Tree-Based Models

Decision Trees, Random Forests, and Gradient Boosting

Tree-based algorithms are among the most practical and trusted tools in modern predictive analytics. They work much like human decision-making — breaking complex problems into smaller, logical choices to uncover patterns and insights hidden in data. These algorithms are highly versatile, performing both classification and regression tasks with ease. One of their greatest advantages is interpretability: trees reveal how different input features impact predictions, a transparency that most AI models lack.

From foundational Decision Trees to advanced ensemble methods like Random Forests and Gradient Boosting, tree-based models help businesses forecast outcomes, identify key drivers, and make smarter, data-backed decisions.

1. Algorithm Selection Framework

Understanding the differences between Decision Trees, Random Forests, and Gradient Boosting is essential for selecting the right tool for your problem. Each algorithm builds on tree-based foundations but uses different strategies to balance accuracy, stability, and interpretability.

Quick Comparison

Algorithm	Structure	Strengths	Limitations
Decision Tree	Single tree structure	Fast, interpretable, handles mixed data types	Unstable; prone to overfitting
Random Forest	Ensemble of independent trees (bagging)	Robust, minimal tuning, handles high dimensions	Less interpretable; slower inference
Gradient Boosting	Sequential ensemble correcting errors	Highest accuracy on structured data	Requires careful tuning; sensitive to overfitting

1.1 Decision Trees

At their core, decision trees mimic how humans make decisions — by asking a series of questions that gradually narrow down possibilities. The tree begins at a root question and branches out into smaller decisions until a clear outcome is reached.

How They Work

The algorithm divides data into groups that are as pure as possible — meaning each branch ideally contains observations belonging to one dominant category (for classification) or a narrow value range (for regression). To decide which feature to split on, the algorithm measures how much uncertainty (or impurity) a feature reduces.

Key Characteristics

- Fully interpretable — you can trace every prediction back through explicit rules
- Handles both numerical and categorical features well
- No assumptions about data distribution or linearity
- Unstable — small changes in data can restructure the entire tree
- Prone to overfitting unless constrained through pruning

Best Practice

Use decision trees when: (1) interpretability is non-negotiable (e.g., regulatory compliance, stakeholder explanations), (2) you need a quick baseline, or (3) you're building an ensemble method like Random Forest or Gradient Boosting.

1.2 Random Forest

Random Forest tackles a fundamental problem with decision trees: they're unstable. Change one data point, and the entire tree structure might restructure. Random Forest solves this by building not one tree, but hundreds (or thousands) of them — each slightly different — and letting them vote on the final prediction.

How It Works

Instead of trusting one tree's opinion, Random Forest creates an ensemble through:

- **Bootstrap Aggregating (Bagging):** Each tree is trained on a random sample of the data (with replacement). This means some observations appear multiple times in a tree's training set, while others don't appear at all. Because each tree sees a slightly different subset, the influence of outliers and noise is diluted.
- **Feature Randomness:** At each split, instead of considering all features, the algorithm randomly selects a subset (typically $\sqrt{n}$ features for classification, n/3 for regression). This forces trees to be diverse — they can't all rely on the same dominant feature.
- **Majority Vote:** For classification, each tree casts a vote, and the most popular class wins. For regression, predictions are averaged.

Mathematical Intuition

If each tree has an error rate ε and trees are uncorrelated, the ensemble error drops dramatically. The key is decorrelation — random sampling and feature selection ensure trees make different mistakes. Random Forest reduces variance (overfitting) without substantially increasing bias.

Key Characteristics

- Robust and stable — handles noisy data, outliers, and high-dimensional features
- Minimal tuning required — often works well with default settings
- Provides feature importance scores for interpretation
- Rarely overfits due to averaging across many trees
- Slower inference than single trees due to ensemble voting

Business Context

Random Forest is often called the "Swiss Army knife" of machine learning. It handles mixed data types, produces reliable predictions with minimal effort, and works across a wide range of business problems. Use it for credit scoring, customer segmentation, and predictive maintenance when you need a robust default choice.

