Lecture 10: Ensemble and Boosting Methods

Luu Minh Sao Khue

From bagging to boosting — theory and practice of ensemble learning, including AdaBoost, Gradient Boosting, XGBoost, LightGBM, and CatBoost. Comparison of bias–variance tradeoff, speed, and interpretability.

1. Learning Objectives

After this lecture, you will be able to:

- Explain the motivation and intuition behind ensemble methods.
- Distinguish between bagging and boosting.
- Derive the AdaBoost algorithm and interpret its weight updates.
- Understand Gradient Boosting as a stagewise additive model.
- Describe XGBoost, LightGBM, and CatBoost differences.
- Compare ensemble algorithms in accuracy, speed, and interpretability.
- Implement and tune Gradient Boosting models in Python.

2. Intuition: Why Ensembles?

A single model may have high variance or bias. **Ensemble methods** combine multiple models to reduce error by averaging (bagging) or sequential correction (boosting).

- **Bagging** reduces variance by training many models on different bootstrap samples and averaging their predictions.
- **Boosting** reduces bias by sequentially focusing on the hardest examples and combining weak learners adaptively.

If individual models are diverse and independent, averaging them stabilizes predictions:

$$\mathrm{Var}(\bar{f}) = \frac{1}{M^2}\sum_{i=1}^{M}\mathrm{Var}(f_i) + \frac{1}{M^2}\sum_{i\neq j}\mathrm{Cov}(f_i, f_j).$$

Low covariance between learners gives large variance reduction.

3. Bagging (Bootstrap Aggregating)

3.1 Mechanism

Given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$:

1. Draw B bootstrap samples $\mathcal{D}_1, \ldots, \mathcal{D}_B$ (each of size n).
2. Train base model f_b on each $\mathcal{D}_b$.
3. For regression: $\hat{f}(x) = \frac{1}{B} \sum_b f_b(x)$.
4. For classification: majority vote of $f_b(x)$.

3.2 Variance Reduction

Bagging decreases variance approximately by $1/B$ when learners are uncorrelated. Random Forests extend bagging by adding random feature selection at each split to reduce correlation.

Out-of-Bag (OOB) Error. Each model f_b leaves out roughly 37% of samples; their predictions on these unseen points can estimate generalization error efficiently.

4. Boosting: Sequential Model Correction

4.1 Concept

Boosting constructs an additive model:

$$F_M(x) = \sum_{m=1}^{M} \alpha_m f_m(x),$$

where f_m are weak learners (e.g., shallow trees), added one by one to minimize a chosen loss $L(y, F(x))$.

Each new model focuses on samples that were mispredicted by previous ones. Learning rate ν controls the contribution of each stage.

5. AdaBoost (Adaptive Boosting)

5.1 Algorithmic Idea

For binary classification $y_i \in \{-1, +1\}$:

1. Initialize equal sample weights $w_i^{(1)} = \frac{1}{n}$.
2. For each round $m = 1, \ldots, M$:
   (a) Fit weak learner $f_m(x)$ minimizing weighted error

$$\varepsilon_m = \frac{\sum_i w_i^{(m)} \mathbf{1}[y_i \neq f_m(x_i)]}{\sum_i w_i^{(m)}}.$$

(b) Compute model weight

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \varepsilon_m}{\varepsilon_m}.$$

(c) Update sample weights:

$$w_i^{(m+1)} = w_i^{(m)} \exp(-\alpha_m y_i f_m(x_i)),$$

then normalize so $\sum_i w_i^{(m+1)} = 1$.

3. Final prediction:

$$F_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m f_m(x)\right).$$

5.2 Loss Interpretation

AdaBoost minimizes the exponential loss:

$$L(y, F(x)) = \exp(-yF(x)).$$

Its update rule is equivalent to gradient descent in function space on this loss.

6. Gradient Boosting

6.1 Additive Model View

We build model iteratively:

$$F_M(x) = F_{M-1}(x) + \nu h_M(x),$$

where $h_M(x)$ approximates the negative gradient of the loss function:

$$r_i^{(m)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}.$$

Each stage fits $h_M(x)$ to pseudo-residuals $r_i^{(m)}$ and updates predictions with learning rate ν.

6.2 Examples of Loss Functions

- Regression (squared error): $L = \frac{1}{2}(y - F(x))^2$.
- Classification (log-loss): $L = \log(1 + \exp(-2yF(x)))$.

6.3 Regularization and Shrinkage

To avoid overfitting:

- Small learning rate $\nu \in [0.01, 0.1]$.
- Limit tree depth (`max_depth`).
- Use subsampling (stochastic gradient boosting).

