Ensemble Methods

Mohammad Salameh and Pat Virtue

Ensemble methods:
what are they?

An **ensemble method** learns a collection of models $h_1, \ldots, h_T$ and **combines their predictions** on each test point, usually by a (weighted)vote or an average.

classification

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \, h_t(x)\right)$$

regression

$$H(x) = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$$

The methods differ in two choices:

Where do the models come from?

Given in advance (weighted majority), trained one after another on reweighted data (boosting), or trained independently on resampled data (bagging, random forests).

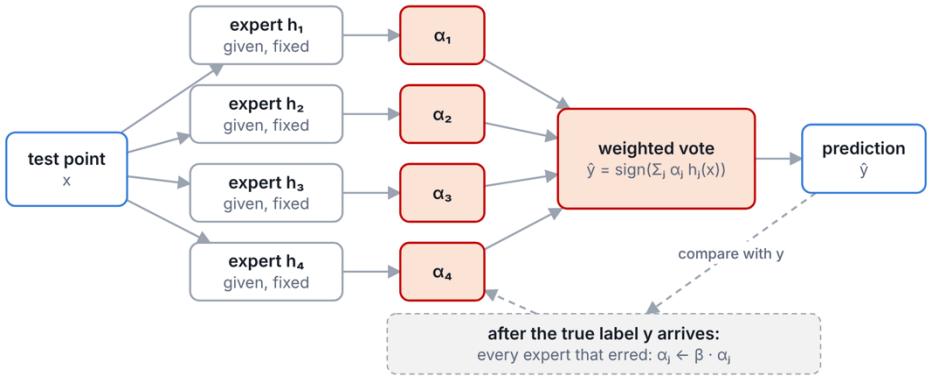
How are they combined?

A plain vote or average, a weighted vote with learned weights α_t, or a second model that learns the combination (stacking).

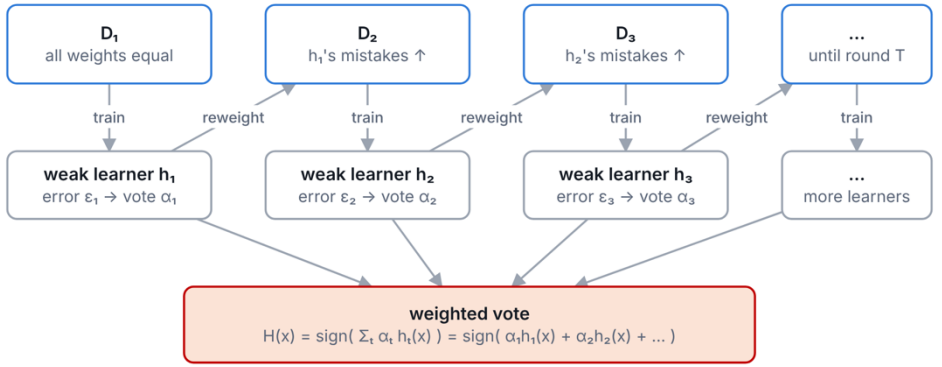
Ensemble methods:

how do the four families differ?

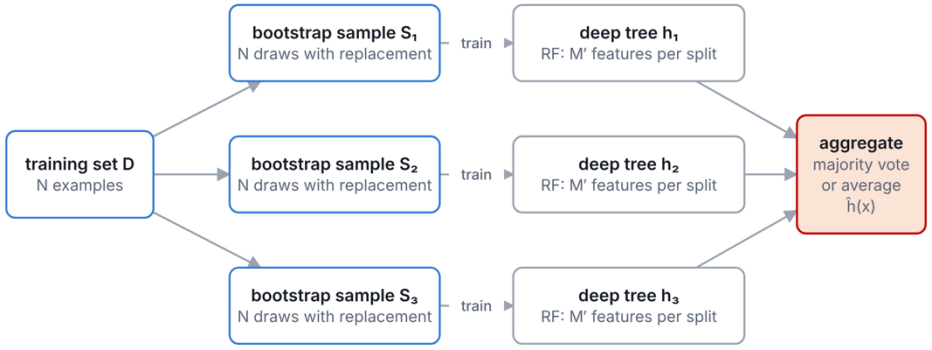
Weighted majority: given experts, learn vote weights



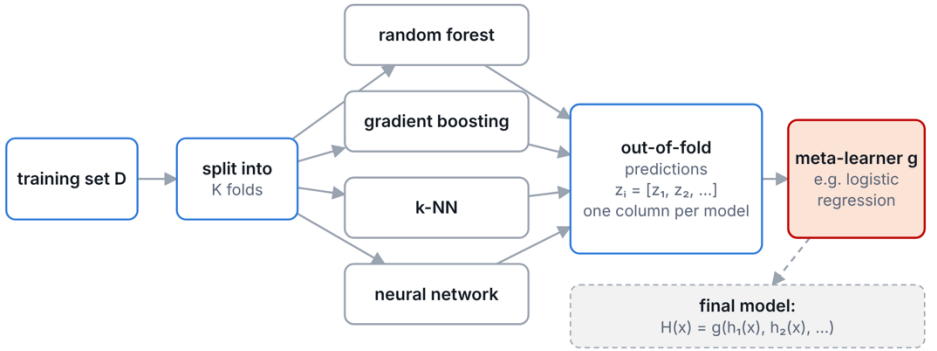
Boosting: serial, reweight the data



Bagging / random forests: parallel, resample



Stacking: learn the combiner



Definitions and notation:

what notation will we use?

SYMBOL	MEANING
$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$	training set of N examples; each $x^{(i)}$ has M features
$y \in \{-1, +1\}$	binary label: with ±1, **y·h(x) = +1 exactly when h is correct**
$h : \mathbb{R}^M \to \{-1, +1\}$	one hypothesis (classifier); an ensemble combines h_1, …, h_T
$\text{sign}(z)$	+1 if z ≥ 0, else −1: turns a weighted sum of votes into a label
$\mathbf{1}[\cdot]$	indicator: 1 if the condition holds, 0 otherwise
$\widehat{\text{err}}(h) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[h(x^{(i)}) \neq y^{(i)}]$	training error (average 0-1 loss)
$\text{err}(h) = P[h(x) \neq y]$	true error on the data distribution, estimated on a test set

Definitions and notation:

what do the key terms mean?

TERM	DEFINITION
Ensemble method	learns a collection of models and combines their predictions on each test instance
Base learner	the learning algorithm that produces each member h_t of the ensemble
Weak learner	returns a hypothesis only slightly better than random guessing (error ≤ ½ − γ on any distribution)
Strong learner	returns a hypothesis of arbitrarily low error
Bagging / boosting	learn models in parallel on resampled data / serially on reweighted data
Diversity	how differently the members err; lower error correlation = more diversity
Online learning	examples arrive one at a time; the learner is judged by its number of mistakes
Bootstrap / out-of-bag	N draws with replacement from N examples / an example a model never saw

Definitions and notation:

how are the members' predictions combined?

Majority vote

$$H(x) = \arg\max_y \sum_t \mathbf{1}[h_t(x) = y]$$

Bagging and random forests

Weighted majority vote

$$H(x) = \text{sign}\left(\sum_t \alpha_t \, h_t(x)\right)$$

Weighted majority and AdaBoost: better members get a bigger say

Averaging

$$H(x) = \frac{1}{T}\sum_t h_t(x)$$

Regression; gradient boosting uses a shrunken sum

Soft voting

$$\bar{p}(y \mid x) = \frac{1}{T}\sum_t p_t(y \mid x)$$

Average class probabilities, then take the arg max (deep ensembles)

Probability toolkit:

how much does averaging models reduce variance?

Fix a test point x. Each prediction $h_b(x)$ is a **random variable** because it depends on random choices made during training:

- which training set was drawn
- which bootstrap sample
- which features were sampled at each split, or
- which random initialization was used

Variance : how much one model's prediction at x would change if you retrained it with different random choices.

$$\sigma^2 = Var(h_b(x))$$

Probability toolkit:

how much does averaging models reduce variance?

Fix a test point x. Each prediction $h_b(x)$ is a **random variable** because it depends on random choices made during training:

- which training set was drawn
- which bootstrap sample
- which features were sampled at each split, or
- which random initialization was used

Correlation: for two different models $b \neq b'$.

Whether they tend to err in the *same direction* at the same point.

$$\rho = Corr(h_b(x), h_{b'}(x))$$

- $\rho = 1$ means they always move together (identical models)
- $\rho = 0$ means one model being too high tells you nothing about the other.

Probability toolkit:

how much does averaging models reduce variance?

The variance of a sum includes the covariances:

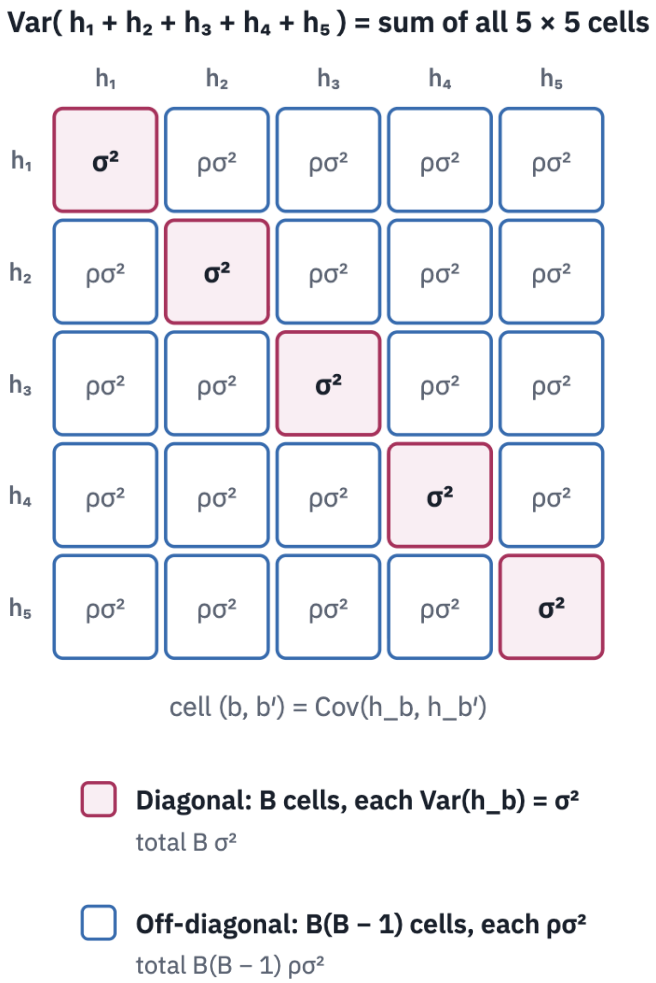
$$\mathrm{Var}\left(\sum_{b=1}^{B} Z_b\right) = \sum_b \mathrm{Var}(Z_b) + \sum_{b \neq b'} \mathrm{Cov}(Z_b, Z_{b'}).$$

$$\mathrm{Var}\left(\frac{1}{B}\sum_b h_b\right) = \frac{1}{B^2}\Big(\underbrace{B\sigma^2}_{B \text{ variances}} + \underbrace{B(B-1)\,\rho\sigma^2}_{B(B-1) \text{ covariances}} \Big)$$

$$= \frac{\sigma^2}{B} + \frac{B-1}{B}\rho\sigma^2$$

$$= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

- **ρ = 0:** σ^2/B, so averaging can drive the variance to 0
- **ρ = 1:** σ^2, so averaging copies does nothing
- **Example:** $\sigma^2 = 0.1$, $\rho = 0.3$: 10 trees give 0.037; 1,000 trees give 0.030



Probability toolkit:
how accurate is a majority vote?

Whether one classifier is correct on a random test point is a **Bernoulli** random variable.

If n classifiers are right **independently**, the number correct K is **binomial**.

$$P(K = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad \mathbb{E}[K] = np, \quad \mathrm{Var}(K) = np(1-p)$$

$$P(\text{vote correct}) = \sum_{k>n/2} \binom{n}{k} p^k (1-p)^{n-k}.$$

Example: five classifiers, each 70% accurate

$$P(K = 3) = \binom{5}{3} (0.7)^3 (0.3)^2 = 10 \times 0.343 \times 0.09 = 0.309$$

Summing such terms over every majority (k = 3, 4, 5) gives the accuracy of a majority vote: the key to Part II.

Probability toolkit:

How accurate is a vote of independent classifiers?

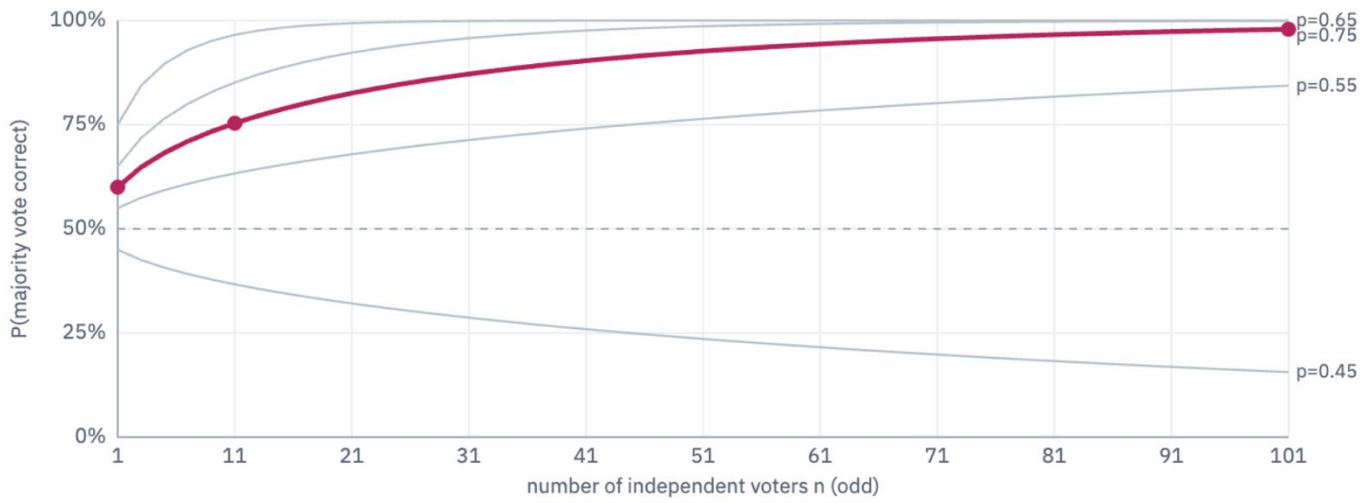
Suppose n classifiers each predict the right label with probability $p > \frac{1}{2}$, and their errors are **independent**.
The majority vote is wrong only when more than half of them err at once.
By the binomial distribution:

$$P(\text{majority correct}) = \sum_{k=\lceil (n+1)/2 \rceil}^{n} \binom{n}{k} p^k (1-p)^{n-k} \xrightarrow{n\to\infty} 1 \quad \text{if } p > \tfrac{1}{2}.$$

For $n = 3$ and $p = 0.6$, the majority is right when 2 or 3 classifiers are right:

$$\underbrace{\binom{3}{2}(0.6)^2(0.4)}_{k=2:\ 3\times 0.36\times 0.4 = 0.432} + \underbrace{\binom{3}{3}(0.6)^3}_{k=3:\ 0.216} = 0.648.$$

Three 60%-accurate independent voters give 64.8%. With $n = 5$ the same sum over $k = 3,4,5$ gives $0.346 + 0.259 + 0.078 = 0.683$
with $n = 11$ about 75%.



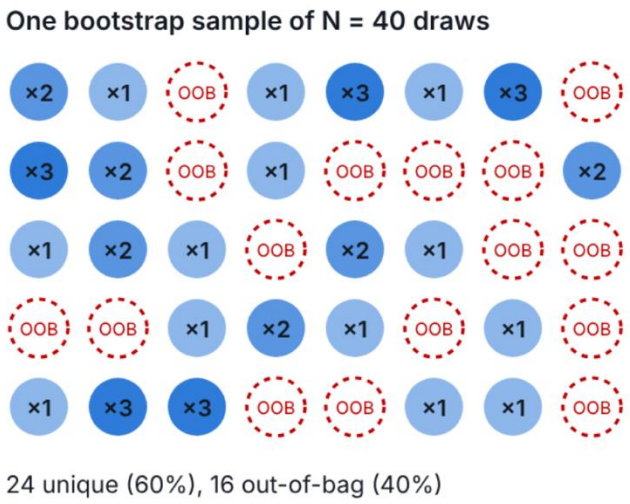
Each voter 60% accurate → 1 voter: 60% · 11 voters: 75.3% · 101 voters: 97.9%

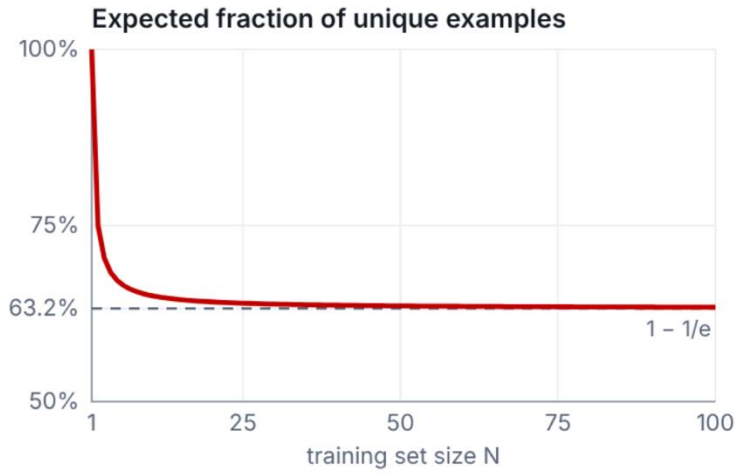
Probability toolkit:

how much data does sampling with replacement keep?