1.3 Gradient Boosting

While Random Forest builds trees independently and averages them, Gradient Boosting takes a sequential approach: each new tree focuses on fixing the mistakes of all previous trees combined.

How It Works

Gradient Boosting builds trees one at a time, with each tree targeting the residual errors (the gap between current predictions and actual values). The "gradient" in Gradient Boosting comes from the fact that it's performing gradient descent in function space — each new tree approximates the negative gradient of the loss function with respect to the current predictions.

Key Characteristics

- Highest accuracy on structured/tabular data among tree-based methods
- Sequential learning — each tree corrects errors from previous trees
- Requires careful tuning of learning rate, tree depth, and number of trees
- Sensitive to overfitting if too many trees are added without regularization

When to Use

Gradient Boosting excels when maximum predictive accuracy is the priority and you have the resources for hyperparameter tuning. Common applications include loan default prediction, demand forecasting, and ranking systems (e.g., search engines, recommendation engines).

> ⚠ **Warning**
> Gradient Boosting is powerful but can easily overfit if misconfigured. Always use early stopping with a validation set, tune the learning rate carefully, and monitor validation performance during training.

2. Split Criteria

Split criteria are the metrics that determine how a decision tree evaluates and selects the best feature and threshold for partitioning a node. The choice of criterion depends on whether the task is regression or classification.

Comparison Summary

Criterion	Task Type	Strengths	Limitations
MSE (Squared Error)	Regression	Default; penalizes large errors heavily	Sensitive to outliers
Absolute Error	Regression	Robust to outliers; equal treatment of errors	Less aggressive penalty for large errors
Poisson	Regression (counts)	Designed for count data; non-negative predictions	Requires Poisson-distributed targets
Gini Impurity	Classification	Fast computation; balanced splits	Less sensitive to class distributions
Entropy	Classification	Sharper penalty for disorder; interpretable	Slightly slower; similar to Gini in practice

2.1 Regression Criteria

Mean Squared Error (MSE)

MSE is the most commonly used splitting criterion for regression trees. It measures the average of the squared differences between predicted and actual values. Because errors are squared, larger mistakes are penalized disproportionately.

When to Use

- Default choice for most regression tasks
- Effective when large deviations are particularly costly (e.g., financial forecasting, demand estimation)
- Works well when the target is well-behaved and approximately normally distributed

Limitations

- Sensitive to outliers — extreme values can dominate tree structure
- May produce suboptimal splits if the dataset contains significant noise or anomalies

Absolute Error (MAE)

Absolute error evaluates the magnitude of deviations without amplifying them. It takes the absolute difference rather than the squared difference, treating all errors proportionally.

When to Use

- Datasets with outliers or natural irregularities (real estate prices, medical costs, user-reported values)
- When robustness matters more than penalizing occasional large mistakes

Characteristics

- More robust than MSE in the presence of extreme values
- May converge more slowly due to constant gradient

Poisson Criterion

Poisson criterion is specifically designed for count-based target variables where the outcome represents event frequency (e.g., website clicks, call-center volumes, number of purchases).

When to Use

- Target variable is a count (discrete, non-negative integer)
- Underlying process follows or approximates a Poisson distribution
- Examples: number of customer complaints, daily transactions, equipment failures

Characteristics

- Ensures all predictions remain non-negative
- Models the rate of occurrence as a function of features

2.2 Classification Criteria

Gini Impurity

Gini impurity measures the degree of class mixing at a node. It aims to create child nodes that are as "pure" as possible — a node with a single dominating class has low impurity, whereas a node with an even mixture of classes has high impurity.

