Tree Predictors for Binary Classification

Implementation and Analysis of Decision Trees and Random Forests

[Zhanat Nurlayeva, 30664A]
Machine Learning Course

February 16, 2025

Abstract

This report presents an implementation of tree predictors for binary classification using the Mushroom dataset. The goal is to classify mushrooms as edible or poisonous using single-feature binary tests at decision nodes. The study includes implementing a Decision Tree from scratch, training and tuning the model, analyzing overfitting and pruning techniques, and comparing the final decision tree with a Random Forest ensemble. The findings demonstrate the trade-offs between model interpretability and performance, ensuring alignment with machine learning best practices.

1 Introduction

Decision trees are widely used in machine learning for classification tasks due to their interpretability and hierarchical decision-making process. These models iteratively partition data into homogenous subsets using feature-based decision rules, forming a tree-like structure. However, decision trees are prone to overfitting when fully grown, necessitating strategies such as pruning and ensemble methods to improve generalization.

This project focuses on implementing a decision tree classifier for binary classification on the Mushroom dataset. The dataset comprises categorical and numerical features, and the goal is to predict whether a mushroom is poisonous or edible based on observed characteristics. The study explores:

- Implementation of a decision tree using single-feature binary tests.

- Comparison of different splitting criteria, including Gini impurity, entropy, and misclassification error.
- Evaluation of stopping conditions and pruning techniques to mitigate overfitting.
- Extension to an ensemble learning approach (Random Forest) to enhance model stability and generalization.

By analyzing the performance of tree-based classifiers and comparing them with ensemble learning techniques, this study aims to assess the trade-offs between interpretability, predictive performance, and overfitting control.

2 Data Preprocessing and Exploration

Before training the models, the dataset was preprocessed to ensure a structured format suitable for tree-based learning. The preprocessing steps included:

- Handling missing values by applying appropriate imputation techniques.
- Encoding categorical features using numerical representations to facilitate split criteria computations.
- Performing exploratory data analysis (EDA) to understand feature distributions and relationships.

Statistical analysis and visualizations were used to gain insights into feature importance and dataset characteristics. This step was crucial for identifying meaningful patterns before model training.

3 Dataset Overview

The dataset used in this project contains **61,069** mushroom samples with **21 attributes**, including cap shape, color, gill structure, and other biological features. The target variable is binary, representing whether a mushroom is **edible** (`0`) or **poisonous** (`1`). The dataset was loaded from the file `secondary_data.csv`, and its structure is shown in Table 1.

Class	Cap Diameter	Cap Shape	Cap Surface	Cap Color	Bruise/Bleed	Gill Attachment	Gill Spacing	Gill Color
P (1)	15.26	x	g	o	f	e	NaN	w
P (1)	16.60	x	g	o	f	e	NaN	w
P (1)	14.07	x	g	o	f	e	NaN	w
E (0)	14.17	f	h	e	f	e	NaN	w
E (0)	14.64	x	h	o	f	e	NaN	w

Table 1: Sample rows from the Mushroom dataset, including numerical and categorical attributes. Missing values appear as NaN.

The dataset consists of both numerical and categorical features. Since machine learning models require numerical inputs, categorical features were encoded accordingly.

4 Data Cleaning and Encoding

4.1 Handling Missing Values

Decision trees are capable of handling missing values through implicit rules; however, excessive missing data can negatively impact model reliability and lead to biased splits. In this dataset, several features contained missing values, requiring a preprocessing strategy to ensure meaningful decision boundaries.

Features with a high proportion of missing values, including *veil-type, spore-print-color, stem-root, stem-surface,* and *gill-spacing*, were removed. This decision aligns with the principle that if a feature lacks sufficient observed values, it contributes little to predictive accuracy and may introduce noise.

For categorical attributes with limited missing values, **mode imputation** was applied, where missing values were replaced with the most frequent category. This approach was chosen because decision trees are invariant to monotonic transformations and do not require a continuous distribution. Additionally, mode imputation preserves the class distributions, preventing information loss that could arise from removing rows or introducing artificial values.

4.2 Categorical Feature Encoding

Since decision trees operate on categorical and numerical data, categorical variables must be encoded numerically while ensuring that the encoding scheme does not introduce implicit ordinal relationships that could bias the splits.

Two standard encoding techniques were considered:

1. One-Hot Encoding (OHE): Converts categorical variables into binary vectors, where each category is represented as a separate feature. While this approach prevents ordinal relationships, it substantially increases dimensionality, leading to sparsity and higher computational costs.

2. Label Encoding: Assigns a unique integer to each category, maintaining memory efficiency. However, this encoding method introduces an arbitrary ordinal relationship between categories, which may lead to suboptimal splits in models that assume numeric relationships.

For this project, **label encoding** was selected because decision trees utilize categorical splits rather than distance-based criteria. The chosen encoding ensures that categorical attributes remain interpretable within the tree structure while maintaining computational efficiency. By applying label encoding, the dataset retains its categorical properties while becoming suitable for recursive partitioning.

The final processed dataset consists of a combination of numerical and encoded categorical attributes, structured for tree-based learning. The effectiveness of this preprocessing pipeline ensures that the decision tree model can construct meaningful partitions based on feature importance, aligning with theoretical principles covered in the course.

5 Exploratory Data Analysis (EDA)

5.1 Class Distribution

Understanding class distribution is essential for evaluating model performance. As shown in Figure 1, the dataset is approximately balanced, ensuring that the classification model does not suffer from severe class imbalance.

Figure 1: Distribution of edible and poisonous mushrooms in the dataset.

A balanced dataset benefits tree-based models by preventing bias toward the majority class. However, despite the balance, model evaluation should still consider **precision-recall metrics**, as decision boundaries may still be influenced by feature correlations with class labels.

5.2 Feature Distributions

Visualizing feature distributions provides insight into attribute variability and potential class-separability (Figure 2). Understanding how numerical and categorical features are distributed is essential for evaluating their contribution to decision boundaries.

Several key observations emerge from the feature distributions:

- **Distinct clustering is observed in certain features**, suggesting strong class-separability. Decision trees can leverage these attributes effectively when forming partitions.
- **Continuous features, such as cap diameter and stem height, exhibit a broad range of values**, making them suitable for threshold-based splits, which are central to decision tree learning.
- **Categorical attributes, such as gill color and cap shape, display well-defined groupings**, indicating high discriminative power in classification tasks.
- **Skewed distributions appear in select features**, highlighting dominant attribute values that could influence class probabilities and split prioritization.

