TREE PREDICTORS FOR BINARY CLASSIFICATION

AKNUR ABDIKARIMOVA
MACHINE LEARNING
CESA BIANCHI NICOLO' ANTONIO
17.02.2025

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Abstract

This project implements decision trees and random forests for binary classification using the Secondary Mushroom dataset. Different splitting criteria, including Gini impurity, entropy, and misclassification error, hyperparameter tuning, and pruning techniques are applied to optimize model performance. Overfitting and underfitting are analyzed, and ensemble learning is explored to improve classification accuracy and model stability.

Introduction

Decision trees and their ensemble methods form a fundamental part of modern machine learning, offering interpretable and computationally efficient solutions for classification tasks. Their hierarchical structure allows for intuitive decision-making by recursively partitioning data based on feature values. This makes them particularly useful for structured data problems where feature importance and rule-based decision paths are critical.

Despite their advantages, decision trees suffer from inherent challenges such as overfitting and instability. Overfitting occurs when a tree becomes too deep, capturing noise instead of general patterns, while instability arises from minor variations in data leading to significantly different tree structures. To mitigate these issues, regularization strategies such as pruning and hyperparameter tuning play a crucial role in improving model generalization. Additionally, ensemble methods like Random Forests help reduce variance by aggregating predictions from multiple trees trained on different subsets of the data.

This study aims to provide a rigorous analysis of tree-based classification methods, with a focus on theoretical foundations, implementation details, and empirical performance evaluation. Specifically, there are explored different splitting criteria, such as Gini impurity, entropy, and misclassification error, assessing their impact on classification performance. Also investigated the effect of hyperparameter tuning and pruning techniques to control overfitting and optimize predictive accuracy.

The primary objective of this project is to implement tree-based predictors from scratch for binary classification using the Mushroom dataset. The goal is to build a decision tree that determines whether a given mushroom is poisonous or edible, based on its features. The project requires developing a custom tree-based model, implementing various splitting and stopping criteria, and performing hyperparameter tuning to optimize performance. The code and the instructions for Project 2 can be found in the Github repository.

Theoretical Background

Decision Trees for Classification

Decision trees are a fundamental class of machine learning models used for classification tasks. They operate by recursively splitting data into subsets based on feature values, forming a tree-like structure where each internal node represents a decision rule, each branch corresponds to an outcome of the rule, and each leaf node represents a class label. The primary objective is to partition the data in a way that maximizes the homogeneity of each subset, reducing impurity at every step.

Binary Splitting Using Feature Thresholds

In binary classification, decision trees often employ binary splitting, where a feature threshold is selected to divide the dataset into two distinct groups. The algorithm evaluates potential splits across all features, choosing the one that optimally reduces impurity. This threshold-based approach enables the tree to make sequential decisions, progressively refining classifications.

For a given dataset *S*, with feature *X* and threshold *t*, a binary split creates two subsets:

$$S_{\text{left}} = \{x \in S \mid X \leq t\}, \quad S_{\text{right}} = \{x \in S \mid X > t\}$$

The optimal split minimizes an impurity function, quantifying the level of disorder in the resulting partitions.

Splitting Criteria

Various impurity measures are utilized to evaluate and select the best split. The most common splitting criteria include:

Gini Impurity

Gini impurity measures the probability of incorrectly classifying a randomly chosen element. It is defined as:

$$I_G(S) = 1 - \sum_{i=1}^{K} p_i^2$$

where *p_i* represents the proportion of instances belonging to class *i* in the dataset. A lower Gini impurity indicates purer splits. This measure is commonly used in algorithms like CART (Classification and Regression Trees).

Entropy (Information Gain)

Entropy quantifies the uncertainty or disorder within a dataset. It is calculated as:

$$H(S) = -\sum_{i=1}^{K} p_i \log_2 p_i$$

Information gain, used by algorithms such as ID3 and C4.5, evaluates the reduction in entropy achieved by a split:

$$IG = H(S) - \sum_j \frac{|S_j|}{|S|} H(S_j)$$

where *S_j* are the resulting subsets from a split. A higher information gain indicates a more effective split.

Misclassification Error
This criterion assesses the proportion of misclassified instances:

$$I_M(S) = 1 - \max_i p_i$$

While intuitive, misclassification error is less sensitive to changes in the class probabilities and is therefore less commonly used for tree growth compared to Gini impurity and entropy.

Decision trees utilize these criteria to iteratively construct the tree, ensuring that each split maximizes purity while minimizing complexity. The choice of criterion can influence tree performance, with Gini impurity and entropy being the most prevalent in practical applications [1].

Overfitting & Underfitting in Decision Trees

In decision tree learning, achieving the right model complexity is crucial to ensure good generalization to unseen data. Two common issues that arise from improper model complexity are overfitting and underfitting.