Draw N times uniformly, with replacement, from N items.
One draw misses a given item with probability $1 - 1/N$
The draws are independent, so all N draws miss it with probability

$$P(\text{an item is never drawn}) = \left(1 - \frac{1}{N}\right)^N \xrightarrow{N\to\infty} e^{-1} \approx 0.368$$





63.2%

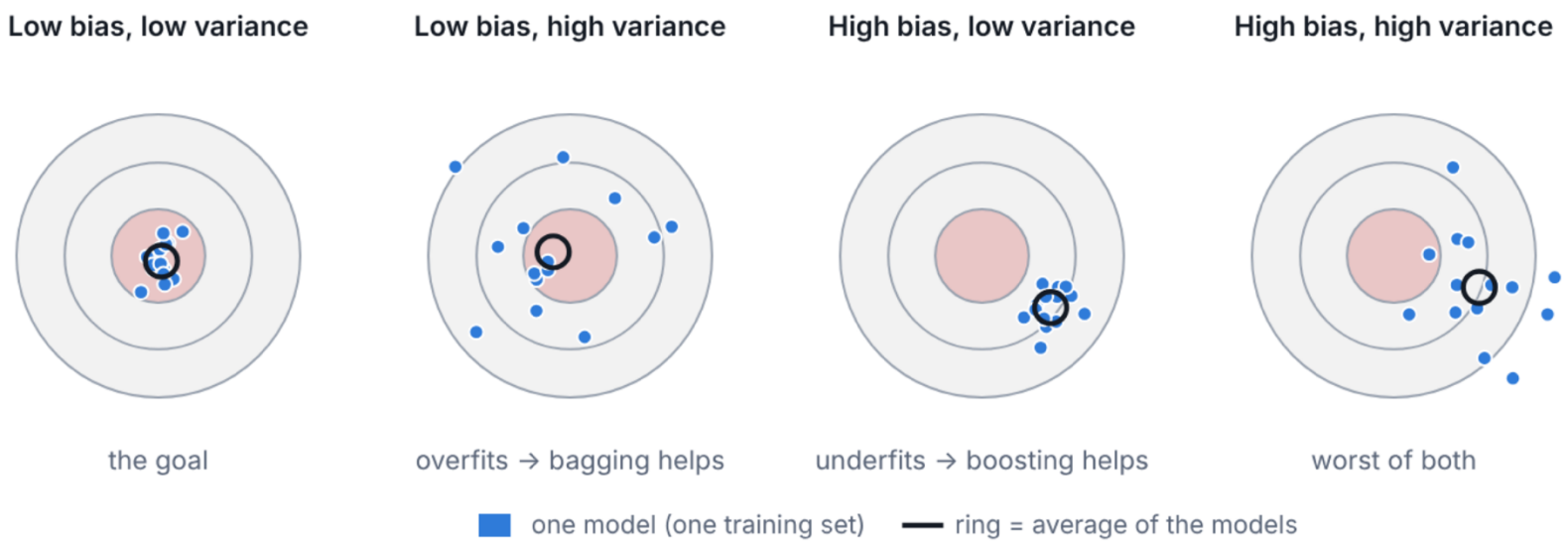
of the distinct examples land in each bootstrap sample

36.8%

are left out: "out-of-bag" for that model

Bias and variance:

When combining models, what do we want the combination to fix?



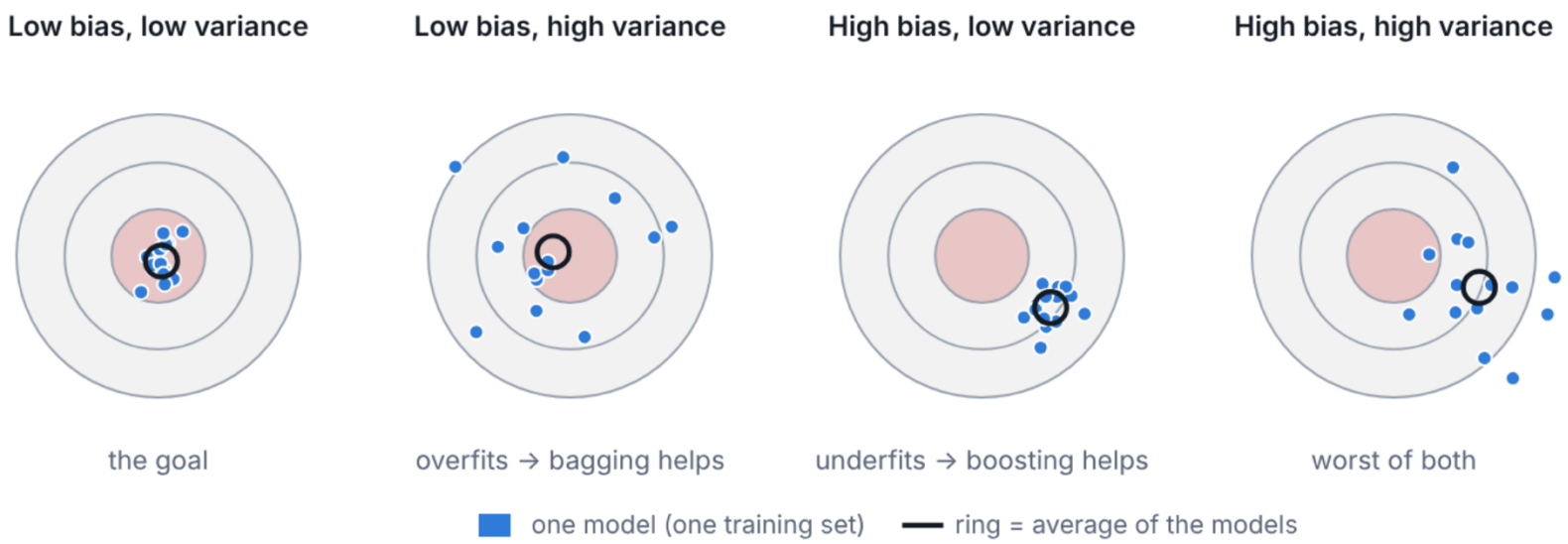
A learned model depends on the training set it happened to see.
Repeat the following experiment many times:

- draw a fresh training set from the same distribution
- train the same learner, and
- record its prediction at one test point

Those predictions form a distribution, and it can be wrong in two different ways.

Bias and variance:

When combining models, what do we want the combination to fix?



$$\text{bias} = \mathbb{E}[h(x)] - c^*(x)$$

How far the **average** prediction is from the truth $c^*(x)$. Too simple: misses the same way every time (**underfitting or systemetic error**).

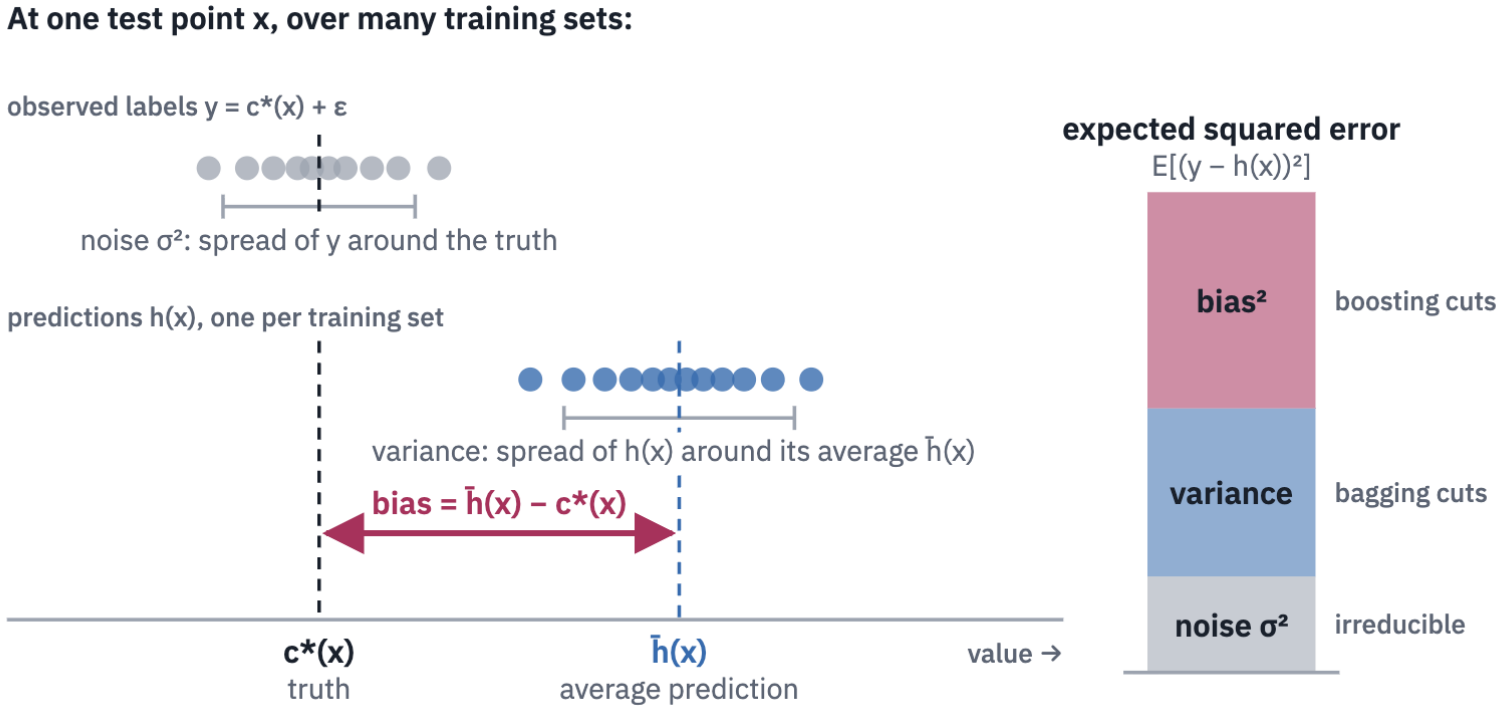
$$\text{variance} = \mathbb{E}\left[(h(x) - \mathbb{E}[h(x)])^2\right]$$

How much the prediction **moves** when the training set changes. Too flexible: fits noise (**overfitting/instability**).

Bias and variance:

how do they add up to the test error?

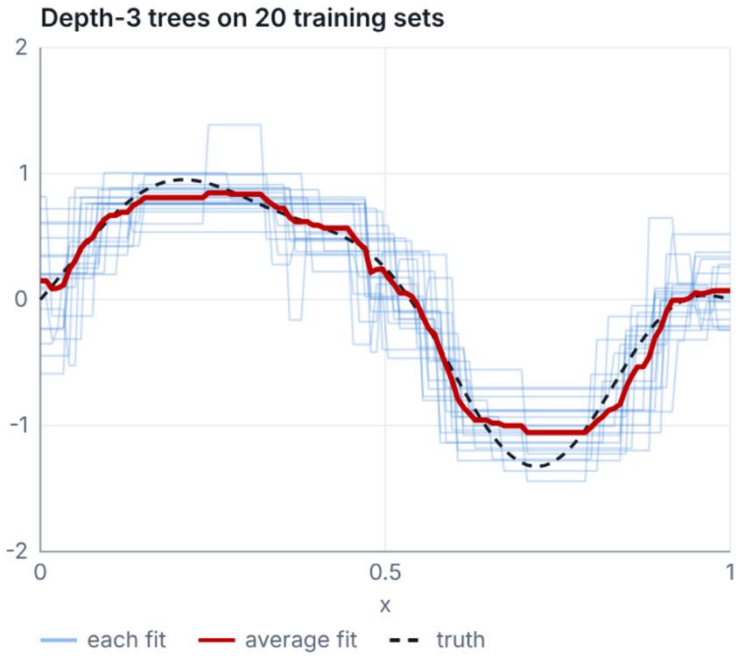
$$\mathbb{E}\left[(y - h(x))^2\right] = \underbrace{\left(\mathbb{E}[h(x)] - c^*(x)\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}\left[(h(x) - \mathbb{E}[h(x)])^2\right]}_{\text{variance}} + \underbrace{\sigma^2}_{\text{noise}}$$

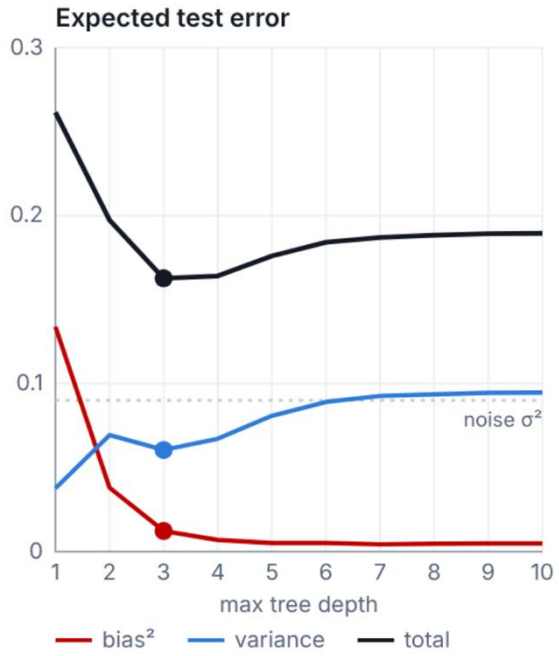


The grey dots are possible labels y. They scatter around the truth $c^*(x)$ because of noise, with variance .
The blue dots are the predictions $h(x)$ of the same learner trained on different training sets. Their average is $\bar{h}(x)$.
Bias is the gap between $\bar{h}(x)$ and $c^*(x)$ (how wrong the learner is on average).
variance is how widely the blue dots spread around $\bar{h}(x)$ (how much it depends on the particular training set).
Squaring and averaging the error $y - h(x)$ gives exactly the three stacked pieces : $bias^2 + variance + noise$.
Making the model more flexible moves toward the truth but spreads the blue dots out

Bias and variance:

how does flexibility trade bias against variance?





- No learner can beat the noise term.
- The other two usually trade off: making a model more flexible lowers bias and raises variance
- Same learner, **20 training sets**
- Depth 1: high bias. Depth 10: high variance
- Total error is a **U-curve**: best single tree at depth 3–4

FLEXIBILITY KNOB	LOW: HIGH BIAS, LOW VARIANCE	HIGH: LOW BIAS, HIGH VARIANCE
Tree depth	stump	fully grown tree
k in k-NN	large k	k = 1
Regularization λ	large λ	λ = 0

Bias and variance:

which error term can an ensemble reduce?

Tuning one model only moves you **along** the U-curve. Ensembles change the curve itself:

cuts variance

Bagging & random forests

Start from **low-bias, high-variance** models (deep trees).

Average them: bias stays low, **variance drops**.

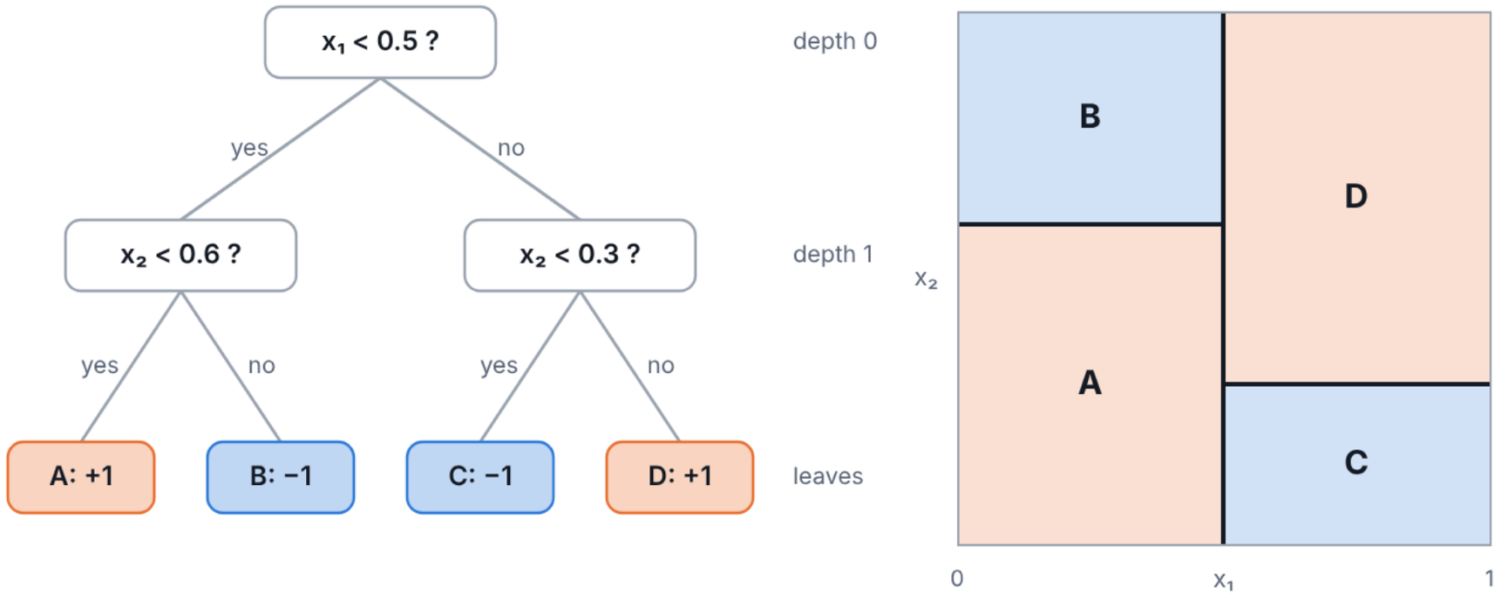
cuts bias

Boosting

Start from **high-bias, low-variance** models (stumps).

Add them up, each correcting the rest: variance stays low, **bias drops**.

Decision trees:



- Each **internal node** tests one feature against a threshold: $x_m < \theta$?
- Each **leaf** predicts a label classification (majority class, or the mean of y for regression)
- A tree cuts the input space into **axis-aligned boxes**
- A **decision stump** is the $depth - 1$ tree: a single split

how does a tree learn, and why is it the usual base learner?
Greedily: each node takes the split that most reduces impurity. Grown deep, a tree fits almost anything but is unstable, which is exactly what ensembles exploit.

Decision trees:

how do we measure how mixed a node is?