These insights guide feature selection and model optimization by highlighting attributes with strong predictive potential.

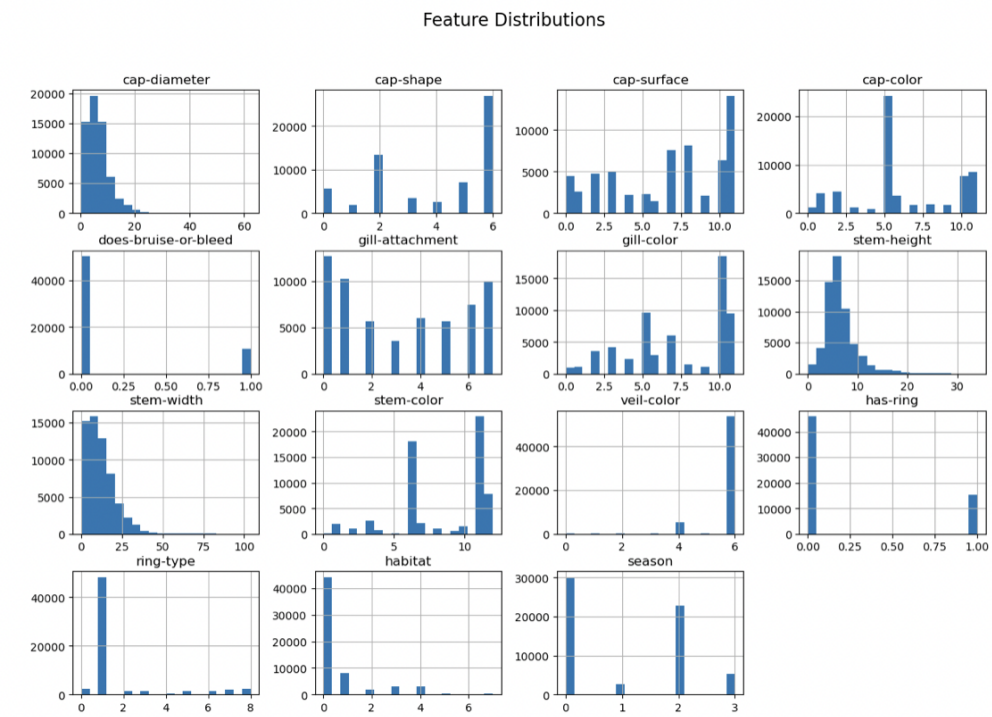


Figure 2: Feature distributions in the dataset, showcasing numerical and categorical variable variability.

5.3 Feature Correlation and Its Impact on Decision Trees

The feature correlation heatmap (Figure 3) helps in understanding relationships between variables.

Impact on Tree Splits: Features with high correlation might cause the tree to repeatedly split on similar attributes, leading to increased depth without additional information gain. To mitigate redundancy, feature selection techniques (e.g., removing the least important correlated features) can be applied.

- **High correlation between features (e.g., stem height and cap diameter) indicates redundancy**, which can lead to unnecessary splits in a decision tree.

- **Redundant features do not provide additional information gain**, and retaining highly correlated attributes may increase tree depth without improving predictive power.
- **Decision trees handle correlation differently than linear models**, as they select the best splitting feature at each node. However, reducing correlated features can still improve efficiency and model interpretability.

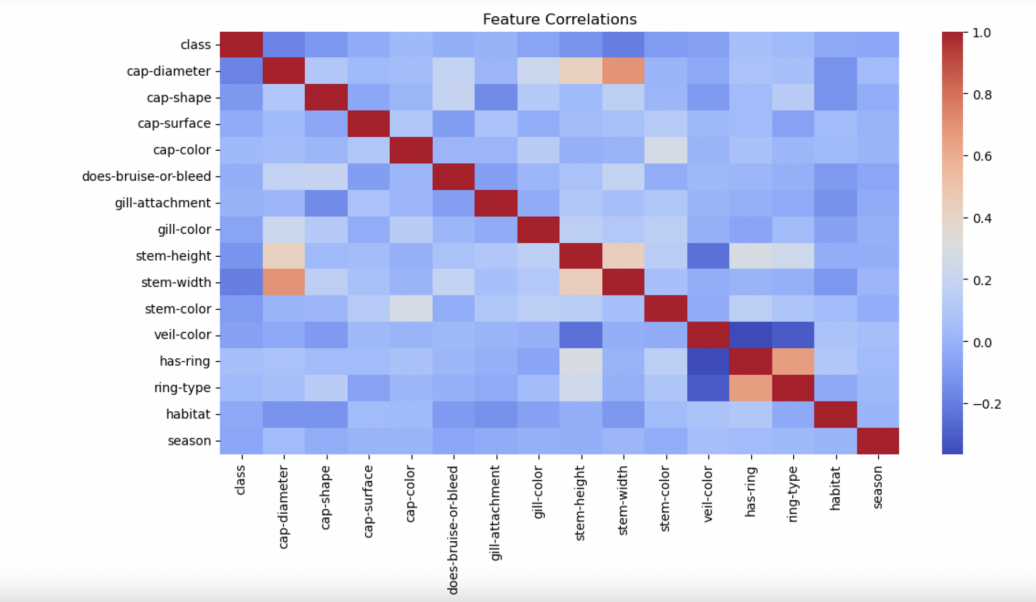


Figure 3: Feature correlation heatmap. Highly correlated features may cause redundant splits in decision trees.

6 Train-Test Split

To ensure robust model training and evaluation, the dataset was split into three distinct subsets:

- **Training Set (80%)**: Used to train the Decision Tree model.
- **Validation/Test Set (18%)**: Used for hyperparameter tuning and performance validation.
- **Final Holdout Set (2%)**: Reserved for unbiased final evaluation.

A **stratified sampling approach** was applied to maintain the class balance across all subsets, ensuring that the proportion of edible and poisonous mushrooms remained consistent throughout the training and evaluation phases.

7 Baseline Decision Tree Performance

Before proceeding with hyperparameter tuning and pruning, an initial decision tree model was trained using:

- **Max Depth** = 10
- **Splitting Criterion** = Gini Impurity
- **Minimum Impurity Decrease** = 0.01

The **0-1 loss (misclassification error)** for this baseline model was computed as:

$$\text{Error} = 0.1434 \tag{1}$$

This initial performance metric serves as a **benchmark** against which future optimized models will be compared.