When to Use

- Default choice for most classification tasks
- Fast computation — preferred for large datasets
- Effective for general-purpose classification where speed and performance must be balanced

Characteristics

- Prefers splits that create strong class distinctions
- Computationally efficient
- May be less sensitive to fine-grained class distributions than entropy

Entropy (Information Gain)

Entropy quantifies the uncertainty or disorder in class labels. It identifies splits that create child nodes with clear and unambiguous class distributions.

When to Use

- When interpretability matters — entropy has a clear information-theoretic interpretation
- Datasets with fine-grained distinctions that must be captured precisely
- When maximizing information gain is conceptually important to stakeholders

Characteristics

- Slightly more sensitive to balanced or borderline class distributions than Gini
- Penalizes disorder more sharply
- Often produces similar splits to Gini in practice

Best Practice

For most practical applications, Gini and Entropy produce similar results. Start with Gini (the default) for speed. Switch to Entropy if you need a criterion with clear probabilistic interpretation or if your dataset has subtle class boundaries.

3. Regularization: Pruning in Decision Trees

Pruning is a fundamental regularization technique in decision trees, designed to prevent the model from growing excessively complex and overfitting to noise in the training data. A fully grown decision tree can perfectly fit the training set by continually splitting until all leaves are pure or contain only a few samples, but such a structure often fails to generalize well.

Most practical implementations rely on pre-pruning, where the training process is restricted through parameters that prevent unnecessary growth. Some implementations also support post-pruning, a technique applied after the full tree has been grown.

> **Note**
> Pre-pruning is the standard approach in production systems because it integrates regularization directly into the tree-building process, making it computationally efficient and highly controllable. Post-pruning (e.g., cost-complexity pruning, reduced-error pruning) is more expensive and typically reserved for research or when maximal interpretability is required.

3.1 Tree Depth Configuration

Tree depth defines how many decision layers a tree can form before reaching a leaf, and it directly shapes the balance between model complexity and accuracy.

Depth Ranges and Recommendations

Depth Range	Typical Use Case	Advantages	Considerations
Shallow (3–6)	Single trees, boosting	Fast, interpretable, resists overfitting	Limited expressive power
Medium (6–12)	Random Forest, general use	Balanced complexity and generalization	May underfit very complex patterns
Deep (15+)	Large datasets, RF with bagging	Captures complex interactions	Prone to overfitting without regularization
Unlimited (None)	Random Forest with bagging	Maximum flexibility	Unsuitable for single unregularized trees

Detailed Guidance

- **Shallow trees (3–6 levels):** Fast to train, resistant to overfitting, and easy to interpret. Ideal for quick baselines, smaller datasets, or scenarios where interpretability is mandatory (e.g., credit risk decisions requiring regulatory scrutiny). Standard choice for Gradient Boosting, where each tree acts as a weak learner.
- **Medium depth (6–12 levels):** Practical balance between predictive power and generalization. Common in Random Forest models, often extending to depth 10–20 depending on data size. Works well on typical business problems with thousands to millions of observations.
- **Deep trees (15+ levels):** Capable of modeling highly complex, non-linear relationships. Useful when extremely large datasets are available or when the data contains intricate feature interactions. Prone to memorizing noise unless strong regularization is applied.

- **Unlimited depth (None):** Allows the tree to grow until no further split is possible. Common in Random Forests where bagging and bootstrapping naturally mitigate overfitting, but generally unsuitable for single, unregularized decision trees.

3.2 Node Split Parameters

Node split parameters determine when a tree is allowed to form new branches. They provide fine-grained control over tree complexity and help prevent splits driven by statistical noise.

Minimum Samples Split

This parameter dictates the minimum number of observations required before the algorithm even considers forming a new split.

Typical Values

- Default: 2 — allows splitting as long as any impurity reduction exists
- Small datasets: 5–20 to preserve subtle patterns
- Medium datasets: 20–100 to avoid noise-driven splits
- Large datasets: 100–1000 to significantly speed up training and keep trees compact

Example

Setting min_samples_split = 50 ensures that any node containing fewer than 50 samples automatically becomes a leaf, even if a split seems mathematically possible. This prevents the model from chasing unreliable distinctions.