A leaf is good when its examples share a label. **Impurity** measures how mixed node S is (p_k= fraction of class k):

MEASURES:

Misclassification error: $1 - \max_k p_k$

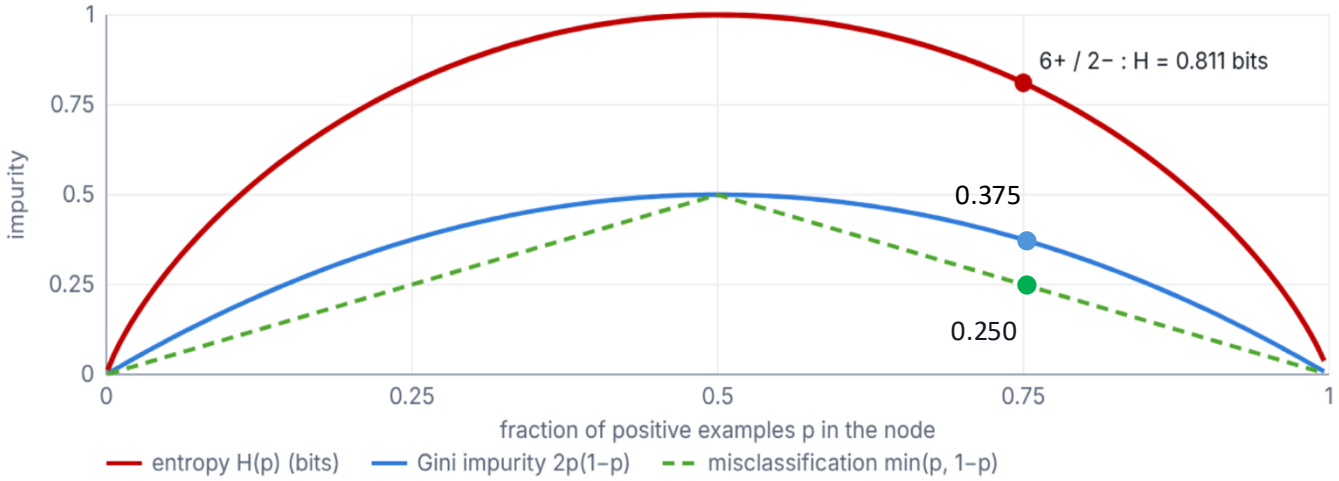
Measures fraction a majority-vote leaf gets wrong

Gini impurity (CART) $1 - \sum_k p_k^2$

Measures the chance two random examples have different labels

Entropy (ID3, C4.5) $H(S) = -\sum_k p_k \log_2 p_k$

Measures the average bits to encode a label from the node



Decision trees:

why are they the favourite base learner for ensembles?

fully grown tree

Low bias: it can fit almost anything.

High variance: move a few training points and the top splits change, and so does everything below.

→ An unstable learner is exactly what **bagging** averages away.

decision stump

High bias: one split is crude.

Low variance: it barely changes from sample to sample.

→ A simple, stable, better-than-chance learner is exactly what **boosting** needs.

Why ensembles work:

why, in general, do ensembles beat single models?

Statistical

With limited data, many hypotheses fit equally well. Averaging them lowers the risk of picking a bad one.

Computational

Greedy learners get stuck in local optima. Different starts land in different optima; averaging smooths over them.

Representational

The truth may lie outside the hypothesis class. A vote of linear classifiers is not linear, so the ensemble can represent more.

Dietterich (2000), Ensemble methods in machine learning.

The methods

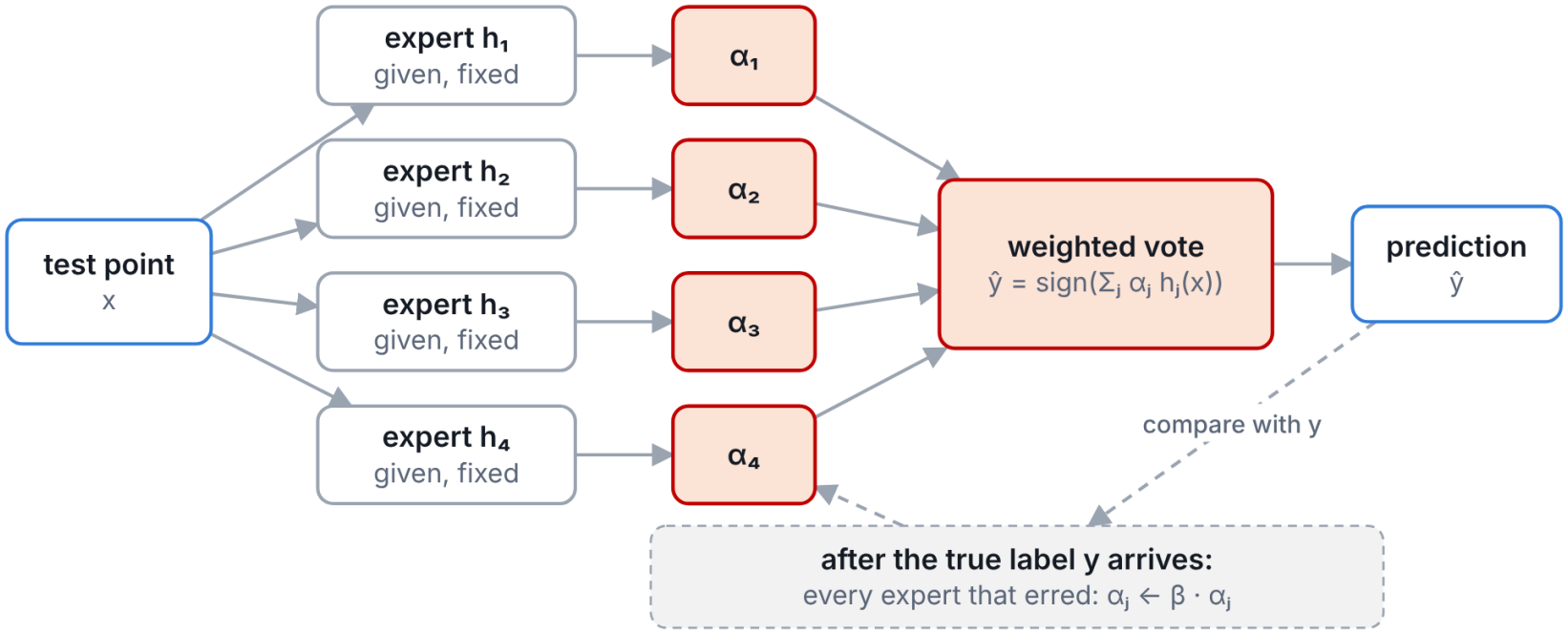
Weighted majority, AdaBoost, bagging, random forests

Weighted majority:
what problem does it solve?

Give each expert a weight, predict by weighted vote, and shrink the weight of every expert that errs. Mistakes stay within a constant factor of the best expert's, plus a log N term.

Setting (Littlestone & Warmuth, 1994)

- We have a pool of binary classifiers ("experts") whose internals we cannot see.
- Examples arrive one at a time (**online learning**).



Goal: We want a learner that uses the pool's predictions and makes almost as few mistakes as the ***best expert in hindsight***, without knowing in advance which expert that is. We learn only the **vote weights**

Weighted majority:
what if one expert is perfect?

Suppose one expert in the pool is perfect, but we don't know which. Predict with the majority of the experts that have never erred, and discard every expert that errs:

How many mistakes can the halving algorithm make?

The algorithm predicts with the majority of the pool, so whenever it is wrong, at least half of the pool voted wrong (red outline) and all of them are discarded. In the worst case exactly half go each time: 16 → 8 → 4 → 2 → 1. The perfect expert (★) is never wrong, so it is never discarded, and once it is the only one left the algorithm cannot err again. So at most $log_2 16 = 4$ mistakes.

Algorithm 4: Halving algorithm

Require: pool $\mathcal{A} = \{h_1, \ldots, h_N\}$, at least one of which never errs

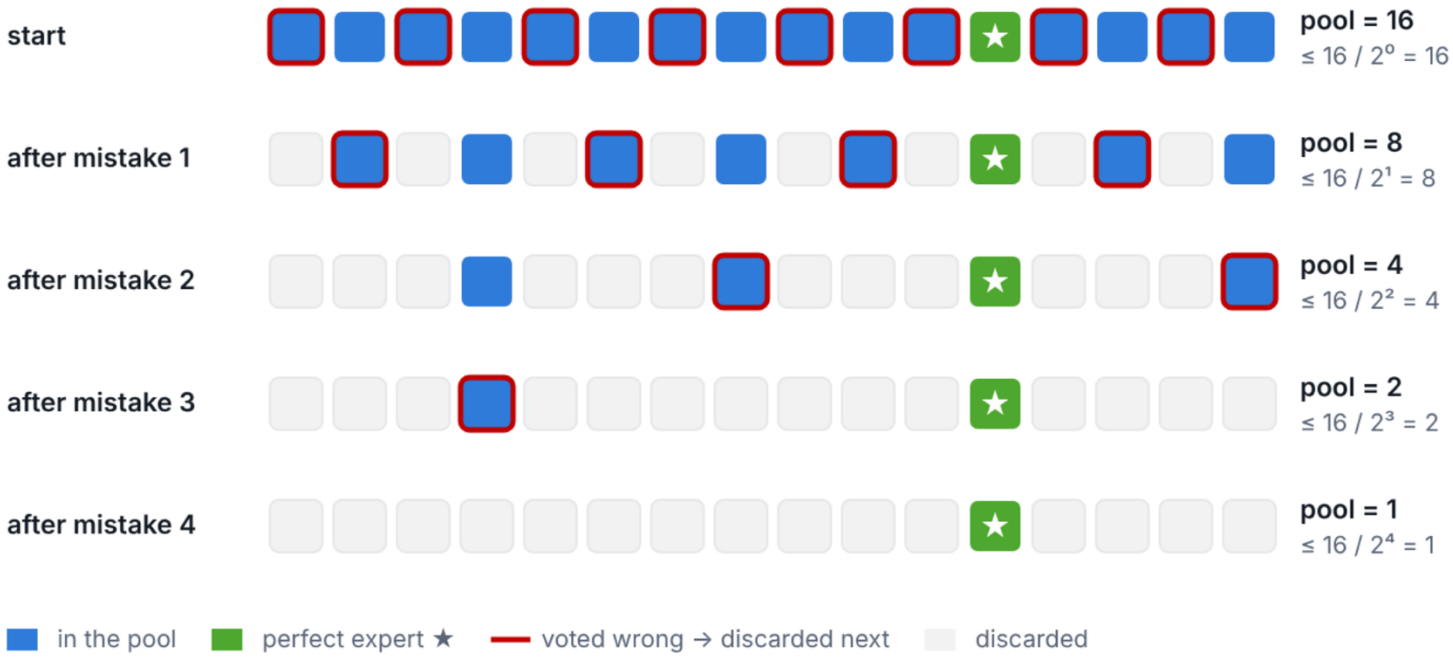
$\mathcal{A}_1 \leftarrow \mathcal{A}$ ▷ *all experts start in the pool*
for each incoming example x **do**
  $\hat{y} \leftarrow$ majority vote of the experts in $\mathcal{A}_t$ on x
  receive the true label y
  $\mathcal{A}_{t+1} \leftarrow \{h \in \mathcal{A}_t : h(x) = y\}$ ▷ *discard every expert that erred*
end for

Weighted majority:
what if one expert is perfect?

Suppose one expert in the pool is perfect, but we don't know which. Predict with the majority of the experts that have never erred, and discard every expert that errs:

How many mistakes can the halving algorithm make?

The algorithm predicts with the majority of the pool, so whenever it is wrong, at least half of the pool voted wrong (red outline) and all of them are discarded. In the worst case exactly half go each time: 16 → 8 → 4 → 2 → 1. The perfect expert (★) is never wrong, so it is never discarded, and once it is the only one left the algorithm cannot err again. So at most $log_2 16 = 4$ mistakes.



$$1 \leq |\mathcal{A}| \leq \frac{N}{2^M} \quad \Longrightarrow \quad M \leq \log_2 N$$

Weighted majority:
what if no expert is perfect?

Weighted Majority (Littlestone & Warmuth, 1994)

Require: pool of binary classifiers $h_1, \ldots, h_N$ (learned beforehand); $\beta \in (0, 1)$

$\alpha_j \leftarrow 1$ for all j ▷ *equal weights*
for each incoming example x **do**
  $\hat{y} \leftarrow \text{sign}\left(\sum_j \alpha_j h_j(x)\right)$ ▷ *weighted majority vote*
  receive the true label y
  for each j with $h_j(x) \neq y$ **do**
    $\alpha_j \leftarrow \beta\, \alpha_j$ ▷ *down-weight by* β
  end for
end for

If no expert is perfect, discarding is too harsh: the best expert would eventually be thrown out.

Weighted majority keeps everyone but multiplies the weight of each expert that errs by β, so **shrinking** each mistake's weight by β

Setting $\beta = 0$ recovers the halving algorithm.

Weighted majority:
what if no expert is perfect?

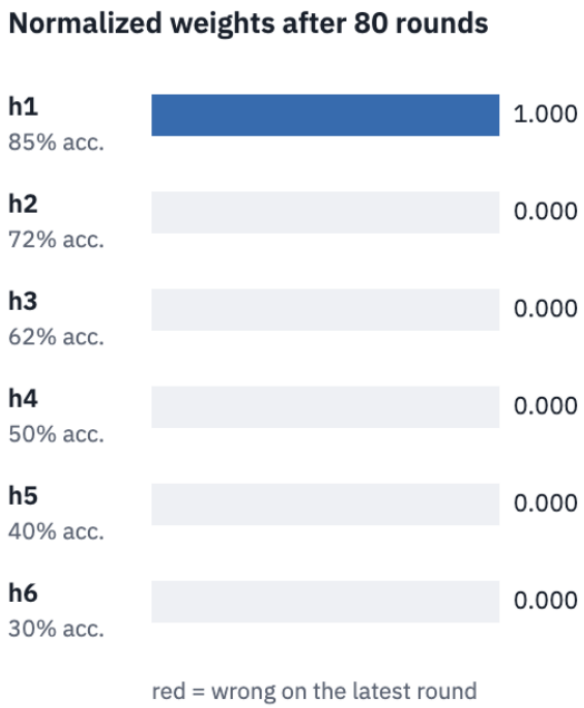
Given six experts with accuracies 85% down to 30% that predict a stream of 80 labels.

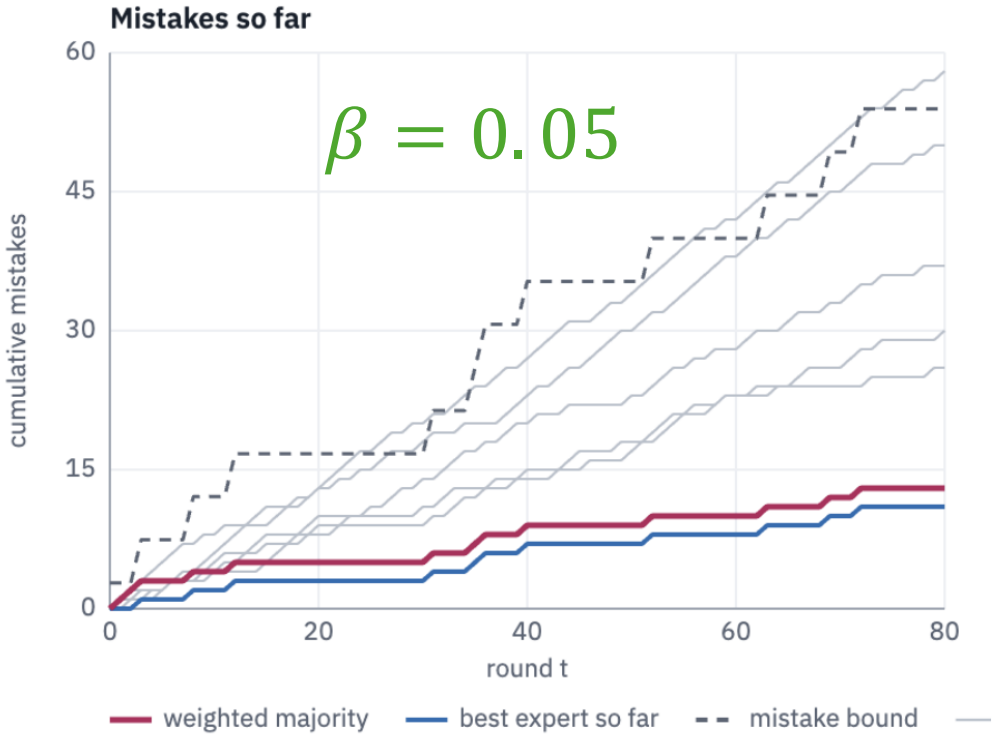
Weights of experts that err are multiplied by β.

The weight mass migrate to h_1.

A small β trusts the current leader quickly

β near 1 forgets slowly.





Round 80: weighted vote +1 gets 0.00, −1 gets 1.00 → predict −1; truth −1. Totals: WM 13 mistakes, best expert 11, bound 53.9.

Weighted majority:
what if no expert is perfect?

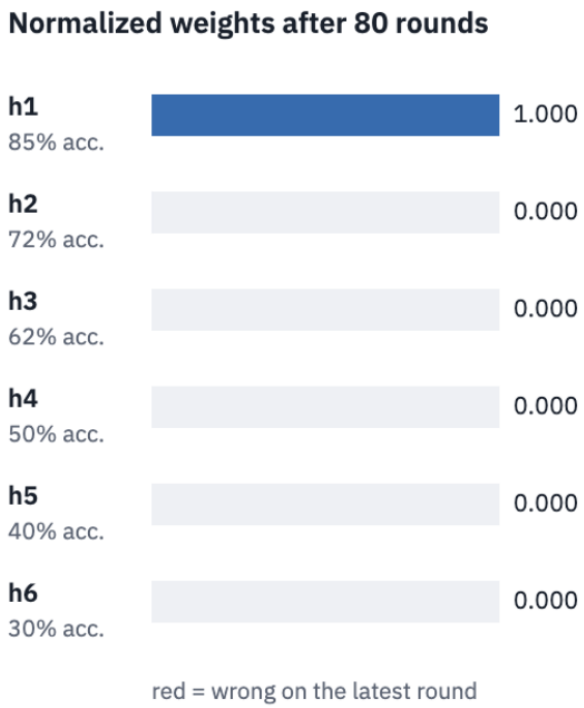
Given six experts with accuracies 85% down to 30% that predict a stream of 80 labels.

