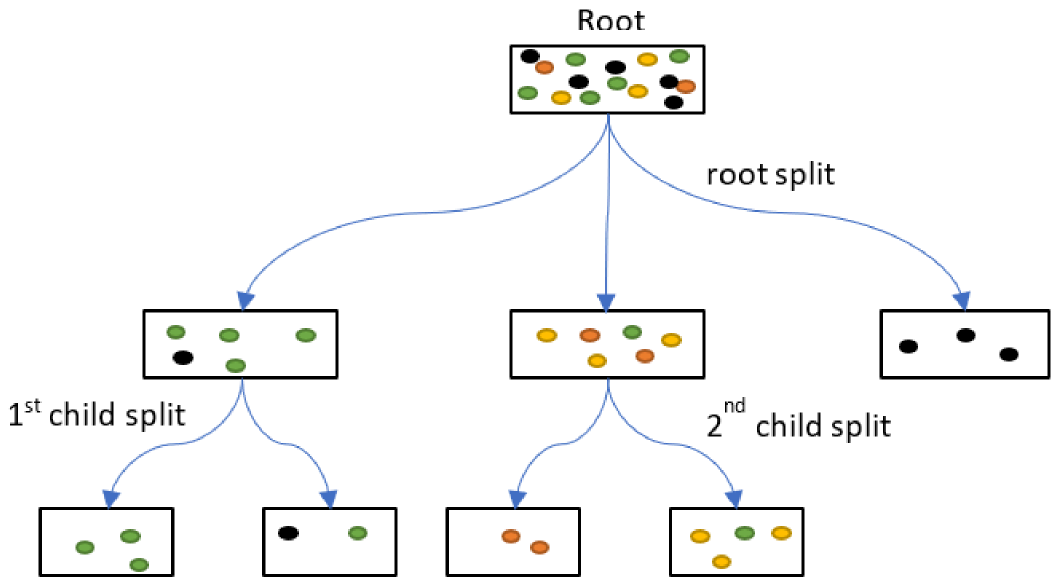
🌳 Decision Tree vs 🌱 Decision Stump



🔹 Decision Tree

A **decision tree**:

- Has **multiple splits**
- Can capture **complex patterns**
- High expressive power
- Risk of **overfitting** if deep

Example:

```
Is age > 30?
 ├── Yes → Is income > 50k?
 |         ├── Yes → Buy
 |         └── No  → Not Buy
 └── No  → Not Buy
```

🔹 Decision Stump

A **decision stump**:

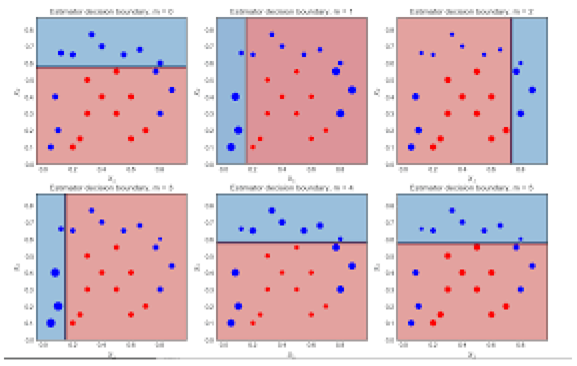
- Tree with **depth = 1**
- Only **one split**
- Extremely simple
- Weak learner by design

Example:

```
Is age > 30?
 ├── Yes → Buy
 └── No  → Not Buy
```

🤖 How They Are Used in Boosting

🟢 AdaBoost (Adaptive Boosting)



Default base learner:
✅ **Decision Stump**

Why stumps?

- AdaBoost works by:
    - Training a weak learner
    - Increasing weight of misclassified points
    - Training another weak learner on hard points
- If learners are **too strong**, boosting becomes unstable

📌 Key idea:

> Many **weak rules** + smart weighting = strong classifier

So AdaBoost usually uses:

```
depth = 1  (decision stumps)
```

🔵 Gradient Boosting

Base learner:
✅ **Shallow decision trees**

Typical depth:

```
depth = 3 to 8
```

Why not stumps only?

