

Ensemble Learning

Aayush Bajaj

July 10, 2026

Contents

1	bagging	1
1.1	the bootstrap	1
1.2	what averaging buys: the correlation formula	2
2	random forests	2
3	boosting	2
3.1	adaboost, the algorithm	3
3.2	why those formulas: forward stagewise additive modelling	3
3.3	gradient boosting	4
3.4	boosting overfits slowly (and margins are why)	4
4	stacking	4
5	one picture	5
6	in code	5
7	see also	7

one model is an opinion; a committee is an estimator. ensemble methods build many imperfect predictors and combine them, and the two great families attack opposite ends of the **bias-variance decomposition**: **bagging** averages low-bias, high-variance models to cancel their wobble; **boosting** stacks up high-bias, low-variance weak learners to build accuracy that none of them has alone.

the committee must actually disagree sometimes — a parliament of clones has the variance of one member.

1 bagging

1.1 the bootstrap

draw B datasets $\mathcal{D}_1^*, \dots, \mathcal{D}_B^*$, each of size n , by sampling *with replacement* from the training set. train one model per replicate and average (regression) or majority-vote (classification) (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009).

- each bootstrap sample omits, in expectation, a fraction $(1 - 1/n)^n \rightarrow e^{-1} \approx 0.368$ of the training points — a fact that becomes useful in a moment.
- the replicates are draws from (approximately) the same distribution over fitted models, so averaging leaves the **bias unchanged** — bagging cannot fix a model that is systematically wrong.

1.2 what averaging buys: the correlation formula

let each committee member's prediction at a fixed point have variance σ^2 , with pairwise correlation ρ between members (they trained on overlapping data, so $\rho > 0$). the variance of the average of B such identically-distributed predictions is

$$\begin{aligned}\text{Var}\left(\frac{1}{B} \sum_{b=1}^B \hat{f}_b\right) &= \frac{1}{B^2} [B \sigma^2 + B(B-1) \rho \sigma^2] \\ &= \boxed{\rho \sigma^2 + \frac{1-\rho}{B} \sigma^2}\end{aligned}$$

- the second term dies as $B \rightarrow \infty$; the first does not. **correlation sets a variance floor** $\rho \sigma^2$ that no amount of extra committee members can dig below (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009).
- so bagging's whole game is: use members with low bias and high σ^2 (deep, unpruned **decision trees** are the canonical choice), crank B until the second term is gone, and then fight ρ .
- bagging a *stable* learner is a waste of electricity: a linear regression refitted on bootstrap replicates barely moves, so there is little variance to average away.

stability is the enemy of ensembling. trees are gloriously unstable, which is why they star in every ensemble.

2 random forests

random forests are bagging plus a deliberate attack on ρ (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):¹

- **decorrelation** at every split of every tree, only a random subset of m features (typically $m \approx \sqrt{p}$ for classification, $p/3$ for regression) is eligible. strong predictors no longer dominate every tree, the trees diverge, ρ drops, and the variance floor $\rho \sigma^2$ drops with it — at the cost of a small bias increase per tree.
- **out-of-bag (oob) error** each tree never saw $\approx 36.8\%$ of the data. predict each training point using only the trees that *didn't* train on it, and you get an approximately unbiased test-error estimate (slightly pessimistic in practice) for free — no held-out set, no extra fitting, morally equivalent to a large **cross validation**.
- **variable importance** comes almost free too: accumulate split-quality improvements per feature, or permute a feature in the oob samples and record the accuracy drop.
- forests barely overfit in B (the variance formula is monotone decreasing in B), so “more trees” is nearly always safe — the tuning action is in m and tree depth.

3 boosting

boosting inverts every design decision of bagging: members are trained *sequentially*, each one on a reweighted dataset that emphasises the mistakes of those before it, and the vote is *weighted*. members can be extremely weak — one-split “stumps” — because the ensemble builds its complexity additively.

¹breiman (2001), *random forests*, machine learning 45(1). bagging is breiman (1996), *bagging predictors*, machine learning 24(2).

