma^2 = 0.5$: 0.45 is out of reach, whatever the number of trees.
3. $T = 5$: $0.2 \times 1.3 + \frac{0.8 \times 1.3}{5} = 0.26 + 0.208 = 0.468$. $T \rightarrow \infty$: $\rho\sigma^2 = 0.26$.
4. No: one tree of the forest is worse ($1.3 > 1$), but already with 5 trees the forest beats bagging ($0.468 < 0.6$), and its floor is about half as large (0.26 vs 0.5).

With correlated models, **adding models** only removes $\frac{1-\rho}{T}\sigma^2$; **decorrelating** the models lowers the floor $\rho\sigma^2$. Random forests do both.

Part 4

Boosting

Question of this part. Can we combine **weak** models, barely better than chance, into a strong one, i.e. reduce the bias?

In this part:

- ▶ AdaBoost, run round by round on six points;
- ▶ why its training error decreases exponentially fast (proof);
- ▶ AdaBoost as the minimization of the exponential loss; margins and noise; two exercises at the end.

Main idea. Boosting builds the models one after the other, each one correcting the previous ones.

Definition – Strong PAC-learnability

A concept class $\mathcal{C}$ is PAC-learnable if there exist an algorithm $\mathcal{A}$ and a polynomial $poly$ such that for all $\varepsilon, \delta > 0$, all distributions $\mathcal{D}$ and all targets $c \in \mathcal{C}$, for $m \geq poly(1/\varepsilon, 1/\delta, d, size(c))$ (d : dimension of $\mathcal{X}$, $size(c)$: cost of representing c): $\mathbb{P}_{S \sim \mathcal{D}^m} [R(h_S) \leq \varepsilon] \geq 1 - \delta$.

Definition – Weak PAC-learnability

Same definition, but only for a fixed **edge** $\gamma > 0$: $\mathbb{P}_{S \sim \mathcal{D}^m} [R(h_S) \leq \frac{1}{2} - \gamma] \geq 1 - \delta$.
 $\mathcal{A}$ is a **weak learner**: slightly better than random guessing.

Strong learners: deep decision trees, over-parameterized neural networks.

Weak learners: decision stumps (depth 1), shallow trees, a linear classifier on non-separable data.

The idea of boosting

- ▶ **Question** (Kearns & Valiant, 1988–89): are weak and strong learnability equivalent? Can a learner slightly better than chance be “boosted” into an arbitrarily accurate one?
- ▶ **Answer:** yes (Schapire, 1990). Practical algorithm: **AdaBoost** (Freund & Schapire, 1995–97), Gödel Prize 2003.
- ▶ **Principle:** learn the weak learners *sequentially*; each one focuses on the examples the previous ones got wrong, by **reweighting** the data.
- ▶ Then: boosting = greedy minimization of a loss (Friedman et al., 2000) $\Rightarrow$ **gradient boosting**, XGBoost, LightGBM, CatBoost.

Bagging: strong learners, built in parallel on resampled data, to reduce the variance.

Boosting: weak learners, built in sequence on reweighted data, to reduce the **bias**.

How can a combination of stumps, which change their prediction only once along one feature, fit a complex boundary?

Algorithm 3: AdaBoost

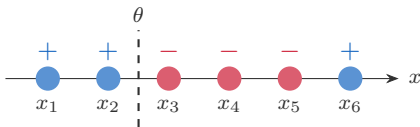
Input : $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, $y_i \in \{-1, +1\}$,
 rounds T

Output : $H_T = \text{sign}(f_T)$, $f_T = \sum_{t=1}^T \alpha_t h_t$

```
1  $w_i^{(1)} = 1/m$  for all  $i$ ;  
2 for  $t = 1$  to  $T$  do  
3   learn  $h_t$  on  $S$  weighted by  $\mathbf{w}^{(t)}$ ;  
4    $\varepsilon_t = \sum_i w_i^{(t)} \mathbb{1}[y_i h_t(\mathbf{x}_i) < 0]$  ;           //  $< 1/2$   
5    $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$ ;  
6    $Z_t = \sum_i w_i^{(t)} e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$ ;  
7    $w_i^{(t+1)} = w_i^{(t)} e^{-\alpha_t y_i h_t(\mathbf{x}_i)} / Z_t$ ;
```

- ▶ ε_t : **weighted** error of h_t .
- ▶ α_t : weight of h_t in the vote; the smaller ε_t , the larger α_t , and $\alpha_t > 0$ iff $\varepsilon_t < \frac{1}{2}$.
- ▶ Misclassified examples: weight $\times e^{\alpha_t} / Z_t$ (**increases**). Well classified: $\times e^{-\alpha_t} / Z_t$ (**decreases**).
- ▶ Z_t normalizes the weights to sum to 1.

Example: AdaBoost on six points, the setting



a stump: one threshold θ , one sign on each side

- ▶ $m = 6$ points $x_i = i$, labels $y = (+1, +1, -1, -1, -1, +1)$.
- ▶ Weak learners: **decision stumps**
 $h_{\theta,s}(x) = s \operatorname{sign}(x - \theta)$, with $\theta \in \{0.5, 1.5, \dots, 6.5\}$ and $s = \pm 1$ ($\theta = 0.5$ or 6.5 : constant classifier).
- ▶ At each round, the weak learner returns the stump with the smallest **weighted** error
 $\varepsilon_t = \sum_i w_i^{(t)} \mathbb{1}[h(x_i) \neq y_i]$.

No stump has a zero error: a stump changes sign at most once, the labels change sign twice.

Useful formula. With $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$ and $Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$, the update multiplies the weight of a misclassified example by $\frac{e^{\alpha_t}}{Z_t} = \frac{1}{2\varepsilon_t}$ and of a well classified one by $\frac{e^{-\alpha_t}}{Z_t} = \frac{1}{2(1-\varepsilon_t)}$. (Check it!)

Example: AdaBoost, round 1

Weights $\mathbf{w}^{(1)} = (\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$. Some candidate stumps:

stump	predictions on $x_1, \dots, x_6$	misclassified	weighted error
+ if $x < 2.5$	+ + - - - -	x_6	$1/6 \simeq 0.167$
+ if $x < 1.5$	+ - - - - -	x_2, x_6	$2/6$
+ if $x > 5.5$	- - - - - +	x_1, x_2	$2/6$
constant +1	+ + + + + +	x_3, x_4, x_5	$3/6$

All the other stumps have a weighted error $\geq 2/6$.

► h_1 : "+1 if $x < 2.5$ ", $\varepsilon_1 = \frac{1}{6}$.

► $\alpha_1 = \frac{1}{2} \ln \frac{5/6}{1/6} = \frac{1}{2} \ln 5 \simeq 0.805$.

► $Z_1 = 2\sqrt{\frac{1}{6} \cdot \frac{5}{6}} = \frac{\sqrt{5}}{3} \simeq 0.745$.

► x_6 (error): $\frac{1}{6} \times \frac{1}{2\varepsilon_1} = \frac{1}{6} \times 3 = 0.5$.

► others: $\frac{1}{6} \times \frac{1}{2(1-\varepsilon_1)} = \frac{1}{6} \times \frac{3}{5} = 0.1$.

► $\mathbf{w}^{(2)} = (0.1, 0.1, 0.1, 0.1, 0.1, \mathbf{0.5})$.

After round 1, the misclassified point x_6 carries half of the total weight.

Example: AdaBoost, round 2

Weights $\mathbf{w}^{(2)} = (0.1, 0.1, 0.1, 0.1, 0.1, 0.5)$. Some candidate stumps:

stump	predictions	misclassified	weighted error
+ if $x < 2.5$ ($= h_1$)	+ + - - - -	x_6	0.5
+ if $x > 5.5$	- - - - - +	x_1, x_2	0.1 + 0.1 = 0.2
+ if $x > 4.5$	- - - - + +	x_1, x_2, x_5	0.3
constant +1	+ + + + + +	x_3, x_4, x_5	0.3

All the other stumps have a weighted error ≥ 0.3 .

► h_1 is now no better than chance (0.5).

► h_2 : “+1 if $x > 5.5$ ”, $\varepsilon_2 = 0.2$.

► $\alpha_2 = \frac{1}{2} \ln \frac{0.8}{0.2} = \ln 2 \simeq 0.693$,
 $Z_2 = 2\sqrt{0.16} = 0.8$.

► x_1, x_2 : $\times \frac{1}{0.4} = 2.5 \Rightarrow 0.25$.

► others: $\times \frac{1}{1.6} = 0.625 \Rightarrow 0.0625$
(x_3, x_4, x_5), 0.3125 (x_6).

► $\mathbf{w}^{(3)} =$
(**0.25, 0.25**, 0.0625, 0.0625, 0.0625, 0.3125).

Example: AdaBoost, round 3

Weights $\mathbf{w}^{(3)} = (0.25, 0.25, 0.0625, 0.0625, 0.0625, 0.3125)$. Some candidate stumps:

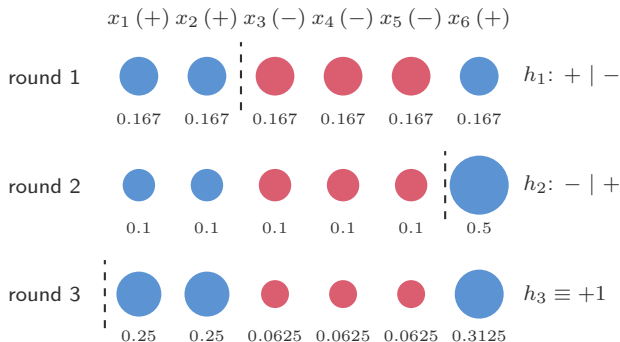
stump	predictions	misclassified	weighted error
constant +1	+ + + + +	x_3, x_4, x_5	$3 \times 0.0625 = 0.1875$
+ if $x < 2.5$ ($= h_1$)	+ + - - -	x_6	0.3125
+ if $x < 3.5$	+ + + - -	x_3, x_6	0.375
+ if $x > 5.5$ ($= h_2$)	- - - - +	x_1, x_2	0.5

All the other stumps have a weighted error ≥ 0.3125 .