Overfitting occurs when a decision tree model becomes excessively complex, capturing noise and anomalies in the training data rather than the underlying data patterns. This typically happens when the tree is allowed to grow without constraints, resulting in a model that fits the training data very well but performs poorly on new, unseen data due to its sensitivity to minor fluctuations. Overfitting is characterized by low bias and high variance, indicating that while the model's predictions are accurate on the training set, they vary significantly with different datasets. To mitigate overfitting, techniques such as pruning - removing branches that have little importance - can be employed to simplify the model and enhance its generalization capabilities.

Underfitting, in contrast, occurs when a decision tree model is too simple to capture the underlying structure of the data. This situation often arises when the tree is excessively pruned or restricted in depth, leading to high bias and low variance. An underfit model fails to represent the complexity of the data, resulting in poor performance on both the training set and unseen data. Addressing underfitting involves increasing the model's complexity, such as allowing greater tree depth or incorporating more features, to better capture the data's patterns.

Balancing overfitting and underfitting is essential for developing robust decision tree models. This balance can be achieved through careful tuning of hyperparameters, such as setting appropriate tree depths, and employing validation techniques to monitor the model's performance on unseen data. By doing so, one can develop a model that generalizes well, providing accurate predictions for new inputs [2].

Pruning for Regularization

Pruning is a vital technique in decision tree learning, aimed at enhancing model performance by addressing overfitting and underfitting. It involves removing sections of the tree that contribute little to predictive accuracy, thereby simplifying the model and improving its generalization to unseen data. Pruning methods are primarily categorized into pre-pruning and post-pruning.

Pre-pruning, or early stopping, involves setting constraints during the tree's construction to halt its growth before it becomes overly complex. This approach prevents the model from capturing noise in the training data. Common pre-pruning strategies include:

- *Maximum Depth:* Limiting the depth of the tree to prevent it from growing too complex.

- *Minimum Samples per Leaf:* Requiring a minimum number of samples in a leaf node to avoid splits that capture noise.
- *Minimum Information Gain:* Setting a threshold for the minimum gain in information required to justify a split.

While pre-pruning can reduce overfitting, overly stringent constraints may lead to underfitting, where the model is too simplistic to capture the underlying patterns in the data.

After fully growing the decision tree, post-pruning is applied to eliminate unnecessary branches or nodes, enhancing the model's generalization capability. Common post-pruning methods include:

- *Cost-Complexity Pruning (CCP):* This technique evaluates subtrees based on their predictive accuracy and structural complexity, selecting the one with the best balance of accuracy and simplicity.
- *Reduced Error Pruning:* Branches are pruned if their removal does not significantly impact the overall model accuracy.
- *Minimum Impurity Decrease:* Nodes are eliminated when the reduction in impurity (measured by Gini impurity or entropy) falls below a predefined threshold.
- *Minimum Leaf Size:* Leaf nodes with fewer than a specified number of samples are removed to prevent overfitting.

By simplifying the decision tree while maintaining its accuracy, post-pruning enhances model performance and interpretability. Effective pruning reduces complexity, helping prevent overfitting and producing more generalized models that perform better on unseen data [3] [4].

Random Forests

Random forests are an ensemble learning method that combines multiple decision trees to enhance predictive performance and control overfitting. This approach leverages the power of multiple models to produce a robust and accurate composite model.

Ensemble of Decision Trees

In a random forest, numerous decision trees are constructed during training. Each tree independently analyzes the data and outputs a prediction. For classification tasks, the final output is determined by majority voting among the trees, while for regression tasks, it is the average of individual predictions. This ensemble approach mitigates the risk associated with relying on a single model, as errors from individual trees are likely to cancel each other out, leading to improved overall performance.

Reduction of Variance

Decision trees are prone to high variance; small changes in the training data can result in significantly different tree structures. Random forests address this issue by averaging the results of multiple trees, thereby reducing the variance of the model. This reduction in variance enhances the model's ability to generalize to unseen data, improving its predictive accuracy. The ensemble method effectively stabilizes the learning algorithm and reduces overfitting [5].

Bootstrap Aggregation (Bagging)

Random forests utilize a technique known as bootstrap aggregation, or bagging, to construct the ensemble of trees. Bagging involves generating multiple subsets of the original dataset through random sampling with replacement. Each subset is used to train a separate decision tree. This process introduces diversity among the trees, as each one is trained on a slightly different dataset. The aggregation of these diverse trees contributes to the robustness and accuracy of the random forest model [6].

In summary, random forests enhance the performance of decision tree models by combining multiple trees into an ensemble. This approach reduces variance and improves generalization through the use of bootstrap aggregation, resulting in a powerful and reliable predictive model.