Minimum Samples Leaf

This parameter enforces a minimum number of observations that must appear in each child node after a split. Because it constrains both sides of the split, it provides stronger regularization than min_samples_split.

Typical Values

- Balanced datasets: 5–20
- Noisy or imbalanced datasets: 20–50

Critical Relationship

min_samples_split must be at least twice min_samples_leaf. For instance, if min_samples_split = 20 and min_samples_leaf = 15, a node with exactly 20 observations is eligible for splitting, but the split will be rejected unless both resulting children contain at least 15 observations each. Even if a split produces children with 18 and 2 samples (totaling 20), this configuration would be forbidden.

Best Practice

Use min_samples_leaf for stronger regularization. It ensures stable predictions by preventing the model from isolating extremely small or unrepresentative groups. This reduces susceptibility to outliers.

3.3 Leaf Node Configuration

Maximum Leaf Nodes

This parameter sets a ceiling on the number of terminal nodes. Rather than controlling the number of layers like max_depth, it regulates complexity by limiting how many distinct prediction regions the tree may create.

Typical Values

- Interpretable models: 10–50 leaf nodes
- Production models: 50–100 leaf nodes
- Ensemble methods (RF, GB): Often left unconstrained

Behavior

When this constraint is applied, the tree grows in a best-first manner — continually selecting the split that yields the highest impurity reduction until the maximum number of leaves is reached. While a tree with a fixed depth of 5 might theoretically generate up to 32 leaves, practical early stopping often results in fewer; however, specifying max_leaf_nodes = 20 ensures the model never exceeds 20 final regions regardless of depth.

Minimum Impurity Decrease

This parameter requires a minimum reduction in impurity before a split is accepted. A split occurs only when the weighted improvement in impurity meets or exceeds this threshold.

Typical Values

- Default: 0.0 — accepts any improvement, no matter how marginal
- Moderate regularization: 0.001 – 0.01
- Strong regularization: 0.1+

Scale Dependency

⚠ Warning

The appropriate scale for min_impurity_decrease depends on the magnitude of the target variable or impurity measure:

• In datasets normalized to small numeric ranges, even 0.01 imposes strong control
• For target values with magnitude in the millions, the threshold may need to rise to 1000 or more to have an equivalent effect

Always validate this parameter empirically by monitoring validation performance during training.

4. Ensemble Configuration

Ensemble methods like Random Forest and Gradient Boosting combine multiple trees to achieve better performance than any single tree. However, the role of ensemble parameters differs significantly between these two approaches.

4.1 Number of Trees / Estimators

The number of trees plays a central but very different role in Random Forest and Gradient Boosting, even though the parameter appears identical.

Random Forest

In Random Forest, trees are independent and built in parallel on resampled or randomized subsets of the data. Increasing the number of trees mainly improves stability through averaging: more trees reduce variance, smooth out noisy decisions, and create a more reliable ensemble.

Typical Values

- Small datasets: 100–300 trees
- Medium datasets: 300–500 trees
- Large datasets: 500–1000+ trees

Characteristics

- Benefits steadily as tree count increases, though improvements become marginal past a certain point
- Generally safe to scale to hundreds or thousands of trees, limited primarily by computational cost
- Acts as a variance reducer

Gradient Boosting

In Gradient Boosting, the number of trees represents the number of sequential correction steps. Each new tree tries to fix the mistakes of the previous one, making the model more accurate but also more sensitive to overfitting.

Typical Values

- With small learning rate (0.01): 500–2000 trees
- With moderate learning rate (0.1): 100–500 trees
- With large learning rate (0.3): 50–200 trees

Characteristics

- Cannot simply add trees indefinitely — must be regulated through learning rate, early stopping, or other regularization
- Higher tree count increases predictive power only when paired with appropriately small learning rate
- Acts as a complexity amplifier

> **Note**
> In Random Forest, more trees = more stability. In Gradient Boosting, more trees = more complexity and potential overfitting. Always use early stopping with Gradient Boosting to prevent adding too many trees.