- ▶ $h_3 \equiv +1$: it errs on x_3, x_4, x_5 , which have become **light** because h_1 and h_2 classify them well.
- ▶ $\varepsilon_3 = \frac{3}{16}$, $\alpha_3 = \frac{1}{2} \ln \frac{13/16}{3/16} = \frac{1}{2} \ln \frac{13}{3} \simeq 0.733$, $Z_3 = 2\sqrt{\frac{3}{16} \cdot \frac{13}{16}} \simeq 0.781$.

Each new weight vector makes the previous stump no better than chance: h_1 has error 0.5 under $\mathbf{w}^{(2)}$, h_2 under $\mathbf{w}^{(3)}$ (this is always true, see “Where does α_t come from?”).

Example: AdaBoost on six points, summary of the three rounds



Labels $+$ $+$ $-$ $-$ $-$ $+$; weak learner:
the stump with the smallest weighted
error (disc area $\propto w_i^{(t)}$).

- ▶ **Round 1:** h_1 errs on x_6 only:
 $\varepsilon_1 = \frac{1}{6}$, $\alpha_1 = \frac{1}{2} \ln 5 \simeq 0.80$. The
weight of x_6 is multiplied by 3.
- ▶ **Round 2:** h_2 errs on x_1, x_2 :
 $\varepsilon_2 = 0.2$, $\alpha_2 = \ln 2 \simeq 0.69$.
- ▶ **Round 3:** the constant $+1$ errs
on x_3, x_4, x_5 , now light:
 $\varepsilon_3 = \frac{3}{16}$, $\alpha_3 \simeq 0.73$.

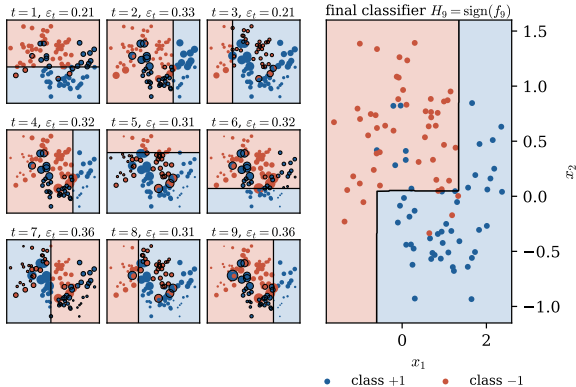
AdaBoost step by step: the final vote

	x_1	x_2	x_3	x_4	x_5	x_6
y	+	+	−	−	−	+
h_1 ($\alpha_1 = 0.80$)	+	+	−	−	−	−
h_2 ($\alpha_2 = 0.69$)	−	−	−	−	−	+
h_3 ($\alpha_3 = 0.73$)	+	+	+	+	+	+
$f_3 = \sum_t \alpha_t h_t$	0.84	0.84	−0.76	−0.76	−0.76	0.62
$H_3 = \text{sign}(f_3)$	+	+	−	−	−	+

- ▶ Each stump errs at least once (a stump changes sign at most once, the labels twice)...
- ▶ ... but the weighted vote of 3 stumps has **zero training error**.
- ▶ Each weak learner corrects the mistakes of the previous ones: boosting **reduces the bias**.

Is this a coincidence? How fast does the training error decrease in general? (Here $\prod_t Z_t \simeq 0.745 \times 0.8 \times 0.781 \simeq 0.465$: see the next theorem.)

AdaBoost at work



- ▶ Decision stumps on $m = 100$ points.
- ▶ Left: stump h_t of round t (background) and its weighted error; the area of a point is proportional to its weight $w_i^{(t)}$; circled points are misclassified by h_t and gain weight.
- ▶ The weak learners progressively concentrate on the hard examples, close to the boundary.
- ▶ Right: $H_9 = \text{sign}(\sum_{t \leq 9} \alpha_t h_t)$ is no longer a stump (training error 0.10).

Theorem (empirical error of AdaBoost)

$$\hat{R}_S(H_T) \leq \prod_{t=1}^T Z_t = \prod_{t=1}^T 2\sqrt{\varepsilon_t(1-\varepsilon_t)} \leq \exp \left[-2 \sum_{t=1}^T \left(\frac{1}{2} - \varepsilon_t \right)^2 \right].$$

If every weak learner has an edge $\frac{1}{2} - \varepsilon_t \geq \gamma > 0$, then $\hat{R}_S(H_T) \leq e^{-2\gamma^2 T}$.

- ▶ As soon as each weak learner beats chance by γ , the training error decreases **exponentially fast** with T .
- ▶ AdaBoost does not need to know γ : it is **adaptive**.

Your turn. Bound the 0–1 loss by the exponential loss, then unroll the weight updates.

Proof (1/3): unrolling the weights

Strategy. bound the 0–1 loss by the exponential loss, and recognize the exponential loss in the final weights.

Step 1 – exponential bound. For any real u : $\mathbb{1}[u \leq 0] \leq e^{-u}$ (if $u \leq 0$, $e^{-u} \geq 1$; if $u > 0$, $0 < e^{-u}$). With $u = y_i f_T(\mathbf{x}_i)$, and since $H_T(\mathbf{x}_i) \neq y_i$ implies $y_i f_T(\mathbf{x}_i) \leq 0$:

$$\hat{R}_S(H_T) \leq \frac{1}{m} \sum_i \mathbb{1}[y_i f_T(\mathbf{x}_i) \leq 0] \leq \frac{1}{m} \sum_i e^{-y_i f_T(\mathbf{x}_i)}.$$

Step 2 – unroll the weights. Apply the update rule twice, starting from $w_i^{(1)} = \frac{1}{m}$:

$$w_i^{(2)} = \frac{1}{m} \cdot \frac{e^{-\alpha_1 y_i h_1(\mathbf{x}_i)}}{Z_1}, \quad w_i^{(3)} = w_i^{(2)} \cdot \frac{e^{-\alpha_2 y_i h_2(\mathbf{x}_i)}}{Z_2} = \frac{1}{m} \cdot \frac{e^{-y_i(\alpha_1 h_1(\mathbf{x}_i) + \alpha_2 h_2(\mathbf{x}_i))}}{Z_1 Z_2}$$

(a product of exponentials is the exponential of the sum), and by induction

$$w_i^{(T+1)} = \frac{1}{m} \frac{\exp\left(-y_i \sum_{t=1}^T \alpha_t h_t(\mathbf{x}_i)\right)}{\prod_{t=1}^T Z_t} = \frac{e^{-y_i f_T(\mathbf{x}_i)}}{m \prod_t Z_t}.$$

Step 3 – conclude. So $\frac{1}{m}e^{-y_i f_T(\mathbf{x}_i)} = (\prod_t Z_t) w_i^{(T+1)}$, and the weights $w_i^{(T+1)}$ sum to 1:

$$\hat{R}_S(H_T) \leq \sum_i \left(\prod_t Z_t \right) w_i^{(T+1)} = \prod_{t=1}^T Z_t.$$

Step 4 – two groups. $y_i h_t(\mathbf{x}_i) = +1$ (correct, total weight $1 - \varepsilon_t$) or -1 (error, total weight ε_t):

$$Z_t = \sum_i w_i^{(t)} e^{-\alpha_t y_i h_t(\mathbf{x}_i)} = (1 - \varepsilon_t) e^{-\alpha_t} + \varepsilon_t e^{\alpha_t}.$$

Step 5 – plug in α_t . $e^{\alpha_t} = \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}$, so $Z_t = (1 - \varepsilon_t) \sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} + \varepsilon_t \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}$.

Step 6 – exponential bound. Write $\gamma_t = \frac{1}{2} - \varepsilon_t$ (the edge): $4\varepsilon_t(1 - \varepsilon_t) = 4(\frac{1}{2} - \gamma_t)(\frac{1}{2} + \gamma_t) = 1 - 4\gamma_t^2$, and $1 - x \leq e^{-x}$:

$$Z_t = \sqrt{1 - 4\gamma_t^2} \leq \sqrt{e^{-4\gamma_t^2}} = e^{-2\gamma_t^2}.$$

Step 7 – product. $\hat{R}_S(H_T) \leq \prod_t Z_t \leq \prod_t e^{-2\gamma_t^2} = \exp(-2 \sum_t \gamma_t^2) \leq e^{-2\gamma^2 T}$ if all $\gamma_t \geq \gamma$. $\square$

Example – six points

$\prod_t Z_t \simeq 0.465$ and $\exp(-2 \sum_t \gamma_t^2) \simeq 0.550$: both bounds hold ($\hat{R}_S(H_3) = 0$), loose after 3 rounds. **Zero training error** as soon as the bound is $< \frac{1}{m}$: after at most $\lfloor \frac{\ln m}{2\gamma^2} \rfloor + 1$ rounds.

Where does α_t come from?