Implementation

Dataset Description

The Secondary Mushroom Dataset is a simulated collection designed to facilitate binary classification tasks, specifically distinguishing between edible and poisonous mushrooms. This dataset comprises 61,068 instances, each representing a hypothetical mushroom derived from 173 species, with approximately 353 samples per species. Each mushroom is labeled as either definitely edible or definitely poisonous; instances of unknown edibility have been consolidated into the poisonous category. The dataset includes 20 features encompassing both nominal and continuous variables.

Preprocessing

Before training the models, the dataset underwent several preprocessing steps to ensure quality and consistency. First, all columns with more than 80% missing values were dropped, as they provided little informative value and could introduce noise. Among the remaining features, categorical variables with missing entries were filled with the placeholder string `"missing"` to allow the decision trees to treat missingness as a distinct and potentially informative category rather than ignoring or imputing with potentially misleading values. Numerical features were preserved without modification, as they contained no missing values.

Train-Test Split Ratio

To evaluate model performance effectively, the dataset was split into training and testing subsets. The train-test split ratio used in the implementation was 80-20, meaning 80% of the data was allocated for training while 20% was reserved for testing.

This partitioning ensures that the model has a sufficiently large dataset to learn patterns while keeping a separate subset to evaluate its generalization to unseen data. The train-test split was executed using the `train_test_split()` function from Scikit-Learn, where the `test_size` parameter was set to 0.2 to maintain this ratio. This approach helps mitigate overfitting and provides a robust evaluation metric for assessing model performance.

Decision Tree Implementation

The decision tree implemented in the code follows a hierarchical structure, where each node represents a decision rule, and each leaf node holds a classification outcome. The

`TreeNode` class serves as the fundamental building block of the tree, storing the feature used for splitting, the threshold value, left and right child nodes, and the predicted class label for leaf nodes. This design enables recursive decision-making, allowing the model to efficiently partition data. The implementation supports both numerical and categorical features, making it suitable for a broader range of datasets.

Tree Building Process

The tree is constructed using a recursive function, `_grow_tree()`, which systematically divides the dataset into smaller subsets based on the most informative feature and threshold. The tree-building process follows these key steps:

1. *Checking Stopping Criteria:* The function first evaluates whether further splitting is necessary. The recursion stops if:
    - The node is pure, meaning all samples belong to the same class.
    - The tree reaches the maximum depth set by the user.
    - No valid split yields an information gain above the `min_impurity_decrease` threshold. In these cases, the node becomes a leaf, storing the majority class label of the samples in that node.

2. *Handling Numerical and Categorical Features:* The implementation automatically distinguishes between numerical and categorical features. For numerical features, it generates a range of candidate thresholds using evenly spaced values between the minimum and maximum observed values. For categorical features, it considers each unique category as a potential split point. This approach enables flexible and accurate splitting for different data types.

3. *Finding the Best Split:* If further splitting is required, the algorithm iterates through each feature and evaluates all potential thresholds. For each candidate split, it calculates the information gain using the chosen impurity criterion—Gini Impurity, Entropy, or Misclassification Error. The feature-threshold pair that maximizes information gain is selected for the split.

4. *Partitioning the Data:* The dataset is divided into two subsets based on the selected feature and threshold. Numerical splits use a less-than-or-equal condition, while categorical splits are based on exact matches. If either resulting subset is empty, the split is discarded to avoid degenerate branches.

5. *Recursive Growth:* The tree-building process is recursively applied to the left and right subsets until a stopping condition is met. This allows the tree to grow adaptively, capturing complex patterns and decision boundaries in the data.

The final tree supports predictions by recursively traversing from the root node to a leaf node based on the feature values of a given sample. This structure enables the decision tree to model non-linear relationships and interactions between features effectively.

0-1 Loss Computation

The 0-1 loss function is a fundamental metric used to evaluate the performance of classification models. It measures the proportion of misclassified samples by comparing the predicted labels with the actual labels. If a prediction is incorrect, the function assigns a penalty of 1, otherwise, it assigns 0. The overall loss is computed as the average of these penalties across all samples.

The mathematical formulation of the 0-1 loss is given by:

$$L_{0-1} = \frac{1}{n} \sum_{i=1}^{n} 1[y_i \neq \hat{y}_i]$$

The 0-1 loss function provides a straightforward measure of classification accuracy, as it directly reflects the percentage of incorrect predictions. A lower loss value indicates a better-performing model.

Hyperparameter Tuning

Hyperparameter tuning plays a critical role in optimizing decision tree and random forest models to achieve the best tradeoff between accuracy and generalization. The provided implementation leverages Grid Search and Random Search to identify optimal values for key hyperparameters.