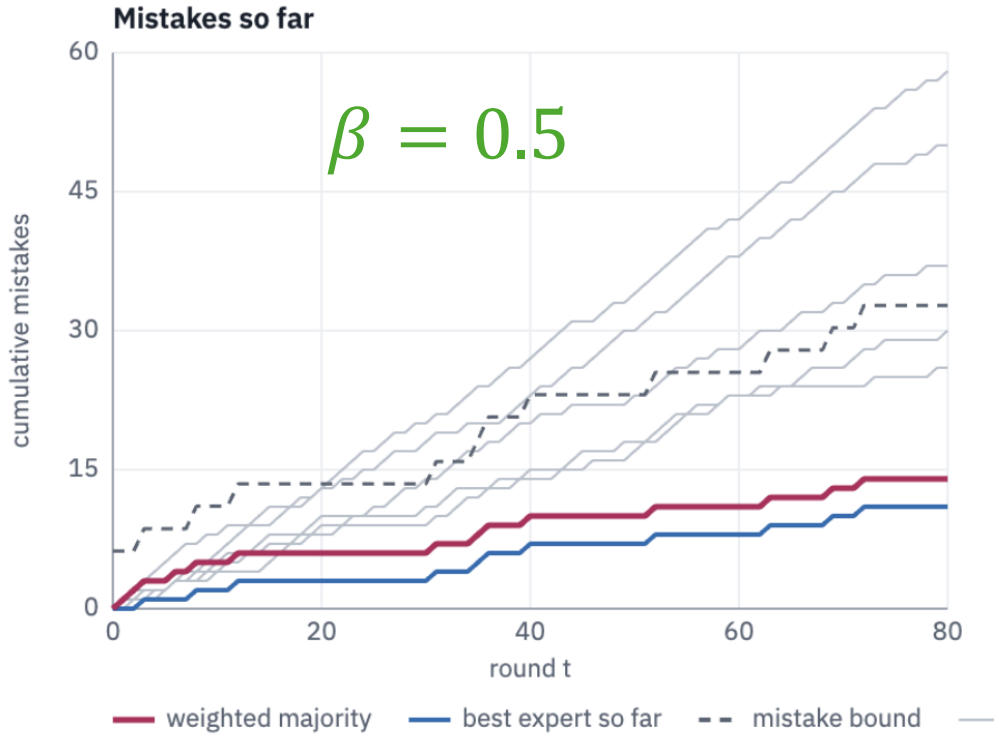
Weights of experts that err are multiplied by β.

The weight mass migrate to h_1.

A small β trusts the current leader quickly

β near 1 forgets slowly.





Round 80: weighted vote +1 gets 0.00, −1 gets 1.00 → predict −1; truth −1. Totals: WM 14 mistakes, best expert 11, bound 32.7.

Weighted majority:
what if no expert is perfect?

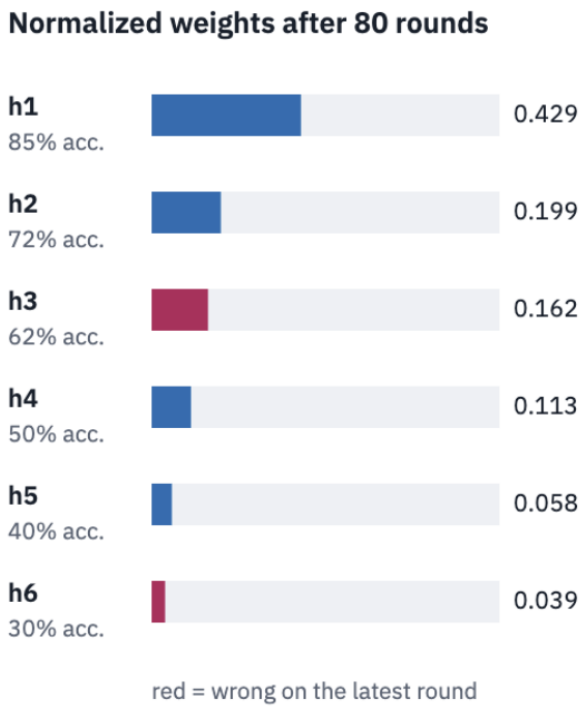
Given six experts with accuracies 85% down to 30% that predict a stream of 80 labels.

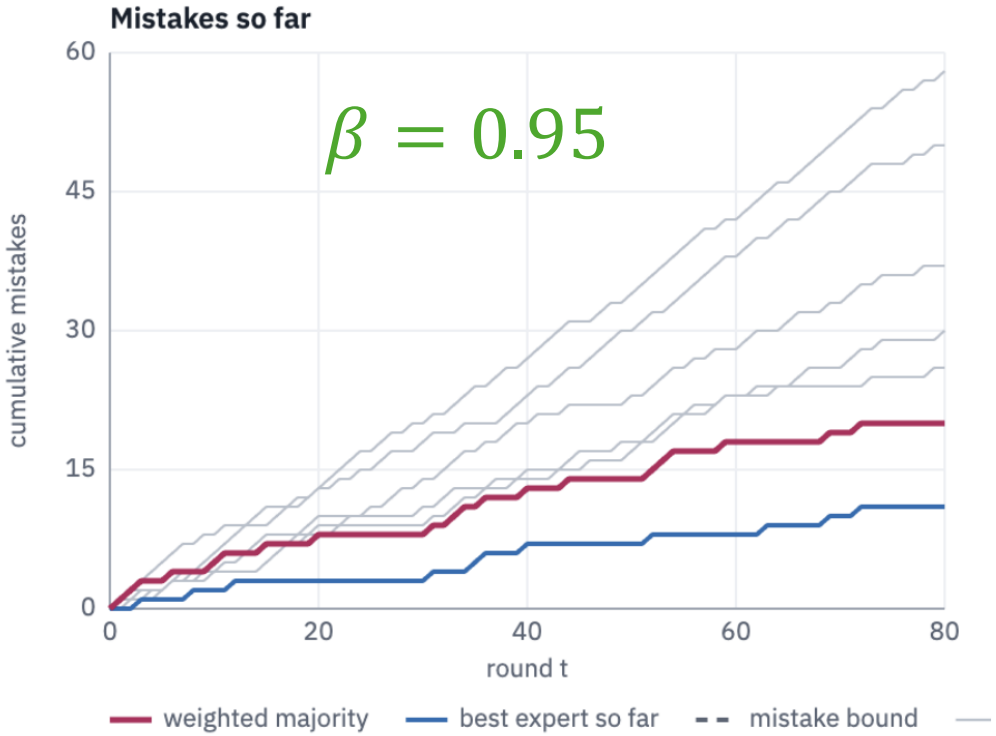
Weights of experts that err are multiplied by β.

The weight mass migrate to h_1.

A small β trusts the current leader quickly

β near 1 forgets slowly.

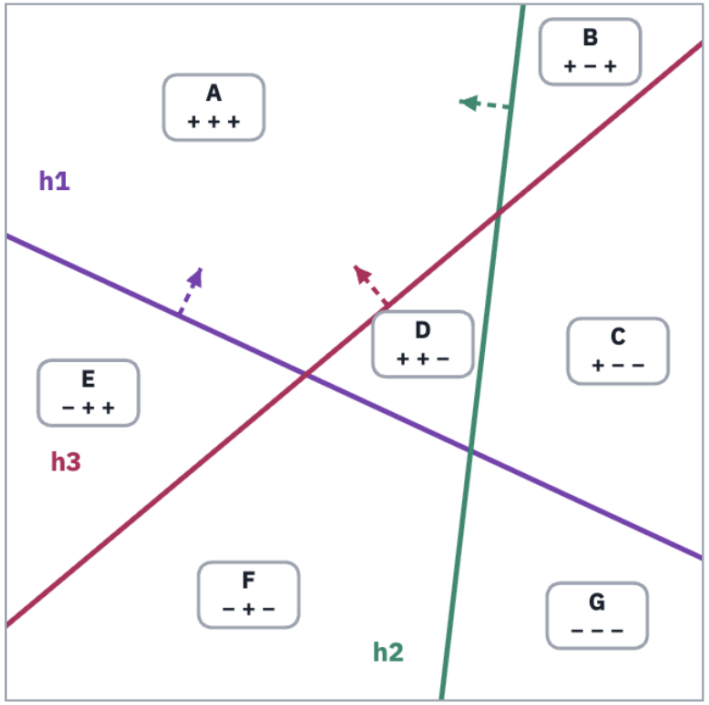


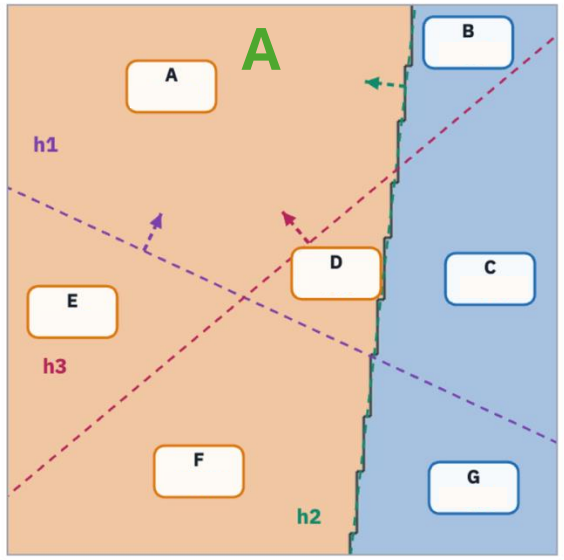


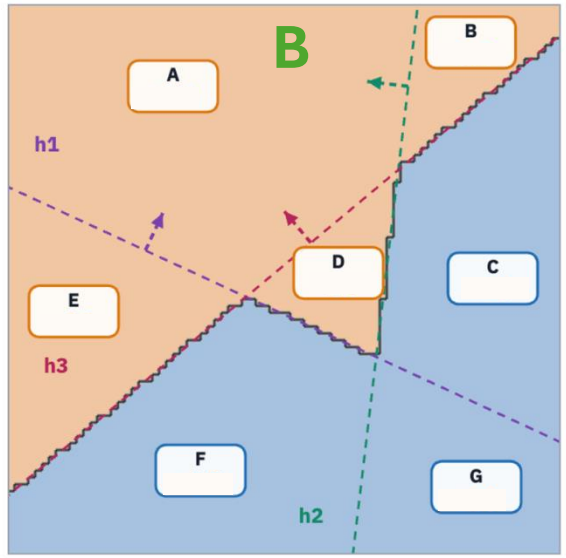
Round 80: weighted vote +1 gets 0.21, −1 gets 0.79 → predict −1; truth −1. Totals: WM 20 mistakes, best expert 11, bound 93.1.

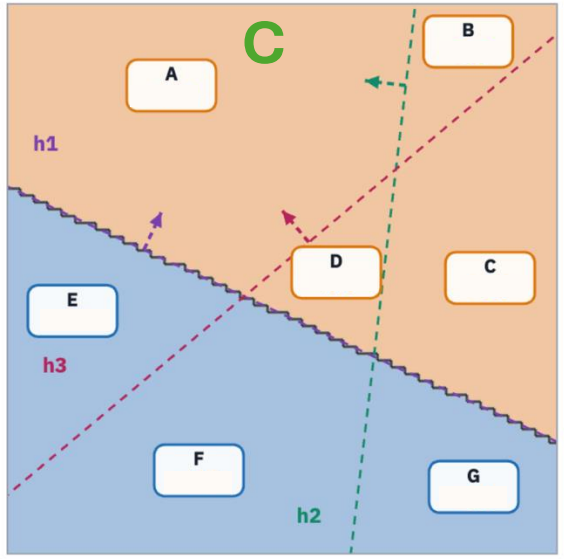
Weighted majority:

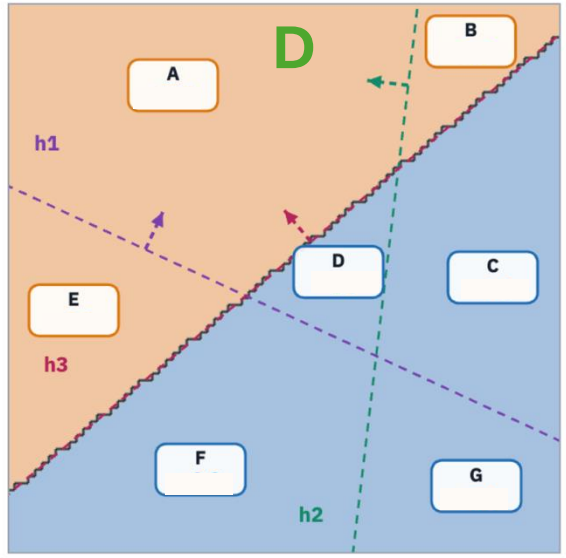
What boundary does a vote of linear classifiers with equal weights $\alpha_1 = \alpha_2 = \alpha_3 = 1$ draw?







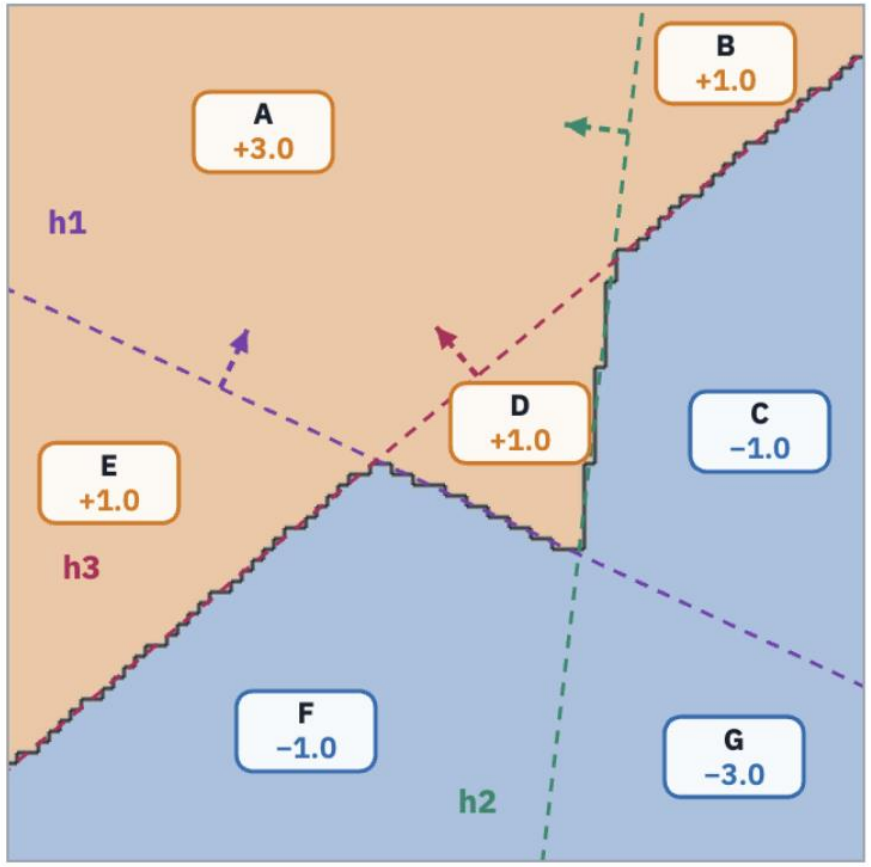




Weighted majority:

what boundary does a vote of linear classifiers draw?

$$\alpha_1 = 1$$
$$\alpha_2 = 1$$
$$\alpha_3 = 1$$



Weighted vote in each region

	h_1	h_2	h_3	$\Sigma\, \alpha_k\, h_k(x)$	label
A	+1	+1	+1	1.0 + 1.0 + 1.0 = +3.0	+1
B	+1	−1	+1	1.0 − 1.0 + 1.0 = +1.0	+1
C	+1	−1	−1	1.0 − 1.0 − 1.0 = −1.0	−1
D	+1	+1	−1	1.0 + 1.0 − 1.0 = +1.0	+1
E	−1	+1	+1	−1.0 + 1.0 + 1.0 = +1.0	+1
F	−1	+1	−1	−1.0 + 1.0 − 1.0 = −1.0	−1
G	−1	−1	−1	−1.0 − 1.0 − 1.0 = −3.0	−1

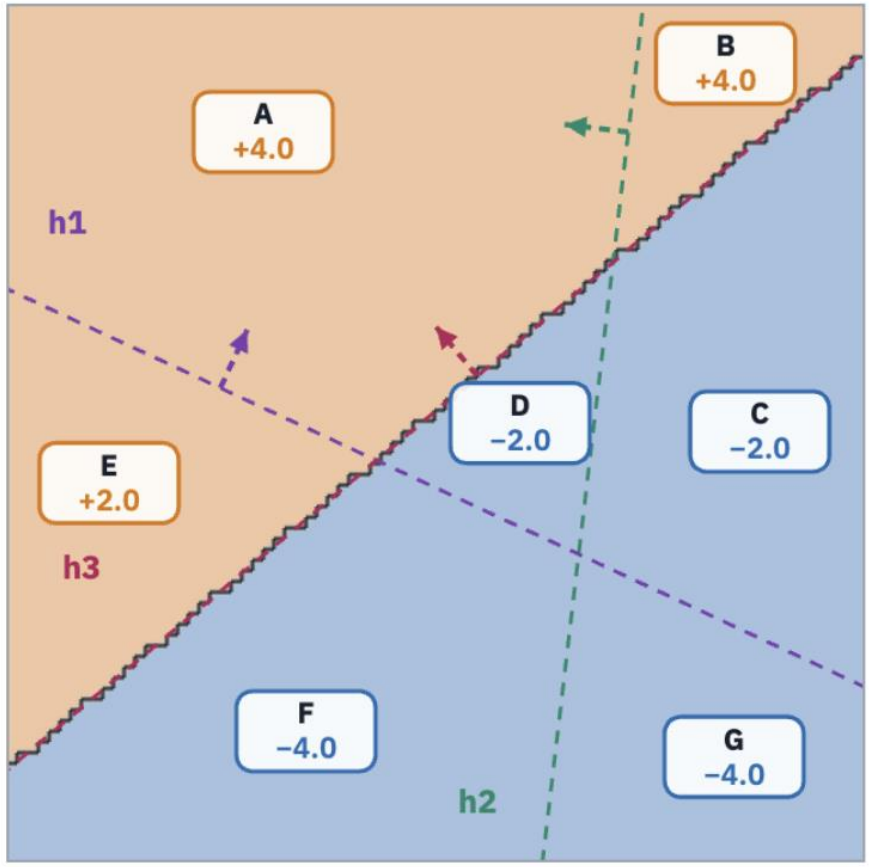
Regions predicted +1: A, B, D, E.

No single α_k outweighs the other two: the vote is a genuine majority, and the boundary is not a line.