Proposition

1. $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$ is the unique minimizer of $\varphi(\alpha) = (1 - \varepsilon_t)e^{-\alpha} + \varepsilon_t e^{\alpha} = Z_t(\alpha)$: AdaBoost chooses α_t to decrease the bound $\prod_t Z_t$ as much as possible, greedily.
 2. After the update, the weighted error of h_t with respect to $\mathbf{w}^{(t+1)}$ is exactly $\frac{1}{2}$.
- 2. means that, for the new weights, h_t is **no better than random guessing**: the next weak learner is forced to bring new information.

Your turn. Try to prove it before the next slide.

Proof: the choice of α_t

Strategy. minimize a one-dimensional convex function by setting its derivative to zero.

Step 1 – convexity. $\varphi(\alpha) = (1 - \varepsilon_t)e^{-\alpha} + \varepsilon_t e^{\alpha}$ has $\varphi''(\alpha) = (1 - \varepsilon_t)e^{-\alpha} + \varepsilon_t e^{\alpha} > 0$: strictly convex, so a critical point is the unique minimizer.

Step 2 – Euler's equation.

$$\varphi'(\alpha) = -(1 - \varepsilon_t)e^{-\alpha} + \varepsilon_t e^{\alpha} = 0 \iff e^{2\alpha} = \frac{1 - \varepsilon_t}{\varepsilon_t} \iff \alpha = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}.$$

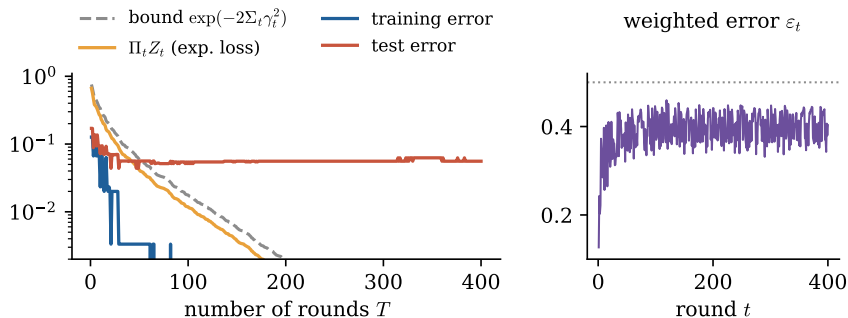
Step 3 – new error of h_t . The weights of the errors are multiplied by $\frac{e^{\alpha_t}}{Z_t}$, so their new total is

$$\frac{\varepsilon_t e^{\alpha_t}}{Z_t} = \frac{\varepsilon_t \sqrt{(1 - \varepsilon_t)/\varepsilon_t}}{2\sqrt{\varepsilon_t(1 - \varepsilon_t)}} = \frac{\sqrt{\varepsilon_t(1 - \varepsilon_t)}}{2\sqrt{\varepsilon_t(1 - \varepsilon_t)}} = \frac{1}{2}. \quad \square$$

Example – six points

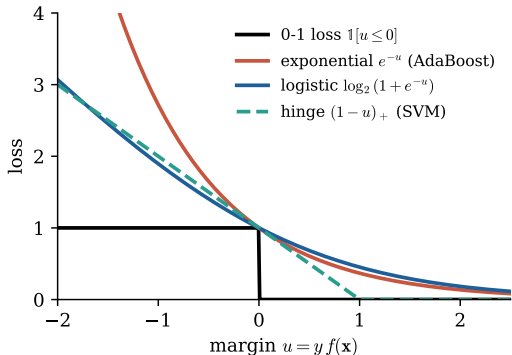
Round 1: $\varepsilon_1 = \frac{1}{6}$; after the update, x_6 (the only error of h_1) weighs 0.5. Round 2: h_1 has error 0.5 for $\mathbf{w}^{(2)}$ (table of round 2).

The guarantee in practice



Two moons, $m = 300$, stumps. Left (log scale): training error $\leq \prod_t Z_t \leq$ bound of the theorem. The training error reaches 0 in less than 100 rounds. Right: the weighted errors ε_t increase (the remaining examples are the hardest ones) but stay below $\frac{1}{2}$, so the bound keeps decreasing.

Which loss does AdaBoost minimize?



- ▶ All these losses depend on the **margin** $u = yf(\mathbf{x})$: $u > 0$ means a correct prediction, $|u|$ is the confidence.
- ▶ The 0–1 loss is not convex: hard to minimize directly.
- ▶ The proof shows that AdaBoost decreases the **empirical exponential loss**

$$F = \frac{1}{m} \sum_i e^{-y_i f_T(\mathbf{x}_i)} = \prod_t Z_t,$$

a convex surrogate.

Is AdaBoost *exactly* an optimization algorithm for this loss? Which one?

AdaBoost is a coordinate descent

Consider a finite family $h_1, \dots, h_N$ of weak learners, $f_\alpha = \sum_{l=1}^N \alpha_l h_l$ and

$$F(\alpha) = \frac{1}{m} \sum_{i=1}^m \exp(-y_i f_\alpha(\mathbf{x}_i)) \quad (\text{convex in } \alpha).$$

Coordinate descent: at each iteration, update only one coordinate, $\alpha^{(k)} = \alpha^{(k-1)} + \rho \mathbf{e}_l$, along the **steepest** direction $\mathbf{e}_l$, with the **best** step ρ .

Proposition

If the weak learner returns the h_l with the smallest weighted error, coordinate descent on F started at $\alpha = 0$ selects, at every iteration, the same classifier and the same weight as AdaBoost.

Your turn. Compute the directional derivative of F along $\mathbf{e}_l$, then the optimal step.

Proof (1/2): the steepest direction

Strategy. compute the slope of F along each coordinate $\mathbf{e}_l$: it is an affine function of the weighted error of h_l .

Let $g_{k-1} = f_{\alpha^{(k-1)}}$, $\tilde{w}_i^{(k)} = e^{-y_i g_{k-1}(\mathbf{x}_i)} / \tilde{Z}_k$ (normalized weights) and $\tilde{\varepsilon}_l^{(k)} = \sum_i \tilde{w}_i^{(k)} \mathbb{1}[y_i h_l(\mathbf{x}_i) < 0]$.

$$\begin{aligned} F'(\alpha^{(k-1)}, \mathbf{e}_l) &= \left. \frac{d}{d\rho} F(\alpha^{(k-1)} + \rho \mathbf{e}_l) \right|_{\rho=0} = -\frac{1}{m} \sum_i y_i h_l(\mathbf{x}_i) e^{-y_i g_{k-1}(\mathbf{x}_i)} \\ &= -\frac{\tilde{Z}_k}{m} \sum_i y_i h_l(\mathbf{x}_i) \tilde{w}_i^{(k)} = -\frac{\tilde{Z}_k}{m} \left[\sum_{y_i h_l(\mathbf{x}_i)=1} \tilde{w}_i^{(k)} - \sum_{y_i h_l(\mathbf{x}_i)=-1} \tilde{w}_i^{(k)} \right] \\ &= \frac{\tilde{Z}_k}{m} (2\tilde{\varepsilon}_l^{(k)} - 1). \end{aligned}$$

The most negative derivative is obtained for the h_l with the **smallest weighted error** $\tilde{\varepsilon}_l^{(k)}$.

By induction, if $g_{k-1} = f_{k-1}$ (the function built by AdaBoost), the unrolled weights of the AdaBoost proof give $\tilde{\mathbf{w}}^{(k)} = \mathbf{w}^{(k)}$: the same classifier as AdaBoost is selected.

Proof (2/2): the optimal step

Strategy. along the chosen coordinate, F is a convex function of the step ρ only: solve Euler's equation.

$\rho \mapsto F(\alpha^{(k-1)} + \rho \mathbf{e}_l)$ is convex; its derivative vanishes when

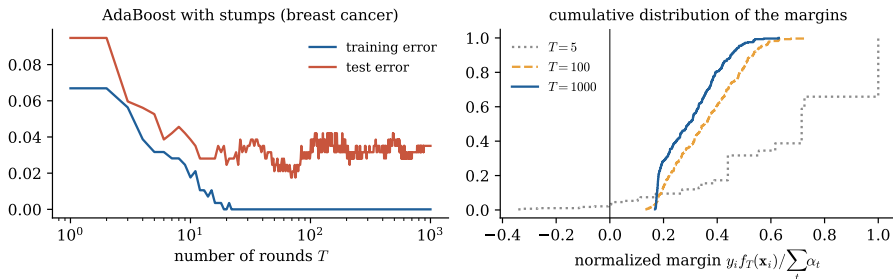
$$\begin{aligned} & -\frac{1}{m} \sum_i y_i h_l(\mathbf{x}_i) e^{-y_i g_{k-1}(\mathbf{x}_i)} e^{-\rho y_i h_l(\mathbf{x}_i)} = 0 \\ \iff & \sum_i y_i h_l(\mathbf{x}_i) \tilde{w}_i^{(k)} e^{-\rho y_i h_l(\mathbf{x}_i)} = 0 \iff (1 - \tilde{\varepsilon}_l^{(k)}) e^{-\rho} - \tilde{\varepsilon}_l^{(k)} e^{\rho} = 0 \\ \iff & \rho = \frac{1}{2} \ln \frac{1 - \tilde{\varepsilon}_l^{(k)}}{\tilde{\varepsilon}_l^{(k)}}. \end{aligned}$$

This is the weight α_k of AdaBoost. Both algorithms start from 0, so they coincide at every iteration.

□

AdaBoost = greedy, stage-wise minimization of the **exponential loss**. Replacing the exponential loss by another loss leads to **gradient boosting**.

Does AdaBoost over-fit? Margins



Breast cancer data, stumps. The training error is 0 after about 20 rounds, yet the test error barely moves (between 0.02 and 0.04) during the next 980 rounds, although f_T becomes more and more complex. Right: AdaBoost keeps increasing the **normalized margins** $y_i f_T(\mathbf{x}_i) / \sum_t \alpha_t$: the vote becomes more confident.

