TREE PREDICTORS FOR BINARY CLASSIFICATION

AKNUR ABDIKARIMOVA
MACHINE LEARNING
CESA BIANCHI NICOLO' ANTONIO
17.02.2025

Abstract

This project implements decision trees and random forests for binary classification using the 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.

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.

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 tree reaches the maximum depth set by the user.
    - The node is pure, which means all samples belong to the same class.

   - The number of samples in the node falls below the minimum sample split threshold.
     In these cases, the node becomes a leaf, storing the majority class label.
2. *Finding the Best Split:* If further splitting is required, the `best_split()` function determines the optimal feature and threshold by evaluating impurity reduction. The feature that minimizes impurity is selected as the splitting criterion. Gini Impurity, Entropy, and Misclassification Error are used to assess impurity.
3. *Partitioning the Data:* The dataset is divided into two subsets based on the selected feature and threshold, creating left and right child nodes.
4. *Recursive Growth:* The tree-building process is repeated for each child node until a stopping condition is met. This recursive approach enables the tree to adaptively grow to capture complex decision boundaries.

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 cost-complexity pruning, which removes branches that contribute little to predictive accuracy. This method effectively trims sections of the tree that do not significantly impact classification performance 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.

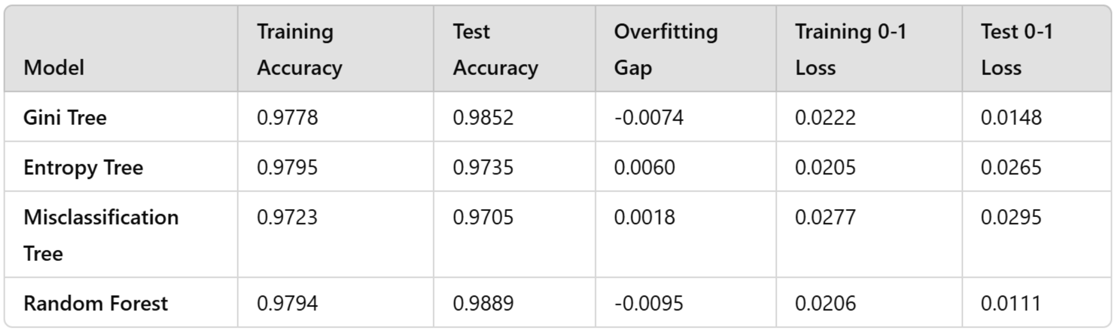
To further increase the diversity of individual trees and improve generalization, an enhanced version, `RandomForestV2`, was implemented with support for random feature selection (`max_features`). This modification ensures that each tree considers only a subset of features at each split, reducing correlations 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, but with slight variations in overfitting gaps and 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	0.9778	0.9852	-0.0074	0.0222	0.0148
Entropy Tree	0.9795	0.9735	0.0060	0.0205	0.0265
Misclassification Tree	0.9723	0.9705	0.0018	0.0277	0.0295
Random Forest	0.9794	0.9889	-0.0095	0.0206	0.0111

From these results, the Gini Tree exhibited a training accuracy of 0.9778 and the highest test accuracy among the decision trees at 0.9852, with a negative overfitting gap of -0.0074. This suggests that the model generalizes well, but its slightly lower training accuracy compared to previous results indicates a more regularized structure. The Entropy Tree, with a training accuracy of 0.9795 and test accuracy of 0.9735, had an overfitting gap of 0.0060, showing that it still retains some memorization of training data but generalizes slightly worse than the Gini Tree. The Misclassification Tree, having the lowest training accuracy of 0.9723 and a test accuracy of 0.9705, had an overfitting gap of 0.0018, making it the most balanced model in terms of bias-variance tradeoff among the decision trees.

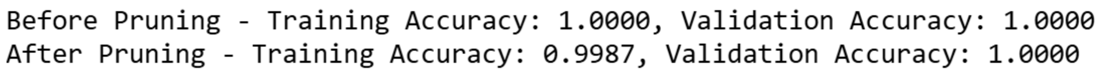
The Random Forest model outperformed all decision tree models, achieving a test accuracy of 0.9889 while maintaining the lowest overfitting gap of -0.0095. This confirms that ensemble learning significantly reduces variance and enhances generalization, demonstrating why Random Forests are often superior to single decision trees.