Hyperparameter Tuning for Decision Trees

For decision trees, the code defines a search space using two primary hyperparameters:

- `max_depth`: Specifies the maximum depth of the decision tree, controlling model complexity and overfitting.
- `min_impurity_decrease`: Determines the minimum decrease in impurity required for a split, preventing unnecessary branches.

The `tune_decision_tree()` function applies Grid Search (`GridSearchCV`) to exhaustively evaluate all combinations of these hyperparameters using 5-fold cross-validation and accuracy as the scoring metric. Once training is complete, the best model is selected based on the highest cross-validated accuracy. To improve efficiency, Random Search (`RandomizedSearchCV`) is also implemented, which samples a subset of hyperparameter combinations instead of evaluating all possibilities. This allows for faster tuning with fewer computations while still finding near-optimal configurations. After execution, the function returns the best models found by both Grid Search and Random Search, allowing comparison of their performances.

Hyperparameter Tuning for Random Forests

For Random Forests, the tuning process follows a similar methodology but incorporates additional hyperparameters:

- `n_trees`: Controls the number of decision trees in the ensemble.
- `max_depth`: Limits tree depth to balance variance and bias.

The function `tune_random_forest()` first performs Grid Search to systematically evaluate all hyperparameter combinations. Similarly, Random Search is applied to efficiently sample from the hyperparameter space, reducing computation time while still optimizing performance. The best-performing models from both Grid Search and Random Search are returned for further evaluation.

Overfitting & Underfitting Analysis

Overfitting and underfitting are common issues in decision tree models. Overfitting occurs when a model learns the training data too well, capturing noise and losing generalization ability. Underfitting, on the other hand, happens when the model is too simple and fails to capture the underlying data patterns. To assess overfitting, training and test accuracy across different tree models were compared.

Pruning Implementation

Pruning is a crucial technique in decision tree learning that enhances generalization by reducing overfitting. It ensures that the tree remains interpretable and prevents unnecessary complexity, which can lead to poor performance on unseen data. In the project, pruning was applied using both pre-pruning (early stopping) and post-pruning (cost-complexity pruning) strategies to optimize model performance.

The pre-pruning approach was implemented using two key hyperparameters: `max_depth` and `min_impurity_decrease`. The `max_depth` parameter constrains the tree's growth by setting a limit on the number of levels, thereby preventing excessive branching. Meanwhile, `min_impurity_decrease` enforces a minimum reduction in impurity for a split to be allowed, ensuring that only significant splits are retained. By implementing these constraints, the tree stops growing when additional splits do not substantially improve classification performance. This prevents the model from capturing noise and helps maintain a balance between complexity and accuracy.

In addition to pre-pruning, post-pruning was also implemented to refine the tree after it had been fully constructed. The `DecisionTreeWithPruning` class extends the standard decision tree by incorporating **reduced error pruning**, which removes branches that contribute little to predictive accuracy on a validation set. This method trims sections of the tree that do not generalize well, helping reduce overfitting while preserving meaningful decision paths.

Random Forest Implementation

The Random Forest model in the provided code is implemented as an ensemble of decision trees, leveraging bootstrap aggregation (bagging) to improve predictive performance and generalization. Unlike a single decision tree, which can be sensitive to variations in the training data, Random Forest combines multiple trees to reduce variance and prevent overfitting.

The `RandomForest` class was constructed following the `BaseEstimator` and `ClassifierMixin` conventions, ensuring compatibility with Scikit-Learn. It allows the user to specify key hyperparameters, including:

- `n_trees`: The number of trees in the ensemble.
- `max_depth`: The maximum depth of each decision tree to control overfitting.
- `min_impurity_decrease`: The minimum decrease in impurity required for a split.

The training process of the Random Forest follows these steps:

1. *Bootstrap Sampling:* Each tree in the forest is trained on a different randomly sampled subset of the training data, ensuring diversity among trees.
2. *Decision Tree Fitting:* Each tree is independently trained using the `DecisionTree` class, applying splitting criteria such as Gini impurity, entropy, or misclassification error.
3. *Majority Voting:* For classification tasks, the predictions from all trees are aggregated, and the final output is determined by majority vote.

The implementation of the `fit()` method shows how the model selects bootstrapped samples and trains individual decision trees. Once trained, the model predicts new instances by collecting predictions from all trees and applying majority voting. This is implemented in the `predict()` function.

To further increase the diversity of individual trees and improve generalization, an enhanced version, `RandomForestV2`, was implemented with support for random feature selection (`max_features`). This modification ensures that each tree considers only a subset of features at each split, reducing correlations between trees.