Large margins lead to generalization bounds independent of T (Schapire et al., 1998): we come back to it with PAC-Bayes.

Pitfall. The weight of a misclassified example is multiplied by $\frac{e^{\alpha_t}}{Z_t} = \frac{1}{2\varepsilon_t} > 1$ at every round. A **mislabeled** example, which the weak learners keep “misclassifying”, gets an exponentially growing weight: AdaBoost ends up spending most of its effort on outliers.

- ▶ This comes from the exponential loss, which penalizes large negative margins much more than the logistic loss (see the figure of the losses).
- ▶ On noisy data: gradient boosting with the logistic loss, early stopping, small learning rate.

Boosting in practice. Weak learners: stumps or trees of depth 2–4. Tune T (early stopping on a validation set). In `scikit-learn`: `AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=T)`.

What to remember – boosting

- ▶ Boosting combines **weak** learners **sequentially**: each one is trained on reweighted data that emphasize the errors of the previous ones. It reduces the **bias**.
- ▶ AdaBoost: $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$; misclassified weights $\times \frac{1}{2\varepsilon_t}$, others $\times \frac{1}{2(1-\varepsilon_t)}$.
- ▶ Training error $\leq \prod_t Z_t \leq e^{-2\gamma^2 T}$: exponential decrease as soon as each weak learner beats chance.
- ▶ AdaBoost = coordinate descent on the exponential loss; it increases the margins, but it is sensitive to label noise.

Exercise 3: two rounds of AdaBoost

Exercise – quick computations

Five points $x_i = i$, labels $y = (+1, -1, -1, +1, +1)$, weak learners: decision stumps (and the two constant classifiers).

1. Round 1 (uniform weights). Show that the best stump is “+1 if $x > 3.5$ ”. Compute ε_1 , α_1 , Z_1 and the new weights $\mathbf{w}^{(2)}$.
2. Check that h_1 has a weighted error of exactly $\frac{1}{2}$ for $\mathbf{w}^{(2)}$.
3. Round 2. Show that “+1 if $x < 1.5$ ” and the constant +1 both have weighted error 0.25, and that no stump does better. We take $h_2 \equiv +1$: compute α_2 and Z_2 .
4. Compute $f_2 = \alpha_1 h_1 + \alpha_2 h_2$ at each point, the training error of $H_2 = \text{sign}(f_2)$, and check the bound $\hat{R}_S(H_2) \leq Z_1 Z_2$.

Reminder: $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$, $Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$; errors $\times \frac{1}{2\varepsilon_t}$, correct $\times \frac{1}{2(1-\varepsilon_t)}$.

Solution

1. “+1 if $x > 3.5$ ” predicts $(-, -, -, +, +)$: one error (x_1), $\varepsilon_1 = 0.2$. Any other stump makes at least 2 errors (e.g. constant +1: x_2, x_3). $\alpha_1 = \frac{1}{2} \ln \frac{0.8}{0.2} = \ln 2 \simeq 0.693$, $Z_1 = 2\sqrt{0.16} = 0.8$.
 x_1 : $0.2 \times \frac{1}{0.4} = 0.5$; others: $0.2 \times \frac{1}{1.6} = 0.125$: $\mathbf{w}^{(2)} = (0.5, 0.125, 0.125, 0.125, 0.125)$.
2. h_1 only errs on x_1 , of weight 0.5. ✓
3. “+1 if $x < 1.5$ ” predicts $(+, -, -, -, -)$: errors x_4, x_5 , 0.25. Constant +1: errors x_2, x_3 , 0.25. Any stump that misclassifies x_1 pays 0.5; the others make at least two errors among $x_2, \dots, x_5$ (≥ 0.25). $\alpha_2 = \frac{1}{2} \ln \frac{0.75}{0.25} = \frac{1}{2} \ln 3 \simeq 0.549$, $Z_2 = 2\sqrt{0.25 \times 0.75} \simeq 0.866$.
4. $f_2(x_1) = f_2(x_2) = f_2(x_3) = -0.693 + 0.549 = -0.144$,
 $f_2(x_4) = f_2(x_5) = 0.693 + 0.549 = 1.242$. $H_2 = (-, -, -, +, +)$: one error (x_1),
 $\widehat{R}_S(H_2) = 0.2 \leq Z_1 Z_2 \simeq 0.693$. ✓

After two rounds x_1 is still misclassified, but its weight is now $w_1^{(3)} = \frac{1}{3}$, the largest one: the next round will focus on it.

Exercise 4: how many rounds?

Exercise – quick computations

All the weak learners of AdaBoost have a weighted error $\varepsilon_t = 0.4$ (edge $\gamma = 0.1$).

1. Compute Z_t . How many rounds guarantee a training error below 5% with the bound $\prod_t Z_t$? With the bound $e^{-2\gamma^2 T}$?
2. With $m = 200$ training examples, after how many rounds is the training error guaranteed to be exactly 0?
3. What happens if a weak learner has $\varepsilon_t = \frac{1}{2}$? And $\varepsilon_t > \frac{1}{2}$?

Your turn. Take logarithms: $q^T \leq b \iff T \geq \ln b / \ln q$ when $0 < q < 1$.

Solution

1. $Z_t = 2\sqrt{0.24} = \sqrt{0.96} \simeq 0.9798$. $Z_t^T \leq 0.05 \iff T \geq \frac{2\ln 0.05}{\ln 0.96} \simeq 146.8$: $T = 147$.
With $e^{-2 \times 0.01 T} \leq 0.05$: $T \geq \frac{\ln 20}{0.02} \simeq 149.8$: $T = 150$ (slightly looser, because of $1 - x \leq e^{-x}$).
2. The training error is a multiple of $\frac{1}{200}$: it is 0 as soon as the bound is $< \frac{1}{200}$.
 $T > \frac{2\ln 200}{-\ln 0.96} \simeq 259.6$: $T = 260$ rounds (with $e^{-2\gamma^2 T}$: $T > \frac{\ln 200}{0.02} \simeq 264.9$, $T = 265$).
3. $\varepsilon_t = \frac{1}{2}$: $\alpha_t = 0$ and $Z_t = 1$: h_t gets no weight and the bound does not decrease, the weak learner brings nothing. $\varepsilon_t > \frac{1}{2}$: $\alpha_t < 0$, i.e. AdaBoost uses $-h_t$, whose error $1 - \varepsilon_t$ is $< \frac{1}{2}$; what matters is the distance $|\frac{1}{2} - \varepsilon_t|$ to random guessing.

Even very weak learners (60% of correct weighted predictions) drive the training error to 0 in a few hundred rounds.

Part 5

Stacking and the Super Learner

Question of this part. Instead of a fixed rule (average, vote), can we **learn** how to combine heterogeneous models?

In this part:

- ▶ stacking: a meta-model fed with the predictions of the base models;
- ▶ why it must use out-of-fold predictions;
- ▶ the Super Learner, on a six-point example.

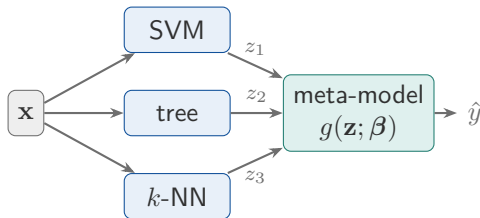
Main idea. The combination itself is learned, with cross-validation.

Stacking (Wolpert, 1992)

Idea. Instead of a fixed rule (average, vote), **learn** how to combine models with different predictive capabilities.

1. Learn base learners $h_1, \dots, h_T$ on the training data; they can all be different (SVM, tree, k -NN, logistic regression...).
2. Their predictions $\mathbf{z} = (h_1(\mathbf{x}), \dots, h_T(\mathbf{x}))$ become the **features of a meta-model** $g(\mathbf{z}; \beta)$, which learns the best combination.

- ▶ Differences with bagging and boosting: heterogeneous models, learned independently on the same data; the weights are learned by a model, not computed by a rule.
- ▶ The outputs $h_t(\mathbf{x})$ can be labels, scores or probabilities.

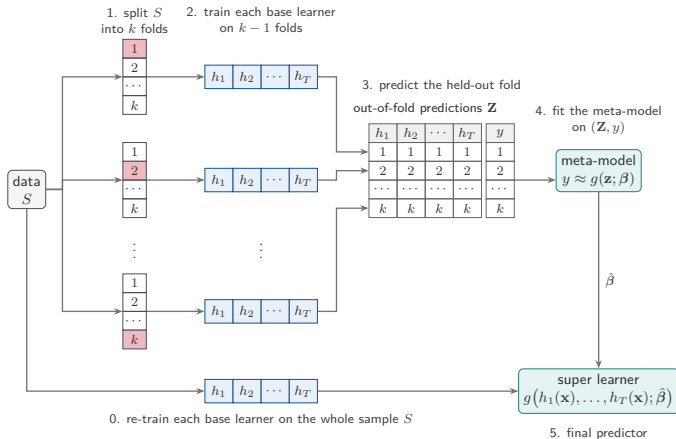


Do not fit the meta-model on the training predictions

If the meta-model is fitted on the predictions of the base learners **on their own training data**, it learns to trust the models that **memorize** the data best (a deep tree, 1-NN: zero training error) and the ensemble over-fits.

- ▶ The meta-model must see predictions for examples that the base learners have **not seen**.
- ▶ Simple solution: split into a training set (base learners) and a validation set (meta-model)... but each part gets less data.
- ▶ Better solution: use **cross-validation** to get a prediction of every example by a model that did not see it: the **Super Learner**.