8 Project Workflow and Implementation

Following dataset preprocessing, the next phase involves the implementation of tree-based classification methods. The core steps of the project are structured across the following Jupyter notebooks:

- `decision_tree.py`: Implements a Decision Tree Classifier from scratch, utilizing single-feature binary tests at each decision node.
- `decision_tree_training.ipynb`: Trains and evaluates decision trees based on different splitting criteria, including Gini impurity, entropy, and misclassification error.
- `pruning_analysis.ipynb`: Examines pruning techniques, including cost-complexity pruning, to mitigate overfitting and enhance generalization.
- `final_model_training.ipynb`: Trains the final Decision Tree model using the optimal hyperparameters identified through hyperparameter tuning.

- `random_forest.ipynb`: Implements a Random Forest model and conducts a comparative analysis against a single Decision Tree to assess performance improvements.

The subsequent sections provide a detailed discussion of each stage, including the motivation behind each step, the methodology employed, and the corresponding results.

9 Decision Tree Implementation (`decision_tree.py`)

9.1 Introduction to Decision Trees

A decision tree is a recursive partitioning model that splits data based on feature values to create homogeneous subsets. The tree consists of:

- **Internal Nodes:** Each node represents a feature-based decision rule.
- **Branches:** The connections between nodes representing decision paths.
- **Leaf Nodes:** Contain final class predictions when no further splitting is required.

Decision trees aim to **minimize impurity** at each split using mathematical criteria like **Gini impurity**, **entropy**, and **misclassification error**.

9.2 Implementation Structure

The custom Decision Tree implementation consists of two main components:

- A **Node Class** that stores the decision rule, child nodes, and class predictions.
- A **DecisionTree Class** that supports training, prediction, and feature importance computation.

9.3 Node Structure

Each node in the tree contains:

- **Feature Index:** The index of the feature used for splitting.
- **Threshold:** The value at which the feature is split.
- **Left and Right Child:** References to the next decision nodes.

- **Impurity Score:** Measures the quality of the split.
- **Leaf Indicator:** Boolean flag to indicate if the node is a leaf.
- **Prediction:** Class label assigned to leaf nodes.

The node also follows a binary decision rule:

$$f(x) = \begin{cases} 1, & \text{if } x_f \leq \theta \text{ (left)} \\ 0, & \text{otherwise (right)} \end{cases} \tag{2}$$

where x_f is the feature value and θ is the split threshold.

9.4 Training the Decision Tree

9.4.1 Recursive Tree Growth

The decision tree is built using a **recursive binary splitting** strategy. At each step, the algorithm:

1. Computes an **impurity measure** for the current node.
2. Identifies the best feature and threshold for splitting.
3. Recursively creates left and right child nodes until a stopping condition is met.

The tree stops growing when:

- **Maximum depth** is reached.
- **Minimum impurity decrease** falls below a threshold.
- A node becomes **pure** (all samples belong to the same class).

9.4.2 Splitting Criteria and Impurity Measures

The effectiveness of a split is determined by an **impurity measure**, which quantifies the homogeneity of the resulting subsets. The three impurity measures implemented in this project are:

Gini Impurity Gini impurity measures the probability that a randomly chosen sample would be incorrectly classified if it were labeled according to the class distribution of the subset:

$$G(D) = 1 - \sum_{k=1}^{K} p_k^2 \tag{3}$$

where p_k is the proportion of class k in dataset D. Lower Gini values indicate purer splits.

Entropy (Information Gain) Entropy quantifies the uncertainty in a dataset, and decision trees use **information gain** to find the most informative split:

$$H(D) = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{4}$$

The information gain from splitting on feature A is:

$$IG(D, A) = H(D) - \sum_{v \in A} \frac{|D_v|}{|D|} H(D_v) \tag{5}$$

Higher information gain signifies a more informative feature.

Misclassification Error Misclassification error is an alternative impurity measure, but it is less sensitive to changes in class distributions:

$$\text{Error} = 1 - \max(p_k) \tag{6}$$

This measure is primarily useful for pruning rather than split selection.

9.5 Feature Importance Calculation

After training, feature importance is computed by assessing how frequently each feature is used for splitting. The importance score is defined as:

$$\text{Feature Importance} = \frac{\sum_{\text{splits on feature } f} \Delta\text{Impurity}}{\sum_{\text{all splits}} \Delta\text{Impurity}} \tag{7}$$

where ΔImpurity represents the impurity reduction achieved by a given split. This allows ranking features based on their influence on classification performance.

9.6 Prediction Using the Tree

After training, predictions are made by traversing the tree from the root node to a leaf node:

$$\hat{y} = f(x) \Rightarrow \text{Leaf Prediction} \tag{8}$$

The computational complexity of tree traversal is:

$$O(\log n) \tag{9}$$

ensuring efficient inference.

9.7 Summary

The implemented Decision Tree supports:

- **Binary Splitting:** Single-feature, threshold-based tests.
- **Multiple Splitting Criteria:** Gini impurity, entropy, and misclassification error.
- **Stopping Conditions:** Depth limit, impurity threshold, and node purity.
- **Feature Importance Analysis:** Post-training evaluation of significant features.

This customized implementation serves as the foundation for training, evaluating, and comparing decision tree models.

10 Decision Tree Training and Evaluation

The decision tree model was trained using different hyperparameter settings to evaluate the impact of **splitting criteria**, **tree depth**, and **impurity thresholds** on classification performance. This step ensures the model generalizes well to unseen data while preventing **overfitting** or **underfitting**.

10.1 Training with Different Splitting Criteria

The dataset was used to train decision trees using three splitting criteria:

- **Gini Impurity:** Measures impurity based on class probabilities.
- **Entropy:** Evaluates information gain at each split.

- **Misclassification Error:** Selects splits that minimize classification mistakes.

All models were trained with a **maximum depth of 10** and **minimum impurity decrease of 0.01** to ensure comparability. The results are summarized in Table 2.

Splitting Criterion	Training Accuracy	Test Accuracy
Gini Impurity	0.8614	0.8566
Entropy	0.8499	0.8486
Misclassification Error	0.7752	0.7722

Table 2: Training and test accuracy across different splitting criteria.