- Gradient Boosting fits **residual errors**
- Needs more expressive power
- Captures **feature interactions**

📌 Key idea:

Each tree corrects the mistakes of the previous model

🔁 Side-by-Side Comparison

Aspect	Decision Tree	Decision Stump
Depth	>1	1
Complexity	High	Very low
Overfitting	Possible	Almost none
Used in AdaBoost	❌ (rare)	✅ (default)
Used in Gradient Boosting	✅	❌ (usually)

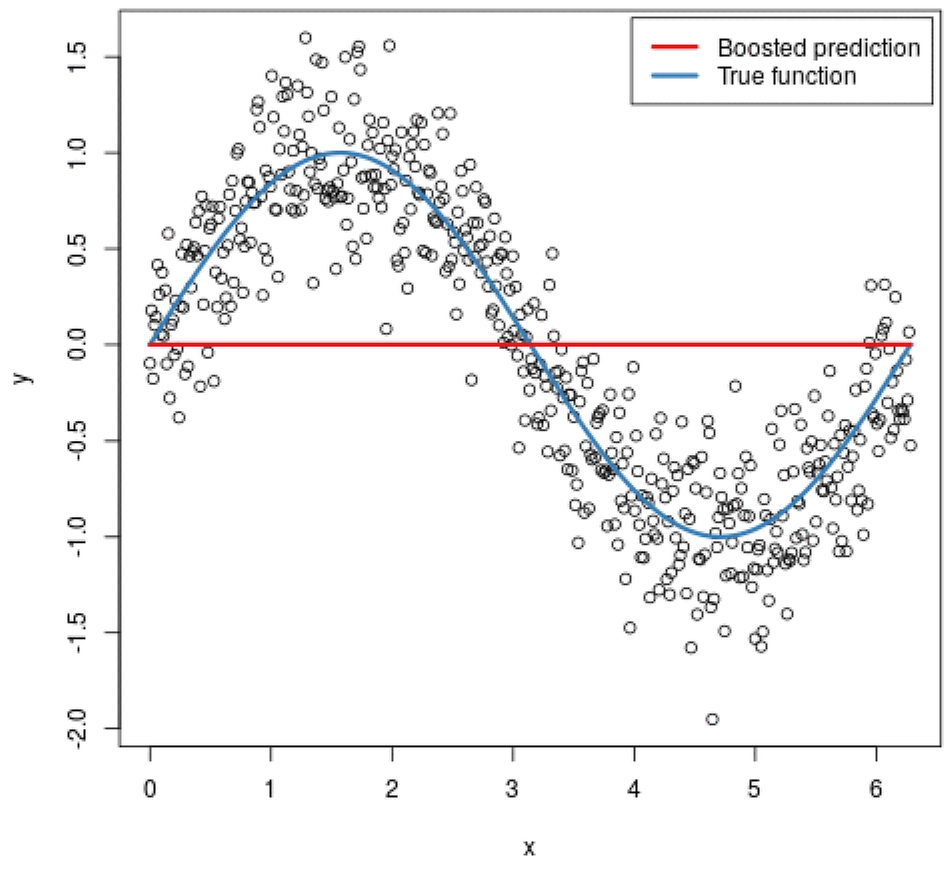
🧠 Why AdaBoost ≠ Gradient Boosting (Important Insight)

AdaBoost	Gradient Boosting
Focuses on **misclassified samples**	Focuses on **residual errors**
Sample-weight based	Gradient-based optimization
Works best with **stumps**	Needs **small trees**

Sensitive to noise	More robust

📘 GRADIENT BOOSTING — COMPLETE NOTES

⚠



1️⃣ What is Gradient Boosting? (Core Idea)

> **Gradient Boosting builds a strong model by adding many weak models sequentially, where each new model learns to correct the errors (residuals) of the previous model using gradient descent.**

In short:

- Models are added **one by one**
- Each model fixes **previous mistakes**
- Uses **gradients of loss function**

2️⃣ Why is it called “Gradient” Boosting?

Because it:

- Minimizes a **loss function**
- Uses **gradient descent** logic
- Fits models to the **negative gradient (residuals)**

📌 Residuals = direction in which prediction should move to reduce loss.

3️⃣ Why not a single Decision Tree?

Single trees:

- Overfit
- Are unstable

Gradient Boosting:

- Uses **many shallow trees**
- Each tree is weak
- Combined model is strong

4️⃣ What base learner does Gradient Boosting use?

✅ **Shallow Decision Trees**

Typical:

- Depth = 3 to 8
- NOT decision stumps usually

Why?

