Ensemble Learning — A Complete Tutorial

Bias–Variance Trade-off, Bagging, and Boosting

From the "why" (bias–variance) to the "how" (algorithms) to the "show me" (worked numericals) — every number verified.

---

1. The foundation: the Bias–Variance Trade-off

Before *any* ensemble method makes sense, you must know *what error we are trying to reduce.* Every supervised model's expected test error splits into three parts.

1.1 The three sources of error

We assume the data is generated as $y = f(\mathbf{x}) + \varepsilon$, where f is the unknown true function and ε is noise with mean 0 and variance σ^2. We train a model $\hat{f}$ on a *random* dataset D; because D is random, $\hat{f}(\mathbf{x})$ is itself a random quantity.

Key idea

For squared-error loss, the expected error at a point $\mathbf{x}$ decomposes *exactly* as

$$\underbrace{\mathbb{E}_D\big[(y - \hat{f}(\mathbf{x}))^2\big]}_{\text{expected test error}} = \underbrace{\big(\mathbb{E}_D[\hat{f}(\mathbf{x})] - f(\mathbf{x})\big)^2}_{\textbf{Bias}^2} + \underbrace{\mathbb{E}_D\big[(\hat{f}(\mathbf{x}) - \mathbb{E}_D[\hat{f}(\mathbf{x})])^2\big]}_{\textbf{Variance}} + \underbrace{\sigma^2}_{\textbf{Irreducible}}$$

Bias — error from wrong *assumptions.* A model too simple to capture f is *consistently* off. High bias ⇒ **underfitting**.

Variance — error from *sensitivity* to the particular training set. A model too flexible changes wildly when D changes. High variance ⇒ **overfitting**.

Irreducible (σ^2) — noise in the data itself. *No* model can remove it.

1.2 Where the decomposition comes from (short derivation)

Write $\bar{f} = \mathbb{E}_D[\hat{f}]$ and drop $\mathbf{x}$ for clarity. Since $\mathbb{E}[\varepsilon] = 0$ and ε is independent of $\hat{f}$,

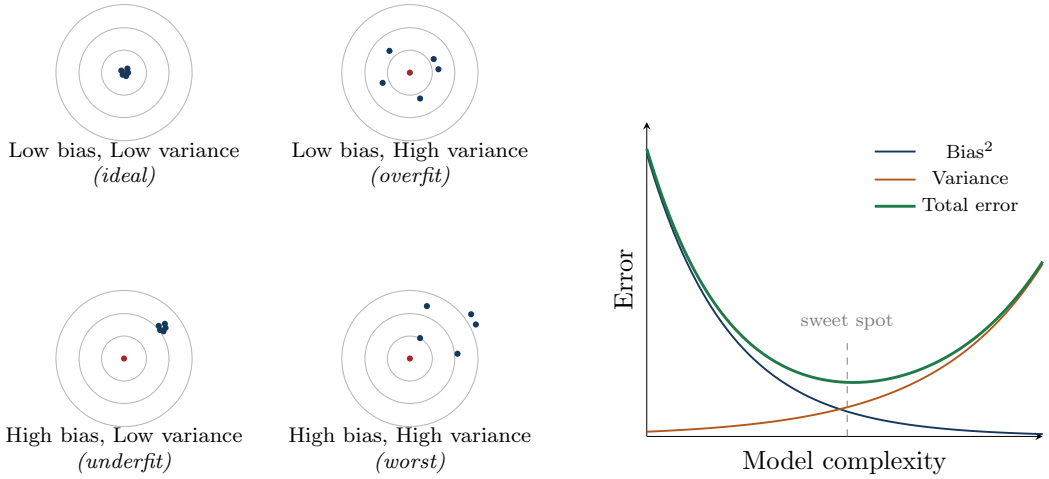
$$\mathbb{E}\big[(y - \hat{f})^2\big] = \mathbb{E}\big[(f + \varepsilon - \hat{f})^2\big] = \underbrace{\mathbb{E}[(f - \hat{f})^2]}_{\text{add/subtract } \bar{f}} + \sigma^2.$$

Now split $f - \hat{f} = (f - \bar{f}) + (\bar{f} - \hat{f})$ and square; the cross term vanishes because $\mathbb{E}[\bar{f} - \hat{f}] = 0$:

$$\mathbb{E}[(f - \hat{f})^2] = \underbrace{(f - \bar{f})^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\bar{f} - \hat{f})^2]}_{\text{Variance}}.$$

That is the whole result. *Bias measures where the average prediction lands; variance measures how much predictions scatter around that average.*

1.3 Two pictures every student should carry



Left: bias is *how far off-centre* the shots land; variance is *how scattered* they are. *Right:* as complexity grows, bias falls but variance rises; total error is U-shaped, and the goal is its minimum.