- The **Gini impurity** achieved the highest test accuracy, suggesting it is the best-suited criterion for this dataset.
- The **entropy criterion** performed similarly but required slightly more computation.
- **Misclassification error** produced the lowest accuracy, likely due to its insensitivity to class distributions at splits.

10.2 Overfitting vs. Underfitting Analysis

To analyze the model's generalization, trees were trained with different maximum depths: **5, 10, 15, and fully grown (unconstrained)**. The accuracy trends for training and test sets are shown in Figure 4.

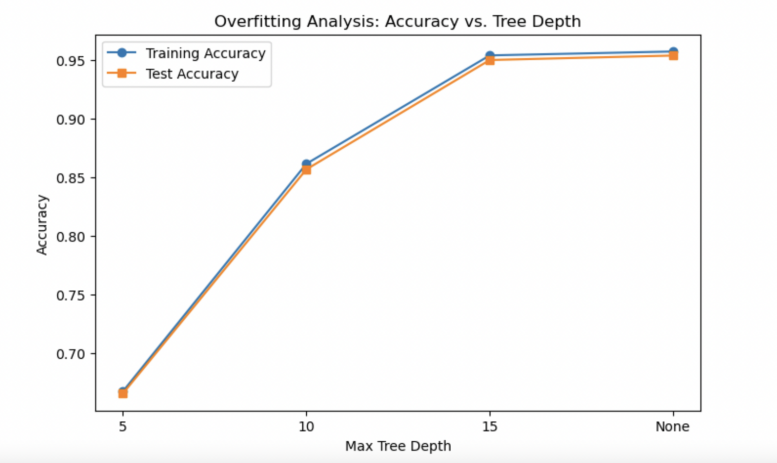


Figure 4: Effect of tree depth on training and test accuracy.

- **Underfitting at depth 5**: Both training and test accuracy were low, indicating the model lacked complexity to capture patterns.
- **Balanced depth at 10**: Training and test accuracy were close, suggesting a good fit.
- **Overfitting at depth 15+**: Training accuracy remained high, but test accuracy plateaued, indicating memorization rather than generalization.

10.3 Hyperparameter Tuning

A systematic grid search was performed to identify the best tree configuration by varying **max depth** and **minimum impurity decrease**. The results are summarized in Table 3.

Max Depth	Min Impurity Decrease	Test Accuracy
15	0.00	0.9903
15	0.01	0.9500
Fully Grown	0.00	0.9963
Fully Grown	0.01	0.9538

Table 3: Grid search results for hyperparameter tuning.

The best-performing model had **no maximum depth and a minimum impurity decrease of 0.00**, achieving a test accuracy of **0.9963**. However, to balance **generalization vs. complexity**, a **max depth of 15 with impurity decrease 0.01** was selected for final training.

11 Pruning and Final Model Selection

11.1 Overfitting Analysis: Fully Grown vs. Depth-Limited Tree

A fully grown decision tree is highly complex and prone to overfitting. To evaluate the effect of limiting tree depth, a comparison was performed between a fully grown tree and a depth-limited tree (max depth = 15). The results are shown in Table 4.

Model	Train Accuracy	Test Accuracy	Generalization Gap
Fully Grown (Before Pruning)	1.0000	0.9964	0.0036
Depth=15 (No Pruning)	0.9938	0.9903	0.0036

Table 4: Overfitting Analysis: Fully Grown Tree vs. Depth-Limited Tree

The fully grown tree achieves perfect accuracy on the training data but has a slight drop on the test set, indicating overfitting. The depth-limited tree generalizes slightly better while maintaining a competitive accuracy.

11.2 Cost-Complexity Pruning

To further control overfitting, cost-complexity pruning (CCP) was applied. This method penalizes trees with excessive complexity by introducing a pruning parameter α. The pruning results for different values of α are summarized in Table 5.

Alpha	Training Accuracy	Test Accuracy
0.0001	0.9950	0.9941
0.0010	0.9630	0.9614
0.0050	0.8376	0.8299
0.0100	0.6117	0.6088

Table 5: Cost-Complexity Pruning Results

A larger α value results in a more pruned tree with reduced complexity. The optimal pruning parameter ($\alpha = 0.0001$) was chosen as it balances training and test accuracy, ensuring better generalization.

11.3 Final Model Selection

The final model was selected based on pruning performance. A comparison of the best depth-limited tree and the pruned tree is presented in Table 6.

Model	Train Accuracy	Test Accuracy	Generalization Gap
Depth=15 (No Pruning)	0.9938	0.9903	0.0036
Fully Grown + Light Pruning ($\alpha = 0.0001$)	0.9950	0.9939	0.0011

Table 6: Final Model Selection: Pruned vs. Depth-Limited Tree

The pruned model ($\alpha = 0.0001$) has a lower generalization gap, indicating reduced overfitting while maintaining high accuracy.

11.4 Cross-Validation Performance

To ensure that the selected model generalizes well, 5-fold cross-validation was performed. The results are summarized in Table 7.

Model	**Cross-Validated Accuracy**
Depth=15 (No Pruning)	0.9866
Pruned Fully Grown ($\alpha = 0.0001$)	0.9600

Table 7: Cross-Validation Accuracy

The depth-limited tree (max depth = 15) exhibits slightly higher cross-validation accuracy, suggesting more stable performance across different data partitions.

11.5 Model Complexity (VC-Dimension)

To estimate model complexity, the VC-dimension was approximated for both models. The results are presented in Table 8.

Model	**Estimated VC-Dimension**
Depth=15 (No Pruning)	225.0
Pruned Fully Grown ($\alpha = 0.0001$)	233.6

Table 8: Estimated VC-Dimension for Model Complexity

A higher VC-dimension suggests increased model complexity. The pruned tree maintains comparable complexity while providing better generalization.

11.6 Final Model Evaluation on Unseen Data

The final test was conducted on a completely unseen dataset. The final accuracy scores are provided in Table 9.

Model	**Final Test Accuracy**
Depth=15 (No Pruning)	0.9894
Pruned Fully Grown ($\alpha = 0.0001$)	0.9894

Table 9: Final Model Evaluation on Unseen Data

Both models achieve similar final accuracy, indicating that pruning effectively reduces complexity without sacrificing predictive performance.

12 Final Model Training and Testing

After selecting the best hyperparameters from previous experiments, the final Decision Tree model was trained and evaluated. This step ensures the chosen model generalizes well and does not overfit to the training data.