- Need some feature interaction
- Residual patterns are complex

5️⃣ Mathematical Form (Simple)

The final prediction function is given by:

```
F(x) = F₀(x) + η ∑ (from m = 1 to M) h (x)
```

Where:

- **F(x)** is the final strong model
- **F_0(x)** is the initial model
- **h□(x)** is the m-th weak learner
- **η** is the learning rate (shrinkage parameter)
- **M** is the total number of weak learners

Where:

- F0F_0F0 → initial prediction
- hmh_mhm → m-th tree
- η\etaη → learning rate
- MMM → number of trees

6️⃣ Gradient Boosting Algorithm (Step-by-Step)

STEP 0️⃣: Initialize model

For regression:

F0=mean of target valuesF_0 = \text{mean of target values}F0=mean of target values
For classification:

F_0 = log(p / (1 - p))

STEP 1️⃣: Compute residuals

Residuals are **negative gradient of loss**.

For regression (MSE loss):

$r_i = y_i - \hat{y}_i$

STEP2️⃣: Train a tree on residuals

- Fit a **small decision tree**
- Input = original features
- Target = residuals

STEP3️⃣: Update prediction

- $F_m(x) = F_{m-1}(x) + \eta * h_m(x)$
-
- η = learning rate (0.01 – 0.1)
-
-

STEP4️⃣: Repeat

Repeat steps 1–3 for **M trees**

7️⃣Worked Example (Regression)

Dataset

```
X : [1, 2, 3]
y : [10, 20, 30]
```

🔹 Initial prediction

F0=mean=20F_0 = \text{mean} = 20F0=mean=20

🔹 Residuals

```
r = [10-20, 20-20, 30-20]
r = [-10, 0, 10]
```

🔹 Train Tree$_1$ on residuals

Tree predicts:

```
h₁(x) = [-8, 1, 7]
```

🔹 Update prediction (η = 0.1)

```
New prediction = 20 + 0.1 × h₁
               = [19.2, 20.1, 20.7]
```

🔹 Compute new residuals

```
r = [0.8, -0.1, 9.3]
```

➡️ Tree$_2$ fits these residuals
➡️ Repeat until error is minimized

8️⃣ Classification Gradient Boosting (Binary)

Uses **Log Loss**:

Log Loss:
L = -[y * log(p) + (1 - y) * log(1 - p)]

Residuals:
r_i = y_i - p_i

Prediction Update (Sigmoid):
p = 1 / (1 + e^(-F(x)))

9️⃣ Role of Learning Rate (VERY IMPORTANT)

Learning Rate	Effect
Small (0.01)	Slow but stable
Large (0.3)	Fast but risky
Default	0.1

📌 Rule:

Smaller learning rate → more trees needed → better generalization

---

🔟 Overfitting Control Parameters

Parameter	Purpose
n_estimators	Number of trees
max_depth	Tree complexity
learning_rate	Step size
subsample	Row sampling
min_samples_leaf	Leaf size

1️⃣1️⃣ Gradient Boosting vs AdaBoost

Aspect	Gradient Boosting	AdaBoost
Error focus	Residuals	Misclassified samples
Math	Gradient descent	Weight update
Noise handling	Better	Sensitive
Base learner	Small trees	Stumps

1️⃣2️⃣ Advantages

✅ High accuracy
✅ Handles non-linear data
✅ Flexible loss functions
✅ Works for regression & classification

Gradient Boosting Parameters — Explained Clearly

You wrote:

```
gradient_params = {
    "loss": ['log_loss','deviance','exponential'],
    "criterion": ['friedman_mse','squared_error','mse'],
    "min_samples_split": [2, 8, 15, 20],
    "n_estimators": [100, 200, 500],
    "max_depth": [5, 8, 15, None, 10]
}
```

Let’s go **one by one**.

1️⃣ `loss` — *WHAT mistake are we trying to reduce?*

Think of **loss** as the **judge**.

Loss tells Gradient Boosting **what “wrong” means**.

🔹 `log_loss` / `deviance` (MOST IMPORTANT)

Where used?

✔ **Classification (binary / multi-class)**

What does it measure?

Not just *right vs wrong*, but **how confident you were**.

Example:

Actual	Predicted prob	Loss
Yes	0.51	small
Yes	0.90	very small
Yes	0.10	🔥 huge

📌 Being **confident and wrong** is punished badly.

