Ensemble Learning

Fundamentals of Machine Learning

Xufeng Zhang

Inria
xufeng.zhang@inria.fr

December 11, 2025

Outline

What is Ensemble Learning?

- **Ensemble learning**: combine multiple base models to obtain a better predictive model.
- Key intuition:
  - Different models make different errors.
  - If their errors are not perfectly correlated, averaging can reduce overall error.
- Works for both classification and regression.
- Often one of the most effective ways to improve performance without changing features.

Voting Classifiers

Hard Voting

- Each classifier outputs a class label.
- Final prediction is the **majority class**.
- Ties are broken by a fixed class order or additional rules.

Soft Voting

- Each classifier outputs class probabilities $p_{m,c}$.
- Weighted average:

$$\hat{y} = \arg\max_{c} \sum_{m=1}^{M} w_m p_{m,c},$$

  where $\sum_m w_m = 1$.
- More informative than hard voting if probabilities are well calibrated.

Why Does Ensembling Work?

- Reduces **variance**: unstable models (e.g., deep trees) change a lot with data perturbations.
- Can reduce **bias**: sequential methods (Boosting) focus on residual errors.
- Main requirements:
  - Base models must be **better than random**.
  - Base models must be **diverse** (uncorrelated errors).
- Diversity can come from:
  - Different training samples.
  - Different model architectures.
  - Different hyperparameters or feature subsets.

Bias–Variance Perspective

- Prediction error (expected squared error):

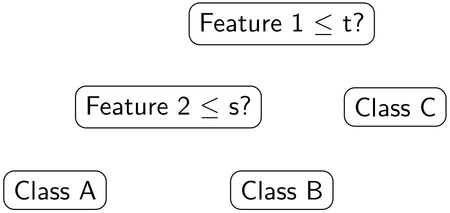
$$\mathbb{E}\big[(Y-\hat{f}(X))^2\big] = \underbrace{\text{Bias}^2}_{\text{systematic error}} + \underbrace{\text{Variance}}_{\text{sensitivity to data}} + \text{Noise}.$$

- **Bagging / Random Forest**:
  - Typically use high-variance, low-bias models.
  - Aim: reduce variance by averaging.
- **Boosting**:
  - Typically use low-variance, high-bias base learners.
  - Aim: reduce bias by combining many weak learners.

Decision Trees: Representation

- Tree structure with **internal nodes** = tests on features.
- **Leaves** correspond to regions of feature space.
- Classification: leaf stores class probabilities (relative frequencies).
- Regression: leaf stores the mean target value.
- Advantage: easy to visualize and interpret.

Example: Simple Classification Tree



- Each root-to-leaf path defines a **decision rule**.
- Trees partition the space using axis-aligned splits.

Splitting Criteria and Impurity

Consider a leaf with class probabilities $\hat{p}_c$.

Gini Impurity

$$G(\hat{\mathbf{p}}) = \sum_c \hat{p}_c(1 - \hat{p}_c) = 1 - \sum_c \hat{p}_c^2.$$

Entropy

$$H(\hat{\mathbf{p}}) = -\sum_c \hat{p}_c \log_2 \hat{p}_c.$$

- Splits are chosen to maximize **information gain**.
- Information gain $=$ impurity(parent) $-$ weighted impurity(children).

Regression Trees

- For regression, each leaf predicts:

$$\mu_L = \frac{1}{|L|} \sum_{i \in L} y_i.$$

- Split chosen to minimize squared error in leaves:

$$\sum_L \sum_{i \in L} (y_i - \mu_L)^2.$$

- Predictions are piecewise constant: non-smooth and cannot extrapolate.
- Still powerful as base learners inside ensembles.

Feature Importance from Trees

- Trees can compute **impurity-based feature importance**.
- Importance of feature j:
  - Sum of impurity reductions over all splits on feature j,
  - Weighted by the number of samples at each node.
- In Random Forests / Gradient Boosting:
  - Importances are averaged over many trees.
  - More robust than using a single tree.