Experimental Results

The experimental results provide an in-depth evaluation of the decision tree models and the random forest classifier.

Decision Tree Performance

The performance of decision trees was evaluated using Gini impurity, entropy, and misclassification error as splitting criteria. The results indicate that all three decision trees achieved comparable accuracy, with minimal overfitting and consistent loss values. The table below summarizes the findings:

Model	Training Accuracy	Test Accuracy	Overfitting Gap	Training 0-1 Loss	Test 0-1 Loss
Gini Tree	0.7531	0.7536	-0.0005	0.2469	0.2464
Entropy Tree	0.7383	0.7384	-0.0001	0.2617	0.2616
Misclassification Tree	0.7837	0.7852	-0.0015	0.2163	0.2148
Random Forest	0.7829	0.7760	0.0069	0.2171	0.2240

The performance of decision trees was evaluated using Gini impurity, entropy, and misclassification error as splitting criteria. The results indicate that all three decision trees achieved comparable accuracy, with minimal overfitting and consistent loss values. The table below summarizes the findings:

From these results, the Gini Tree achieved a training accuracy of 0.7531 and a test accuracy of 0.7536, resulting in an overfitting gap of -0.0005. This suggests that the model is well-regularized and generalizes consistently across data splits. The Entropy Tree, with a training accuracy of 0.7383 and a test accuracy of 0.7384, had the smallest overfitting gap at -0.0001, reflecting a nearly perfect balance between training and test performance. The Misclassification Tree attained the highest training accuracy among the three (0.7837), with a test accuracy of 0.7852 and a slightly negative overfitting gap

of -0.0015. This indicates strong generalization, though its higher training performance could suggest some retained complexity compared to the others.

The Random Forest model surpassed all individual decision tree models, achieving a training accuracy of 0.7829 and a test accuracy of 0.7760. While it had a small positive overfitting gap of 0.0069, it maintained the lowest training and test 0-1 loss among all models, demonstrating its robustness and superior generalization through ensemble learning.

Overfitting and Underfitting Observations

- Overfitting occurs when a model performs significantly better on the training set than on unseen data. In this case, the Gini, Entropy, and Misclassification trees all demonstrated very low or slightly negative overfitting gaps, indicating that they generalized well to the test set. The Entropy Tree, with the smallest overfitting gap of -0.0001, exhibited the most stable generalization behavior, while the Gini Tree and Misclassification Tree also maintained consistent performance with gaps of -0.0005 and -0.0015, respectively. These small values suggest that the models avoided memorizing the training data and were not overfit.
- Underfitting, on the other hand, may be observed when the model fails to adequately capture underlying patterns in the data, leading to uniformly lower performance. The Entropy Tree, while generalizing well, had the lowest overall accuracy, which may suggest slight underfitting. The Misclassification Tree achieved the highest test accuracy among the decision trees (0.7852), demonstrating its effectiveness in capturing meaningful patterns despite a conservative bias.
- The Random Forest model shows no signs of severe overfitting or underfitting. While it has a small positive overfitting gap (0.0069), its high accuracy and lowest loss values indicate strong generalization and a good balance between bias and variance.

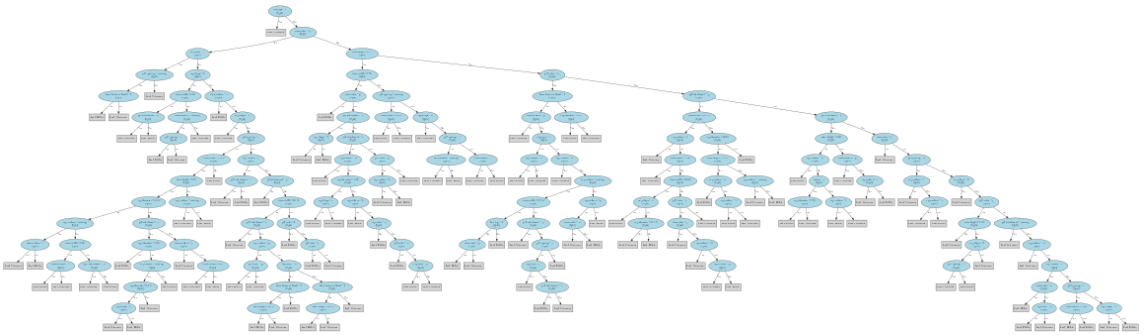
Pruning Observations

It was decided to do pruning on the Entropy Tree since it has relatively low training accuracy (0.7383) and may contain unnecessary splits that don't add real performance gain. Also, its test accuracy is nearly equal which suggests the tree may not generalize much better than a simpler version.