Weighted majority:

what boundary does a vote of linear classifiers draw?

$$\alpha_1 = 1$$
$$\alpha_2 = 0$$
$$\alpha_3 = 3$$



Weighted vote in each region

	h_1	h_2	h_3	$\Sigma\, \alpha_k\, h_k(x)$	label
A	+1	+1	+1	1.0 + 0.0 + 3.0 = +4.0	+1
B	+1	−1	+1	1.0 − 0.0 + 3.0 = +4.0	+1
C	+1	−1	−1	1.0 − 0.0 − 3.0 = −2.0	−1
D	+1	+1	−1	1.0 + 0.0 − 3.0 = −2.0	−1
E	−1	+1	+1	−1.0 + 0.0 + 3.0 = +2.0	+1
F	−1	+1	−1	−1.0 + 0.0 − 3.0 = −4.0	−1
G	−1	−1	−1	−1.0 − 0.0 − 3.0 = −4.0	−1

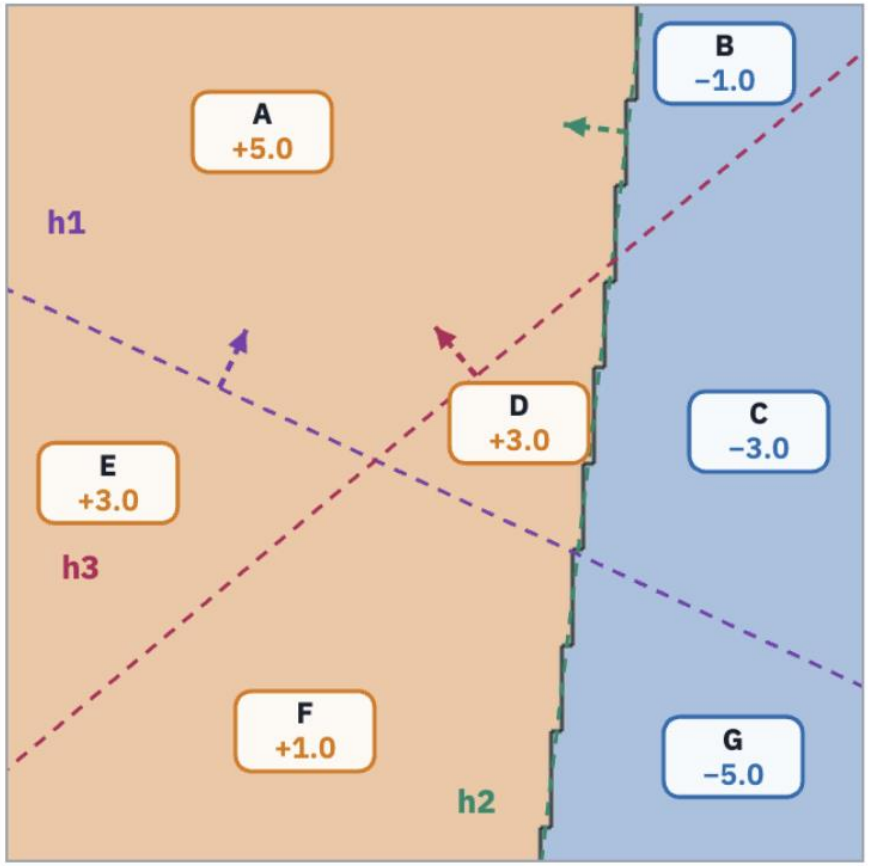
Regions predicted +1: A, B, E.

α_3 outweighs the other two combined, so h_3 alone decides.

Weighted majority:

what boundary does a vote of linear classifiers draw?

$$\alpha_1 = 1$$
$$\alpha_2 = 3$$
$$\alpha_3 = 0$$



Weighted vote in each region

	h_1	h_2	h_3	$\Sigma\, \alpha_k\, h_k(x)$	label
A	+1	+1	+1	1.0 + 3.0 + 1.0 = +5.0	+1
B	+1	−1	+1	1.0 − 3.0 + 1.0 = −1.0	−1
C	+1	−1	−1	1.0 − 3.0 − 1.0 = −3.0	−1
D	+1	+1	−1	1.0 + 3.0 − 1.0 = +3.0	+1
E	−1	+1	+1	−1.0 + 3.0 + 1.0 = +3.0	+1
F	−1	+1	−1	−1.0 + 3.0 − 1.0 = +1.0	+1
G	−1	−1	−1	−1.0 − 3.0 − 1.0 = −5.0	−1

Regions predicted +1: A, D, E, F.

α_2 outweighs the other two combined, so h_2 alone decides.

AdaBoost:
what are weak and strong learners?

Weighted majority assumes the classifiers are learned beforehand and learns only their vote weights.

Boosting learns both the classifiers h_t and their weights α_t together.

Weak learner

Returns a hypothesis only slightly better than random guessing: weighted error $\varepsilon \leq \frac{1}{2} - \gamma$, on **any** distribution over the data.

Strong learner

Returns a hypothesis with **arbitrarily low error**.

Question (Kearns & Valiant, 1988): Can weak learners be efficiently combined into a strong learner?

Schapire (1990) proved **yes**.

Freund & Schapire's **AdaBoost** (1995) made it practical.

Yes. AdaBoost trains each new weak learner on data reweighted toward the previous mistakes and gives it a vote weight based on its error.

AdaBoost:

how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

$D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
for $t = 1, \ldots, T$ **do**
  $h_t \leftarrow$ train the weak learner on distribution D_t
  $\varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
  for $i = 1, \ldots, N$ **do**
    $D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
  end for
end for
return $H(x) = \operatorname{sign}\big(\sum_{t=1}^{T} \alpha_t h_t(x)\big)$

AdaBoost keeps a weight on every training example.
In each round it trains one weak learner on the weighted data, measures how good that learner is, and then shifts weight *toward the examples it got wrong*.
The next learner is therefore forced to concentrate on the cases the ensemble still struggles with.
At the end, all the learners vote, and better learners get a bigger say

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad h_t \leftarrow$ train the weak learner on distribution D_t
4: $\quad \varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
5: $\quad \alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
6: $\quad$ **for** $i = 1, \ldots, N$ **do**
7: $\quad\quad D_{t+1}(i) \leftarrow D_t(i) \exp\big(- \alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
8: $\quad$ **end for**
9: **end for**
10: **return** $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Start with equal weights $D_1(i) = 1/N$

Before any learner has been trained, no example is harder than another.

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

$D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
for $t = 1, \ldots, T$ **do**
    $h_t \leftarrow$ train the weak learner on distribution D_t
    $\varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
    $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
    for $i = 1, \ldots, N$ **do**
        $D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
    end for
end for
return $H(x) = \operatorname{sign}\big(\sum_{t=1}^{T} \alpha_t h_t(x)\big)$

Each round adds one weak learner to the ensemble.

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

$D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
for $t = 1, \ldots, T$ **do**
    $h_t \leftarrow$ train the weak learner on distribution D_t
    $\varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
    $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
    for $i = 1, \ldots, N$ **do**
        $D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
    end for
end for
return $H(x) = \operatorname{sign}\big(\sum_{t=1}^{T} \alpha_t h_t(x)\big)$

The learner minimizes *weighted* error, so heavy examples matter more.
It only needs to beat random guessing.

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad h_t \leftarrow$ train the weak learner on distribution D_t
4: $\quad \varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
5: $\quad \alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
6: $\quad$ **for** $i = 1, \ldots, N$ **do**
7: $\quad\quad D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
8: $\quad$ **end for**
9: **end for**
10: **return** $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Weighted error ϵ_t = total weight of the examples h_t gets wrong.

Measures how good h_t is *on the examples that currently matter*, not on the original uniform data.

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad h_t \leftarrow$ train the weak learner on distribution D_t
4: $\quad \varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
5: $\quad \alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
6: $\quad$ **for** $i = 1, \ldots, N$ **do**
7: $\quad\quad D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
8: $\quad$ **end for**
9: **end for**
10: **return** $H(x) = \operatorname{sign}\big(\sum_{t=1}^{T} \alpha_t h_t(x)\big)$

Small error gives a large positive α (a trusted voter)
Error near ½ gives $\alpha \approx 0$ (no say).

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad h_t \leftarrow$ train the weak learner on distribution D_t
4: $\quad \varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
5: $\quad \alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
6: $\quad$ **for** $i = 1, \ldots, N$ **do**
7: $\quad\quad D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
8: $\quad$ **end for**
9: **end for**
10: **return** $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Multiply each weight by $e^{-\alpha_t y_i h_t(x_i)}$, then divide by Z_t.

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

$D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
for $t = 1, \ldots, T$ **do**
  $h_t \leftarrow$ train the weak learner on distribution D_t
  $\varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
  for $i = 1, \ldots, N$ **do**
    $D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
  end for
end for
return $H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

A weighted majority vote of all T learners, but with weights learned from the errors.

AdaBoost:
how does it work?

AdaBoost (Freund & Schapire)

Require: $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ with $y^{(i)} \in \{-1, +1\}$; rounds T; a weak learner

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad h_t \leftarrow$ train the weak learner on distribution D_t
4: $\quad \varepsilon_t \leftarrow \sum_i D_t(i)\, \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$
5: $\quad \alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
6: $\quad$ **for** $i = 1, \ldots, N$ **do**
7: $\quad\quad D_{t+1}(i) \leftarrow D_t(i) \exp\big(-\alpha_t\, y^{(i)} h_t(x^{(i)})\big)/Z_t$
8: $\quad$ **end for**
9: **end for**
10: **return** $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

vote weight

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$$

reweighting

$$D_{t+1}(i) \propto D_t(i) \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \text{ (shrink)} \\ e^{+\alpha_t} & \text{if } h_t(x_i) \neq y_i \text{ (grow)} \end{cases}$$

Reweight toward mistakes; better learners get a bigger vote; weak is enough.

AdaBoost:

how is Z_t calculated?

$$
\begin{aligned}
Z_t &= \sum_i D_t(i)\, e^{-\alpha_t y_i h_t(x_i)} \\
&= (1-\varepsilon_t)\, e^{-\alpha_t} + \varepsilon_t\, e^{\alpha_t} \\
&= 2\sqrt{\varepsilon_t(1-\varepsilon_t)}
\end{aligned}
$$

Z_t = sum of all updated weights **before** normalizing, so dividing by it makes them sum to 1.

Correct examples hold weight $1-\varepsilon_t$, scaled by $e^{(-\alpha)}$

Mistakes hold ε_t, scaled by $e^{(+\alpha)}$. Substituting α_t gives the last line.

Round 1 of the toy example

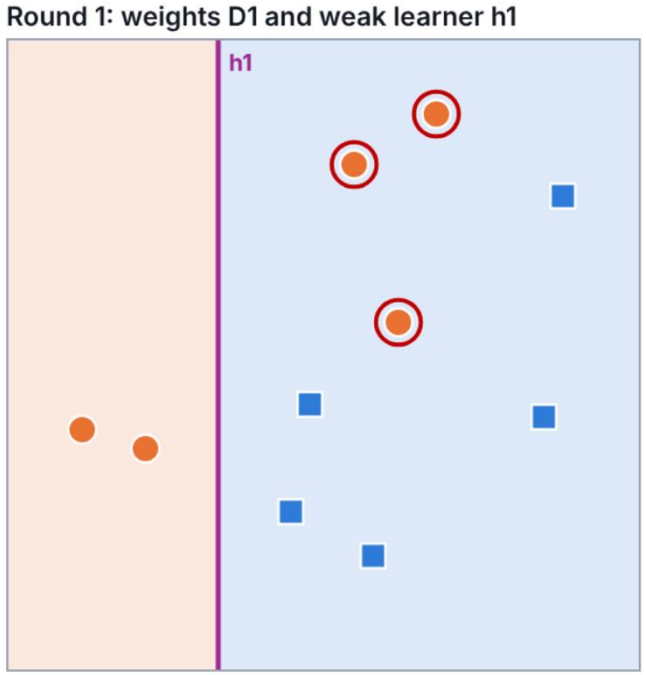
$$Z_1 = 0.7 \times 0.655 + 0.3 \times 1.528 = 0.917 = 2\sqrt{0.3 \times 0.7}$$

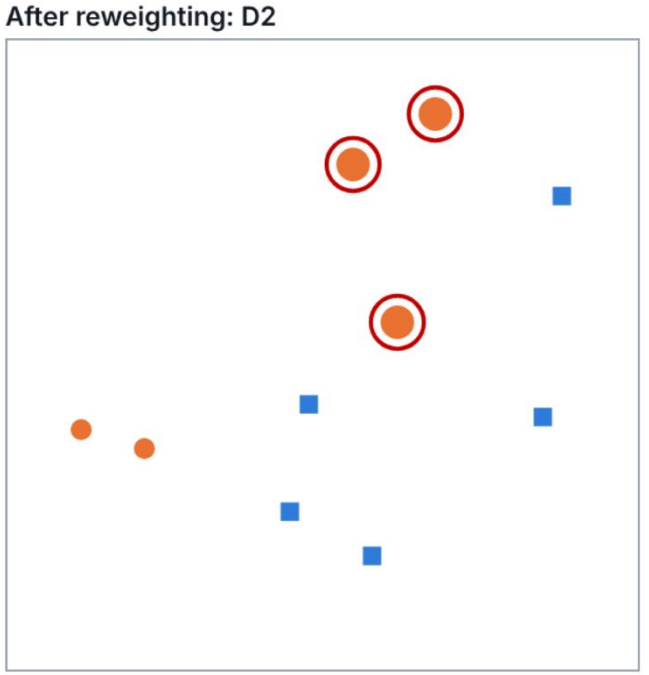
$Z_t < 1$ whenever $\varepsilon_t < ½$, and the product of the Z_t is exactly AdaBoost's training-error bound.

AdaBoost:

$$\varepsilon_1 = 0.30, \qquad \alpha_1 = \tfrac{1}{2}\ln\tfrac{0.7}{0.3} = 0.42$$

- Uniform weights 0.1
- The best stump misses 3 points
- Mistakes ×1.53, correct points ×0.65, then ÷ $Z_1 = 0.917$
- Afterwards the mistakes hold **exactly half** the weight



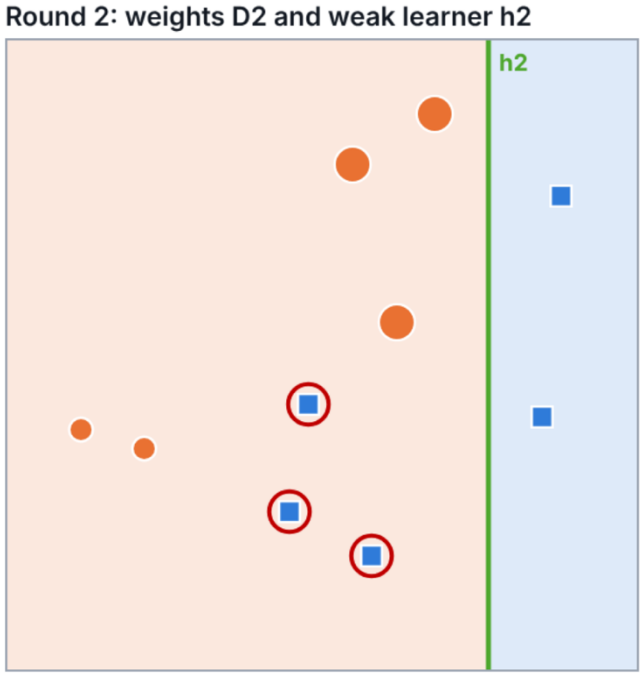


t	stump h_t (+1 side)	ε_t	α_t	Z_t	$2\sqrt{(\varepsilon_t(1-\varepsilon_t))}$
1	**vertical, $x_1 < 0.335$**	**0.300**	**0.424**	**0.917**	**0.917**
2	vertical, $x_1 < 0.765$	0.214	0.650	0.821	0.821
3	horizontal, $x_2 > 0.485$	0.136	0.923	0.686	0.686
final	$H = \text{sign}(\Sigma\, \alpha_t\, h_t)$, training error 0 / 10			$\prod Z_t = 0.516$	

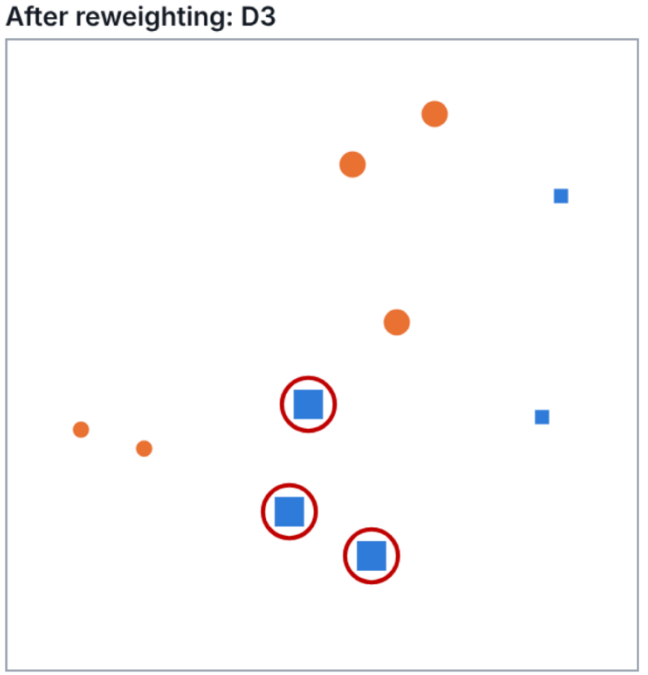
AdaBoost:

$$\varepsilon_2 = 3 \times \tfrac{1}{14} = 0.21, \qquad \alpha_2 = 0.65$$

- Heavy points (1/6 each) must be right now
- The new stump misses 3 **light** points (1/14 each)



ε2 = 0.21 α2 = ½ ln((1−ε)/ε) = 0.65



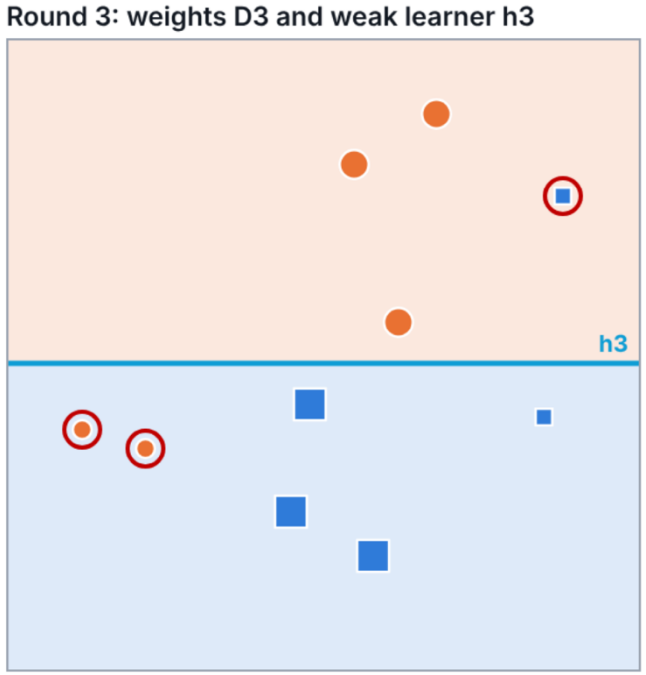
ringed = misclassified by h2 · point area ∝ weight