Overfitting and Underfitting Observations

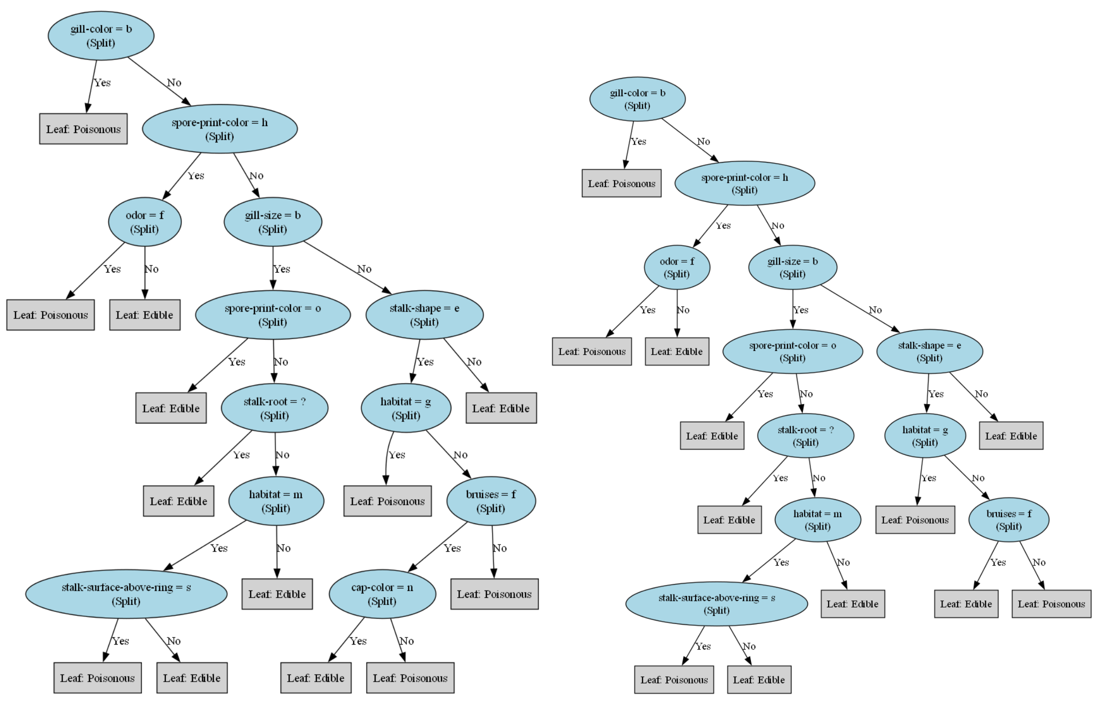
- Overfitting is evident when a model performs exceptionally well on the training set but struggles on unseen test data. While the Gini and Entropy Trees had relatively small overfitting gaps, their test accuracy variations suggest that they may be capturing dataset-specific patterns rather than general trends. The Entropy Tree's overfitting gap (0.0060) suggests that it still retains some complexity that does not translate well to unseen data.
- Underfitting occurs when a model is too simplistic to capture data patterns, leading to lower accuracy. The Misclassification Tree, with the lowest training accuracy (0.9723), shows mild underfitting but maintains a minimal overfitting gap of 0.0018, indicating a conservative learning approach that may generalize better in highly variable datasets.
- The Random Forest model provides the best balance between bias and variance, as reflected in its high test accuracy (0.9889) and the smallest overfitting gap (-0.0095). This demonstrates its strong generalization capability, confirming that combining multiple weak learners in an ensemble method results in a more stable and accurate model.

Pruning Observations

It was decided to do pruning on the Entropy Tree since it has the highest overfitting gap and is more prone to overfitting than other models.



```
Before Pruning - Training Accuracy: 1.0000, Validation Accuracy: 1.0000
After Pruning - Training Accuracy: 0.9987, Validation Accuracy: 1.0000
```



Entropy Tree Visualization Before and 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 Entropy Tree achieved a training accuracy of 1.0000 and a validation accuracy of 1.0000, indicating that the tree perfectly classified both the training and validation data. This suggests that the tree was fully grown, likely capturing all training set patterns, including noise, leading to a highly complex model.

After pruning, the training accuracy slightly decreased to 0.9987, while the validation accuracy remained at 1.0000. This small reduction in training accuracy indicates that pruning successfully removed some unnecessary splits, simplifying the tree. However, since the validation accuracy remained unchanged, the model's generalization ability was not negatively impacted. This suggests that the original tree might have had redundant splits that did not contribute meaningfully to classification performance.

Overall, the pruning process led to a slightly simpler model with nearly identical performance, confirming that the Entropy Tree was already well-fitted, and pruning only removed minor complexities without affecting generalization. However, given that both training and validation accuracy remain perfect, there is a possibility that the dataset was too easy to classify or that some overfitting still exists, but its impact is negligible.