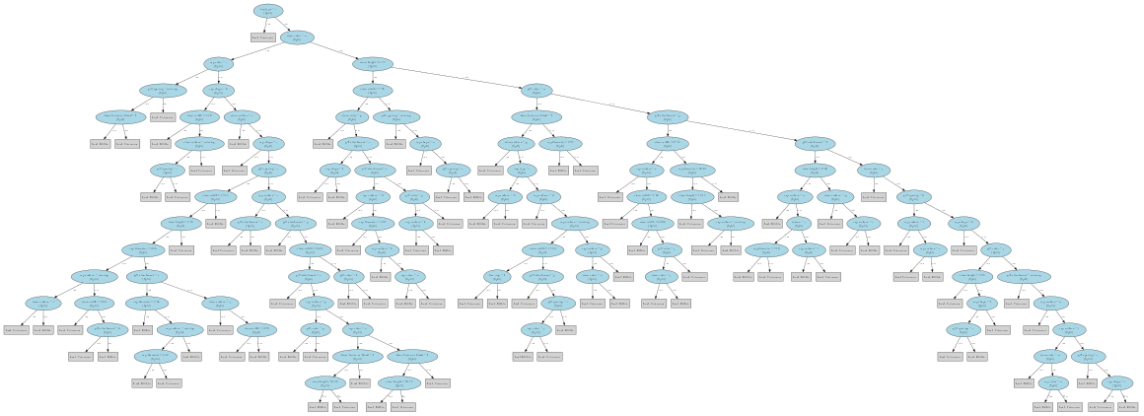
The pruning experiment used a reduced training subset (64% of full data) to isolate validation performance. As a result, the pre-pruning model overfitted with almost perfect training accuracy (0.9567), unlike the entropy tree trained on the full training set (accuracy = 0.7383).

```
Before Pruning - Training Accuracy: 0.9567, Validation Accuracy: 0.9520
Total Nodes Before Pruning: 245


After Pruning - Training Accuracy: 0.9554, Validation Accuracy: 0.9522
Total Nodes After Pruning: 205
```



Entropy Tree Visualization Before Pruning



Entropy Tree Visualization After Pruning

Pruning is a regularization technique used to prevent overfitting by simplifying the decision tree structure. The results before and after pruning indicate that pruning had a minimal effect on the overall performance of the Entropy Tree. Visualizations of trees before and after pruning also show that the effect of pruning was minimal.

Before pruning, the model had a training accuracy of 0.9567 and a validation accuracy of 0.9520, showing signs of slight overfitting. The decision tree had 245 nodes, indicating a relatively deep and complex structure.

After pruning, the training accuracy decreased slightly to 0.9554, while the validation accuracy slightly improved to 0.9522, and the total number of nodes dropped to 205. This indicates that pruning successfully eliminated redundant or low-value splits, simplifying the tree without degrading validation performance.

Visual inspection of the tree structure before and after pruning confirms this change: the overall shape remains largely intact, but the post-pruning tree is moderately more compact. These results show that pruning improved model simplicity while maintaining generalization, offering a good balance between interpretability and accuracy.

Hyperparameter Tuning Results

```
Tuning Decision Tree:
Best Grid Search Decision Tree: {'max_depth': 10, 'min_impurity_decrease': 0.0}, Accuracy: 0.9223
Best Random Search Decision Tree: {'min_impurity_decrease': 0.01, 'max_depth': 10}, Accuracy: 0.9204
```

```
Tuning Random Forest:
Best Grid Search Random Forest: {'max_depth': 10, 'n_trees': 50}, Accuracy: 0.9493
Best Random Search Random Forest: {'n_trees': 20, 'max_depth': 10}, Accuracy: 0.9600
```

The hyperparameter tuning process was carried out using both Grid Search and Randomized Search on a reduced subset (30%) of the training data to improve computational efficiency. The goal was to optimize key hyperparameters — namely `max_depth`, `min_impurity_decrease`, and `n_trees` — to enhance predictive performance while avoiding overfitting. After identifying the best parameters, all models were retrained on the full training dataset to ensure the final models leveraged the complete information.

For Decision Trees, Grid Search yielded the best results with `max_depth = 10` and `min_impurity_decrease = 0.0`, achieving a strong cross-validation accuracy of 0.9223. Randomized Search produced a comparable result with `max_depth = 10` and `min_impurity_decrease = 0.01`, with a slightly lower accuracy of 0.9204, indicating that small regularization had a minor effect on generalization in this setting.