4.2 Maximum Features

Maximum features controls how many input features a tree can consider at each split. Its purpose differs between Random Forest and Gradient Boosting.

Random Forest

Limiting the number of features (such as using √n for classification or n/3 for regression) is essential for creating diversity across trees. Because Random Forest relies on the principle that many weakly correlated trees average into a strong learner, reducing feature availability helps each tree explore different splitting patterns.

Typical Values

- Classification: √n (square root of total features) — e.g., 10 features if n = 100
- Regression: n/3 (one-third of total features) — e.g., 33 features if n = 100

Effect

- Low values increase diversity but may weaken individual trees
- High values make trees more similar, reducing ensemble benefit

Gradient Boosting

In Gradient Boosting, subsampling features is not required for diversity (since trees are built sequentially), but it can significantly reduce overfitting by preventing every boosting iteration from relying on the same dominant variables.

Typical Values

- Conservative: 0.5 (half of features)
- Moderate: 0.7–0.8
- Full features: 1.0 (use all features)

Effect

- Lower values encourage variation across boosting rounds
- Makes the model more robust to noise
- Reduces risk of chasing the same patterns repeatedly

4.3 Bootstrap (Random Forest)

Bootstrap determines whether trees in a Random Forest receive training data through sampling with replacement. Enabling bootstrap ensures that each tree sees a slightly different

subset of the data, which introduces natural variability and significantly reduces correlation between trees.

When Enabled (Default)

- Each tree trains on a random sample (with replacement) of approximately 63% of the data
- Produces diverse, weakly correlated trees
- Enables Out-of-Bag (OOB) error estimates — an internal validation method

When Disabled

- All trees see the full dataset
- Trees become stronger but more correlated
- May be useful for very small datasets but reduces ensemble benefits

4.4 Early Stopping (Gradient Boosting)

Early stopping is a critical mechanism in Gradient Boosting that halts the training process when additional trees no longer improve validation performance. Because boosting is a sequential learning method, continuing to add trees eventually leads to overfitting as the model begins to learn noise rather than meaningful structure.

How It Works

- Monitor validation loss during training
- If loss does not improve for N consecutive rounds (patience), stop training
- Typical patience values: 10–50 rounds

Benefits

- Prevents overfitting by stopping at the point of maximum generalization
- Improves computational efficiency by avoiding unnecessary boosting rounds
- Automatically balances model complexity

Best Practice

Always use early stopping with Gradient Boosting. Set a high maximum number of trees (e.g., 1000–2000) and let early stopping find the optimal point. This is more reliable than guessing the right number of trees upfront.

5. Evaluation Metrics

Evaluating tree-based models requires selecting metrics that align with the problem type and business objectives. The choice of metric depends on whether the task is regression or classification, and on the specific characteristics of the dataset.

5.1 Regression Metrics

Metric	Strengths	Limitations
MSE	Penalizes large errors heavily	Sensitive to outliers; amplifies extreme deviations
MAE	Robust to outliers; interpretable in target units	Less aggressive penalty for large errors

Mean Squared Error (MSE)

MSE evaluates the average of the squared differences between predicted and actual values. Because errors are squared, larger mistakes are penalized more heavily than smaller ones.

When to Use

- Default metric for most regression tasks
- When large deviations are particularly undesirable (financial forecasting, demand estimation)
- When the target is approximately normally distributed

Limitations

- Very sensitive to outliers — extreme values disproportionately inflate the metric
- Less interpretable than MAE (squared units)

Mean Absolute Error (MAE)

MAE measures the average of the absolute differences between predictions and actual outcomes. It treats all errors proportionally without excessively punishing large deviations.