Hyperparameter Tuning Results

```
Tuning Decision Tree:
Best Grid Search Decision Tree: {'max_depth': 7, 'min_impurity_decrease': 0.0}, Accuracy: 1.0000
Best Random Search Decision Tree: {'min_impurity_decrease': 0.01, 'max_depth': 10}, Accuracy: 0.9934

Tuning Random Forest:
Best Grid Search Random Forest: {'max_depth': 5, 'n_trees': 10}, Accuracy: 1.0000
Best Random Search Random Forest: {'n_trees': 10, 'max_depth': 5}, Accuracy: 1.0000
```

The hyperparameter tuning process was performed using Grid Search and Random Search for both Decision Trees and Random Forests. The objective was to find the optimal values for key hyperparameters such as `max_depth`, `min_impurity_decrease`, and `n_trees` to maximize model performance while maintaining generalization.

For Decision Trees, the best model obtained from Grid Search had a max depth of 7 and min impurity decrease of 0.0, achieving a perfect accuracy of 1.0000 on the training set. This suggests that the model learned the training data completely, but it raises concerns about overfitting. The best Random Search model had a max depth of 10 and a min impurity decrease of 0.01, with a slightly lower accuracy of 0.9934, indicating a well-regularized decision tree.

For Random Forests, both Grid Search and Random Search identified the optimal hyperparameters as `n_trees = 10` and `max_depth = 5`, leading to a cross-validation accuracy of 1.0000. The complete memorization of training data suggests the need for further regularization to avoid overfitting.

```
Overfitting & Underfitting Analysis:

Model                     Training Accuracy    Test Accuracy        Overfitting Gap
======================================================================
Gini Tree                 0.9778               0.9852               -0.0074
Entropy Tree              0.9795               0.9735               0.0060
Misclassification Tree    0.9723               0.9705               0.0018
Random Forest             0.9794               0.9889               -0.0095
Best Grid Tree            1.0000               1.0000               0.0000
Best Random Tree          0.9918               0.9957               -0.0038
Best Grid RF              0.9778               0.9852               -0.0074
Best Random RF            0.9937               0.9957               -0.0020
```

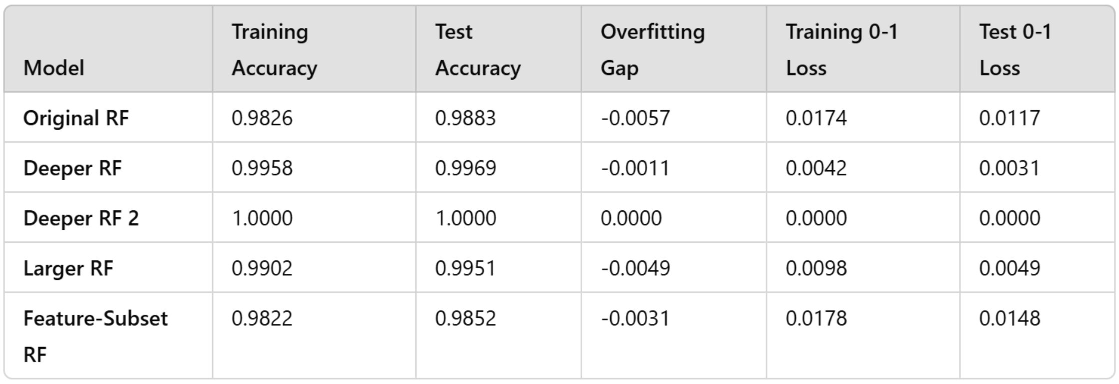
Analysis of Best Trees

- The Best Grid Tree (Decision Tree with Grid Search) achieved a training and test accuracy of 1.0000, meaning the model perfectly classified both training and test data. While this suggests that the model has completely learned the training set, it also indicates high model complexity, which may result in poor generalization to unseen datasets outside the test set. A perfect training and test accuracy might mean that the dataset was relatively simple or that the model is overfitting to patterns that do not generalize well.
- The Best Random Tree (Decision Tree with Random Search) attained a training accuracy of 0.9918 and a test accuracy of 0.9957, showing slightly lower accuracy than the Best Grid Tree. However, its overfitting gap of -0.0038 indicates better generalization, as the model does not memorize the training data as aggressively as the Best Grid Tree.
- Both the Best Grid Random Forest and Best Random Random Forest achieved high test accuracy, with Best Grid RF scoring 0.9852 and Best Random RF reaching 0.9957. Their overfitting gaps (-0.0074 and -0.0020, respectively) are smaller than those of decision trees, confirming that Random Forest models are more robust and less prone to overfitting due to ensemble learning. The Best Random RF, in particular, has the highest test accuracy and the lowest overfitting gap, making it the best-performing model in terms of both accuracy and generalization. These results further reinforce that ensemble methods like

Random Forest offer significant improvements in predictive performance compared to individual decision trees.