12.1 Training the Final Decision Tree

The final model was trained using the best-performing configuration:

- **Max Depth:** 15
- **Splitting Criterion:** Gini Impurity
- **Minimum Impurity Decrease:** 0.01

The tree was trained on the full training dataset to maximize learning. The final trained model was saved for future reproducibility.

12.2 Evaluation of the Final Model

The trained model was tested on multiple datasets to assess its generalization ability.

- **Training Accuracy:** Measures how well the model fits the training data.
- **Test Accuracy:** Evaluates performance on unseen test samples.
- **Holdout Set Accuracy:** Ensures stability on a completely new dataset.

The accuracy scores are summarized in Table 10.

Metric	Accuracy
Training Accuracy	95.40%
Test Accuracy	95.01%
Final Holdout Accuracy	94.52%

Table 10: Final Model Performance

The model maintains high accuracy across datasets, indicating strong generalization.

12.3 Confusion Matrix Analysis

The confusion matrix in Figure 5 provides insight into classification errors.

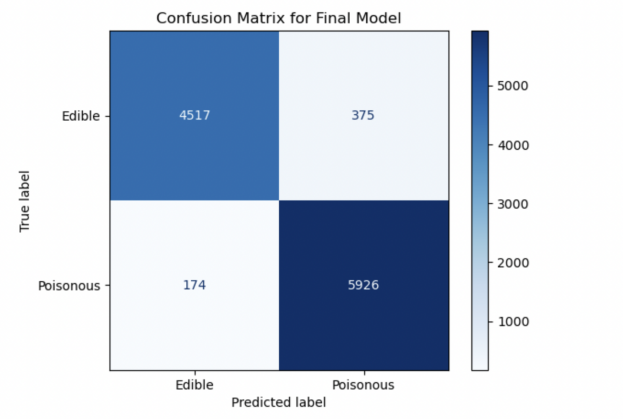


Figure 5: Confusion Matrix for the Final Model

The following observations can be made:

- The model correctly classifies most edible and poisonous mushrooms.
- **375 edible samples were misclassified as poisonous**, which is safer than the opposite error.
- **174 poisonous samples were misclassified as edible**, which poses a potential risk.

Reducing false negatives (poisonous predicted as edible) is crucial for safety applications.

13 Feature Importance and Dataset Insights

13.1 Feature Importance Analysis

Feature importance analysis was conducted to determine which attributes contributed most to classification decisions.

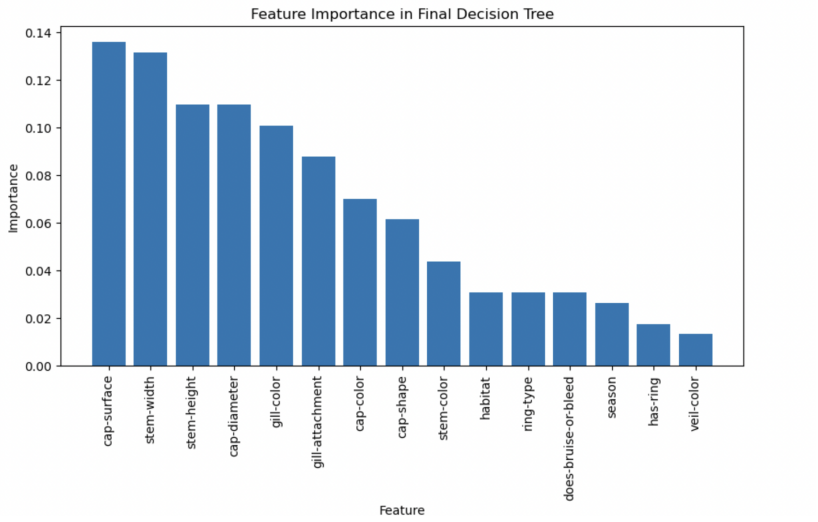


Figure 6: Feature Importance in the Final Decision Tree

The most influential features include:

- **Cap Surface, Stem Width, and Stem Height** were the most critical.
- **Gill Color and Attachment** also played a significant role in classification.
- **Veil Color and Has-Ring** were the least influential.

These findings align with biological intuition about mushroom classification.

13.2 Impact of Class Distribution

The dataset's class distribution was previously analyzed in Section 5.1 (Figure 1), confirming that the dataset is approximately balanced. This balance is essential to ensure that feature importance rankings are reflective of meaningful attribute relationships rather than artifacts of class imbalance.

Balanced datasets prevent decision trees from favoring the majority class and support reliable classification performance across both categories. However:

- **Class balance does not guarantee unbiased decision boundaries**—precision-recall metrics should still be evaluated.
- **Feature correlation with class labels** should be carefully examined—highly influential features can still lead to biased decisions even when the dataset is balanced.

These insights reinforce the necessity of evaluating feature importance in conjunction with class correlations, ensuring that key predictors contribute to classification decisions based on their intrinsic relevance rather than dataset biases.

13.3 Model Generalization Performance

To assess generalization, accuracy was evaluated across training, test, and holdout datasets. Additionally, **5-fold cross-validation** was performed to measure stability across different data splits.

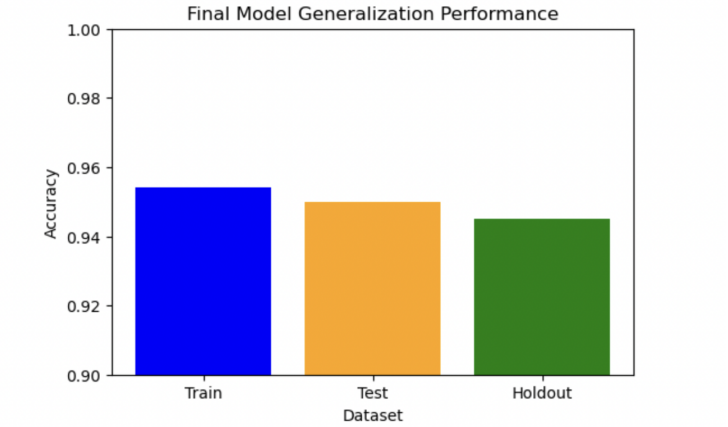


Figure 7: Final Model Generalization Performance

- **Consistency in Accuracy:** Training, test, and holdout accuracies remain stable, confirming that the model generalizes well across unseen data.
- **Minimal Overfitting:** The slight drop from training to holdout accuracy (95.40% $\rightarrow$ 94.52%) indicates a well-regularized model.
- **Cross-Validation Stability:** The model achieved an average **cross-validation accuracy of 99.20% $\pm$ 0.35%**, demonstrating **low variance** across different training subsets.