The Super Learner (van der Laan et al., 2007)



1. Split S into k folds.
2. Train each base learner on $k - 1$ folds, predict the held-out fold; repeat for every fold.
3. The **out-of-fold** predictions form a new design matrix $\mathbf{Z}$ ($m \times T$).
4. Fit the meta-model on $(\mathbf{Z}, y)$.
5. Re-train the base learners on the whole S and combine them with the fitted weights $\hat{\beta}$ (in the example below, β^*).

Example: a Super Learner on six points (1/2)

Data $x = (1, \dots, 6)$, $y = (1, 2.5, 2, 4, 3, 5.5)$. Two base learners: **A** = the mean of the training targets (very stable, biased); **B** = 1-nearest neighbour (ties: left neighbour). Three folds: $F_1 = \{1, 4\}$, $F_2 = \{2, 5\}$, $F_3 = \{3, 6\}$.

i	fold	y_i	z_{iA} (mean)	z_{iB} (1-NN)
1	F_1	1	3.25	2.5 (x_2)
2	F_2	2.5	3.125	1 (x_1)
3	F_3	2	2.625	2.5 (x_2)
4	F_1	4	3.25	2 (x_3)
5	F_2	3	3.125	4 (x_4)
6	F_3	5.5	2.625	3 (x_5)

- ▶ Row 1: $x_1 \in F_1$, so A and B are trained on $\{2, 3, 5, 6\}$. A predicts $\frac{2.5+2+3+5.5}{4} = 3.25$; the nearest neighbour of x_1 is x_2 : B predicts 2.5.
- ▶ Each prediction is made by a model that has **not seen** the example: the columns of **Z** are honest predictions.
- ▶ Cross-validated MSE: A: 2.45, B: 2.67.

Example: a Super Learner on six points (2/2)

Meta-model: a convex combination $\hat{y} = \beta z_A + (1 - \beta) z_B$, $\beta \in [0, 1]$, fitted by least squares on $(\mathbf{Z}, y)$.

Minimize $\sum_i (y_i - z_{iB} - \beta(z_{iA} - z_{iB}))^2$: a 1D least-squares problem, so

$$\beta^* = \frac{\sum_i (y_i - z_{iB})(z_{iA} - z_{iB})}{\sum_i (z_{iA} - z_{iB})^2} \simeq 0.59.$$

Super Learner: $0.59 A + 0.41 B$. Its MSE on the out-of-fold predictions is 2.23, below 2.45 and 2.67 (by construction, since $\beta = 0$ and $\beta = 1$ are allowed; an honest estimate needs an outer cross-validation).

With the training predictions instead: 1-NN predicts every training point perfectly ($z_{iB} = y_i$, MSE 0), so the meta-model puts $\beta = 0$: **all the weight on 1-NN**, the most over-fitted model.

The out-of-fold predictions tell the meta-model how each base learner behaves on **new** data. The final model re-trains A and B on the six points and combines them with $\beta^* = 0.59$.

Stacking in practice

```
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

stack = StackingClassifier(
    estimators=[("rf", RandomForestClassifier()),
                ("svm", SVC(probability=True)),
                ("knn", KNeighborsClassifier(15))],
    final_estimator=LogisticRegression(), # meta
    cv=5) # out-of-fold predictions
stack.fit(X_train, y_train)
```

- ▶ Typical meta-model: a (non-negative) linear or logistic regression; with non-negative weights summing to 1, it is a **weighted vote**.
- ▶ Gain: exploits the **complementarity** of different models.
- ▶ Cost: each base learner is trained $k + 1$ times.

Open questions: how do we tune the base learners with the same data? How do we estimate the error of the whole procedure?

What to remember – stacking

- ▶ Stacking learns the combination of heterogeneous models with a meta-model whose features are the predictions of the base learners.
- ▶ The meta-model must be fitted on **out-of-fold** predictions (Super Learner, k -fold cross-validation), otherwise it trusts the models that over-fit.
- ▶ With a convex combination as meta-model, the Super Learner is a learned **weighted vote**.

Part 6

Gradient boosting

Question of this part. How do we boost with **any** loss function, for regression and classification?

In this part:

- ▶ gradient boosting as a gradient descent in the space of functions, with worked examples;
- ▶ regularization: learning rate, early stopping;
- ▶ XGBoost: second-order approximation, leaf values and split gain; two exercises at the end.

Main idea. Each new tree fits the negative gradient of the loss: generalized residuals.

From AdaBoost to gradient boosting

- ▶ AdaBoost minimizes the **exponential loss**, which is not suited to every task: regression, noisy labels, ranking, custom costs. . .
- ▶ **Gradient boosting** (Friedman, 2001) minimizes **any differentiable loss** ℓ (written $\ell(y, H)$ from now on, H being the prediction) with the same stage-wise idea:

$$H_t = H_{t-1} + \alpha_t h_t.$$

- ▶ The new weak learner h_t is fitted to the **pseudo-residuals**, the negative gradient of the loss at the current predictions:

$$\tilde{y}_i = -\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}, \quad h_t \approx \arg \min_h \sum_{i=1}^m (\tilde{y}_i - h(\mathbf{x}_i))^2.$$

The golfer. A golfer rarely reaches the hole in one stroke: each stroke starts where the ball stopped and aims at the remaining distance. Each new weak learner aims at **what the current model still gets wrong**.

Algorithm 4: Gradient boosting

Input : S , number of models T , differentiable loss ℓ

Output : $H_T = H_0 + \sum_{t=1}^T \alpha_t h_{\mathbf{a}_t}$

1 $H_0(\mathbf{x}) = \arg \min_{\rho \in \mathbb{R}} \sum_i \ell(y_i, \rho)$; // a constant

2 **for** $t = 1$ **to** T **do**

3 **negative gradient:** $\tilde{y}_i = -\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}$;

4 **approximate it:** $\mathbf{a}_t = \arg \min_{\mathbf{a}} \sum_i (\tilde{y}_i - h_{\mathbf{a}}(\mathbf{x}_i))^2$;

5 **line search:**

$\alpha_t = \arg \min_{\alpha \geq 0} \sum_i \ell(y_i, H_{t-1}(\mathbf{x}_i) + \alpha h_{\mathbf{a}_t}(\mathbf{x}_i))$;

6 **update:** $H_t = H_{t-1} + \alpha_t h_{\mathbf{a}_t}$;

- ▶ **A gradient descent in the space of functions:**
the gradient, known only at the training points, is approximated by a weak learner (a shallow regression tree).
- ▶ ℓ is used for the gradient and the step; the squared loss only to fit the pseudo-residuals.

Proposition

1. Squared loss $\ell(y, H) = \frac{1}{2}(y - H)^2$: $\tilde{y} = y - H_{t-1}(\mathbf{x})$ (the residual), $H_0 = \bar{y}$.
2. Logistic loss $\ell(y, H) = \ln(1 + e^{-2yH})$, $y \in \{-1, +1\}$: $\tilde{y} = \frac{2y}{1 + e^{2yH_{t-1}(\mathbf{x})}}$.
3. Cross-entropy, $y \in \{0, 1\}$, $\ell(y, H) = -y \ln \sigma(H) - (1 - y) \ln(1 - \sigma(H))$:
 $\tilde{y} = y - \sigma(H_{t-1}(\mathbf{x}))$.
4. Absolute loss $\ell(y, H) = |y - H|$: $\tilde{y} = \text{sign}(y - H_{t-1}(\mathbf{x}))$, $H_0 = \text{a median of the } y_i$.

- Pseudo-residuals are **generalized residuals**: in which direction, and how strongly, the prediction should move at each training point to decrease the loss.

Your turn. Differentiate with respect to H ; for 3., $\sigma' = \sigma(1 - \sigma)$.

Proof: pseudo-residuals (1/2)

Strategy. differentiate the loss with respect to the prediction H (chain rule), and minimize $\rho \mapsto \sum_i \ell(y_i, \rho)$ for H_0 .

Step 1 – squared loss. $\frac{\partial}{\partial H} \frac{1}{2}(y-H)^2 = \frac{1}{2} \cdot 2(y-H) \cdot (-1) = -(y-H)$, so $\tilde{y} = y - H_{t-1}(\mathbf{x})$.
For H_0 : $\frac{d}{d\rho} \sum_i \frac{1}{2}(y_i - \rho)^2 = -\sum_i (y_i - \rho) = 0 \iff \rho = \bar{y}$.

Step 2 – logistic loss, $y \in \{\pm 1\}$. With $u = -2yH$, $\frac{\partial}{\partial H} \ln(1 + e^u) = \frac{e^u}{1+e^u} \cdot \frac{\partial u}{\partial H} = \frac{e^{-2yH}}{1+e^{-2yH}} \cdot (-2y)$. Multiplying numerator and denominator by e^{2yH} :

$$\tilde{y} = \frac{2y e^{-2yH}}{1 + e^{-2yH}} = \frac{2y}{e^{2yH} + 1}.$$

When yH is large (correct with a large margin), $\tilde{y} \approx 0$: the example is “done”.