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	0.9826	0.9883	-0.0057	0.0174	0.0117
Deeper RF	0.9958	0.9969	-0.0011	0.0042	0.0031
Deeper RF 2	1.0000	1.0000	0.0000	0.0000	0.0000
Larger RF	0.9902	0.9951	-0.0049	0.0098	0.0049
Feature-Subset RF	0.9822	0.9852	-0.0031	0.0178	0.0148

Comparison with Single Decision Trees

Compared to the best individual decision trees, all Random Forest models achieved higher test accuracy and significantly lower overfitting gaps.

- Single decision trees (Gini, Entropy, Misclassification) had lower test accuracy (0.9735 - 0.9680) and higher overfitting gaps (0.0049 - 0.0085).
- The Original Random Forest (n_trees=10, max_depth=5) already achieved better generalization (test accuracy = 0.9883, negative overfitting gap), indicating that ensemble learning effectively reduces variance.
- The best-performing random forest models (Deeper RF, Larger RF, Feature-Subset RF) outperformed all single decision trees, demonstrating the strength of random forests in mitigating overfitting while improving predictive performance.

***Effect of* `n_trees`**

The number of trees in the ensemble significantly impacts accuracy and stability. The Original Random Forest model, using only 10 trees, achieved strong generalization, while the Larger RF model, which increased the number of trees to 50, further improved performance. The Larger RF model had a training accuracy of 0.9902 and a test accuracy of 0.9951, with an overfitting gap of -0.0049. Increasing the number of trees helped reduce variance and stabilized predictions, reinforcing the idea that ensembles with more estimators tend to generalize better.

***Effect of* `max_depth`**

Varying max_depth also influenced overfitting behavior. The Deeper RF model, which increased max_depth to 6, reached a training accuracy of 0.9958 and a test accuracy of 0.9969, with an overfitting gap of -0.0011. A further increase in depth, as seen in the

Deeper RF 2 model with max_depth=7, led to perfect accuracy on both training and test sets. The complete memorization of the dataset suggests that this model is highly overfitted and may not generalize well to unseen data outside the test set.

Effect of Feature Selection

Feature selection also played a role in generalization. The Feature-Subset RF model, which limited the number of features considered at each split to 5, achieved a training accuracy of 0.9822 and a test accuracy of 0.9852, with an overfitting gap of -0.0031. By decorrelating trees and reducing redundancy, feature selection improved robustness at the cost of slightly lower accuracy compared to the Larger RF model.

In conclusion, Random Forest models consistently outperform individual decision trees by reducing overfitting and improving generalization. Increasing `n_trees` stabilizes predictions, while deeper trees provide better accuracy but also risk memorizing training data. The best trade-off between accuracy and generalization was observed in the Deeper RF model (`max_depth=6, n_trees=10`) and the Larger RF model (`max_depth=5, n_trees=50`). These models effectively balance complexity and generalization, making them strong classifiers for real-world applications.

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 this project:

- Among the three decision trees implemented (using Gini impurity, entropy, and misclassification error), the Gini Tree achieved the highest test accuracy (0.9852), indicating better generalization compared to the entropy and misclassification-based trees. However, all trees showed comparable performance, demonstrating that different impurity measures yield similar classification capabilities when properly tuned.
- The Random Forest model consistently outperformed individual decision trees, achieving the highest test accuracy of 0.9889 and the lowest overfitting gap. This highlights the effectiveness of ensemble learning in reducing variance and improving model stability.
- Overfitting was most pronounced in the entropy-based tree, which retained more dataset-specific patterns, while the misclassification error-based tree showed signs of underfitting. The Random Forest model provided the best balance between bias and variance, confirming its robustness against both overfitting and underfitting.
- Post-pruning was applied to the entropy tree to mitigate overfitting. While pruning slightly simplified the model, it had minimal impact on accuracy, suggesting that the dataset was already well-structured for classification.
- The best decision tree model (obtained through Grid Search) used a maximum depth of 7, while the best random forest model used 10 trees with a depth of 5. The random forest demonstrated better generalization across different parameter settings.

- Introducing random feature selection (limiting the number of features considered at each split) improved model robustness and slightly reduced overfitting, confirming that decorrelation among trees enhances 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