Under- and Overfitting with Trees

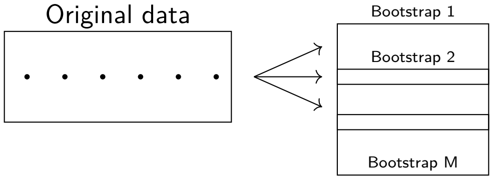
- **Shallow trees**:
  - High bias, low variance.
  - Capture only simple patterns (underfitting).
- **Deep trees**:
  - Low bias, high variance.
  - Fit training data very well (risk of overfitting).
- Trees are ideal base learners for ensembles:
  - Deep trees for Bagging / Random Forest (variance reduction).
  - Shallow trees (stumps) for Boosting (bias reduction).

Bagging: Bootstrap Aggregating

- Given training set D of size n.
- For $m = 1, \ldots, M$:
    - Draw bootstrap sample D_m of size n with replacement.
    - Train base model h_m on D_m.
- For a new point x:
    - Classification: majority vote or soft vote.
    - Regression: average prediction $\hat{y}(x) = \frac{1}{M} \sum_m h_m(x)$.
- Reduces variance by averaging many high-variance estimators.

Bootstrap Sampling Illustration

- Each bootstrap sample leaves out about $\approx 36\%$ of instances (out-of-bag).
- Different samples lead to different fitted trees.
- Averaging the predictions smooths the decision boundary.



Random Forests

- Extension of Bagging using **randomized trees**.
- At each split:
  - Consider only a random subset of features of size `max_features`.
  - Choose the best split among those features.
- Further de-correlates trees:
  - Trees become more diverse.
  - Ensemble variance decreases more effectively.
- Extremely randomized trees (ExtraTrees):
  - Also randomize split thresholds.
  - Faster and sometimes more regularized.

Effect on Bias and Variance

- Increasing the number of trees M:
  - **Variance** decreases and stabilizes.
  - **Bias** is mostly unchanged (trees are still high-capacity).
- With very many trees:
  - Decision boundary becomes smoother.
  - Too much smoothing can slightly increase bias.
- In practice:
  - Use large M (e.g., 100–1000) until performance plateaus.
  - Training/prediction time is the main limitation.

Important Hyperparameters (Random Forest)

- `n_estimators`:
  - Number of trees; higher is usually better.
  - Diminishing returns after some point.
- `max_features`:
  - Typical defaults: $\sqrt{p}$ for classification, $\log_2(p)$ or p for regression.
  - Smaller values increase decorrelation but may increase bias.
- Tree parameters:
  - `max_depth`, `min_samples_split`, `min_samples_leaf`, etc.
  - Pre-pruning reduces tree size and training time.
- `bootstrap`, `oob_score` (out-of-bag error).

Out-of-Bag (OOB) Error

- For each tree, about $1/3$ of training samples are **not** included in its bootstrap sample.
- These are **out-of-bag** (OOB) for that tree.
- For each training point:
  - Aggregate predictions over trees for which the point is OOB.
- Compute OOB error using these aggregated predictions.
- OOB error:
  - Comparable to cross-validation performance.
  - Saves computation during model selection.

Random Forests: Strengths and Weaknesses

Strengths

- Strong baseline; often high accuracy with minimal tuning.
- Handles heterogeneous feature types and scales.
- Provides feature importance estimates.
- Parallelizable across trees.

Weaknesses

- Less interpretable than single trees.
- Larger memory footprint, slower predictions.
- Not ideal for very high-dimensional sparse data (e.g., text).
- Probability estimates can be poorly calibrated.

Boosting: High-Level Idea

- Build an ensemble sequentially.
- Each new model focuses on examples that previous models got wrong.
- Final model is an additive combination:

$$F_M(x) = \sum_{m=1}^{M} \alpha_m h_m(x).$$

- Two major families:
  - **AdaBoost**: reweight samples based on previous errors.
  - **Gradient Boosting**: fit to residuals (gradient steps).

AdaBoost: Intuition

- Use simple base learners: usually decision stumps (depth-1 trees).
- Start with equal weights on all training examples.
- After each iteration:
  - Increase weights of misclassified examples.
  - Decrease weights of correctly classified examples.
- New weak learner is trained on this reweighted dataset.
- Hard examples get more influence over time.