7. Modern Implementations

7.1 XGBoost

- Adds L_1/L_2 regularization on leaf weights.
- Employs second-order (Newton) approximation of loss.
- Efficient parallel computation and sparsity-aware split finding.

7.2 LightGBM

- Uses **leaf-wise** tree growth with depth limit.
- Gradient-based one-side sampling (GOSS) for speed.
- Histogram-based feature binning for memory efficiency.

7.3 CatBoost

- Handles categorical features natively via target statistics.
- Uses ordered boosting to avoid target leakage.
- Excellent performance with minimal tuning.

8. Comparison of Ensemble Methods

Method	Type	Speed	Interpretability	Accuracy
Bagging	Parallel	High	High	Moderate
Random Forest	Parallel	Moderate	Medium	High
AdaBoost	Sequential	Slow	Low	High
Gradient Boosting	Sequential	Slow	Low	Very High
XGBoost	Sequential	Fast	Low	Very High
LightGBM	Sequential	Very Fast	Low	Very High
CatBoost	Sequential	Fast	Medium	Very High

Interpretability can be improved via feature importance, SHAP values, and partial dependence plots.

9. Python Implementation: Gradient Boosting and Variants

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
```

```python
from sklearn.metrics import accuracy_score
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostClassifier

# === Load dataset ===
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# === Gradient Boosting (sklearn) ===
gb = GradientBoostingClassifier(
    learning_rate=0.05, n_estimators=200, max_depth=3
)
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_test)
print("GradientBoosting:", accuracy_score(y_test, y_pred_gb))

# === XGBoost ===
xgb_clf = xgb.XGBClassifier(
    learning_rate=0.05, n_estimators=300, max_depth=3,
    subsample=0.8, colsample_bytree=0.8, eval_metric="logloss"
)
xgb_clf.fit(X_train, y_train)
print("XGBoost:", accuracy_score(y_test, xgb_clf.predict(X_test)))

# === LightGBM ===
lgb_clf = lgb.LGBMClassifier(
    learning_rate=0.05, n_estimators=300, max_depth=-1
)
lgb_clf.fit(X_train, y_train)
print("LightGBM:", accuracy_score(y_test, lgb_clf.predict(X_test)))

# === CatBoost ===
cat_clf = CatBoostClassifier(
    iterations=300, learning_rate=0.05, depth=6,
    verbose=False
)
cat_clf.fit(X_train, y_train)
print("CatBoost:", accuracy_score(y_test, cat_clf.predict(X_test)))
```

10. Bias–Variance and Overfitting Control

- Bagging: reduces variance, increases stability.
- Boosting: reduces bias, but can overfit without shrinkage.
- Learning rate and number of estimators must be balanced.

$$\text{Test Error} = \text{Bias}^2 + \text{Variance} + \text{Noise}.$$

Boosting starts with high bias, low variance, and gradually reduces bias until overfitting increases variance.

11. Practical Tips

- Use shallow trees (depth 3–6) as weak learners.
- Always tune learning rate ν and `n_estimators`.
- Subsampling and column sampling improve generalization.
- Use early stopping to prevent overfitting.
- Normalize numeric features; encode categoricals if not using CatBoost.
- Interpret models with feature importance or SHAP values.

12. Summary

- **Ensembles** combine multiple models to reduce error.
- **Bagging** (e.g., Random Forest) reduces variance.
- **Boosting** (e.g., AdaBoost, Gradient Boosting) reduces bias.
- **AdaBoost**: weighted re-sampling with exponential loss.
- **Gradient Boosting**: stagewise additive model minimizing arbitrary differentiable loss via pseudo-residuals.
- **XGBoost, LightGBM, CatBoost**: fast, regularized, and optimized implementations.
- Tradeoffs:
  - Accuracy $\uparrow$ from Bagging $\rightarrow$ Boosting $\rightarrow$ XGBoost.
  - Interpretability $\downarrow$ with complexity.
  - Speed: LightGBM $>$ CatBoost $>$ XGBoost $>$ classic GBM.

13. Exercises

1. Derive AdaBoost's weight update rule from exponential loss.
2. Implement Gradient Boosting manually using pseudo-residuals.
3. Compare performance of Gradient Boosting vs. Random Forest.
4. Tune learning rate and number of estimators using GridSearchCV.
5. Plot feature importance for XGBoost and CatBoost models.
6. Analyze bias–variance tradeoff as learning rate varies.