1.4 Worked numerical — computing bias and variance

Suppose the true value at a point is $f = 10$ and the noise variance is $\sigma^2 = 0.5$. We train the same model on 5 different training sets and record its prediction at that point.

Model	Predictions over 5 datasets					
Complex (flexible)	12	9	11	13	10	$\bar{f} = 11$
Simple (rigid)	8	9	8	9	8	$\bar{f} = 8.4$

Complex model. Bias $= \bar{f} - f = 11 - 10 = 1 \Rightarrow \text{Bias}^2 = 1.0$. Var $= \frac{1}{5}\left[(1)^2 + (-2)^2 + 0^2 + (2)^2 + (-1)^2\right] = \frac{10}{5} = 2.0$.

$$\text{Total} = \text{Bias}^2 + \text{Var} + \sigma^2 = 1.0 + 2.0 + 0.5 = \mathbf{3.5}.$$

Simple model. Bias $= 8.4 - 10 = -1.6 \Rightarrow \text{Bias}^2 = 2.56$. Var $= \frac{1}{5}\left[(-0.4)^2 + (0.6)^2 + (-0.4)^2 + (0.6)^2 + (-0.4)^2\right] = \frac{1.2}{5} = 0.24$.

$$\text{Total} = 2.56 + 0.24 + 0.5 = \mathbf{3.3}.$$

The complex model has *lower bias but higher variance*; the simple model the reverse. Neither dominates — that tension is the trade-off. **Ensembles attack it directly.**

1.5 Why ensembles help — the variance-reduction formula

Average M models, each with variance σ^2 and *pairwise correlation* ρ. The variance of the average is

$$\boxed{\text{Var}\left(\frac{1}{M}\sum_i \hat{f}_i\right) = \rho\,\sigma^2 + \frac{1-\rho}{M}\,\sigma^2}$$

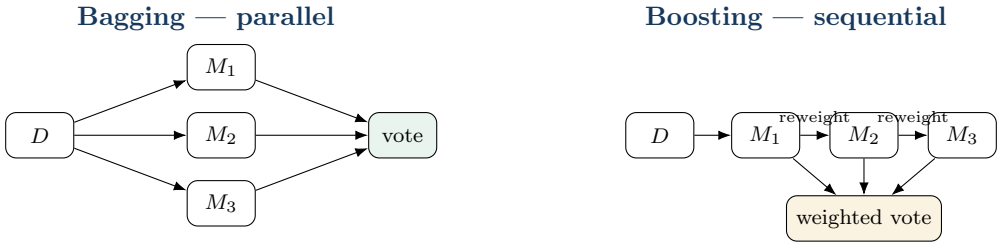
Intuition

Two knobs reduce variance: **more models** ($M \uparrow$ shrinks the second term) and **less correlation** ($\rho \downarrow$ shrinks the first). Bagging supplies the first; Random Forests add the second.

Numerical. With $\sigma^2 = 2$ and $M = 10$: if models were independent ($\rho = 0$) $\to$ Var $= 0.2$; realistically correlated ($\rho = 0.2$) $\to 0.2(2) + \frac{0.8}{10}(2) = 0.4 + 0.16 = \mathbf{0.56}$; identical ($\rho = 1$) $\to 2$ (no gain). So averaging cut variance from 2 to 0.56 — *only because the models differ.*

2. Ensemble learning at a glance

An **ensemble** combines many "base" models into one stronger predictor. It works when the base models are individually *good enough* (better than chance) and *diverse* (they make different mistakes), so their errors partly cancel on aggregation.



3. Bagging (Bootstrap Aggregating)

Key idea

Bootstrap many datasets by sampling the original *with replacement* → train one independent model on each → **aggregate** by majority vote (classification) or average (regression). Bagging chiefly reduces **variance**, so it shines on low-bias, high-variance learners such as deep decision trees.

3.1 The algorithm

Input: dataset D of N tuples; ensemble size k; a base learner.

Training — for $i = 1, \ldots, k$:

1. Draw a bootstrap sample D_i of size N from D *with replacement*.
2. Train base model M_i on D_i.

Prediction for a new $\mathbf{x}$:

3. Collect $M_1(\mathbf{x}), \ldots, M_k(\mathbf{x})$.
4. Return the **majority class** (or the **mean** for regression).

One line: $D \to D_1, \ldots, D_k \to M_1, \ldots, M_k \to$ vote $\to$ final class.