When to Use

- Datasets with outliers or natural irregularities (real estate prices, medical costs)
- When consistency and robustness matter more than penalizing occasional large mistakes
- When interpretability is important (same units as target variable)

Best Practice

If outliers are present, prefer MAE over MSE.

5.2 Classification Metrics

Metric	Strengths	Limitations
Accuracy	Overall correctness	Misleading for imbalanced datasets
Precision	Minimizes false positives	Ignores false negatives

Metric	Strengths	Limitations
Recall	Minimizes false negatives (sensitivity)	Ignores false positives
F1 Score	Balances precision and recall	Less interpretable than individual metrics

Accuracy

Accuracy reflects the proportion of total predictions that were correct. It provides a straightforward measure of overall performance.

When to Use

- Balanced datasets where all classes are equally important
- Simple binary or multi-class problems without significant class imbalance

Limitations

> ⚠ **Warning**
> Accuracy can be highly misleading for imbalanced datasets. For example, if 95% of samples belong to Class A, a model that always predicts Class A achieves 95% accuracy while learning nothing useful. In such cases, use precision, recall, or F1 score instead.

Precision

Precision measures how many of the predicted positive cases are truly positive. It answers: "Of all instances we predicted as positive, how many were actually positive?"

When to Use

- When false positives carry significant consequences
- Examples: spam detection (don't flag legitimate emails), loan approval (don't incorrectly reject applicants)

Recall (Sensitivity)

Recall focuses on the ability of the model to identify all actual positive cases. It answers: "Of all actual positive instances, how many did we correctly identify?"

When to Use

- When missing a positive instance is particularly harmful
- Examples: disease detection (don't miss sick patients), fraud detection (don't miss fraudulent transactions)

F1 Score

F1 score provides a balanced single measure by taking the harmonic mean of precision and recall. It becomes especially valuable when dealing with imbalanced classes or when both false positives and false negatives must be minimized.

When to Use

- Imbalanced datasets where both precision and recall matter
- When a single metric is needed for model comparison or hyperparameter tuning
- When accuracy alone does not reveal true model performance

Best Practice

For classification tasks, always examine precision, recall, and F1 score alongside accuracy. For imbalanced datasets, accuracy is often meaningless — focus on precision/recall trade-offs based on business costs of false positives vs false negatives.

6. Practical Recommendations

The following guidelines synthesize best practices from the sections above into a concise decision framework.

Algorithm Selection

- Start with Random Forest as your default — it's robust, requires minimal tuning, and works well across most problems
- Use Gradient Boosting when maximum accuracy is the priority and you have resources for hyperparameter tuning
- Reserve single Decision Trees for scenarios requiring full interpretability or as weak learners in ensembles

Split Criteria

- Use MSE (regression) or Gini (classification) as defaults — they work well for most cases
- Switch to MAE or Absolute Error if your dataset contains significant outliers
- Use Poisson criterion only when the target is truly count-based and follows a Poisson distribution

Regularization (Pruning)

- Always constrain tree depth: 3–6 for Gradient Boosting, 10–20 for Random Forest
- Set min_samples_leaf between 5–20 for balanced datasets, 20–50 for noisy data
- Use max_leaf_nodes for interpretable models requiring a fixed number of decision regions
- Tune min_impurity_decrease carefully — its scale depends on target magnitude

Ensemble Configuration

- Random Forest: Use 300–500 trees, enable bootstrap, set max_features to $\sqrt{n}$ (classification) or n/3 (regression)

- Gradient Boosting: Set high max trees (1000–2000), use small learning rate (0.01–0.1), always enable early stopping
- Monitor validation performance during training to detect overfitting early

Evaluation

- Regression: Report MSE/MAE for error magnitude and R^2 for explained variance
- Classification (balanced): Use accuracy as primary metric
- Classification (imbalanced): Focus on precision, recall, and F1 score; ignore accuracy
- Always validate on held-out test data — never tune on the test set

Best Practice

Tree-based models are production-ready when configured properly. Random Forest provides reliability with minimal effort. Gradient Boosting provides maximum accuracy with careful tuning. Both are widely deployed across industries for mission-critical applications.