Why Gradient Boosting loves this

Gradient Boosting:

- Predicts **probabilities**
- Improves predictions gradually

So it needs a loss that says:

"You're wrong AND overconfident → fix this urgently"

Mental picture

```
Prediction too confident but wrong

        ↓

Large loss

        ↓

Tree focuses strongly on this sample
```

Why `deviance` = `log_loss`

- Old sklearn name = `deviance`
- New name = `log_loss`
- Same meaning, same math

🔹 `exponential` (AdaBoost-style)

What does it do?

- Heavily increases importance of **misclassified samples**
- Even **one bad outlier** can dominate training

Example:

```
Correctly classified → small penalty

Wrong → exponentially large penalty
```

Why risky?

- Noise or label error → model chases it endlessly
- Overfitting risk

📌 That's why:

Gradient Boosting usually prefers `log_loss` over `exponential`

When is `exponential` OK?

✔ Clean dataset
✔ Few outliers
✔ Simple classification

🎯 SUMMARY for `loss`

Loss	Meanin g	When to use

log_loss	Probability quality	Default, safest
deviance	Same as log_loss	Old name
exponential	Hard punishment	Clean data only

2️⃣ `criterion` — *HOW does each tree split the data?*

Now imagine the tree is asking:

> “Which question should I ask to best reduce error?”

That’s what **criterion** controls.

🔹 `friedman_mse` (🔥 KEY IDEA)

This criterion is **designed for Gradient Boosting**, not normal trees.

What does it consider?

Not just:

> “Does this split make labels pure?”

But:

“Does this split help reduce the overall boosting loss?”

📌 It looks **one step ahead**.

Simple intuition

Normal tree split:

```
Split to separate Yes / No
```

Boosting-aware split:

```
Split to best reduce remaining mistakes
```

That’s the difference.

Why it’s best for Gradient Boosting

- Gradient Boosting learns **residuals**
- Friedman MSE evaluates splits using **residual improvement**
- Leads to faster convergence & better accuracy

🔹 `squared_error` / `mse`

What does it do?

Classic decision tree logic:

- Split to reduce variance
- Ignores boosting context

Why weaker for GB?

- Doesn’t consider previous models
- Treats each tree as independent

📌 Works, but not optimal.

🎯 SUMMARY for `criterion`

Criterion	Meaning	Use
friedman_mse	Boosting-aware split	✅ Best
squared_error	Normal tree split	OK
mse	Same as squared_error	Old name

🧠 Put them together (IMPORTANT)

```
loss = 'log_loss'

criterion = 'friedman_mse'
```

Means:

> “I want to improve probability predictions, and I want each tree split to focus on reducing remaining mistakes.”

This is **why Gradient Boosting is powerful**.

3️⃣ `min_samples_split` — *When is a node allowed to split?*

Minimum number of samples required to split a node.

Value	Effect
Small (2)	More splits, complex trees
Large (15–20)	Fewer splits, simpler trees

📌 Controls overfitting:

- Higher value → more regularization

4️⃣ `n_estimators` — *How many trees?*

Number of boosting stages (trees).

Trees	Behavior
100	Fast, may underfit
500	Strong, slower
Too many	Overfitting risk

📌 Rule:

> More trees = better fit **if learning rate is small**

5️⃣`max_depth` — *How complex each tree is*

Depth of each decision tree.

Depth	Meaning
1	Decision stump
3–5	Best in practice
10+	Risky (overfitting)
None	Full tree (❌ bad for GB)

📌 Important:

> Gradient Boosting prefers **shallow trees**, not deep ones.

🔁 How ALL these work together (VERY IMPORTANT)

Imagine this setting:

```
n_estimators = 500
max_depth = 5
min_samples_split = 15
loss = log_loss
criterion = friedman_mse
```

Meaning:

- Build **500 small trees**
- Each tree:
    - depth ≤ 5
    - split only if ≥ 15 samples

- Trees focus on **reducing probability error**
- Each tree corrects previous mistakes

➡️ Result: **Strong but controlled model**

🎯 Common Good Defaults (Interview-safe)

```
GradientBoostingClassifier(
    loss='log_loss',
    criterion='friedman_mse',
    n_estimators=200,
    max_depth=3,
    min_samples_split=10,
    learning_rate=0.1
)
```

🚨 Mistakes to Avoid (Important)

❌ Using `max_depth=None`
❌ Using deep trees with many estimators
❌ Using `exponential` loss on noisy data

🧠 One-line intuition per parameter

Parameter	One-liner
loss	What error am I minimizing?
criterion	How to split nodes inside trees
min_samples_split	When to stop splitting
n_estimators	How many corrections

max_depth	How powerful each correction