Step 3 – cross-entropy, $y \in \{0, 1\}$. Write $p = \sigma(H)$, with $\sigma'(H) = \sigma(H)(1 - \sigma(H)) = p(1 - p)$:

$$\frac{\partial}{\partial H} [-y \ln p - (1 - y) \ln(1 - p)] = \left(-\frac{y}{p} + \frac{1 - y}{1 - p} \right) p(1 - p) = -y(1 - p) + (1 - y)p = p - y,$$

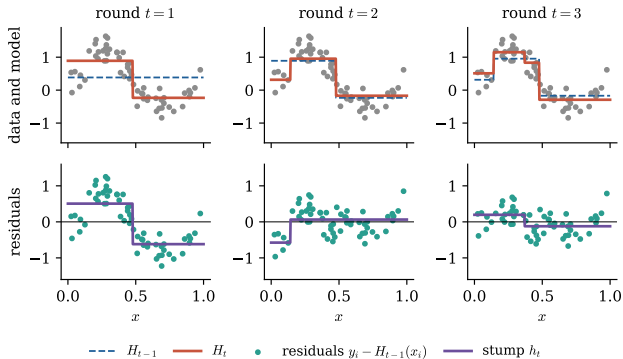
so $\tilde{y} = y - \sigma(H_{t-1}(\mathbf{x}))$: label minus predicted probability.

Step 4 – absolute loss. For $y \neq H$, $\frac{\partial}{\partial H} |y - H| = -\text{sign}(y - H)$, so $\tilde{y} = \text{sign}(y - H_{t-1}(\mathbf{x}))$. For H_0 : the slope of $\rho \mapsto \sum_i |y_i - \rho|$ is $\#\{y_i < \rho\} - \#\{y_i > \rho\}$; it is negative below a median and positive above: the minimum is reached at a median. $\square$

Example – robustness to outliers

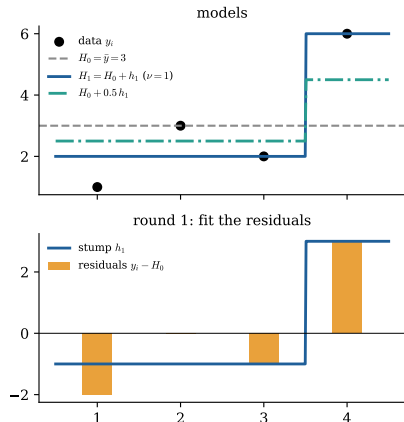
$y = (1, 2, 3, 4, 100)$. Squared loss: $H_0 = 22$, $\tilde{y} = (-21, -20, -19, -18, 78)$: the next tree is driven by the outlier. Absolute loss: $H_0 = 3$ (median), $\tilde{y} = (-1, -1, 0, 1, 1)$: the outlier counts like any other point.

Gradient boosting at work



- ▶ Squared loss, decision stumps, $H_0 = \bar{y}$.
- ▶ Bottom: the residuals of H_{t-1} and the stump h_t fitted to them by least squares.
- ▶ Top: the model before (H_{t-1} , dashed) and after (H_t , solid) round t .
- ▶ The residuals shrink from one round to the next; the model gets progressively closer to the data.

A first round by hand



For the squared loss, $\alpha_t = 1$ for any tree whose leaf values are the means of the residuals.

Example – squared loss, stumps

$x = (1, 2, 3, 4)$, $y = (1, 3, 2, 6)$.

- ▶ $H_0 = \bar{y} = 3$, residuals $(-2, 0, -1, 3)$.
- ▶ Best stump (sum of squared errors): threshold 1.5: 8.67; 2.5: 10; 3.5: **2**. So $h_1 = -1$ if $x < 3.5$, 3 otherwise.
- ▶ Line search: $\alpha_1 = \frac{\sum_i \tilde{y}_i h_1(x_i)}{\sum_i h_1(x_i)^2} = \frac{12}{12} = 1$.
- ▶ $H_1 = (2, 2, 2, 6)$: training MSE from 3.5 to 0.5. With a learning rate $\nu = 0.5$ (shrinkage $H_1 = H_0 + \nu \alpha_1 h_1$, see “Regularizing gradient boosting”): $H_1 = (2.5, 2.5, 2.5, 4.5)$, MSE 1.25.

Example: the second round

After round 1 ($\nu = 1$): $H_1 = (2, 2, 2, 6)$, residuals $\tilde{y}^{(2)} = y - H_1 = (-1, 1, 0, 0)$.

threshold	left leaf	right leaf	SSE
1.5	$\{-1\}: -1$	$\{1, 0, 0\}: \frac{1}{3}$	0.67
2.5	$\{-1, 1\}: 0$	$\{0, 0\}: 0$	2
3.5	$\{-1, 1, 0\}: 0$	$\{0\}: 0$	2

SSE for 1.5: $0 + (1 - \frac{1}{3})^2 + 2(\frac{1}{3})^2 = \frac{4}{9} + \frac{2}{9} = \frac{2}{3}$.

- ▶ $h_2(x) = -1$ if $x < 1.5$, $\frac{1}{3}$ otherwise; again $\alpha_2 = 1$.
- ▶ $H_2 = H_1 + h_2 = (1, 2.33, 2.33, 6.33)$.
- ▶ Residuals $(0, 0.67, -0.33, -0.33)$; training MSE $\frac{1}{4}(0 + \frac{4}{9} + \frac{1}{9} + \frac{1}{9}) \simeq 0.17$.

Training MSE: $3.5 (H_0) \rightarrow 0.5 (H_1) \rightarrow 0.17 (H_2)$. Each round fits what is left. With enough rounds the model interpolates the data: hence the need for a learning rate and early stopping.

Example: gradient boosting for classification

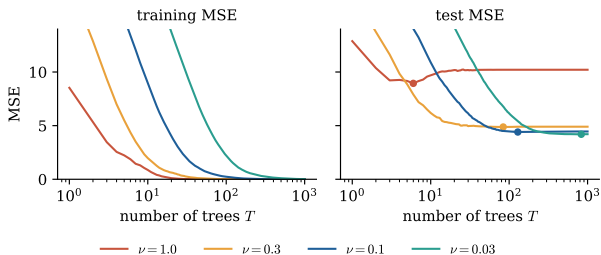
Cross-entropy, $x = (1, 2, 3, 4)$, $y = (1, 0, 1, 1)$, stumps, learning rate $\nu = 0.5$. Predictions are log-odds H , probabilities $p = \sigma(H)$.

1. $H_0 = \ln \frac{\bar{y}}{1-\bar{y}} = \ln 3 \simeq 1.10$, i.e. $p = 0.75$ everywhere.
2. Pseudo-residuals
 $y - p = (0.25, -0.75, 0.25, 0.25)$.
3. Best stump on the residuals: split at 2.5 (SSE 0.5, vs 0.67 for 1.5 and 3.5).
4. Leaf values by one **Newton step** on the log-loss (sum of the negative gradients divided by the sum of the second derivatives $p_i(1 - p_i)$; justified in the XGBoost slides):
$$\frac{\sum (y_i - p_i)}{\sum p_i(1 - p_i)}: \text{left } \frac{-0.5}{0.375} = -1.33, \text{right } \frac{0.5}{0.375} = +1.33.$$

	x_1	x_2	x_3	x_4
y	1	0	1	1
p before	0.75	0.75	0.75	0.75
$H_1 = H_0 + 0.5 h_1$	0.43	0.43	1.77	1.77
p after	0.61	0.61	0.85	0.85

Log-loss: $0.562 \rightarrow 0.437$. The probability of x_2 (label 0) decreases; x_1 is "sacrificed" for now, a later stump can separate it from x_2 .

Regularizing gradient boosting



- ▶ Unlike bagging, boosting **can over-fit** when T grows: each round decreases the training loss.
- ▶ **Learning rate** (shrinkage) $\nu \in (0, 1]$: $H_t = H_{t-1} + \nu \alpha_t h_t$.
- ▶ **Early stopping**: choose T on a validation set.
- ▶ Small ν : more rounds, but a lower minimum of the test error.

Defaults: $\nu \in [0.01, 0.1]$ with early stopping, ν and T tuned *together*; trees of depth 3–6.
Stochastic gradient boosting (Friedman, 2002): fit each tree on a random half of the data, faster and more accurate.

Trees $f_1, \dots, f_T$, prediction $\hat{y}_i = \sum_t f_t(\mathbf{x}_i)$, twice differentiable loss ℓ , and a **regularized** objective:

$$\min_{f_1, \dots, f_T} \sum_{i=1}^m \ell(y_i, \hat{y}_i) + \sum_{t=1}^T \left(\gamma \mathcal{L}_t + \frac{\lambda}{2} \sum_{j=1}^{\mathcal{L}_t} (f_t^{(j)})^2 \right),$$

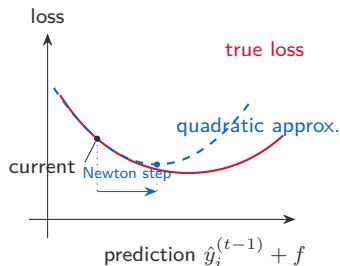
where $\mathcal{L}_t$ is the number of leaves of tree t , $f_t^{(j)}$ the value of its leaf j , and $\gamma, \lambda \geq 0$ are the parameters gamma and lambda of XGBoost. At round t , with $\hat{y}_i^{(t-1)} = H_{t-1}(\mathbf{x}_i)$, the new tree f_t minimizes

$$R_t = \sum_{i=1}^m \ell(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \gamma \mathcal{L}_t + \frac{\lambda}{2} \sum_j (f_t^{(j)})^2.$$

Second-order approximation around $\hat{y}_i^{(t-1)}$, with $g_i = \partial_{\hat{y}} \ell(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial_{\hat{y}}^2 \ell(y_i, \hat{y}_i^{(t-1)})$ (here h_i is a second derivative, not a weak learner):

$$R_t \simeq \sum_i \left[\ell(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t(\mathbf{x}_i)^2 \right] + \gamma \mathcal{L}_t + \frac{\lambda}{2} \sum_j (f_t^{(j)})^2.$$

Why a second-order approximation?