AdaBoost: Algorithm (Binary Classification)

1. Initialize sample weights $w_i^{(1)} = 1/N$.
2. For $m = 1, \ldots, M$:
   1. Train weak learner h_m using weights $w^{(m)}$.
   2. Compute weighted error:
      $$\varepsilon_m = \frac{\sum_i w_i^{(m)} \mathbf{1}\{h_m(x_i) \neq y_i\}}{\sum_i w_i^{(m)}}.$$
   3. Compute learner weight:
      $$\alpha_m = \frac{1}{2} \log \frac{1 - \varepsilon_m}{\varepsilon_m}.$$
   4. Update sample weights:
      $$w_i^{(m+1)} = w_i^{(m)} \exp\big(- \alpha_m y_i h_m(x_i)\big),$$
      then renormalize.
3. Final prediction: $\operatorname{sign}\left(\sum_m \alpha_m h_m(x)\right)$.

AdaBoost: Bias–Variance Behaviour

- Weak learners have high bias, low variance.
- AdaBoost reduces bias by combining many such learners.
- Initially:
  - Strong improvements in both training and test error.
- If we keep boosting:
  - Training error can reach zero.
  - Variance may increase; risk of overfitting.
- In practice:
  - Control complexity via `n_estimators` and `learning_rate`.

Gradient Boosting: General Framework

- Want to minimize a differentiable loss $L(y, F(x))$.
- Initialize model $F_0(x)$ (e.g., constant).
- For $m = 1, \ldots, M$:
  1. Compute pseudo-residuals:
  $$r_i^{(m)} = - \left. \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right|_{F=F_{m-1}}.$$
  2. Fit base learner $h_m(x)$ to predict $r^{(m)}$.
  3. Choose step size η (learning rate).
  4. Update:
  $$F_m(x) = F_{m-1}(x) + \eta\, h_m(x).$$
- Base learners: typically shallow regression trees.

Gradient Boosting for Regression

- Loss: squared error $L(y, F(x)) = \frac{1}{2}(y - F(x))^2$.
- Pseudo-residuals:

$$r_i^{(m)} = y_i - F_{m-1}(x_i),$$

  i.e., the regression residuals.
- Each tree approximates the residuals of the current ensemble.
- Final prediction:

$$F_M(x) = F_0(x) + \eta \sum_{m=1}^{M} h_m(x).$$

- Early stopping: monitor validation error and stop when it stops improving.

Gradient Boosting for Classification

- Often implemented via **logistic loss**.
- For binary labels $y_i \in \{0, 1\}$, model log-odds:

$$F(x) \approx \log \frac{p(y = 1 \mid x)}{p(y = 0 \mid x)}.$$

- Pseudo-residuals:

$$r_i^{(m)} = y_i - p_{m-1}(x_i),$$

  where $p_{m-1} = \sigma(F_{m-1}(x_i))$ is the predicted probability.
- Trees fit these residuals; ensemble updates log-odds.
- Final probability via sigmoid:

$$p_M(x) = \sigma\big(F_M(x)\big).$$

Gradient Boosting: Hyperparameters

- `n_estimators`:
  - Number of boosting iterations.
  - Too many iterations can overfit.
- `learning_rate` (η):
  - Smaller η requires more trees but can improve generalization.
  - Common range: 0.01 to 0.3.
- Tree complexity:
  - `max_depth` or `max_leaf_nodes`.
  - Usually 2–5 for base learners.
- Regularization:
  - Subsample rows or columns.
  - Early stopping via `n_iter_no_change`.

Gradient Boosting: Pros and Cons

Pros

- Among the most accurate off-the-shelf models.
- Handles different feature types and scales.
- Flexible via choice of loss function.

Cons

- Training is sequential and harder to parallelize.
- Sensitive to hyperparameters; more tuning required than Random Forests.
- Can overfit if `n_estimators` is large and regularization is weak.

Extreme Gradient Boosting (XGBoost)

- Highly optimized gradient boosting implementation.
- Key features:
  - Uses second-order derivatives (Hessian) for faster convergence.
  - Regularization on tree weights and leaf scores.
  - Efficient handling of sparse inputs and missing values.
  - Histogram-based split finding (feature binning).
- Supports parallel and distributed training.
- Widely used in machine learning competitions and industry.

LightGBM and CatBoost

LightGBM

- Gradient boosting framework from Microsoft.
- Uses gradient-based one-side sampling:
  - Keep all instances with large gradients.
  - Randomly sample among those with small gradients.
- Leaf-wise tree growth with depth constraints.
- Efficient handling of large-scale datasets.

CatBoost

- Gradient boosting framework optimized for categorical features.
- Uses ordered target encoding and symmetric trees.
- Strong default settings; less need for heavy tuning.
- Supports monotonicity constraints for selected features.