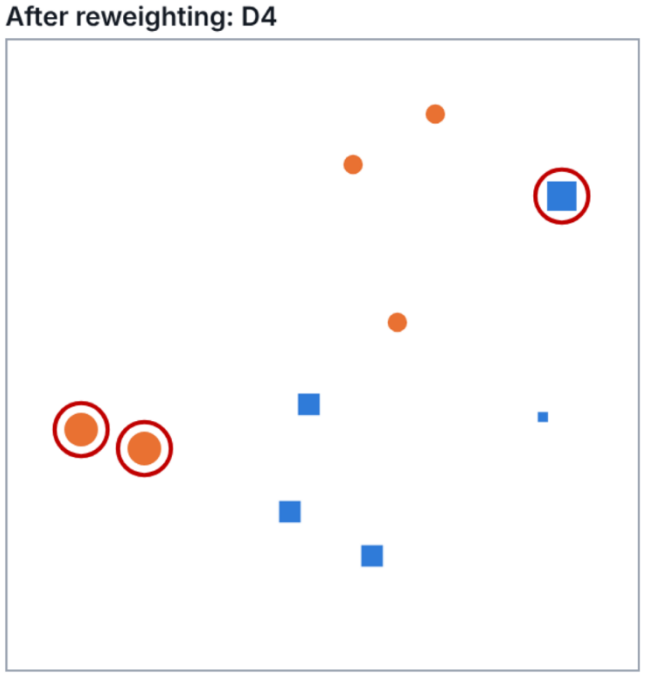
t	stump h_t (+1 side)	ε_t	α_t	Z_t	$2\sqrt{(\varepsilon_t(1-\varepsilon_t))}$
1	vertical, $x_1 < 0.335$	0.300	0.424	0.917	0.917
2	**vertical, $x_1 < 0.765$**	**0.214**	**0.650**	**0.821**	**0.821**
3	horizontal, $x_2 > 0.485$	0.136	0.923	0.686	0.686
final	$H = \text{sign}(\Sigma\, \alpha_t\, h_t)$, training error 0 / 10			$\prod Z_t = 0.516$	

AdaBoost:

$$\varepsilon_3 = 3 \times \tfrac{1}{22} = 0.14, \qquad \alpha_3 = 0.92$$

- Weights now favour points both earlier stumps found hard
- A horizontal stump





ε3 = 0.14 α3 = ½ ln((1−ε)/ε) = 0.92

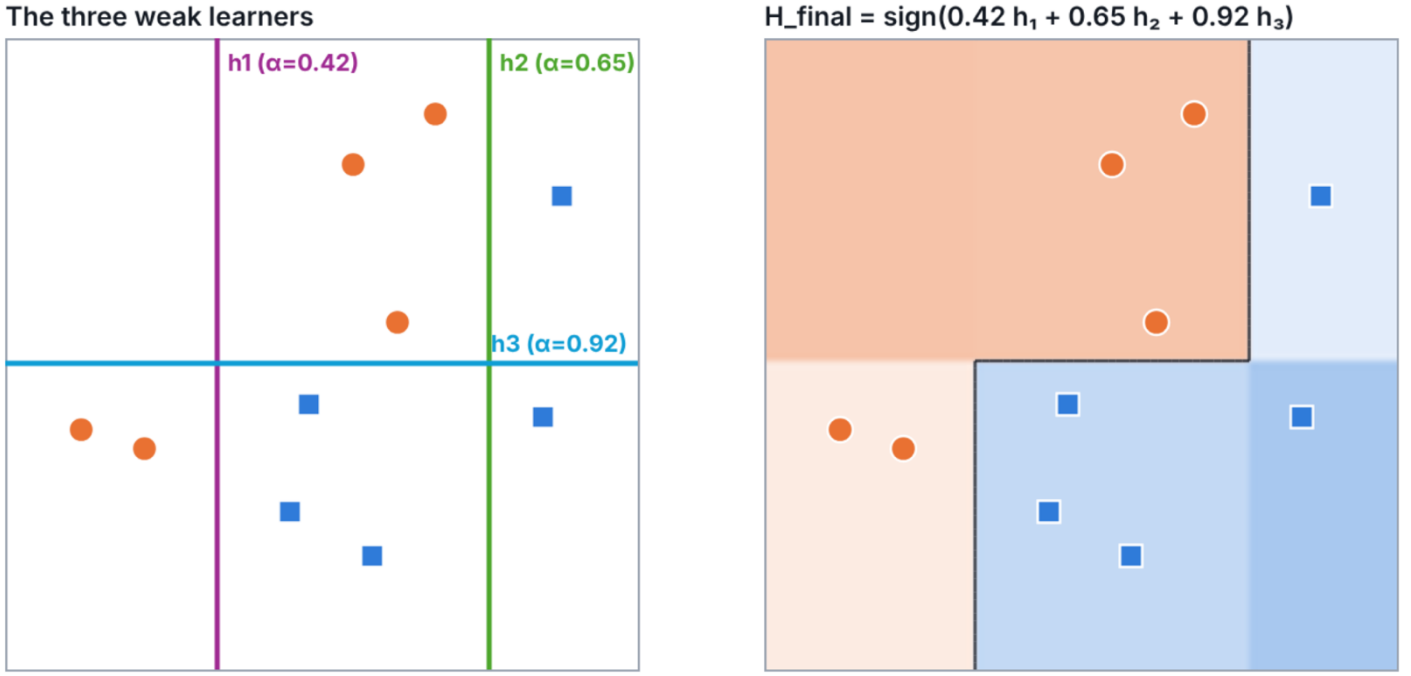
ringed = misclassified by h3 · point area ∝ weight

t	stump h_t (+1 side)	ε_t	α_t	Z_t	$2\sqrt{(\varepsilon_t(1-\varepsilon_t))}$
1	vertical, $x_1 < 0.335$	0.300	0.424	0.917	0.917
2	vertical, $x_1 < 0.765$	0.214	0.650	0.821	0.821
3	**horizontal, $x_2 > 0.485$**	**0.136**	**0.923**	**0.686**	**0.686**
final	$H = \text{sign}(\Sigma\, \alpha_t\, h_t)$, training error 0 / 10			$\prod Z_t = 0.516$	

AdaBoost:

$$H(x) = \text{sign}\left(0.42\,h_1 + 0.65\,h_2 + 0.92\,h_3\right)$$

- Where h_1 and h_2 agree they outvote h_3 (1.07 > 0.92)
- Where they disagree they cancel, and h_3 decides
- **0 training errors**: a staircase no single stump can draw



No single stump separates the data; the weighted vote of three does (0 training errors).

t	stump h_t (+1 side)	ε_t	α_t	Z_t	$2\sqrt{(\varepsilon_t(1-\varepsilon_t))}$
1	vertical, $x_1 < 0.335$	0.300	0.424	0.917	0.917
2	vertical, $x_1 < 0.765$	0.214	0.650	0.821	0.821
3	horizontal, $x_2 > 0.485$	0.136	0.923	0.686	0.686
final	**H = sign(Σ α_t h_t), training error 0 / 10**			**∏ Z_t = 0.516**	

AdaBoost:

how do α and the weights respond to a learner's error?

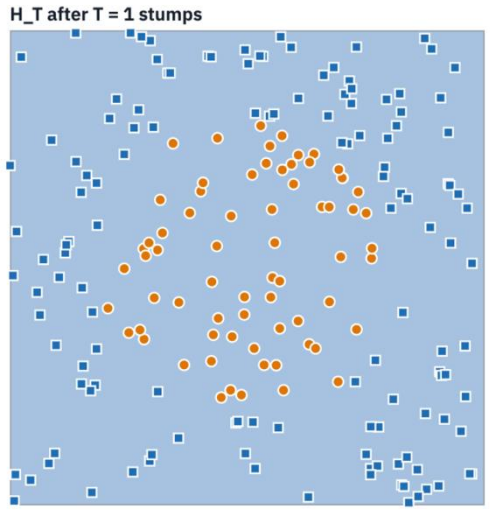


	effect on α_t	effect on weights D_{t+1}
h_t has low error ($\varepsilon_t \to 0$)	large and positive: a trusted voter	its few mistakes are boosted sharply
h_t has error near ½	≈ 0: barely votes	weights barely change
h_t has high error ($\varepsilon_t > \frac{1}{2}$)	negative: its vote is flipped	(rare in practice; flipping h_t gives a good learner)
$h_t(x_i) = y_i$ (correct)	–	$D_{t+1}(i) \propto D_t(i)\, e^{-\alpha_t}$ decreases
$h_t(x_i) \neq y_i$ (mistake)	–	$D_{t+1}(i) \propto D_t(i)\, e^{+\alpha_t}$ increases

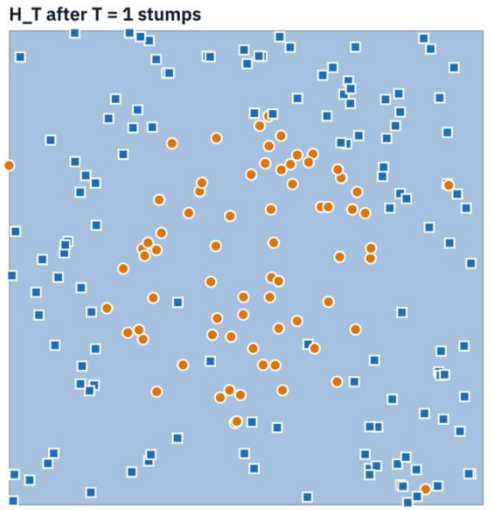
AdaBoost:

what happens on a harder problem, and with noisy labels?

clean labels, T = 1



5% flipped labels, T = 1



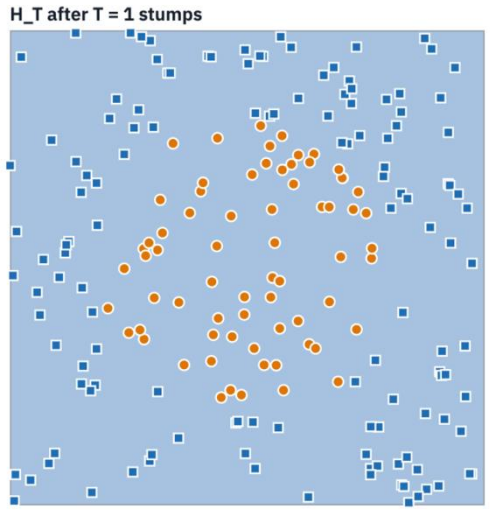
A circle needs **many** axis-aligned stumps. **Clean:** test error keeps falling for a while after training error reaches 0.

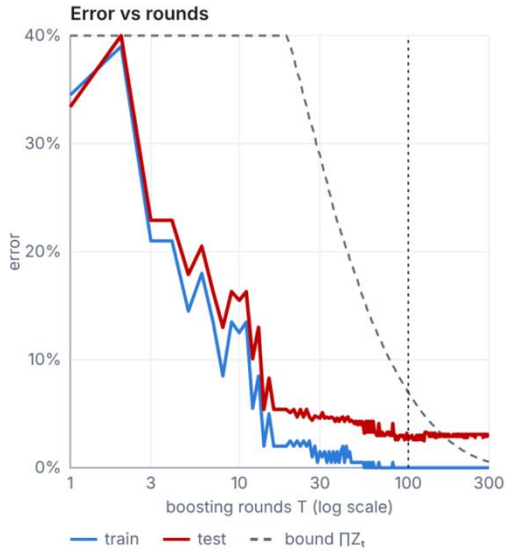
Noisy: the exponential loss piles weight onto mislabeled points and test error creeps back up.

AdaBoost:

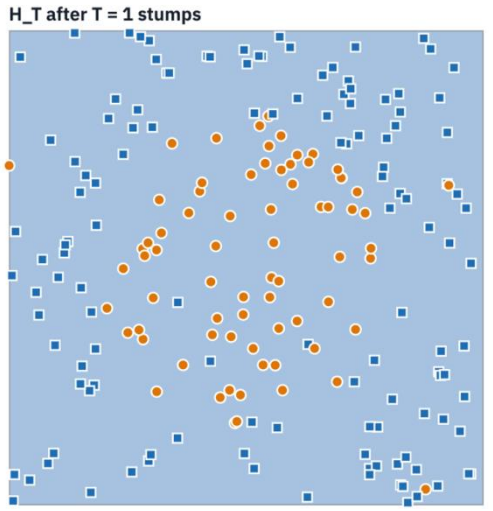
what happens on a harder problem, and with noisy labels?

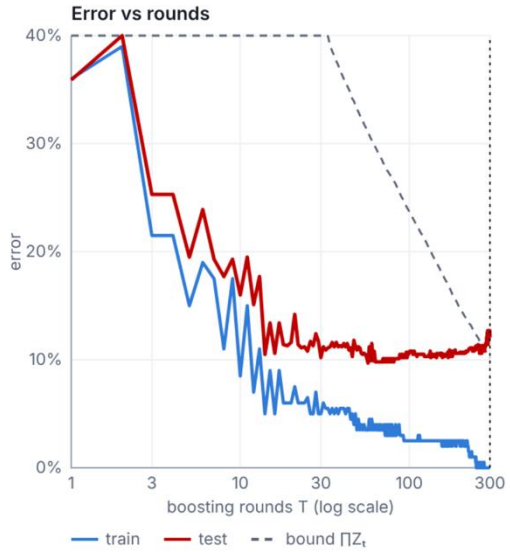
clean labels, T = 100





5% flipped labels, T = 300





A circle needs **many** axis-aligned stumps. **Clean:** test error keeps falling for a while after training error reaches 0.

Noisy: the exponential loss piles weight onto mislabeled points and test error creeps back up.

AdaBoost theory:
why does AdaBoost work?

It greedily minimizes exponential loss, which bounds the training error and drives it down exponentially fast. Growing margins explain why test error often keeps improving.

That AdaBoost's training error after T rounds is at most a product of the normalizers Z_t , and that this product shrinks exponentially in T. The proof has 6 steps:

AdaBoost theory:

why does training error fall exponentially? (steps 1–3)

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x), \qquad H(x) = \operatorname{sign}(f(x))$$

y f(x) > 0 exactly when H is correct

Step 1. Unroll the weight update (start from $1/N$, apply T times):

$$D_{T+1}(i) = \frac{1}{N} \cdot \frac{e^{-\alpha_1 y_i h_1(x_i)}}{Z_1} \cdot \frac{e^{-\alpha_2 y_i h_2(x_i)}}{Z_2} \cdots \frac{e^{-\alpha_T y_i h_T(x_i)}}{Z_T}.$$

$$D_{T+1}(i) = \frac{1}{N} \prod_{t=1}^{T} \frac{e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

Step 2. Collect the exponents: the sum in the exponent is exactly $y_i f(x_i)$,so

$$D_{T+1}(i) = \frac{1}{N} \frac{e^{-y_i \sum_t \alpha_t h_t(x_i)}}{\prod_t Z_t} = \frac{1}{N} \frac{e^{-y_i f(x_i)}}{\prod_t Z_t},$$

$$\frac{1}{N} e^{-y_i f(x_i)} = D_{T+1}(i) \prod_t Z_t$$

Step 3. Bound each mistake: if H is wrong, $y_i f(x_i) \le 0$, so the exponential is ≥ 1:

$$\mathbf{1}[H(x_i) \neq y_i] \le e^{-y_i f(x_i)}$$

AdaBoost theory:

why does training error fall exponentially? (steps 4–6)

Step 4. Average over the training set and use Step 2 (the weights sum to 1):

$$\frac{1}{N}\sum_i \mathbf{1}[H(x_i) \neq y_i] \leq \sum_i \tfrac{1}{N} e^{-y_i f(x_i)} = \sum_i D_{T+1}(i) \prod_t Z_t = \prod_t Z_t$$

Step 5. Write each Z_t with the edge $\gamma_t = \frac{1}{2} - \varepsilon_t$, since $\varepsilon_t(1 - \varepsilon_t) = \frac{1}{4} - \gamma_t^2$:

$$Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)} = \sqrt{1-4\gamma_t^2}, \qquad \gamma_t = \tfrac{1}{2} - \varepsilon_t$$

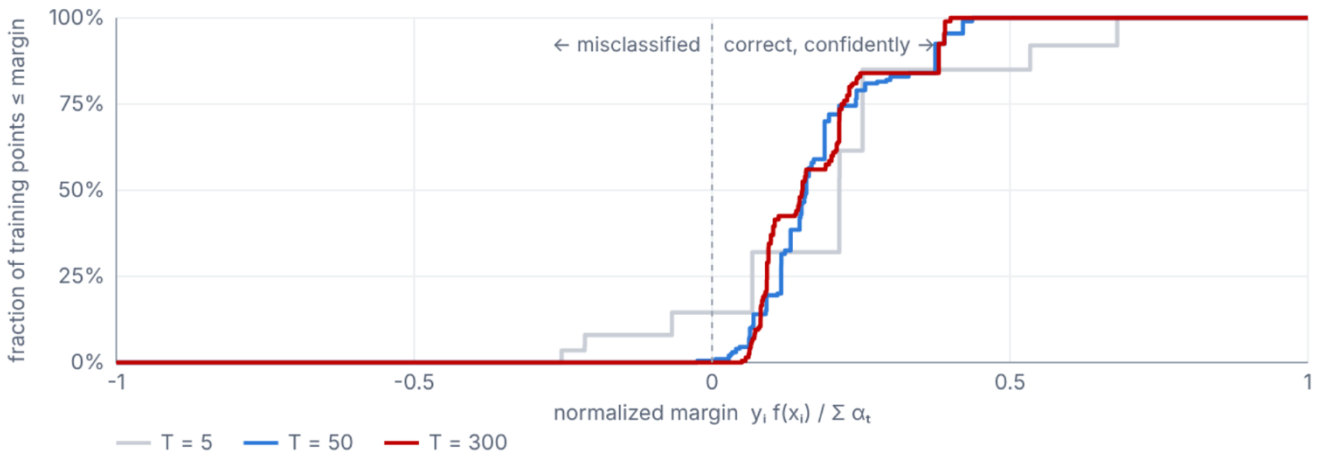
Step 6. Use $1 + z \leq e^z$ with $z = -4\gamma_t^2$:

$$\text{training error} \leq \prod_t \sqrt{1-4\gamma_t^2} \leq \prod_t e^{-2\gamma_t^2} = \exp\Big(-2\sum_t \gamma_t^2\Big)$$

Weak is enough: with edge ≥ γ every round, training error ≤ e^(−2γ²T). It hits 0 once the bound drops below 1/N, i.e. for T > ln N / (2γ²): about 346 rounds for 60%-accurate learners on 1,000 points.