3.1 adaboost, the algorithm

binary labels $y_i \in \{-1, +1\}$, weak learners h_m , weights w_i on training points (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):²

- **input** data $\{(x_i, y_i)\}_{i=1}^n$, rounds M ; initialise $w_i = 1/n$.
- **loop** for $m = 1, \dots, M$:
 1. fit h_m to minimise the *weighted* error $\text{err}_m = \sum_i w_i \mathbf{1}[h_m(x_i) \neq y_i] / \sum_i w_i$.
 2. set the vote weight

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \text{err}_m}{\text{err}_m}, \quad (1)$$
 positive as long as the learner beats coin-flipping.
 3. reweight the data: $w_i \leftarrow w_i \exp(-\alpha_m y_i h_m(x_i))$, then normalise. mistakes get heavier, solved points fade.
- **output** the weighted committee $F_M(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(x) \right)$.

3.2 why those formulas: forward stagewise additive modelling

adaboost looks like folklore until you see it as greedy optimisation. build an additive model $F_m = F_{m-1} + \beta h$ one term at a time, minimising the **exponential loss** $L(y, F) = e^{-yF}$ (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):

$$\begin{aligned} (\beta_m, h_m) &= \arg \min_{\beta, h} \sum_{i=1}^n \exp(-y_i(F_{m-1}(x_i) + \beta h(x_i))) \\ &= \arg \min_{\beta, h} \sum_{i=1}^n w_i^{(m)} \exp(-\beta y_i h(x_i)), \quad w_i^{(m)} = e^{-y_i F_{m-1}(x_i)}, \end{aligned}$$

so the “mistake weights” are just the accumulated exponential loss. since $y_i h(x_i) = \pm 1$, split the sum over correct and incorrect points (weights normalised to sum to one):

$$e^{-\beta} (1 - \text{err}) + e^{\beta} \text{err}. \quad (2)$$

- minimising over h at any $\beta > 0$: minimise the weighted error — **step 1 of adaboost**.
- setting the β -derivative to zero: $-e^{-\beta}(1 - \text{err}) + e^{\beta} \text{err} = 0$, i.e. $\beta = \frac{1}{2} \ln \frac{1 - \text{err}}{\text{err}}$ — **step 2**.
- the induced weight recursion $w_i^{(m+1)} = w_i^{(m)} e^{-\beta_m y_i h_m(x_i)}$ — **step 3**.

adaboost is exact greedy coordinate descent on exponential loss over the span of weak learners. the population minimiser of $\mathbb{E}[e^{-yF(x)}]$ is $F^*(x) = \frac{1}{2} \ln \frac{P(y=1|x)}{P(y=-1|x)}$ —

²freund and schapire (1997), *a decision-theoretic generalization of on-line learning and an application to boosting*, jcss 55(1). the textbook variant uses $\alpha_m = \ln \frac{1 - \text{err}_m}{\text{err}_m}$ and reweights only the mistakes — identical classifier after normalisation, since the two conventions differ by a factor of 2 in α and a constant factor in the weights.

half the log-odds — which is why sign F is the right read-out, and why exponential loss sits next to its better-behaved siblings on the [loss functions](#) page.

3.3 gradient boosting

swap “exponential” for *any* differentiable loss and the same greedy scheme survives as **functional gradient descent** (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):³

- **pseudo-residuals** at round m , compute $r_i = -[\partial L(y_i, F(x_i))/\partial F(x_i)]_{F=F_{m-1}}$ — the direction, per training point, in which the loss falls fastest. for squared loss r_i is literally the residual $y_i - F_{m-1}(x_i)$; for absolute loss, its sign; for logistic deviance, $y_i - p_{m-1}(x_i)$.
- **fit** a small regression tree h_m to (x_i, r_i) — projecting the gradient onto the space of representable functions — and take a line-search step in it.
- **shrink**: update $F_m = F_{m-1} + \nu \gamma_m h_m$ with learning rate $\nu \in (0, 1]$, typically 0.01–0.1. small ν plus many rounds regularises far better than either alone. subsampling rows per round (“stochastic gradient boosting”) decorrelates the steps for another variance discount.
- tree depth sets the **interaction order** of the final model: stumps give an additive model, depth-2 trees allow pairwise interactions, and so on. this dial, plus ν , plus early stopping via a validation set, is the entire tuning surface of the xgboost/lightgbm family.