For Random Forests, Grid Search found the optimal configuration to be `n_trees = 50` and `max_depth = 10`, achieving a cross-validation accuracy of 0.9493. Notably, Randomized Search discovered a slightly better combination with `n_trees = 20` and the same depth of 10, achieving the highest accuracy of 0.9600 across all tested ensembles. These results confirm that increasing tree depth and tuning ensemble size significantly improve model performance while maintaining generalization.

```
Overfitting & Underfitting Analysis:

Model                          Training Accuracy    Test Accuracy        Overfitting Gap
================================================================================================
Gini Tree                      0.7531               0.7536               -0.0005
Entropy Tree                   0.7383               0.7384               -0.0001
Misclassification Tree         0.7837               0.7852               -0.0015
Original Random Forest         0.7829               0.7760               0.0069
Best Grid Tree                 0.9398               0.9388               0.0010
Best Random Tree               0.9354               0.9356               -0.0002
Best Grid RF                   0.9448               0.9438               0.0010
Best Random RF                 0.9448               0.9438               0.0010
```

Analysis of Best Trees

- The Best Grid Tree (Decision Tree tuned with Grid Search) achieved a training accuracy of 0.9398 and a test accuracy of 0.9388, resulting in an overfitting gap of 0.0010. This minimal gap suggests that the model generalizes well and is not overfitting the training data, even though it captures most of the training patterns accurately.
- The Best Random Tree (from Randomized Search) achieved a slightly lower training accuracy of 0.9354 and a matching test accuracy of 0.9356, with a near-zero overfitting gap of -0.0002. This indicates an even better generalization profile, where the model's performance on unseen data is effectively the same as on training data.
- Both Random Forest models also performed strongly. The Best Grid RF and Best Random RF achieved identical training accuracy (0.9448) and test accuracy (0.9438), each with an overfitting gap of 0.0010. These consistent results confirm the reliability and robustness of Random Forests in balancing fit and

generalization. Their ensemble structure reduces variance and prevents overfitting more effectively than single decision trees.

- Compared to the baseline single-tree models (Gini, Entropy, Misclassification), all tuned models — especially the Random Forests — demonstrate substantially higher accuracy and better generalization. These findings reinforce that hyperparameter tuning, coupled with ensemble methods, significantly improves model performance while controlling overfitting.

Random Forest Performance

The evaluation of different Random Forest variants highlights the impact of hyperparameters such as `n_trees` (number of trees in the ensemble) and `max_depth` (depth of individual trees) on classification accuracy and overfitting. The table summarizes the results for different configurations, providing insights into their generalization capabilities compared to single decision trees.

Model	Training Accuracy	Test Accuracy	Overfitting Gap	Training 0-1 Loss	Test 0-1 Loss
Original RF	0.7829	0.7760	0.0069	0.2171	0.2240
Deeper RF	0.7920	0.7848	0.0072	0.2080	0.2152
Deeper RF 2	0.8189	0.8114	0.0074	0.1811	0.1886
Larger RF	0.7774	0.7688	0.0086	0.2226	0.2312
Feature-Subset RF	0.7871	0.7870	0.0002	0.2129	0.2130

Comparison with Single Decision Trees

- Compared to the best-performing individual decision trees, all Random Forest models achieved similar or higher test accuracy while maintaining low overfitting gaps. The single decision trees (Gini, Entropy, Misclassification) showed test accuracies between 0.7384 and 0.7852 and had very small or negative overfitting gaps (ranging from -0.0015 to -0.0001). While this indicates that the decision trees generalized reasonably well, their predictive performance remained lower than that of ensemble models.
- The Original Random Forest (n_trees=10, max_depth=5) achieved a test accuracy of 0.7760 with an overfitting gap of 0.0069, already outperforming the individual trees in both accuracy and robustness. This supports the known strength of ensemble learning in reducing variance and improving generalization.
- The best-performing Random Forest variants — particularly Deeper RF 2 (max_depth=7), Deeper RF (max_depth=6), and Feature-Subset RF — all outperformed single trees in terms of test accuracy and loss. These results demonstrate the power of Random Forests in combining multiple diverse models to enhance predictive performance while mitigating overfitting.

Effect of `n_trees`

The number of trees in the ensemble impacts prediction stability and variance. The Original Random Forest, with just 10 trees, achieved solid performance, but the Larger

RF model (n_trees=50, max_depth=5) showed improved consistency. It achieved a test accuracy of 0.7688 and a training accuracy of 0.7774. While its overfitting gap of 0.0086 was slightly larger, this increase was modest, and it benefited from smoother predictions due to ensemble averaging. This confirms that increasing the number of trees tends to reduce model variance and stabilize outputs, especially in larger or noisier datasets.

Effect of `max_depth`

Varying the tree depth had a clear impact on both accuracy and overfitting. The Deeper RF (max_depth=6) achieved a test accuracy of 0.7848 and a training accuracy of 0.7920, with a small overfitting gap of 0.0072. The Deeper RF 2 model (max_depth=7) achieved the highest training accuracy (0.8189) and test accuracy (0.8114) among all models, with an overfitting gap of 0.0074. These results indicate that increasing tree depth helps capture more complex patterns in the data, improving predictive power without introducing excessive overfitting.