Stacking: Idea

- Combine **heterogeneous** base models (e.g., RF, GBM, SVM, NN).
- For each base model h_j, obtain predictions $\hat{y}_j(x)$.
- Train a **meta-model** (stacker) on these predictions:

$$z(x) = [\hat{y}_1(x), \dots, \hat{y}_K(x)], \quad \tilde{y} = g(z(x)).$$

- Often use cross-validation to create out-of-fold predictions for training the stacker.
- Popular stackers: linear models, logistic regression, gradient boosting.

Stacking: Design Considerations

- Base models should be:
    - **Strong** performers individually.
    - **Diverse** (different biases and errors).
- Avoid information leakage:
    - Use cross-validated predictions for meta-training.
    - Do not train meta-model on predictions from the same folds used to fit base models.
- Complexity and speed:
    - Stacking can be expensive at prediction time (many models).
    - Useful when accuracy is the main objective.

Other Ensemble Techniques

- **Hyper-ensembles**:
  - Same algorithm with different hyperparameter settings.
  - Example: multiple Random Forests with different depths.
- **Deep ensembles**:
  - Train multiple neural networks with different initializations or subsets.
  - Improve robustness and uncertainty estimates.
- **Bayesian model averaging**:
  - Average over models weighted by posterior probabilities.
  - More theoretical; often approximated by ensembles.
- **Cross-validation selection**:
  - Select the single best model via internal cross-validation (not strictly an ensemble).

When to Use Which Ensemble?

- **Random Forests**:
  - Strong, robust baseline.
  - Limited need for tuning; good with small to medium tabular data.
- **Gradient Boosting / XGBoost / LightGBM / CatBoost**:
  - Often best performance on structured/tabular data.
  - Requires more tuning; good when accuracy is critical.
- **AdaBoost**:
  - Useful for simple weak learners.
  - Historically important; less used than modern GBMs.
- **Stacking**:
  - Final refinement when diverse high-performing models are available.

Calibration, Warm Starts, and Early Stopping

- **Calibration**:
  - Tree ensembles often produce poorly calibrated probabilities.
  - Use methods like isotonic regression or Platt scaling.
- **Warm starting**:
  - Continue training by adding trees to an existing ensemble.
  - Useful for incremental improvement on similar data.
- **Early stopping**:
  - Monitor validation loss; stop adding trees when performance stops improving.
  - Avoids overfitting and saves computation.

Typical Ensemble Workflow

1. Start with a simple model (e.g., logistic regression, decision tree) for baseline.
2. Train a Random Forest; tune `n_estimators`, `max_depth`, `max_features`.
3. Train a Gradient Boosting or XGBoost model with early stopping.
4. Compare performance on validation / cross-validation.
5. Optionally:
    - Calibrate probabilities.
    - Build a small stacking ensemble from the best models.
6. Evaluate final ensemble on a held-out test set.

Algorithm Overview

Method	Base Learners	Loss	Optimization
Decision Trees	Single tree	Gini / Entropy / MSE	Greedy splitting
Random Forest	Many trees	Gini / Entropy / MSE	Bagging + feature subsampling
AdaBoost	Stumps / shallow trees	Exponential	Greedy reweighting
Grad. Boosting (Reg.)	Regression trees	MSE	Gradient descent on residuals
Grad. Boosting (Clf.)	Regression trees	Log loss	Gradient descent on log-odds
XGBoost / LightGBM / CatBoost	Small trees	Task-specific	2nd order gradients + regularization
Stacking	Any models	Task-specific	Meta-model on predictions

Key Takeaways

- Ensembles leverage multiple models to reduce error via bias and variance control.
- Bagging / Random Forest:
    - Variance reduction by averaging many overfitting models.
- Boosting (AdaBoost, Gradient Boosting):
    - Bias reduction by focusing on hard examples or residuals.
- Modern boosting libraries (XGBoost, LightGBM, CatBoost) are state-of-the-art for tabular data.
- Stacking combines heterogeneous models using a learned combiner.
- Always consider computational cost, interpretability, and deployment constraints.

References

- Lecture 4: *Ensemble Learning — ML Engineering*, Joaquin Vanschoren.
- L. Breiman, "Random Forests," Machine Learning, 2001.
- J. Friedman, T. Hastie, R. Tibshirani, "The Elements of Statistical Learning."
- T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," KDD 2016.