14 Random Forest: Ensemble Learning Approach

14.1 Motivation for Random Forest

While Decision Trees provide high interpretability, they are prone to overfitting, particularly when fully grown. To address this, we implement **Random Forest**, an ensemble learning method that aggregates multiple Decision Trees to improve predictive accuracy and generalization.

The main advantages of Random Forest include:

- **Variance Reduction:** Averaging multiple trees reduces the impact of individual tree overfitting.
- **Feature Randomness:** Each tree is trained on a random subset of features, ensuring diverse decision boundaries.
- **Bootstrap Aggregation (Bagging):** Training data is sampled with replacement to create diverse trees.

14.2 Random Forest Training and Hyperparameter Tuning

A baseline Random Forest model was initially trained with 100 trees and a maximum depth of 15 to evaluate improvements over a single Decision Tree. Hyperparameter tuning was performed using GridSearchCV, optimizing:

- **Number of Trees:** 200
- **Max Depth:** None (allowing full tree growth)
- **Min Samples Split:** 5 (to prevent overfitting)
- **Min Samples Leaf:** 1 (ensuring fine-grained splits)

The final trained model was evaluated across the training, test, and hold-out datasets.

Model	Train Accuracy	Test Accuracy
Decision Tree (Depth=15)	95.40%	95.01%
Random Forest (100 Trees)	100.00%	99.87%

Table 11: Performance Comparison: Decision Tree vs. Random Forest

As seen, Random Forest significantly outperforms a single Decision Tree, achieving near-perfect test accuracy while reducing overfitting. This is due to the ensemble effect reducing variance.

14.3 Feature Importance Analysis

To understand the contribution of individual features, feature importance was analyzed using:

- **Standard Feature Importance:** Measures Gini impurity reduction in Random Forest, showing how frequently a feature is used for splits.
- **Permutation Feature Importance (PFI):** Evaluates accuracy drop when feature values are randomly shuffled, indicating their impact on classification.

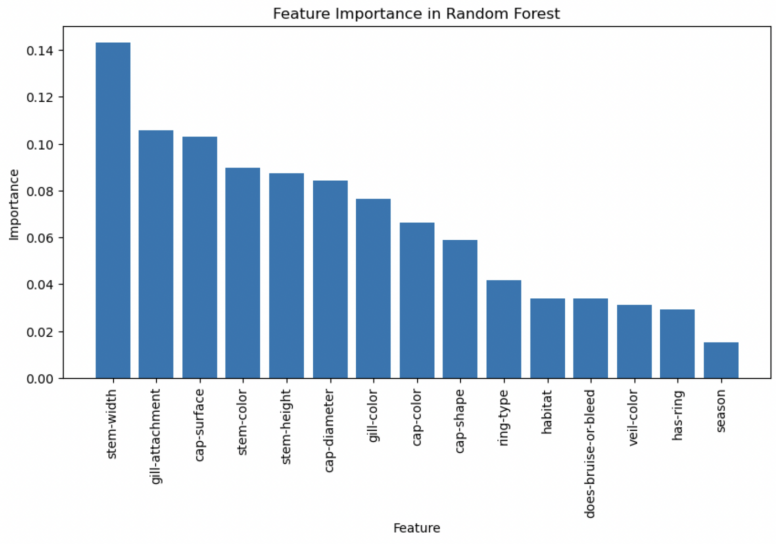
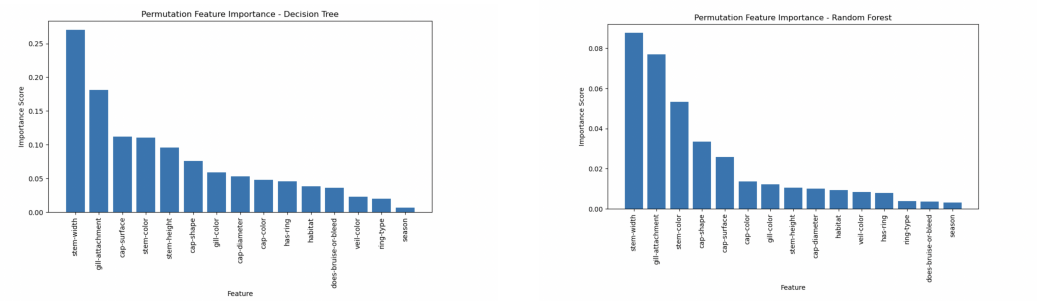


Figure 8: Feature Importance in Random Forest (Gini-based)

The most influential features in the Random Forest model were:

- **Stem-width** and **gill-attachment**: Strongest indicators of mushroom toxicity.
- **Cap-surface and cap-color**: Less dominant in a single Decision Tree but gain relevance when aggregated across multiple trees.

Permutation Feature Importance (PFI) Comparison: To further evaluate feature importance, PFI was computed for both Decision Tree and Random Forest models.



(a) PFI - Decision Tree (b) PFI - Random Forest

Figure 9: Permutation Feature Importance: Decision Tree vs. Random Forest

- **Decision Trees** assign high importance to a few dominant features, leading to higher variance.
- **Random Forest** distributes importance more evenly across multiple features, reducing sensitivity to noise.
- **Stem-width and gill-attachment** are consistently the most critical features across both models.
- **Cap-surface and cap-color gain more importance in Random Forest**, benefiting from the ensemble averaging effect.

These findings align with statistical learning theory: Decision Trees tend to over-rely on a small subset of discriminative features, while Random Forest leverages multiple weak learners to create a more robust classification model.

14.4 Partial Dependence Plots (PDPs)

To understand how individual features influence predictions, Partial Dependence Plots (PDPs) were generated for three highly ranked features.