7. Underfitting – Overfitting Trade-off

Achieving strong model performance is a critical stage in any machine learning pipeline, where the goal is not just to learn patterns from training data, but to ensure those patterns hold on unseen data. While high training accuracy may appear promising, real-world success depends on balancing underfitting and overfitting.

7.1. Decision Trees

Decision Trees partition the feature space through a series of binary splits, building an interpretable hierarchy of rules. An unconstrained tree will grow until every leaf is pure — perfectly memorising the training data. Conversely, a tree that is pruned or constrained too aggressively will produce rules too coarse to capture real patterns. The key lever is controlling how deeply the tree is allowed to grow.

Aspect	▽ UNDERFITTING		△ OVERFITTING	
	Causes	Actions to Fix	Causes	Actions to Fix
Maximum Tree Depth	Tree is too shallow to represent the multi-level feature interactions present in the data	Allow the tree to grow deeper; start with no depth constraint and prune back based on validation performance	Tree grows without any depth limit, creating leaves that correspond to individual training points	Set a maximum depth; cross-validate to find the shallowest depth that preserves validation performance
Minimum Samples per Leaf	Minimum leaf size is set too high, forcing the	Reduce the minimum leaf size to allow the tree to make finer	Leaves are allowed to contain single	Require each leaf to contain a meaningful minimum number of

	tree to generalise too broadly and miss local patterns	distinctions in dense regions of the data	examples, making every training point its own decision region	samples to ensure rules generalise
Split Threshold	The requirement for a split to occur is too strict, causing the tree to stop splitting before useful structure is found	Lower the split threshold so the tree can continue partitioning where moderate improvement is available	Tree splits on features with negligible impurity gain, creating branches that model noise	Require a minimum impurity improvement before making a split; apply post-training pruning to remove weak branches
Feature Quality	Available features lack discriminative power; the tree cannot find splits that meaningfully separate classes or reduce error	Engineer more informative features; bring in additional data sources that carry the missing signal	Noisy or irrelevant features are available as split candidates and the tree exploits them	Pre-select features using domain knowledge or feature importance; remove features that are mostly noise
Pruning	Pruning is applied too aggressively, cutting back branches that encoded real patterns in the training data	Reduce pruning strength; tune the pruning criterion using validation data rather than training data alone	No pruning is applied to the fully grown tree, leaving all noise-fitting branches intact	Apply post-pruning (cost-complexity pruning) to remove branches whose contribution does not justify their complexity

7.2. Random Forest

Random Forests reduce the high variance of individual decision trees by averaging the predictions of many trees, each trained on a random bootstrap sample and a random subset of features. While the ensemble mechanism naturally combats overfitting compared to a single tree, underfitting can arise if individual trees are too weak or too few in number. Overfitting is still possible when trees are very deep and the dataset is small.

Aspect	▽ UNDERFITTING		△ OVERFITTING	
	Causes	Actions to Fix	Causes	Actions to Fix
Number of Trees in Ensemble	Too few trees means the ensemble average is noisy and unstable, with high variance across	Increase the number of trees; ensemble performance continues to improve until a plateau is reached	More trees alone do not cause overfitting — this is rarely the source of the problem in random forests	Focus on tree depth and feature controls rather than reducing the number of trees

