Implementing Decision Trees and Random Forests for Mushroom Classification

Sara Miraglia (45669A)
sara.miraglia@studenti.unimi.it

October 2024

Applying Decision Trees and Random Forests to Determine Mushroom Edibility

Abstract

This study explores the creation and application of decision tree and random forest classifiers aimed at predicting the edibility of mushrooms based on their attributes. Both models were developed manually in Python, encompassing stages like data preprocessing, feature selection, model training, hyperparameter optimization, and performance assessment. The decision tree and random forest classifiers both demonstrated exceptional accuracy, highlighting the effectiveness of the implemented algorithms and the dataset's quality. Additionally, the report delves into the theoretical aspects of decision trees and random forests, discussing split criteria, methods to prevent overfitting, and a comparative analysis of the models' performances.

Contents

1 Introduction

The goal of this project is to construct a decision tree classifier from the ground up, designed to predict whether a mushroom is poisonous or edible based on its features. Decision trees function by recursively partitioning the dataset using specific criteria, creating a hierarchical structure where each internal node represents a decision based on a single feature. In this project, the trees employ binary tests on individual features at each node. For numerical features, splits are based on threshold values, while categorical features are handled through membership tests.

Beyond decision trees, the project extends to the implementation of a random forest model. Random forests aggregate the predictions of multiple decision trees, enhancing robustness and reducing overfitting by introducing randomness during the training process. Improvements to the random forest algorithm led to significantly better performance, comparable to that of the decision tree.

Throughout the project, various splitting criteria such as Gini impurity and entropy were utilized, alongside multiple stopping rules to control tree growth. Hyperparameter tuning was conducted to fine-tune the models' performance. This report covers the methodology, theoretical background, and performance evaluation of both decision trees and random forests.

2 Dataset Description

The dataset comprises 61,069 entries, each representing a unique mushroom sample characterized by 20 distinct features. The target variable, denoted as `class`, specifies whether a mushroom is poisonous (`p`) or edible (`e`). The following table provides a summary of the dataset's features:

Variable	Type	Description
class	Categorical	Target variable indicating edibility: poisonous (p) or edible (e).
cap-diameter	Numerical	Diameter of the mushroom cap in centimeters.
cap-shape	Categorical	Shape of the mushroom cap: bell (b), conical (c), convex (x), flat (f), sunken (s), spherical (p), others (o).
cap-surface	Categorical	Surface texture of the cap: fibrous (i), grooves (g), scaly (y), smooth (s), shiny (h), leathery (l), etc.
cap-color	Categorical	Color of the cap: brown (n), buff (b), gray (g), green (r), pink (p), red (e), white (w), yellow (y), etc.
does-bruise-or-bleed	Categorical	Indicates if the mushroom bruises or bleeds: yes (t), no (f).
gill-attachment	Categorical	Attachment of the gills to the stem: adnate (a), adnexed (x), decurrent (d), free (e), none (f).
gill-spacing	Categorical	Spacing of the gills: close (c), distant (d), none (f).
gill-color	Categorical	Color of the gills, similar to cap colors.
stem-height	Numerical	Height of the mushroom stem in centimeters.
stem-width	Numerical	Width of the mushroom stem in millimeters.
stem-root	Categorical	Root structure: bulbous (b), swollen (s), club (c), cup (u), equal (e), rhizomorphs (z), rooted (r).
stem-surface	Categorical	Surface texture of the stem, similar to cap surface.
stem-color	Categorical	Color of the stem, similar to cap colors.
veil-type	Categorical	Type of veil: partial (p), universal (u).
veil-color	Categorical	Color of the veil, similar to cap colors.
has-ring	Categorical	Presence of a ring on the stem: ring (t), none (f).
ring-type	Categorical	Type of ring: cobwebby (c), evanescent (e), flaring (r), grooved (g), etc.
spore-print-color	Categorical	Color of the spore print, similar to cap colors.
habitat	Categorical	Habitat where the mushroom grows: grasses (g), leaves (l), meadows (m), paths (p), etc.
season	Categorical	Season when the mushroom is commonly found: spring (s), summer (u), autumn (a), winter (w).

Table 1: Summary of Features in the Mushroom Dataset

3 Methodology and Implementation

This section outlines the systematic approach undertaken to preprocess the data, split it appropriately, and implement the decision tree and random forest models.