3.2 Bootstrap sampling and the "free" validation set (OOB)

Because sampling is with replacement, some tuples repeat and some are left out. The probability a given tuple is *never* chosen in N draws is

$$\left(1 - \tfrac{1}{N}\right)^N \xrightarrow[N\to\infty]{} \tfrac{1}{e} \approx 0.368.$$

Key idea

About **63.2%** of the unique tuples appear in each bootstrap sample and **36.8%** are *out-of-bag* (OOB). Each OOB tuple can be predicted by exactly the models that did *not* train on it, giving a **built-in validation estimate** at no extra cost.

Check: $N = 5 \to 0.328$, $N = 10 \to 0.349$, $N = 100 \to 0.366$, $N = 1000 \to 0.368$ — converging to $1/e$.

3.3 Worked numerical 1 — majority vote

Predict whether a new customer will **Buy (Yes/No)**. Three bagged trees are trained on bootstrap samples D_1, D_2, D_3 drawn from 6 customers $\{A, \ldots, F\}$. The membership table shows the effect of sampling with replacement:

Tuple	D_1	D_2	D_3
A	✓	✓	—
B	✓	—	✓✓
C	✓✓	—	✓
D	✓	✓✓	✓
E	✓	✓	✓
F	✓	✓	✓

✓ once ✓✓ twice — omitted

For a new tuple X the three models predict $M_1 = \text{Yes},\ M_2 = \text{No},\ M_3 = \text{Yes}$.

$$\text{Yes} = 2, \quad \text{No} = 1 \quad \Rightarrow \quad \boxed{\text{Final} = \text{YES}} \quad \text{(equal voting weight)}.$$

3.4 Worked numerical 2 — regression by averaging

Bagging is not only for votes. Predict a house price with 5 bagged regressors giving (in lakh) 52, 55, 49, 58, 51.

$$\hat{y} = \tfrac{52+55+49+58+51}{5} = \frac{265}{5} = \mathbf{53} \text{ lakh}.$$

The spread of the five (49–58) is exactly the *variance* bagging averages away: any single tree could be off, but the mean is steadier.

3.5 From Bagging to Random Forests

A Random Forest is bagging with decision trees *plus* one extra trick: at each split, only a random subset of features (typically $\sqrt{p}$ of p features) is considered. This *decorrelates* the trees — lowering ρ in the variance formula of §1.6 — so the ensemble variance drops further than bagging alone.

4. Boosting and AdaBoost

Key idea

Build models **sequentially**; each new model pays *more attention to the tuples the previous ones got wrong.* Combine them by a **weighted** vote, where more accurate models count more. Boosting chiefly reduces **bias**, turning many *weak* learners into one *strong* learner.

$$D \to M_1 \to \text{find mistakes} \to M_2 \to \text{focus on mistakes} \to M_3 \to \cdots \to \text{weighted vote}$$

4.1 The AdaBoost algorithm

Input: d tuples; ensemble size k; a weak learner.

1. Initialise tuple weights equally: $w_i = \dfrac{1}{d}$.

For $i = 1, \ldots, k$:

2. Train M_i on the data sampled/weighted by w.
3. Weighted error: $\text{error}(M_i) = \sum_j w_j \,\text{err}(X_j)$, where $\text{err}(X_j) = 1$ if X_j is misclassified, else 0.

(if $\text{error}(M_i) > 0.5$*, discard and retry / stop.)*

4. Model vote weight: $\alpha_i = \ln \dfrac{1 - \text{error}(M_i)}{\text{error}(M_i)}$.
5. Re-weight tuples: multiply each *correctly* classified tuple's weight by $\dfrac{\text{error}(M_i)}{1 - \text{error}(M_i)}$ (misclassified unchanged), then **normalise** so $\sum_j w_j = 1$.

Prediction: return the class maximising $\displaystyle\sum_{i:\, M_i(\mathbf{x}) = c} \alpha_i$ (weighted vote).

Intuition

Correct tuples get *shrunk* (factor < 1 since error < 0.5); after normalising, the *misclassified* tuples' share rises — so the next model concentrates on them. A model with small error earns a large α and dominates the final vote.

4.2 Fully worked numerical — 3 rounds, 5 tuples

Start: $d = 5$ tuples $\{A, B, C, D, E\}$, all weights $w_i = \frac{1}{5} = 0.2$.