AdaBoost theory:

why doesn't it overfit as the rounds increase?



$$P\big[yf(x) \le 0\big] \;\le\; \hat{P}\big[yf(x) \le \theta\big] + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2}}\right)$$

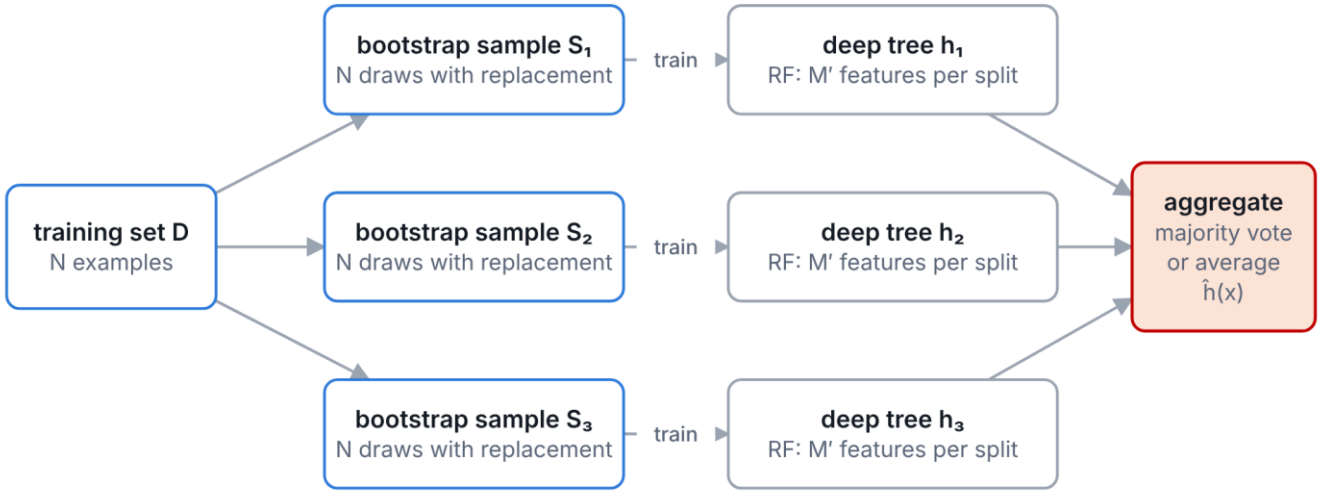
- Old bound $\tilde{O}(\sqrt{(Td/N)})$ predicted overfitting as T grows; experiments disagreed
- The **margin** measures how decisive the vote is:

$$\text{margin}(x_i) = \frac{y_i f(x_i)}{\sum_t |\alpha_t|} \in [-1, 1]$$

- Extra rounds keep pushing margins up after every margin is positive
- **The margin bound does not depend on T**

Schapire, Freund, Bartlett & Lee (1998), Boosting the margin. d = VC dimension of the weak learner; holds for every $\theta > 0$.

Bagging:
which parts of the data can we resample?



Sample bagging

each model sees N rows drawn with replacement

Feature bagging

random subspace method: all rows, a random subset of features

Random forests

bootstrap rows and a fresh random feature subset at every split

Boosting reweights **one** training set serially

Bagging builds **many** training sets in parallel and averages the models.

Breiman (1996), Bagging predictors · Ho (1998), The random subspace method.

Bagging:

how do sample bagging and feature bagging work?

SampleBagging($\mathcal{D}$, T, S)

Require: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, models T, sample size S (usually N)

for $t = 1, \ldots, T$ **do**
  for $s = 1, \ldots, S$ **do**
    $i_s \sim \text{Uniform}(1, \ldots, N)$ ▷ *with replacement*
  end for
  $S_t \leftarrow \{(x^{(i_s)}, y^{(i_s)})\}_{s=1}^{S}$ ▷ *bootstrap sample*
  $h_t \leftarrow \text{train}(S_t)$
end for
return $\hat{h}(x) = \text{aggregate}(h_1, \ldots, h_T)$

Training set D

#	x_1	x_2	x_3	x_4	y
1	1	0	1	0	+
2	0	1	1	1	−
3	1	1	0	0	+
4	0	0	1	1	−
5	1	0	0	1	+

Projected data for h1: {x_1, x_2}

#	x_1	x_2	y
1	1	0	+
2	0	1	−
3	1	1	+
4	0	0	−
5	1	0	+

Projected data for h2: {x_1, x_3}

#	x_1	x_3	y
1	1	1	+
2	0	1	−
3	1	0	+
4	0	1	−
5	1	0	+

Projected data for h3: {x_1, x_2}

#	x_1	x_2	y
1	1	0	+
2	0	1	−
3	1	1	+
4	0	0	−
5	1	0	+

Every model sees all 5 examples but only a random 2 of the 4 features (the random subspace method).

Bagging:

how do sample bagging and feature bagging work?

SampleBagging($\mathcal{D}$, T, S)

Require: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, models T, sample size S (usually N)

for $t = 1, \ldots, T$ **do**
  for $s = 1, \ldots, S$ **do**
    $i_s \sim \text{Uniform}(1, \ldots, N)$ ▷ *with replacement*
  end for
  $S_t \leftarrow \{(x^{(i_s)}, y^{(i_s)})\}_{s=1}^{S}$ ▷ *bootstrap sample*
  $h_t \leftarrow \text{train}(S_t)$
end for
return $\hat{h}(x) = \text{aggregate}(h_1, \ldots, h_T)$

FeatureBagging($\mathcal{D}$, T, S)

Require: $\mathcal{D}$, models T, features per model $S < M$

for $t = 1, \ldots, T$ **do**
  for $s = 1, \ldots, S$ **do**
    $m_s \sim \text{Uniform}(1, \ldots, M)$ ▷ *feature indices*
  end for
  $\tilde{x}^{(i)} \leftarrow [x^{(i)}_{m_1}, \ldots, x^{(i)}_{m_S}]^T$ for all i ▷ *project*
  $h_t \leftarrow \text{train}\big(\{(\tilde{x}^{(i)}, y^{(i)})\}_{i=1}^{N}\big)$
end for
return $\hat{h}(x) = \text{aggregate}(h_1, \ldots, h_T)$

Training set D

#	x1	x2	x3	x4	y
1	1	0	1	0	+
2	0	1	1	1	−
3	1	1	0	0	+
4	0	0	1	1	−
5	1	0	0	1	+

Bootstrap sample S1 → h1

#	x1	x2	x3	x4	y
1	1	0	1	0	+
4	0	0	1	1	−
5	1	0	0	1	+
5	1	0	0	1	+
5	1	0	0	1	+

out-of-bag: #2, #3

Bootstrap sample S2 → h2

#	x1	x2	x3	x4	y
2	0	1	1	1	−
2	0	1	1	1	−
4	0	0	1	1	−
5	1	0	0	1	+
5	1	0	0	1	+

out-of-bag: #1, #3

Bootstrap sample S3 → h3

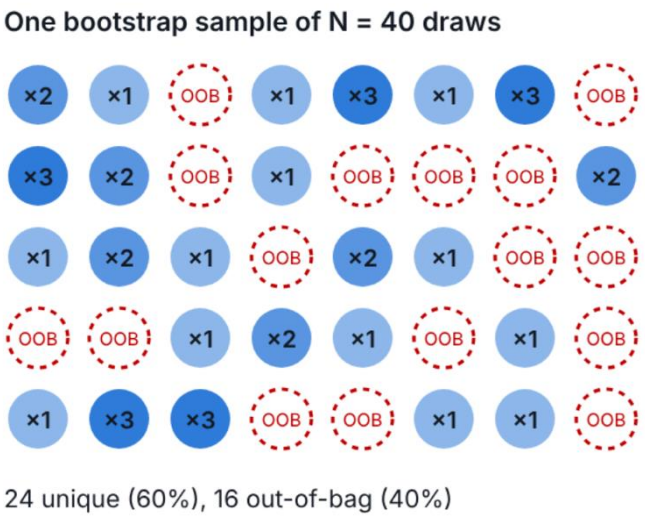
#	x1	x2	x3	x4	y
2	0	1	1	1	−
3	1	1	0	0	+
4	0	0	1	1	−
4	0	0	1	1	−
5	1	0	0	1	+

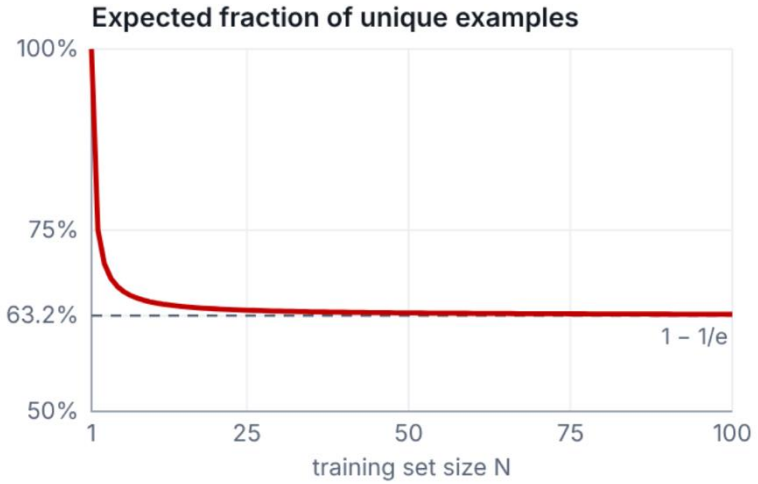
out-of-bag: #1

Shaded rows appear more than once. Each sample has 5 rows drawn with replacement, so some examples repeat and others are left out (out-of-bag).

Bagging:

how many distinct examples does each model see?





≈ 63%

distinct examples per bootstrap sample

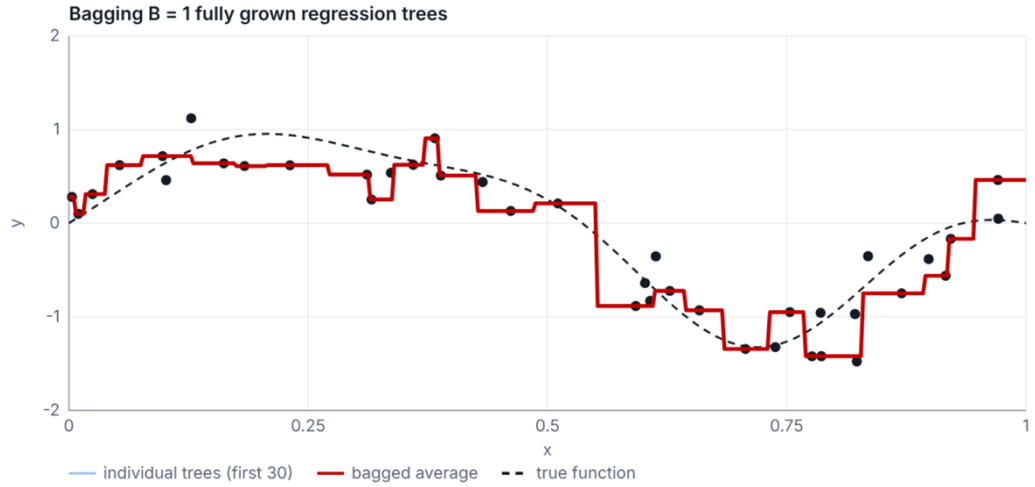
≈ 37%

out-of-bag for each model: a free validation set

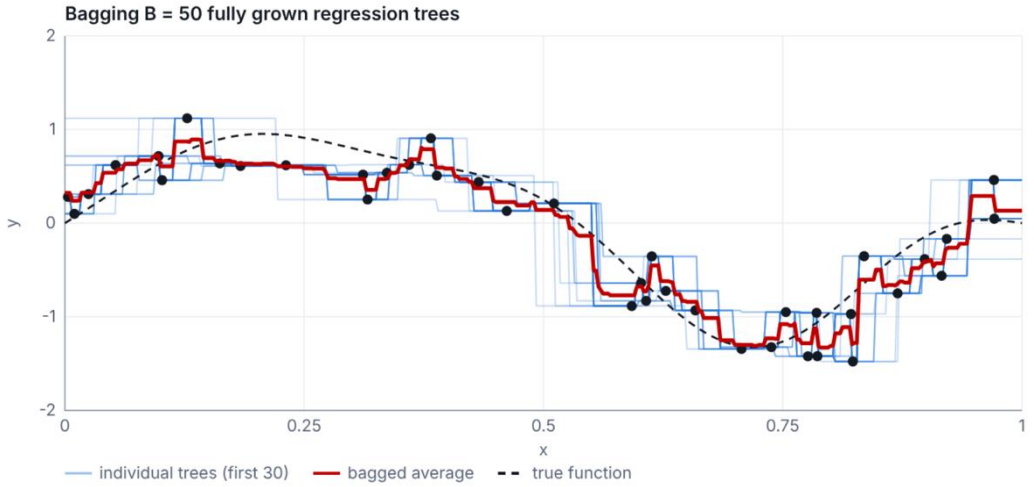
$$P(\text{an item is never drawn}) = \left(1 - \frac{1}{N}\right)^N \xrightarrow{N\to\infty} e^{-1} \approx 0.368$$

Bagging:
which models does it help, and why?

B = 1 tree



B = 50 trees

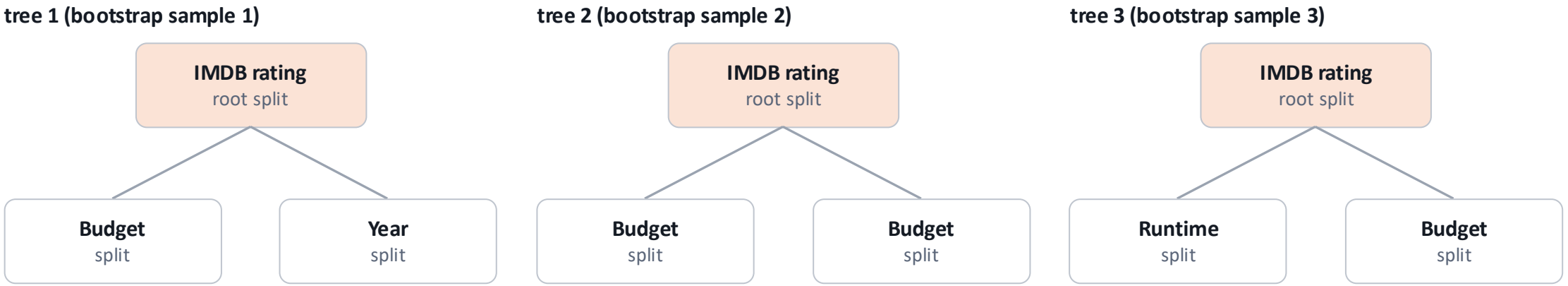


Bagging helps **unstable** learners: each deep tree interpolates its sample and wiggles, but their **average is smooth**. Bagging a stable learner (linear regression) changes almost nothing.

Random forests:

why do bagged trees still look alike?

Three bagged trees on the movie data. Each sees a different bootstrap sample, yet the strongest feature, **IMDB rating**, wins the root split every time:



Same top split → similar trees → **highly correlated** predictions (large ρ).

Averaging removes only the uncorrelated part, so the ensemble quickly hits the floor $\rho\sigma^2$.

$$\mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B} h_b(x)\right) = \frac{1}{B^2}\left[B\sigma^2 + B(B-1)\rho\sigma^2\right]$$

$$= \rho\sigma^2 + \frac{1-\rho}{B}\,\sigma^2$$

Random forests:

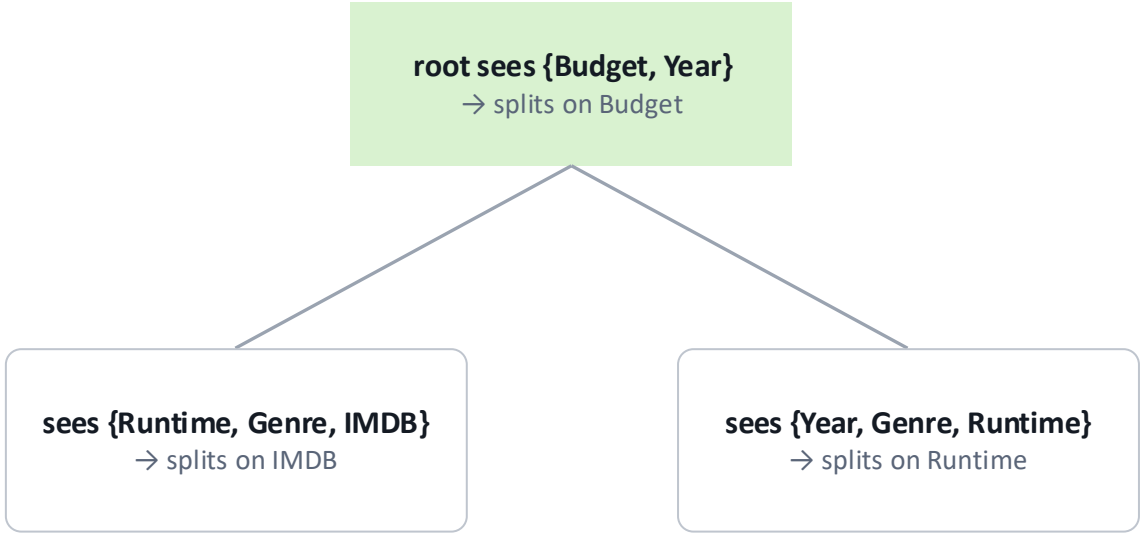
how do random forests decorrelate the trees?

split-feature randomization

At **every node**, draw a fresh random subset of M' of the M features and pick the best split **among those only**.