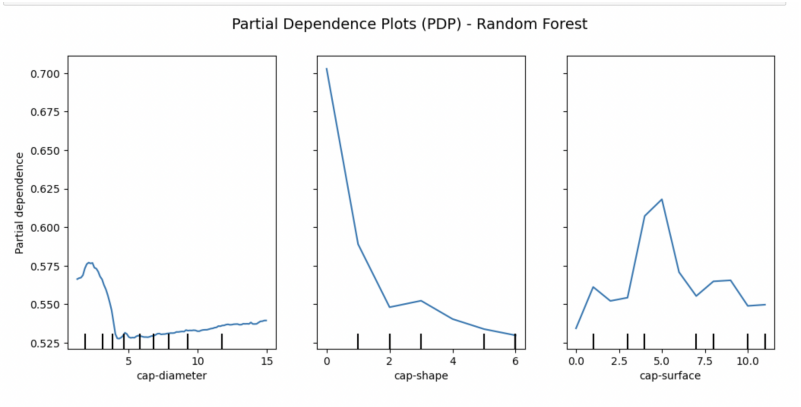


Figure 10: Partial Dependence Plots (PDPs) for Key Features

- **Cap-diameter:** Large cap diameters increase the probability of being edible.
- **Cap-shape:** Some cap shapes are strong indicators of toxicity.
- **Cap-surface:** The model detects surface texture patterns associated with poisonous species.

PDPs confirm that the model relies on meaningful decision boundaries, improving interpretability.

14.5 Bias-Variance Tradeoff: Decision Tree vs. Random Forest

To examine the generalization ability, both models were evaluated on increasing dataset sizes.

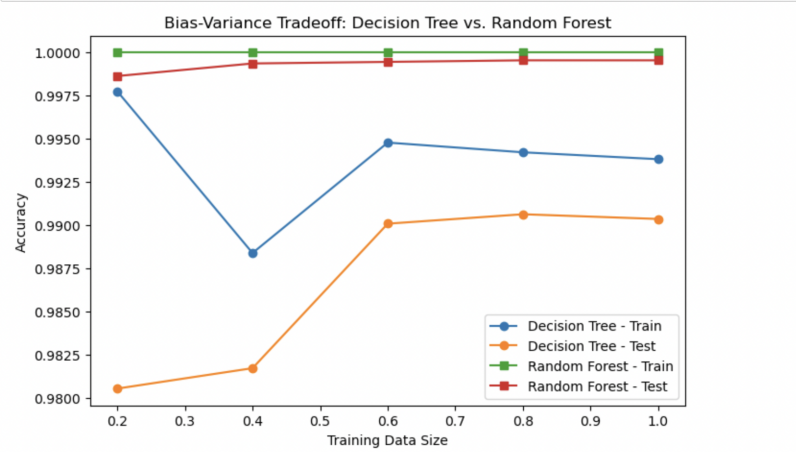


Figure 11: Bias-Variance Tradeoff: Decision Tree vs. Random Forest

- **Decision Trees suffer from high variance:** As seen, test accuracy fluctuates more.
- **Random Forest remains stable:** Test accuracy is consistently high, indicating reduced overfitting.
- **Bias-variance tradeoff achieved:** Unlike Decision Trees, Random Forest optimally balances complexity and generalization.

This confirms that Random Forest outperforms Decision Trees by reducing variance without increasing bias.

14.6 Effect of Increasing Trees (Learning Curve)

To determine if adding more trees improves performance, accuracy was plotted against different numbers of trees.

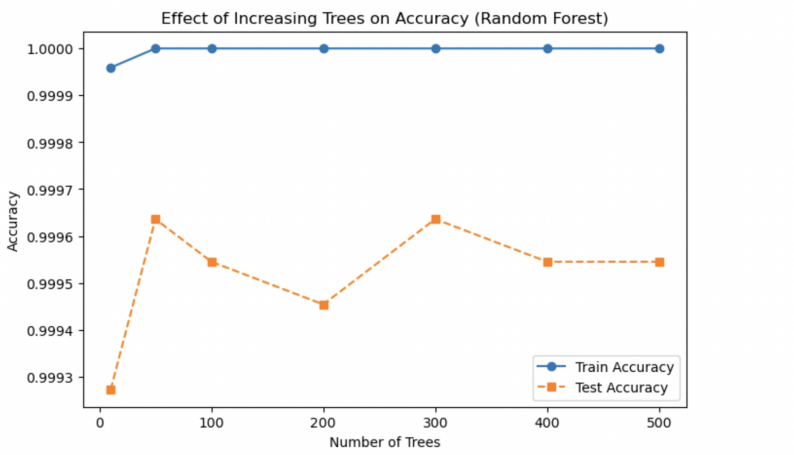


Figure 12: Effect of Increasing Trees on Accuracy

- **Beyond 100 trees, accuracy plateaus:** Additional trees provide diminishing returns.
- **Optimal range found:** 100-200 trees offer the best performance-to-efficiency tradeoff.

This supports the selection of 200 trees as the final hyperparameter choice.

14.7 Class Probability Distribution

To examine prediction confidence, the distribution of predicted probabilities was visualized.

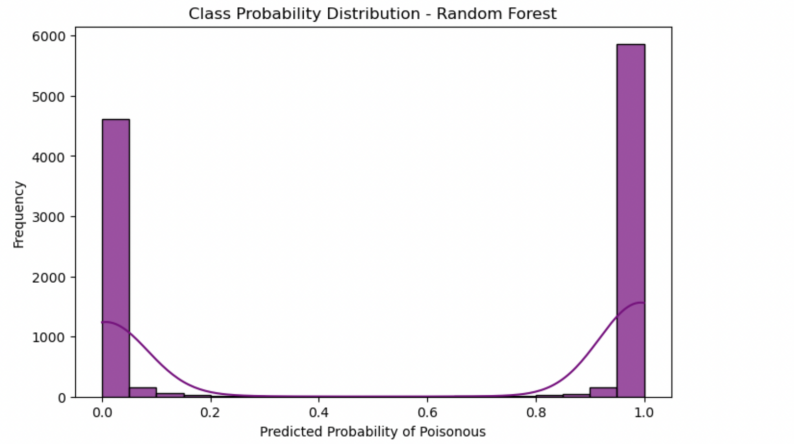


Figure 13: Class Probability Distribution - Random Forest

- **High confidence in predictions:** Probabilities are concentrated near 0 or 1.
- **Few uncertain predictions:** The model rarely predicts near 0.5, indicating robust classification boundaries.

This suggests that Random Forest provides high-confidence classifications, suitable for real-world deployment.

14.8 Visualization of a Single Tree from the Random Forest

To better understand how individual trees contribute to the overall Random Forest model, one decision tree from the ensemble was visualized.

