Gradient Boosting – Complete Beginner-to-Advanced Revision Guide

---

1. Introduction

What is Gradient Boosting?

Gradient Boosting (GB) is an **ensemble learning technique** that builds a strong predictive model by **sequentially adding weak learners**, where each new learner is trained to **correct the errors (residuals)** of the previous ensemble.

It generalises AdaBoost by framing boosting as a **numerical optimisation problem**.

Core Intuition

> "Fit the model where it currently performs poorly."

- Models are added **one at a time**
- Each model fits the **negative gradient of the loss**
- The ensemble improves in a **stage-wise manner**

Real-World Applications

- Credit risk prediction
- Fraud detection
- Ranking systems
- Medical diagnosis
- Winning solutions in Kaggle competitions

---

2. Mathematical Foundations

2.1 Additive Model Representation

Gradient Boosting builds an additive model:

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

Where: - $h_m(x)$ are weak learners (typically shallow trees) - γ_m are step sizes

---

2.2 Loss Function

Gradient Boosting can optimise **any differentiable loss function**.

Examples: - Regression: squared error - Classification: log-loss

---

2.3 Stage-Wise Optimisation

At stage m, we add a new learner:

$$ F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) $$

2.4 Gradient Descent View

We minimise the empirical risk:

$$ \mathcal{L} = \sum_{i=1}^n L(y_i, F(x_i)) $$

Compute pseudo-residuals:

$$ r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F} $$

Train $h_m(x)$ to predict r_{im}.

2.5 Example: Squared Error Loss

Loss:

$$ L(y, F) = \frac{1}{2}(y - F)^2 $$

Gradient:

$$ r_{im} = y_i - F_{m-1}(x_i) $$

Thus, gradient boosting fits **residuals**.

---

3. Gradient Boosting for Classification

Log-Loss (Binary Classification)

$$ L(y, F) = \log(1 + e^{-2yF(x)}) $$

Gradient:

$$ r_{im} = \frac{2y_i}{1 + e^{2y_i F(x_i)}} $$

---

4. Learning Rate (Shrinkage)

Update rule with learning rate ν:

$$ F_m(x) = F_{m-1}(x) + \nu \gamma_m h_m(x) $$

- Smaller $\nu \rightarrow$ slower but more robust learning
- Requires more trees

---

5. Model Training Algorithm

---

1. Initialise model $F_0(x)$
2. For $m = 1 \ldots M$:
3. Compute residuals
4. Fit weak learner
5. Compute step size
6. Update model

---

6. Bias–Variance Tradeoff

- Reduces bias aggressively
- Risk of overfitting if too many trees
- Controlled via learning rate and depth

---

7. Model Evaluation

Metrics

Classification: - Accuracy - Precision - Recall - F1-score - ROC-AUC

Regression: - MSE - RMSE - MAE

---

8. Interpretation

- Feature importance from split gains
- Partial Dependence Plots (PDP)
- SHAP values (advanced)

---

9. Assumptions

- Weak learners can capture local structure
- Loss is differentiable
- Enough data to avoid overfitting

---

10. Common Pitfalls & Misconceptions

- ❌ Too large learning rate
- ❌ Deep trees cause overfitting
- ❌ Too many estimators without regularisation
- ❌ Misinterpreting feature importance

---

11. Python Implementation

11.1 From Scratch (Regression)

```
import numpy as np

# Simplified gradient boosting (squared loss)
def gradient_boosting(X, y, M=50, lr=0.1):
    F = np.zeros(len(y))
    models = []

    for m in range(M):
```

```
        residuals = y - F
        stump = train_tree(X, residuals)
        pred = stump.predict(X)
        F += lr * pred
        models.append(stump)

    return models
```

11.2 Using scikit-learn

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=3,
    random_state=42
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

11.3 Training Loss Visualization

```
import matplotlib.pyplot as plt

plt.plot(model.train_score_)
plt.title("Training Loss Over Iterations")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()
```

---

12. Advanced Topics

- Gradient Boosting vs AdaBoost
- XGBoost, LightGBM, CatBoost
- Second-order methods (Newton boosting)

---

13. When NOT to Use Gradient Boosting

- Very small datasets
- Extremely noisy labels
- Strict real-time inference

---

14. Best Practices

- Use small learning rate
- Limit tree depth
- Use early stopping
- Cross-validate hyperparameters

---

15. Summary

Gradient Boosting is: - Powerful - Flexible - Bias-reducing - Foundation of modern ML

It underpins: - XGBoost - LightGBM - CatBoost

End of Revision Guide