- Strong features are sometimes **unavailable**, so other features get a chance
- Each tree starts differently → trees **decorrelate** (smaller ρ)
- Each tree is a little worse alone, but the **average** is better

Movie example (from the slides): features available at each node



IMDB rating cannot win the root here, so this tree differs from the bagged trees on the previous slide.

Random forests:

how does a random forest work?

Algorithm 10: Random Forest (Breiman, 2001)

Require: $\mathcal{D}$, number of trees T, features per split $M' < M$

1: **for** $t = 1, \ldots, T$ **do**
2: $\quad \mathcal{D}_t \leftarrow$ sample N points from $\mathcal{D}$ with replacement ▷ *sample bagging*
3: $\quad h_t \leftarrow \text{BuildTree}(\mathcal{D}_t)$, where at every node: ▷ *usually unpruned*
4: $\quad\quad$ sample M' of the M features uniformly at random ▷ *split-feature randomization*
5: $\quad\quad$ choose the best split among those M' features only
6: **end for**
7: **return** $\bar{h}(x) = \text{aggregate}(h_1, \ldots, h_T)$

Random forest = sample bagging + split-feature randomization

$M' = M$ is plain bagged trees

Trees are usually grown deep (unpruned)

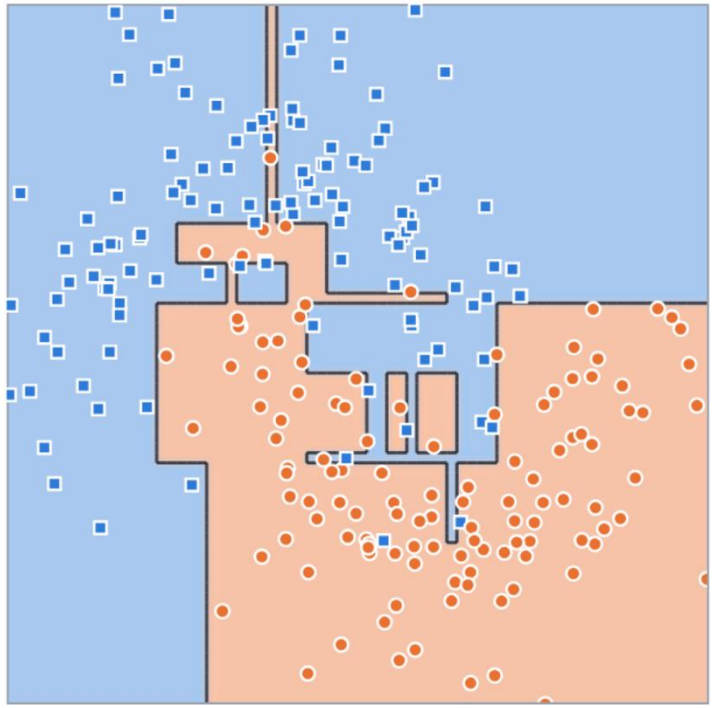
common defaults

$$M' = \sqrt{M} \quad \text{(classification)}, \qquad M' = \frac{M}{3} \quad \text{(regression)}$$

Random forests:

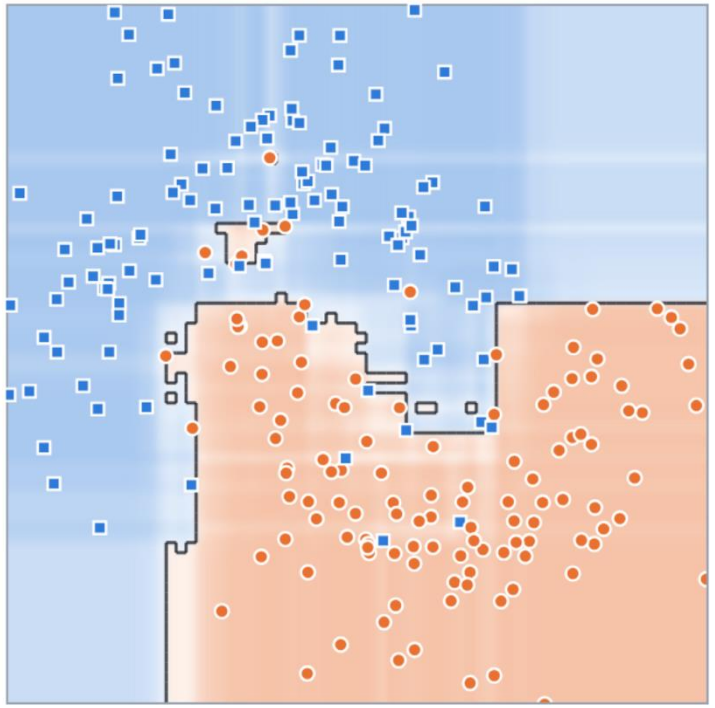
what does a forest's decision boundary look like?

One fully grown decision tree



test error 15.0%

Random forest: B = 100, m = 1



test error 10.5% · OOB error 9.2%

Data: two noisy interleaved half-moons, 240 points

Left: one unpruned tree carves tiny boxes around noisy points

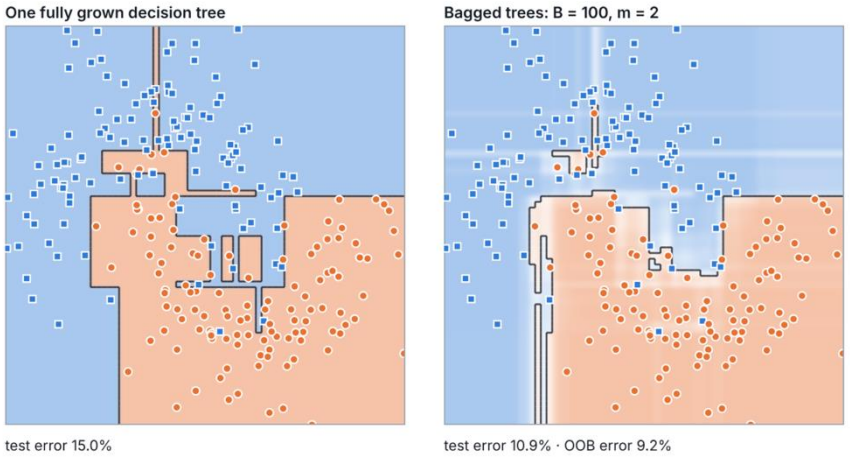
Right: 100 trees, M' = 1. Shading = share of trees voting each class: deep colour = agreement, pale = trees split

One randomized tree alone is worse than a normal tree: the benefit appears only in the average

Random forests:

which is better: one tree, plain bagging, or a forest?

MODEL	TEST ERROR	OOB ERROR
One fully grown tree	15.0%	—
Random forest, B = 1, M′ = 1	13.2%	12.4%
Bagged trees, B = 100, M′ = 2	10.9%	9.2%
Random forest, B = 100, M′ = 1	**10.5%**	**9.2%**



Bagged trees (M′ = 2), B = 100

The forest wins. Each tree overfits in different places, so the idiosyncratic boxes are outvoted; the boundary that survives is the one most trees agree on.

- One randomized tree alone is **not reliably** better; the gain comes from averaging
- With only 2 features, M′ = 1 vs 2 differ little; decorrelation matters most with many features

Out-of-bag error:

What is it?

- For each training point, $x^{(i)}$, there are some decision trees which $x^{(i)}$ was not used to train
- OOB error (solid) tracks error on 1,000 held-out points (dashed) using **no extra data**
- Error drops fast over the first ~20 trees, then plateaus: choose T where it flattens
- Tune M' by OOB error, starting from $\sqrt{M}$. The optimum is usually mid-range

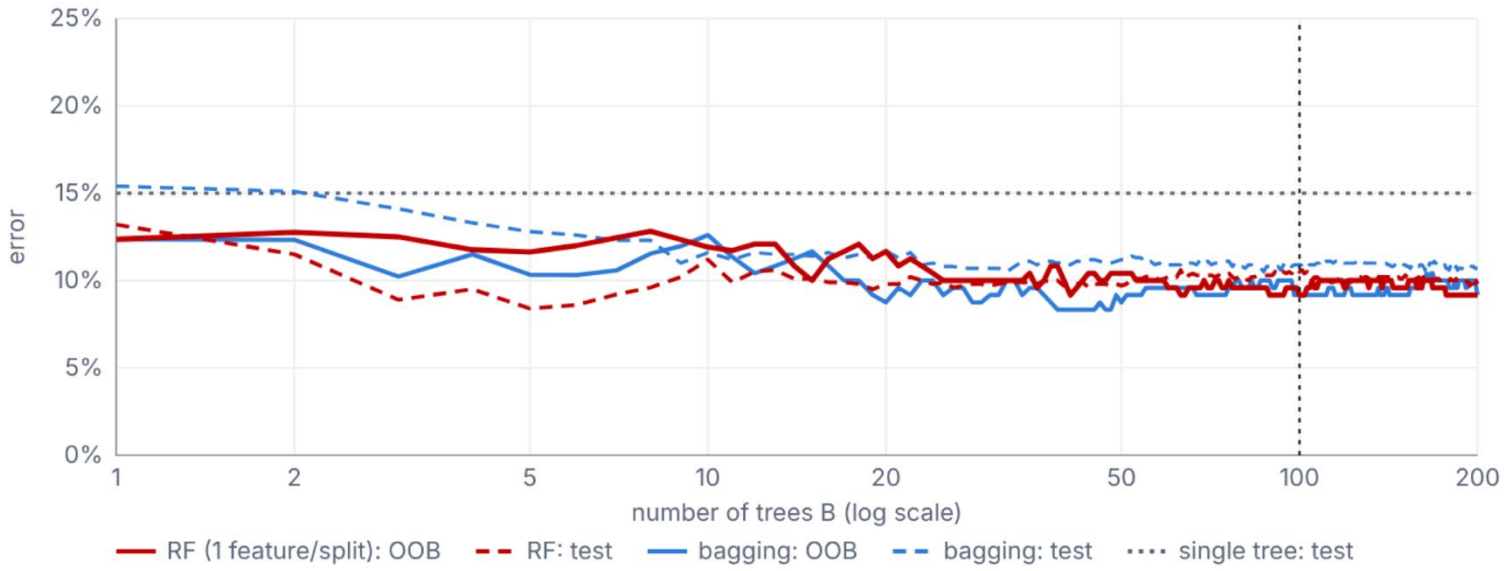
Out-of-bag (OOB) error

Require: trees $h_1, \ldots, h_T$ and their bootstrap samples $\mathcal{D}_1, \ldots, \mathcal{D}_T$

1: **for** $i = 1, \ldots, N$ **do**
2: $\quad \mathcal{T}_i \leftarrow \{t : (x^{(i)}, y^{(i)}) \notin \mathcal{D}_t\}$ ▷ *about 37% of the trees*
3: $\quad \hat{y}_i \leftarrow \text{aggregate}\{h_t(x^{(i)}) : t \in \mathcal{T}_i\}$
4: **end for**
5: **return** $E_{\text{OOB}} = \frac{1}{N} \sum_i \mathbf{1}[\hat{y}_i \neq y^{(i)}]$ ▷ *or squared error*

classification

regression



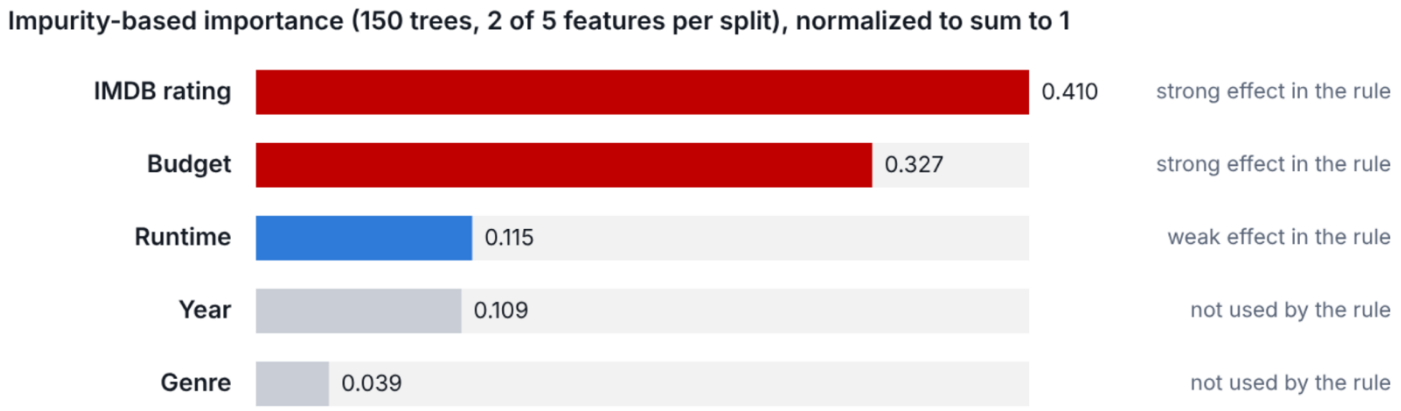
Out-of-bag error:

Feature Importance: which features does a forest rely on?

Algorithm 12: Impurity-based feature importance

Require: a trained forest $h_1, \ldots, h_T$

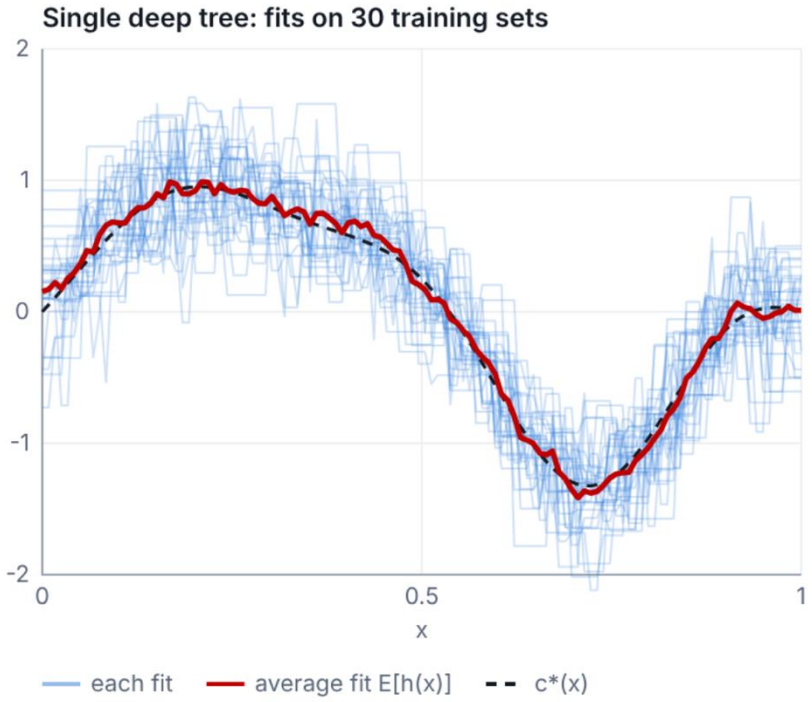
$\text{imp}(m) \leftarrow 0$ for every feature m
for each tree h_t and each internal node v of h_t **do**
    $m \leftarrow$ feature that v splits on; $S_v \leftarrow$ training points reaching v
    $\text{imp}(m) \leftarrow \text{imp}(m) + |S_v| \cdot \text{IG}(S_v, \text{split at } v)$
end for
normalize so that $\sum_m \text{imp}(m) = 1$

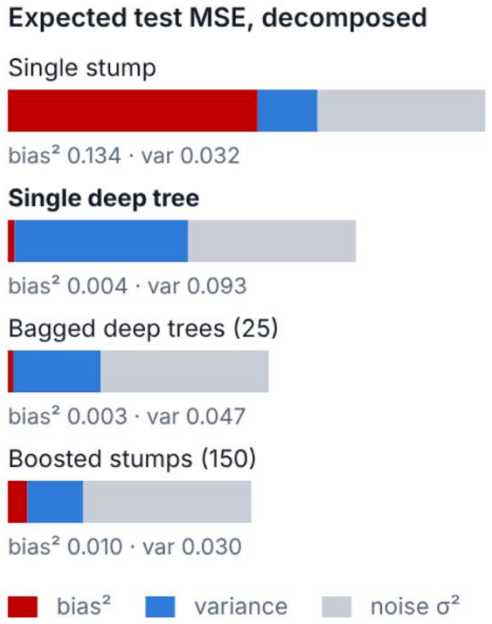


- A forest of 500 trees is not interpretable the way one tree is.
- **Impurity-based importance** recovers part of the picture
    - start every feature at 0
    - every time a feature is used to split a node, add the impurity (entropy) reduction of that split, weighted by the number of training points reaching the node
    - sum over all trees and normalize.

Bias–variance revisited:

what happens when we measure it for each method?





- 30 training sets of 40 points
- **Stump:** high bias. **Deep tree:** high variance
- **Bagging** deep trees roughly **halves the variance**
- **Boosting** stumps cuts bias **more than tenfold**

Bag high-variance learners (deep trees, 1-NN)

Boost high-bias learners (stumps, shallow trees).

We can tune and interpret a forest without extra data. Simply, use each tree's left-out (out-of-bag) examples as a free validation set, and add up each feature's impurity reductions to rank the features.

In practice:

when should I use bagging and when boosting?

	BAGGING / RANDOM FORESTS	BOOSTING (ADABOOST, GBM)
Training	parallel, independent models	sequential; each depends on the previous
Data per model	bootstrap sample (+ random features)	full data, reweighted (or residuals)
Best base learner	low bias, high variance: deep trees	high bias, low variance: stumps, depth 3–8
Mainly reduces	variance	bias (and often variance)
More models	never hurts; error plateaus	can overfit: shrinkage, subsampling, early stopping
Label noise	robust	AdaBoost sensitive (exponential loss)
Tuning effort	low: defaults usually good	higher: learning rate × rounds × depth
Free validation	OOB error	none: validation split or CV

Summary:

Diversity is everything A vote helps only if members beat chance and err differently; correlation ρ sets the floor $\rho\sigma^2$.

Weighted majority Shrink wrong experts by β; mistakes within $O(m + logN)$ of the best expert.

Boosting cuts bias Reweight toward mistakes; α_t from the error; training error falls exponentially.

Bagging cuts variance Bootstrap and average unstable learners; random forests add random features per split. OOB is free validation.

Learn the combiner Stacking trains a meta-model on out-of-fold predictions.