Round 1. M_1 misclassifies $\{B, D\}$.

$$\text{error}(M_1) = w_B + w_D = 0.2 + 0.2 = 0.4, \qquad \alpha_1 = \ln \tfrac{1-0.4}{0.4} = \ln 1.5 = \mathbf{0.4055}.$$

$$\text{factor} = \tfrac{0.4}{0.6} = 0.6667: \ \text{correct } \{A, C, E\} \to 0.2(0.6667) = 0.1333, \ \text{wrong } \{B, D\} \to 0.2.$$

$$Z = 3(0.1333) + 2(0.2) = 0.8 \ \Rightarrow \ \text{normalise: } A{=}C{=}E{=}0.1667, \ B{=}D{=}0.25.$$

Round 2. With those weights, M_2 misclassifies $\{A, E\}$.

$$\text{error}(M_2) = w_A + w_E = 0.1667 + 0.1667 = 0.3333, \qquad \alpha_2 = \ln \tfrac{0.6667}{0.3333} = \ln 2 = \mathbf{0.6931}.$$

$$\text{factor} = 0.5: \ B, C, D \to \{0.125,\ 0.0833,\ 0.125\}, \ A, E \to 0.1667.$$

$$Z = 0.6667 \ \Rightarrow \ A{=}0.25, \ B{=}0.1875, \ C{=}0.125, \ D{=}0.1875, \ E{=}0.25.$$

Round 3. M_3 misclassifies $\{C, D\}$.

$$\text{error}(M_3) = w_C + w_D = 0.125 + 0.1875 = 0.3125, \qquad \alpha_3 = \ln \tfrac{0.6875}{0.3125} = \ln 2.2 = \mathbf{0.7885}.$$

Tuple	w start	after M_1	after M_2	α per model
A	0.20	0.1667	0.25	$\alpha_1 = 0.4055$
B	0.20	0.25	0.1875	$\alpha_2 = 0.6931$
C	0.20	0.1667	0.125	$\alpha_3 = 0.7885$
D	0.20	0.25	0.1875	
E	0.20	0.1667	0.25	

4.3 Final weighted vote

For a new tuple X, suppose $M_1 = \text{Yes}$, $M_2 = \text{No}$, $M_3 = \text{No}$. Unlike bagging, we *do not count heads* — we sum the α's:

$$\text{Yes}: \ \alpha_1 = 0.4055 \qquad \text{No}: \ \alpha_2 + \alpha_3 = 0.6931 + 0.7885 = 1.4816.$$

$$1.4816 > 0.4055 \quad \Rightarrow \quad \boxed{\text{Final} = \text{NO}}.$$

Note that a 2–1 *head-count* also favours No here, but the mechanism is the weighted sum: had M_1 been far more accurate, its single large α could *overturn* a majority.

Watch out

The attached boosting example stops at "correct weight = 0.1333" and never *normalises*. Always finish step 5: divide by Z so weights sum to 1 (here $Z = 0.8$, giving 0.1667 and 0.25). Without normalisation the "weighted error" in the next round is meaningless.

4.4 Advanced note (optional): the exponential form

Classic AdaBoost (Freund & Schapire, labels ± 1) uses $\alpha_i = \frac{1}{2}\ln\frac{1-\text{error}}{\text{error}}$ and the update $w_j \leftarrow w_j\, e^{-\alpha_i y_j M_i(x_j)}$ (then normalise). This is *equivalent* to the multiplicative update above: after normalisation both produce the *same* weight distribution, and the extra $\frac{1}{2}$ only rescales all α's, leaving the arg-max vote unchanged. Gradient Boosting (XGBoost, LightGBM) generalises the same idea by fitting each new model to the *residual gradient* of a differentiable loss.

5. Bagging vs. Boosting — the comparison to memorise

	Bagging	**Boosting**
How models train	Independently, **in parallel**	Sequentially; each fixes the last
Data to each model	Bootstrap sample (with replacement)	Full data with **re-weighted** tuples
Focus	Treats all tuples equally	**Focuses on misclassified** tuples
Combining rule	**Majority** vote / mean (equal weight)	**Weighted** vote (α_i by accuracy)
Mainly reduces	**Variance** (fights overfitting)	**Bias** (fights underfitting)
Base learner	Strong, low-bias (deep trees)	Weak, high-bias (stumps)
Risk	Little extra; very robust	Can **overfit noise** if over-boosted
Typical method	Random Forest	AdaBoost, Gradient Boosting

Key idea

$$\textbf{BAGGING} = \underbrace{\text{BOOTSTRAP}}_{\text{sample with replacement}} + \underbrace{\text{AGGREGATE}}_{\text{majority vote}}$$

$$\textbf{BOOSTING} = \underbrace{\text{re-weight mistakes}}_{\text{sequential, focus on errors}} + \underbrace{\text{weighted vote}}_{\alpha_i = \ln((1-e)/e)}$$

Bagging averages away variance; boosting chips away bias.

Understand the bias–variance trade-off, and both bagging and boosting stop being formulas — they become the two obvious ways to move on the U-curve.