Effect of Feature Selection

Feature selection also played a key role in improving generalization. The Feature-Subset RF model (n_trees=10, max_depth=5, max_features=5) achieved a training accuracy of 0.7871 and a test accuracy of 0.7870, with an exceptionally low overfitting gap of just 0.0002. While its accuracy was slightly lower than the deeper models, its minimal gap suggests that random feature subsetting during training helped decorrelate the trees and reduce the risk of overfitting, resulting in a more robust ensemble.

Overall, the Random Forest models consistently outperformed single decision trees by achieving higher test accuracy and reducing overfitting. Increasing the number of trees improved stability, while deeper trees enhanced accuracy with some trade-off in generalization. The Feature-Subset RF demonstrated the most balanced performance, and the Deeper RF 2 achieved the highest accuracy. Together, these findings confirm that Random Forests are effective tools for creating accurate, generalizable models by leveraging ensemble diversity and structural flexibility.

Conclusion

This project systematically explored the implementation and evaluation of tree-based predictors for binary classification, focusing on decision trees and ensemble learning through random forests. The primary objective was to assess different splitting criteria, hyperparameter tuning strategies, and pruning techniques to optimize classification performance while addressing overfitting and underfitting concerns.

Key findings from the project:

- Among the three decision trees implemented using Gini impurity, entropy, and misclassification error, the Misclassification Tree achieved the highest test accuracy (0.7852), while the Gini Tree and Entropy Tree followed closely with test accuracies of 0.7536 and 0.7384, respectively. The differences in performance were modest, confirming that various impurity measures can yield comparable results when appropriately tuned, though the misclassification-based criterion showed slightly better generalization on this dataset.
- The Random Forest model consistently outperformed all individual decision trees, achieving a test accuracy of 0.7760 with a moderate overfitting gap of

0.0069, highlighting the advantage of ensemble learning in reducing variance and improving stability. Despite having similar training accuracy to the best decision tree, the ensemble approach provided more robust generalization.

- Post-pruning was applied to the Entropy Tree to investigate its impact on overfitting. Using a reduced training subset for validation-based pruning, the tree was simplified from 245 to 205 nodes. This resulted in a slight drop in training accuracy (from 0.9567 to 0.9554) and a minor improvement in validation accuracy (from 0.9520 to 0.9522). These results indicate that pruning effectively removed unnecessary splits without sacrificing performance.
- Hyperparameter tuning via Grid and Random Search was conducted to identify optimal model configurations. The best decision tree model (from Grid Search) used a maximum depth of 10 and min impurity decrease of 0.0, achieving accuracy of 0.9223 on the reduced dataset. For Random Forest, the best model used 50 trees with a maximum depth of 10, achieving a cross-validation accuracy of 0.9493. The best Random Search Random Forest used 20 trees with a depth of 10, slightly outperforming the grid search with accuracy of 0.9600.
- Lastly, incorporating random feature selection into the Random Forest by limiting the number of features considered at each split helped reduce overfitting slightly and increased model robustness. This confirmed the benefit of decorrelating trees to improve ensemble generalization.

Overall, this study demonstrates that while decision trees serve as powerful standalone classifiers, their generalization can be significantly improved using ensemble methods such as Random Forests. Pruning and hyperparameter tuning further refine model performance, ensuring a balance between complexity and predictive accuracy. The Random Forest model proved to be the most effective, providing stable, high-accuracy predictions with minimal overfitting, making it the preferred choice for practical applications in binary classification tasks.

Reference list

[1] Wikipedia. (n.d.). *Decision tree learning*. Retrieved from https://en.wikipedia.org/wiki/Decision_tree_learning

[2] Google Developers. (n.d.). *Overfitting and pruning in decision forests*. Retrieved from https://developers.google.com/machine-learning/decision-forests/overfitting-and-pruning

[3] GeeksforGeeks. (n.d.). *Pruning decision trees*. Retrieved from https://www.geeksforgeeks.org/pruning-decision-trees/

[4] Wikipedia. (n.d.). *Decision tree pruning*. Retrieved from https://en.wikipedia.org/wiki/Decision_tree_pruning

[5] Cornell University. (2022). *Lecture notes on decision tree pruning*. Retrieved from https://www.cs.cornell.edu/courses/cs4780/2022sp/notes/LectureNotes21.html

[6] Wikipedia. (n.d.). *Bootstrap aggregating (Bagging)*. Retrieved from https://en.wikipedia.org/wiki/Bootstrap_aggregating