3.4 boosting overfits slowly (and margins are why)

boosting keeps adding capacity, yet its test error routinely keeps *falling* long after training error hits zero. the margin explanation: further rounds keep increasing the confidence $y_i F(x_i)/\|\alpha\|_1$ of the classifications, pushing points away from the decision boundary — the same maximum-margin instinct as the [support vector machine](#), reached by entirely different means. it is not immune though: boosting will eventually fit label noise (exponential loss punishes outliers viciously), which is where shrinkage and early stopping earn their keep.

4 stacking

bagging averages clones and boosting chains weaklings; **stacking** combines *different species* — a forest, a kernel machine, a neural net — by learning the combination itself (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):⁴

- **level 0** train the base models. **level 1** train a simple combiner (linear or logistic regression, often constrained to non-negative weights) whose inputs are the base models’ predictions.
- the cardinal rule: the combiner must be trained on **out-of-fold** predictions — each base model predicts points it never trained on, k-fold style. train it on in-sample predictions and it will learn to trust whichever base model overfits hardest.

³friedman (2001), *greedy function approximation: a gradient boosting machine*, annals of statistics 29(5).

⁴wolpert (1992), *stacked generalization*, neural networks 5(2).

- simple averaging and majority voting are stacking with the combiner frozen; most competition-winning pipelines are stacks with two or three levels.

5 one picture

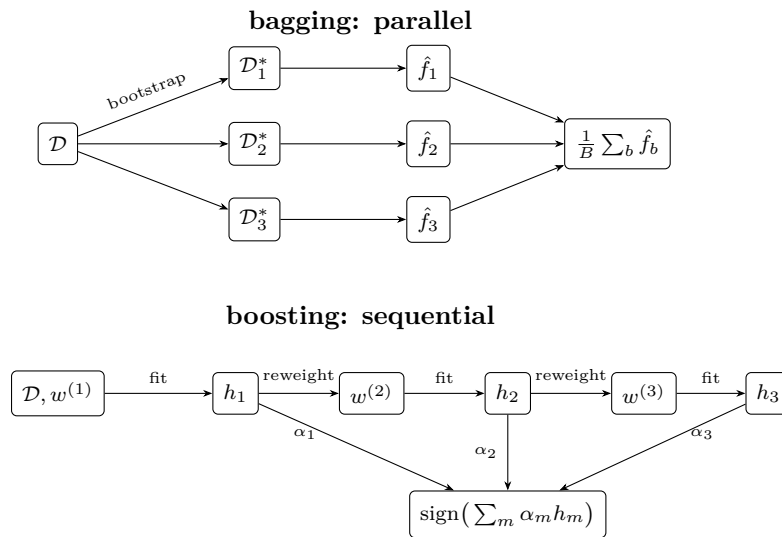


Figure 1: the two architectures. bagging: independent bootstrap worlds in parallel, combined by an unweighted average — variance falls to the correlation floor. boosting: one sequential conversation with the errors, combined by a weighted vote — bias falls, one weak learner at a time.

6 in code

adaboost from scratch — exact decision stumps as weak learners, on the classic “nested spheres” toy from ch 10 of (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009): ten standard-normal features, $y = +1$ iff $\|x\|^2$ exceeds the median of a χ^2_{10} , so the boundary is a sphere no single axis-aligned stump can see:

```

1 import numpy as np
2
3 rng = np.random.default_rng(1)
4 # the ESL ch 10 toy: x ~ N(0, I_10), y = +1 iff ||x||^2 > 9.34,
5 # the median of a chi-squared with 10 dof -- classes exactly balanced
6 def make(n):
7     X = rng.normal(size=(n, 10))
8     y = np.where((X**2).sum(1) > 9.34, 1.0, -1.0)
9     return X, y
10
11 Xtr, ytr = make(2000)
12 Xte, yte = make(2000)
13 n, d = Xtr.shape
14 order = np.argsort(Xtr, axis=0)      # sort each feature once
15
16 def best_stump(w):
17     """exact best decision stump (feature, threshold, polarity) under weights
18     ↪ w."""
19     best = (np.inf, 0, 0.0, 1)
20     for j in range(d):
21         idx = order[:, j]
22         ys, ws = ytr[idx], w[idx]
23         pos = np.cumsum(ws * (ys > 0))      # weight of +1s at or below split