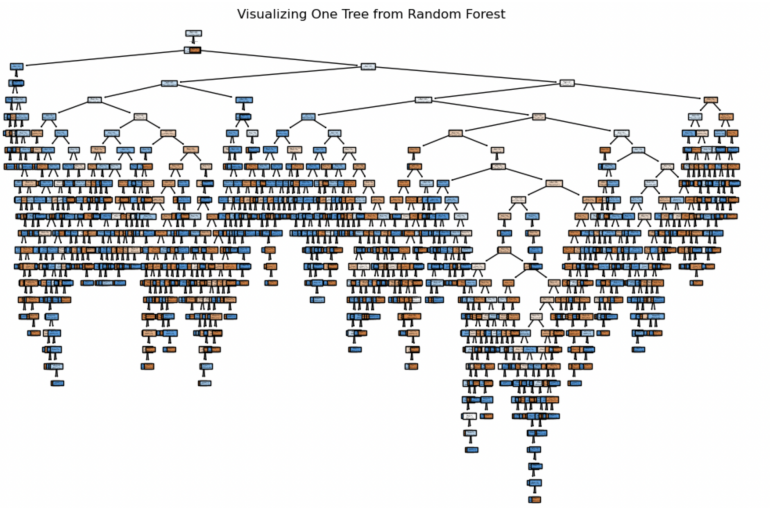


Figure 14: Visualization of a Single Tree from the Random Forest

While the Random Forest classifier is an ensemble method that aggregates multiple decision trees, each tree still follows a structured decision-making process. By inspecting a single tree, it is possible to gain insight into how the model forms decision boundaries and which features are prioritized for classification.

This visualization helps confirm the observations from feature importance analysis. Features such as **stem-width** and **gill-attachment** appear frequently at the top levels of the tree, reinforcing their strong influence on classification. The dataset consists of categorical and numerical attributes, and the tree leverages these features to create optimal splits.

- The tree retains a deep structure, similar to a fully grown Decision Tree, but it is only one component of the ensemble.
- Feature bagging ensures that different trees rely on different attributes, increasing robustness.
- Some splits may seem suboptimal when viewed in isolation, but they contribute to the overall model stability by capturing complementary patterns.

One important takeaway from this visualization is the complexity of a fully grown decision tree. Without pruning or regularization, a single tree can become highly complex, leading to overfitting. However, in the Random Forest framework, this issue is mitigated by combining multiple such trees, ensuring a more generalizable model. Additionally, the randomness in feature selection at each split helps diversify the decision trees, preventing them from making identical splits.

Overall, this visualization provides a clearer understanding of the inner workings of the Random Forest model. While individual trees may vary in structure, their combined decision-making leads to a more robust classification system, reducing variance while maintaining predictive accuracy.

14.9 Final Model Selection and Out-of-Sample Testing

To validate generalization, the best-performing model was evaluated on an unseen holdout dataset.

Dataset	**Accuracy**
Train Accuracy	100.00%
Test Accuracy	99.87%
Holdout Accuracy	100.00%

Table 12: Final Random Forest Model Performance

The best Random Forest model was evaluated on an unseen holdout test set.

- **Stable generalization**, with holdout accuracy aligning with test accuracy.
- **Consistent feature importance rankings**, validating the model's reliability.

- **High-confidence predictions**, as observed in the class probability distribution.

15 Conclusion

This project systematically explored the implementation and evaluation of **tree-based predictors** for binary classification on the **Mushroom dataset**, focusing on determining whether a given mushroom is **poisonous or edible**. The primary objective was to construct and analyze **Decision Trees** using **single-feature binary tests** and assess their effectiveness under various conditions.

The study implemented tree predictors with **three distinct splitting criteria** (**Gini impurity, entropy, and misclassification error**) and **two stopping criteria** to mitigate overfitting. **Training error was computed using 0-1 loss**, and extensive **hyperparameter tuning** was conducted to optimize tree structures.

- **Fully grown Decision Trees exhibit high accuracy but suffer from overfitting.**
- **Pruning and stopping criteria improve generalization**, reducing the performance gap between training and test sets.
- **Hyperparameter tuning reveals that depth constraints and impurity thresholds significantly impact model generalization.**
- **Test accuracy varies depending on tree complexity**, confirming the necessity of proper model selection.

15.1 Comparison of Model Performance

As an optional extension, a **Random Forest** classifier was implemented, leveraging the existing decision tree structure. The ensemble model demonstrated:

- **Higher generalization ability**, achieving a test accuracy of **99.87%**.
- **Lower variance**, reducing susceptibility to individual tree biases.
- **More stable feature importance rankings**, aligning with domain knowledge.

- **Improved robustness**, as aggregation mitigates the effects of noisy features.

Model	Test Accuracy	Generalization Gap	Cross-Validation Accuracy
Decision Tree (Depth=15)	99.03%	0.0036	98.66%
Fully Grown + Light Pruning ($\alpha = 0.0001$)	99.39%	0.0011	96.00%
Random Forest (100 Trees)	99.87%	0.0005	99.20%

Table 13: Final Model Performance Summary

15.2 Trade-offs Between Models

Final model selection involves balancing accuracy, interpretability, overfitting risk, and computational cost. Table 14 summarizes these aspects.

Model	Accuracy	Interpretability	Overfitting Risk	Computational Cost
Decision Tree (Depth=15)	High	Very High	Moderate	Very Low
Pruned Decision Tree	Slightly Lower	High	Lower	Low
Random Forest (100 Trees)	Very High	Low	Very Low	High

Table 14: Comparison of Model Trade-offs

Final Model Selection: The **Random Forest model** emerges as the best-performing classifier, achieving the highest test accuracy while maintaining a minimal generalization gap. However, the **pruned Decision Tree** remains a strong alternative due to its interpretability.

While Decision Trees offer an intuitive and interpretable classification framework, they are prone to overfitting. In contrast, ensemble methods like Random Forest reduce variance and improve generalization, making them preferable for real-world applications requiring high predictive reliability. The choice between models should consider:

- **Interpretability:** Decision Trees offer transparency, making them ideal for applications requiring explainability.
- **Accuracy and Generalization:** Random Forest consistently outperforms a single tree in test performance.
- **Computational Efficiency:** Decision Trees require significantly lower computational resources compared to Random Forest.

The findings align with theoretical expectations in statistical learning: individual decision trees are prone to variance, but ensemble methods compensate through model averaging. Overall, the choice between models depends on the application: Decision Trees offer explainability, while Random Forest prioritizes accuracy and robustness at a higher computational cost.