Red: loss as a function of the prediction. Blue: its order-2 Taylor expansion at the current prediction (same value, slope and curvature). Newton's step jumps to the minimum of the blue parabola.

- ▶ Taylor at order 2 around the current prediction $\hat{y}_i^{(t-1)}$:
$$\ell(y_i, \hat{y}_i^{(t-1)} + f) \simeq \ell(y_i, \hat{y}_i^{(t-1)}) + g_i f + \frac{1}{2} h_i f^2.$$
- ▶ A **quadratic** in f is minimized in closed form: this is **Newton's method**, using the curvature h_i in addition to the slope g_i (plain gradient boosting fits the negative slopes $-g_i$ only, and may refine the leaf values afterwards, as in the classification example).
- ▶ For the squared loss the approximation is **exact** ($h_i = 1$).
- ▶ Only g_i and h_i depend on the loss: one algorithm for every twice differentiable loss.

Example – one leaf, no regularization

Minimizing $\sum_i (g_i f + \frac{1}{2} h_i f^2)$ gives $f = -\sum_i g_i / \sum_i h_i$. Cross-entropy at $p = 0.75$, $y = (1, 0, 1, 1)$: $\sum g_i = \sum (p - y_i) = 0$, so $f = 0$: the constant $\ln 3$ is already optimal.

XGBoost: the values of the leaves

Fix the structure of the tree. Let I_j be the examples falling in leaf j , $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$ (notation of XGBoost: here h_i and H_j are second derivatives, not models).

Proposition (optimal leaf values)

If $H_j + \lambda > 0$, the value of leaf j minimizing the approximate objective is

$$f^{(j)\star} = -\frac{G_j}{H_j + \lambda}, \quad \text{and the leaf then contributes} \quad V_j^\star = -\frac{1}{2} \frac{G_j^2}{H_j + \lambda}.$$

- ▶ Squared loss ($g = \hat{y} - y$, $h = 1$): $f^{(j)\star} = \frac{\sum_{I_j} (y_i - \hat{y}_i^{(t-1)})}{|I_j| + \lambda}$, the mean residual, **shrunk towards** 0 by λ .
- ▶ Cross-entropy ($g = p - y$, $h = p(1 - p)$, $p = \sigma(\hat{y}^{(t-1)})$): $f^{(j)\star} = \frac{\sum_{I_j} (y_i - p_i)}{\lambda + \sum_{I_j} p_i(1 - p_i)}$, a Newton step.

Your turn. Group the terms of the approximation by leaf.

Proof of the optimal leaf values

Strategy. in the approximation, the leaves do not interact: minimize one quadratic per leaf.

Step 1 – group by leaf. The term $\ell(y_i, \hat{y}_i^{(t-1)})$ does not depend on f_t , and $f_t(\mathbf{x}_i) = f^{(j)}$ for $i \in I_j$. Grouping the examples by leaf, the approximate objective is $\sum_j V_j + \gamma \mathcal{L}_t + \text{const}$, with

$$V_j = \left(\sum_{i \in I_j} g_i \right) f^{(j)} + \frac{1}{2} \left(\lambda + \sum_{i \in I_j} h_i \right) (f^{(j)})^2 = G_j f^{(j)} + \frac{1}{2} (H_j + \lambda) (f^{(j)})^2.$$

Step 2 – one quadratic per leaf. The leaves are decoupled: minimize each V_j separately. V_j is a convex quadratic when $H_j + \lambda > 0$ (e.g. convex loss, so $h_i \geq 0$, and $\lambda > 0$); Euler's equation $V'_j = G_j + (H_j + \lambda) f^{(j)} = 0$ gives $f^{(j)*} = -G_j / (H_j + \lambda)$.

Step 3 – optimal value. Plugging $f^{(j)*}$ into V_j :

$$V_j^* = -\frac{G_j^2}{H_j + \lambda} + \frac{1}{2} \frac{G_j^2}{H_j + \lambda} = -\frac{1}{2} \frac{G_j^2}{H_j + \lambda}. \quad \square$$

V_j^* measures the quality of a leaf (the lower, the better): a generalization of impurity measures such as the Gini index to any loss.

Proposition (gain of a split)

Splitting a leaf with examples I into $I_L \cup I_R$ decreases the approximate objective by

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G_I^2}{H_I + \lambda} \right] - \gamma.$$

- ▶ The tree is grown greedily: at each leaf, choose the split with the largest gain.
- ▶ A leaf is **not split** if no split has a positive gain: γ acts as a **pruning** threshold, λ shrinks the leaf values.

Your turn. Compare the objective with one leaf and with two leaves.

By the previous proposition, a leaf with examples J contributes at best $V_J^* + \gamma = -\frac{1}{2} \frac{G_J^2}{H_J + \lambda} + \gamma$ to the approximate objective (the other leaves are unchanged).

- ▶ Before the split: one leaf, $V_I^* + \gamma$.
- ▶ After the split: two leaves, $V_L^* + V_R^* + 2\gamma$.

The decrease of the objective is

$$\text{Gain} = V_I^* - V_L^* - V_R^* - \gamma = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G_I^2}{H_I + \lambda} \right] - \gamma. \quad \square$$

The same formula, with g_i and h_i , works for **any** twice differentiable loss: only the computation of g_i and h_i changes. This is what makes XGBoost generic and fast.

Example: the first tree of XGBoost

Cross-entropy, $y \in \{0, 1\}$, $\hat{y}^{(0)} = 0$, $\lambda = 1$, $\gamma = 0$; six examples sorted by one feature:
 $y = (1, 1, 1, 0, 0, 1)$.

- ▶ $p = \sigma(0) = \frac{1}{2}$: $g_i = -\frac{1}{2}$ if $y_i = 1$,
 $+\frac{1}{2}$ if $y_i = 0$; $h_i = \frac{1}{4}$.
- ▶ One leaf: $G = -1$, $H = 1.5$,
 $f = \frac{1}{2.5} = 0.4$, probability
 $\sigma(0.4) \simeq 0.60$ (without λ : $\frac{2}{3}$,
 $\sigma \simeq 0.66$).
- ▶ Score $Q = G^2/(H + \lambda)$; root:
 $Q_I = 0.4$; Gain = $\frac{1}{2}(Q_L + Q_R - Q_I)$.

split after	Q_L	Q_R	Gain
x_1	0.2	0.111	-0.044
x_2	0.667	0	0.133
x_3	1.286	0.143	0.514
x_4	0.5	0	0.05
x_5	0.111	0.2	-0.044

Best split: $\{x_1, x_2, x_3\} | \{x_4, x_5, x_6\}$, leaf values $f_L = \frac{1.5}{1.75} \simeq 0.86$, $f_R = \frac{-0.5}{1.75} \simeq -0.29$.
With $\gamma = 0.6$ the gain $0.514 - 0.6 < 0$: no split.

Example: the first tree of XGBoost, detail of two splits

Recall $g = -\frac{1}{2}$ for $y = 1$, $g = +\frac{1}{2}$ for $y = 0$, $h = \frac{1}{4}$, $\lambda = 1$; labels $y = (1, 1, 1, 0, 0, 1)$; root:
 $G_I = -1$, $H_I = 1.5$, $Q_I = \frac{1}{2.5} = 0.4$.

Example – split after x_3 (the best)

Left $\{1, 2, 3\}$: $G_L = 3 \times (-\frac{1}{2}) = -1.5$,
 $H_L = 0.75$, $Q_L = \frac{2.25}{1.75} \simeq 1.286$.

Right $\{4, 5, 6\}$: $G_R = \frac{1}{2} + \frac{1}{2} - \frac{1}{2} = 0.5$,
 $H_R = 0.75$, $Q_R = \frac{0.25}{1.75} \simeq 0.143$.

Gain $= \frac{1}{2}(1.286 + 0.143 - 0.4) \simeq \mathbf{0.514}$.

Example – split after x_1

Left $\{1\}$: $G_L = -0.5$, $H_L = 0.25$, $Q_L = \frac{0.25}{1.25} = 0.2$.

Right $\{2, \dots, 6\}$: $G_R = -0.5$, $H_R = 1.25$,
 $Q_R = \frac{0.25}{2.25} \simeq 0.111$.

Gain $= \frac{1}{2}(0.2 + 0.111 - 0.4) \simeq -0.044 < 0$:
worse than no split, because λ penalizes small leaves.

The best split isolates a pure group of positives: $f_L = -\frac{G_L}{H_L + \lambda} = \frac{1.5}{1.75} \simeq 0.86$ (probability $\sigma(0.86) \simeq 0.70$), $f_R = -\frac{0.5}{1.75} \simeq -0.29$ (probability 0.43). Without λ , $f_L = 2$: λ **shrinks** the leaf values.

Gradient boosting in practice

```
from sklearn.ensemble import \
    HistGradientBoostingClassifier as HGB
# or: from xgboost import XGBClassifier
gb = HGB(
    loss="log_loss",
    learning_rate=0.05, # nu
    max_iter=2000, # T (upper bound)
    max_depth=4, # weak learners
    l2_regularization=1.0, # lambda
    early_stopping=True, # T chosen on a
    validation_fraction=0.1, # validation set
    random_state=0)
gb.fit(X_train, y_train)
print(gb.n_iter_, gb.score(X_test, y_test))
```

- ▶ Libraries: XGBoost, LightGBM, CatBoost, HistGradientBoosting: histograms of the features, missing values, categorical features, GPU...
- ▶ Main hyper-parameters: ν , T (early stopping), depth of the trees, λ , γ , subsampling.