```

```

23     neg = np.cumsum(ws * (ys < 0))
24     # split after position k: predict +1 for x > thr, -1 below
25     err_plus = pos[:-1] + (neg[-1] - neg[:-1])
26     k = np.argmin(err_plus)
27     if err_plus[k] < best[0]:
28         best = (err_plus[k], j, (Xtr[idx[k], j] + Xtr[idx[k+1], j]) / 2, 1)
29     k = np.argmax(err_plus) # flipped polarity
30     if 1 - err_plus[k] < best[0]:
31         best = (1 - err_plus[k], j, (Xtr[idx[k], j] + Xtr[idx[k+1], j]) /
32             ↪ 2, -1)
33     return best
34
35 def stump_pred(X, j, thr, s):
36     return s * np.where(X[:, j] > thr, 1.0, -1.0)
37
38 w = np.full(n, 1 / n)
39 Ftr, Fte = np.zeros(n), np.zeros(len(yte))
40 print(f"{'round':>6} {'w-err':>7} {'alpha':>7} {'train':>7} {'test':>7}")
41 for t in range(1, 801):
42     err, j, thr, s = best_stump(w)
43     alpha = 0.5 * np.log((1 - err) / max(err, 1e-12))
44     h = stump_pred(Xtr, j, thr, s)
45     Ftr += alpha * h
46     Fte += alpha * stump_pred(Xte, j, thr, s)
47     w *= np.exp(-alpha * ytr * h) # upweight the mistakes
48     w /= w.sum()
49     if t in (1, 10, 50, 100, 200, 400, 800):
50         tr = (np.sign(Ftr) != ytr).mean()
51         te = (np.sign(Fte) != yte).mean()
52         print(f"{'t':>6} {'err':>7.3f} {'alpha':>7.3f} {'tr':>7.3f} {'te':>7.3f}")

```

round	w-err	alpha	train	test
1	0.413	0.177	0.412	0.435
10	0.438	0.124	0.339	0.369
50	0.450	0.101	0.202	0.221
100	0.468	0.064	0.137	0.174
200	0.469	0.063	0.103	0.151
400	0.478	0.045	0.062	0.118
800	0.482	0.036	0.038	0.096

reading the run:

- a **single stump** scores 43.5% test error — barely better than the coin it legally has to beat. eight hundred of them, boosted, reach 9.6%. weak learnability compounding into strong learnability is the entire theorem, live.
- the **weighted** error err_m drifts up toward 0.5 as boosting concentrates weight on ever-harder points — each new stump faces a nastier distribution than the last, and α_m shrinks accordingly.
- **no overfitting in sight**: test error is still falling at round 800 even as training error approaches zero — the margin story in action. (it would eventually creep up; exponential loss has no mercy on label noise.)
- boosting had to build the sphere out of axis-aligned cuts: the additive model $\sum_m \alpha_m h_m$ approximates $\sum_j (\text{bumps in } x_j)$, which can express $\|x\|^2 > c$ exactly because the discriminant is coordinate-additive. a problem needing genuine feature *interactions* would want deeper trees as weak learners.

7 see also

- [decision trees](#) — the raw material of both great ensemble families
- [bias-variance decomposition](#) — the ledger in which bagging and boosting settle different debts
- [loss functions](#) — exponential loss and its calmer siblings
- [cross validation](#) — what oob error gives you for free
- [support vector machines](#) — the other road to large margins

references

Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome (2009). *The Elements of Statistical Learning*, Springer.