	different training runs			
Individual Tree Depth	Trees are constrained to be very shallow, making each individual learner too weak to capture useful patterns	Allow trees to grow deeper; the ensemble averages away much of the variance introduced by deeper individual trees	Fully grown, very deep trees memorise their bootstrap samples; averaged noise still degrades generalisation	Limit maximum tree depth; require a minimum number of samples in leaves to prevent overly specific rules
Feature Subsampling per Split	Too few features are considered at each split, causing trees to make uninformed decisions based on insufficient information	Increase the number of features considered at each split to give each tree access to more signal	Considering all features at every split makes trees correlated with each other, reducing ensemble diversity	Use a square-root or logarithmic fraction of features per split to ensure trees are diverse and decorrelated
Bootstrap Sampling	Training all trees on the full dataset without resampling removes the variance-reducing benefit of the ensemble	Ensure bootstrap sampling is enabled so each tree sees a different random subset of the training data	Individual trees overfit their bootstrap samples; out-of-bag error diverges from in-sample accuracy	Monitor out-of-bag error as an unbiased estimate of generalisation; use it to guide depth and leaf constraints
Class Imbalance	Minority classes appear infrequently across bootstrap samples, causing the ensemble to underfit them systematically	Re-weight classes so minority examples have greater influence; use stratified sampling when constructing bootstraps	After resampling, individual trees can overfit artificially amplified minority patterns	Combine class weighting with depth constraints; evaluate using class-specific metrics such as recall and F1

7.3. Gradient Boosting

Gradient Boosting builds an ensemble sequentially, with each new tree correcting the residual errors of the previous ones. Because it trains greedily on the same data, it is far more prone to overfitting than Random Forests — especially when boosting continues for too many rounds or with complex trees. Underfitting arises when individual trees are too weak or the learning process is halted too early. Controlling the number of rounds, learning rate, and tree complexity simultaneously is essential.

Aspect	▽ UNDERFITTING		△ OVERFITTING	
	Causes	Actions to Fix	Causes	Actions to Fix

Number of Boosting Rounds	Too few rounds mean the model has not had enough iterations to correct its residual errors, leaving high bias	Increase the number of rounds; use a held-out validation set to find the round at which error is minimised	Continuing to boost beyond the optimal round causes later trees to fit the noise in residuals, not the signal	Apply early stopping: monitor validation error and halt boosting when it stops improving
Learning Rate	Learning rate is too high, causing each tree to overcorrect residuals and produce unstable, high-variance updates	Reduce the learning rate and compensate by increasing the number of boosting rounds correspondingly	A very low learning rate combined with excessive rounds causes the model to slowly overfit the training residuals	Use a small learning rate paired with early stopping so the model takes small steps but stops at the right time
Individual Tree Complexity	Trees are too shallow (e.g., stumps) and cannot capture the higher-order interactions needed to reduce residuals effectively	Allow trees to be moderately deeper so each boosting step can correct more complex patterns in the residuals	Deep trees in each boosting step capture noise in the residuals and amplify it across subsequent rounds	Keep individual trees shallow (moderate depth); increase the minimum samples required to make a split
Regularization	Overly strong penalties on tree complexity or leaf values prevent meaningful trees from being added at each step	Reduce regularization strength; verify that training error is decreasing as more trees are added	No regularization allows the ensemble to grow arbitrarily complex without penalty	Apply L1 and L2 penalties on leaf weights; use minimum gain thresholds to prevent trivial splits
Row and Column Subsampling	Too little data or too few features per tree result in trees that are too noisy and weak to correct residuals well	Use a higher fraction of rows and features per tree to ensure each tree has enough information to learn from	Using all rows and all features every round produces highly correlated trees that collectively overfit	Subsample rows and features randomly at each round to introduce stochasticity and reduce tree correlation
Training Data Volume	Insufficient data means gradient estimates are unreliable and the model cannot distinguish signal from noise	Collect more training data; feature engineering can help the model extract more signal from limited samples	Small datasets with many boosting rounds give the model ample opportunity to memorise every training example	Reduce the number of rounds; apply stronger regularization; always evaluate on a held-out set or via cross-validation

Have a question or suggestion?
Reach us at support@tvaritam.ai