On tabular data, gradient-boosted trees remain, to this day, very hard to beat, even by deep networks (Grinsztajn et al., 2022).

What to remember – gradient boosting

- ▶ Gradient boosting = gradient descent in function space: each weak learner fits the **pseudo-residuals** $-\partial\ell/\partial H$ of the current model.
- ▶ Squared loss: residuals; cross-entropy: $y - p$; absolute loss: signs (robust).
- ▶ Regularization: learning rate ν , early stopping on T , shallow trees, subsampling.
- ▶ XGBoost: second-order (Newton) approximation, leaf value $-\frac{G}{H+\lambda}$, gain $\frac{1}{2} \left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G_I^2}{H_I+\lambda} \right] - \gamma$.

Exercise 5: gradient boosting by hand

Exercise – quick computations

Regression stumps (leaf value = mean of the targets of the leaf), data $x = (1, 2, 3, 4, 5)$, $y = (2, 4, 3, 8, 8)$.

1. Squared loss $\frac{1}{2}(y - H)^2$. Compute H_0 , the pseudo-residuals of round 1 and the training MSE of H_0 .
2. Find the stump h_1 fitted to the pseudo-residuals (thresholds 1.5, 2.5, 3.5, 4.5).
3. Compute $H_1 = H_0 + \nu h_1$ and its training MSE for $\nu = 1$ and $\nu = 0.5$.
4. Absolute loss $|y - H|$: what are H_0 and the pseudo-residuals of round 1?

Your turn. For 2., compute the sum of squared errors (SSE) of the two leaf means for each threshold.

Solution

1. $H_0 = \bar{y} = 5$; pseudo-residuals = residuals $y - 5 = (-3, -1, -2, 3, 3)$; $MSE = \frac{9+1+4+9+9}{5} = 6.4$.

threshold	left (mean)	right (mean)	SSE
1.5	$\{-3\} (-3)$	$\{-1, -2, 3, 3\} (0.75)$	20.75
2.5	$\{-3, -1\} (-2)$	$\{-2, 3, 3\} (1.33)$	18.67
3.5	$\{-3, -1, -2\} (-2)$	$\{3, 3\} (3)$	2
4.5	$\{-3, -1, -2, 3\} (-0.75)$	$\{3\} (3)$	20.75

$h_1(x) = -2$ if $x < 3.5$, 3 otherwise.

3. $\nu = 1$: $H_1 = (3, 3, 3, 8, 8)$, residuals $(-1, 1, 0, 0, 0)$, $MSE = 0.4$. $\nu = 0.5$: $H_1 = (4, 4, 4, 6.5, 6.5)$, residuals $(-2, 0, -1, 1.5, 1.5)$, $MSE = \frac{4+0+1+2.25+2.25}{5} = 1.9$. The smaller step leaves more work to the next rounds.
4. $H_0 = \text{median of } y = 4$; pseudo-residuals $\text{sign}(y - 4) = (-1, 0, -1, +1, +1)$: only the direction of the errors counts.

Exercise 6: one split of XGBoost

Exercise – quick computations

Squared loss $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, first tree ($\hat{y}_i^{(0)} = 0$), $\lambda = 1$. Four examples sorted by a feature, with targets $y = (1, 2, 6, 7)$.

1. Compute g_i and h_i for each example.
2. Optimal value of a single leaf containing the four examples. Compare with the mean of the targets.
3. Compute the gain of the three possible splits (with $\gamma = 0$). Which one is chosen? Give its leaf values.
4. Explain why one of the gains is negative. Is the best split performed if $\gamma = 5$?

Reminder: $f^* = -\frac{G}{H+\lambda}$, $Q = \frac{G^2}{H+\lambda}$, $\text{Gain} = \frac{1}{2}(Q_L + Q_R - Q_I) - \gamma$.

Solution

1. $g_i = \hat{y}_i - y_i = -y_i$: $g = (-1, -2, -6, -7)$; $h_i = 1$.
2. $G = -16$, $H = 4$: $f^* = \frac{16}{4+1} = 3.2$, smaller than the mean 4: λ shrinks towards 0.
Root score $Q_I = \frac{256}{5} = 51.2$.

3.	split	(G_L, H_L)	(G_R, H_R)	$Q_L + Q_R$	Gain
	$\{1\} \{2, 6, 7\}$	$(-1, 1)$	$(-15, 3)$	$0.5 + 56.25$	2.78
	$\{1, 2\} \{6, 7\}$	$(-3, 2)$	$(-13, 2)$	$3 + 56.33$	4.07
	$\{1, 2, 6\} \{7\}$	$(-9, 3)$	$(-7, 1)$	$20.25 + 24.5$	-3.23

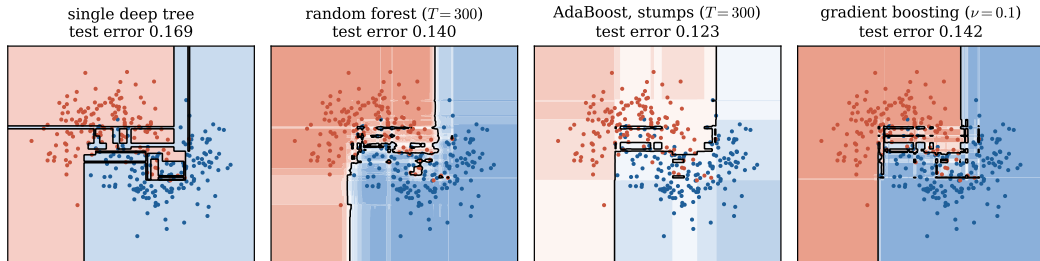
Best: $\{1, 2\}|\{6, 7\}$, leaf values $\frac{3}{3} = 1$ and $\frac{13}{3} \simeq 4.33$ (the means would be 1.5 and 6.5).

4. Without λ , a split can only decrease the objective (gain ≥ 0). With $\lambda = 1$, a leaf with few examples is strongly shrunk ($\{7\}$: value 3.5 instead of 7), so this split loses more than it gains. With $\gamma = 5$: $4.07 - 5 < 0$, the leaf is **not** split.

Part 7

Summary

Comparing the methods on the same problem



Two moons with noise, 300 training examples, test error on 1000 examples. The single deep tree over-fits (blocky, irregular regions). The random forest averages many such trees; boosting builds the boundary from stumps or shallow trees. The three ensembles improve on the tree.

Three ways to combine models

	Bagging / RF	Boosting / GB	Stacking
Base learners	strong, unstable (deep trees)	weak (stumps, shallow trees)	heterogeneous
Diversity from Training	bootstrap (+ features) parallel	reweighting / residuals sequential	different algorithms parallel + meta-model
Reduces	variance	bias (and variance)	both, via complementarity
Over-fits when T grows?	no	yes: early stopping, ν	the meta-model can
Free validation	OOB error	–	out-of-fold predictions

All three build a (weighted) **vote** of models. Why and when a vote is better than its voters, with guarantees: **next lecture, PAC-Bayes and majority votes.**

- ▶ Error = **bias**² + **variance** + noise; Bayes regret = estimation + approximation; the Rademacher complexity measures the ability of a class to fit noise.
- ▶ Averaging: the error of the average is at most the average error; the gain is the **ambiguity** (diversity). Independent voters: Condorcet; correlated voters: much less.
- ▶ Bagging and random forests: bootstrap + random features to decorrelate deep trees; variance $\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$; OOB error for free; feature importances.
- ▶ AdaBoost: $\hat{R}_S(H_T) \leq \prod_t Z_t \leq e^{-2\gamma^2 T}$; α_t minimizes Z_t ; coordinate descent on the exponential loss; margins; sensitive to label noise.
- ▶ Stacking and Super Learner: a meta-model fitted on out-of-fold predictions.
- ▶ Gradient boosting: fit the negative gradient of any loss; XGBoost: second-order approximation, leaf value $-G/(H + \lambda)$, gain of a split; tune ν and T together.

- ▶ P. Bartlett, S. Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *JMLR*, 2002.
- ▶ L. Breiman. Bagging predictors. *Machine Learning*, 1996. — Random forests. *Machine Learning*, 2001.
- ▶ T. Chen, C. Guestrin. XGBoost: a scalable tree boosting system. *KDD*, 2016.
- ▶ B. Efron. Bootstrap methods: another look at the jackknife. *Annals of Statistics*, 1979.
- ▶ Y. Freund, R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS*, 1997.
- ▶ J. Friedman, T. Hastie, R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 2000.
- ▶ J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001. — Stochastic gradient boosting. *CSDA*, 2002.
- ▶ L. Grinsztajn, E. Oyallon, G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *NeurIPS*, 2022.
- ▶ T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning*, Springer, 2009.
- ▶ A. Krogh, J. Vedelsby. Neural network ensembles, cross validation, and active learning. *NIPS*, 1995.
- ▶ M. Mohri, A. Rostamizadeh, A. Talwalkar. *Foundations of Machine Learning*, MIT Press, 2012 (2nd ed. 2018).
- ▶ R. Schapire, Y. Freund, P. Bartlett, W. S. Lee. Boosting the margin. *Annals of Statistics*, 1998.
- ▶ R. Schapire, Y. Freund. *Boosting: Foundations and Algorithms*, MIT Press, 2012.
- ▶ M. van der Laan, E. Polley, A. Hubbard. Super learner. *Stat. Appl. Genet. Mol. Biol.*, 2007.
- ▶ D. Wolpert. Stacked generalization. *Neural Networks*, 1992.