Model Comparison of Decision Tree: Random Forest vs Boosting

Dataset Used: Boston Housing

- **Goal**: Predicting **median home value (MEDV)**.
- **Preprocessing**:
- Dropped missing or irrelevant columns.
- Exploratory analysis with correlation heatmap showed:
    - Strong **negative correlation** between `lstat` and `medv`
    - Strong **positive correlation** between `rm` and `medv`

Decision Tree (Base Learner)

Overview:

A **decision tree** is a supervised learning algorithm used for both regression and classification. It works by recursively splitting the data based on feature values to minimize prediction error.

How It Works:

- At each node, choose the best feature and split point to divide the dataset.
- Continue splitting recursively until stopping criteria is met (e.g., min samples, max depth).
- Final predictions are made at the **leaf nodes**.

Algorithm Steps:

1. Start with the full dataset.
2. For each feature, find the best split point by minimizing the loss (e.g., MSE for regression).
3. Choose the feature and split point with the lowest error.
4. Split the dataset and repeat steps 2-3 recursively.
5. Stop when a condition is met (e.g., min leaf size or max depth).
6. Make predictions using the average target value in each leaf (regression).

Strengths:

- Simple and interpretable.
- Handles both numerical and categorical data.

Limitations:

- Prone to overfitting.
- Sensitive to small data changes.

Random Forest

Overview:

Random Forest is an **ensemble method** that builds multiple decision trees and aggregates their results to improve prediction accuracy and reduce overfitting.

How It Works:

- Combines **bagging** (bootstrap sampling) and **random feature selection**.
- Each tree is trained independently on a different bootstrap sample.
- For each split in a tree, a random subset of features is used.
- Final prediction is the **average** (for regression) or **majority vote** (for classification).

Algorithm Steps:

1. Fix the final train-test split.
2. For each tree (B trees):
3. Randomly select subset of features.
4. Create bootstrap sample from training data.
5. Train a decision tree on the sample.
6. Predict on train and test data.
7. Average the predictions across all trees.
8. Compute metrics (MSE, R^2).

Results:

- **Final Train MSE**: 11.20
- **Final Test MSE**: 12.33
- **Final Train R^2**: 0.87
- **Final Test R^2**: 0.83

Visuals:

- Train/Test MSE & R^2 per tree.
- Histogram of Test R^2 scores.

Pros:

- Reduces variance.
- Robust to overfitting.
- Can handle high-dimensional data.

Cons:

- Slower to predict due to multiple trees.
- Less interpretable.

Boosting (Gradient Boosting)

Overview:

Boosting is a **sequential ensemble method** where each tree tries to fix the errors of the previous trees by predicting the **residuals**.

How It Works:

- Starts with a base prediction (e.g., 0).
- Sequentially adds new trees trained on the residual errors.
- Each tree's contribution is scaled by a **learning rate**.
- Final prediction is the sum of all tree contributions.

Algorithm Steps:

1. Initialize predictions (e.g., all 0).
2. For each iteration:
3. Compute residual = actual - predicted.
4. Train a tree on residuals.
5. Predict on train/test data.
6. Update predictions: previous + learning_rate * current_tree_output.
7. Final prediction = sum of all adjusted tree outputs.
8. Evaluate performance using MSE and R^2.

Results:

- **Final Train MSE**: 3.83
- **Final Test MSE**: 7.81
- **Final Train R^2**: 0.955
- **Final Test R^2**: 0.893

Visuals:

- R^2 and MSE over iterations.
- Actual vs Predicted plot.
- Residual plot and histogram.

Pros:

- Reduces both bias and variance.
- Learns complex patterns.

- Tuning (learning rate, depth) helps control overfitting.

Cons:

- Computationally intensive.
- Sensitive to hyperparameters.

Comparative Analysis

Metric Comparison:

Metric	Random Forest	Boosting	Better Performer
Final Train MSE	11.20	3.83	Boosting
Final Test MSE	12.33	7.81	Boosting
Final Train R^2	0.87	0.955	Boosting
Final Test R^2	0.83	0.893	Boosting

Algorithmic Comparison:

Feature	Random Forest	Boosting
Learning Style	Parallel (Independent Trees)	Sequential (Corrects Residuals)
Data Sampling	Bootstrap Samples	Full Dataset
Feature Selection	Random Subsets per Tree	Full Feature Set
Error Focus	Average Predictions	Focus on Residuals
Overfitting Tendency	Lower (if trees are weak learners)	Higher (if too many deep trees)
Bias Handling	Primarily reduces variance	Reduces bias and variance
Interpretability	Moderate (can view tree structure)	Higher (due to stage-wise learning)
Speed	Faster Training	Slower Training
Accuracy	Moderate	Higher (with tuning)

Final Recommendation

- If your priority is **accuracy** and **learning from complex patterns**, go with **Boosting**.
- If your goal is **speed**, **stability**, and **robustness with less tuning**, use **Random Forest**.
- **Decision Tree** alone is easy to interpret but often overfits and lacks the power of ensemble methods.

Conclusion:

Boosting outperforms both Decision Trees and Random Forest in this use case by offering lower error and higher explanatory power. However, it requires careful tuning and longer training times.