3.1 Data Preprocessing

Effective data preprocessing is crucial for building robust machine learning models. The following steps were meticulously executed to clean and prepare the dataset:

- **Handling Missing Values**: Columns exhibiting more than 25% missing values were excluded from the analysis to maintain data integrity. For the remaining missing values, rows with incomplete data were removed, ensuring that the dataset used for modeling was complete and reliable.
- **Removing Duplicate Entries**: Identical rows were identified and eliminated to prevent bias and ensure that each sample contributed uniquely to the model training process.
- **Encoding the Target Variable**: The categorical target variable, `class`, was transformed into binary numerical values: 1 for edible mushrooms and 0 for poisonous ones. This encoding facilitates efficient processing by the classification algorithms.
- **Separating Features and Target**: The dataset was partitioned into features (`X`) and target (`y`) to streamline the training process. This separation is fundamental for supervised learning tasks, allowing the model to learn the relationship between input features and the target variable.

Dataset Phase	Rows	Columns
Initial Dataset	61,069	21
Cleaned Dataset	36,948	15

Table 2: Dataset Size Before and After Cleaning

3.2 Stratified Splitting of the Dataset

To ensure that the training and test sets accurately reflect the overall class distribution, a stratified splitting approach was employed. This method guarantees that both subsets maintain the same proportion of edible and poisonous mushrooms as present in the original dataset.

3.2.1 Stratified Split Methodology

The stratified splitting process was carried out through the following steps:

- **Identifying Unique Classes**: The distinct classes within the target variable were identified to facilitate balanced sampling.
- **Shuffling Indices**: For each class, the corresponding data indices were shuffled to randomize the distribution and eliminate any inherent order.
- **Splitting Based on Desired Test Size**: The shuffled indices were divided into training and test subsets based on a predefined test size, ensuring proportional representation of each class.
- **Combining Indices**: The indices from all classes were amalgamated to form the final training and test sets, preserving the overall class distribution.

This meticulous approach ensures that both the training and test sets are representative of the entire dataset, thereby enhancing the model's ability to generalize effectively to unseen data.

Dataset Split	Samples
Training Set	29,560
Test Set	7,388

Table 3: Training and Test Set Sizes

4 Decision Tree

This section explores the Decision Tree model, detailing its theoretical basis, implementation, hyperparameter tuning, evaluation, and results.

4.1 Structure and Functionality

A decision tree is a hierarchical, rooted tree consisting of internal nodes and leaves. Internal nodes signify decision points based on feature tests, while leaves represent the final class predictions. Traversing the tree involves making a series of decisions based on input features, leading to a leaf node that provides the prediction.

4.1.1 Example of a Decision Tree

Figure 1 illustrates a simple decision tree used for classifying mushrooms as edible or poisonous based on specific characteristics.

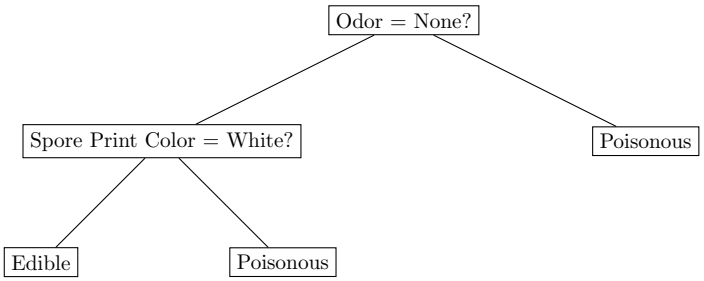

Figure 1: Sample Decision Tree for Mushroom Classification

4.2 Building and Growing the Tree

Creating a decision tree involves selecting the most effective feature to split the data at each node to minimize classification errors. The process starts with a single node and expands by replacing leaves with internal nodes based on feature tests. Key factors in tree growth include:

- **Splitting Criteria**: Metrics like Gini impurity, entropy (information gain), and classification error assess the quality of splits.
- **Preventing Overfitting**: Techniques such as limiting the maximum depth, setting a minimum number of samples for splitting, and pruning help avoid overly complex trees that capture noise in the data.

4.3 Splitting Criteria

Splitting criteria determine how the algorithm selects the best feature and threshold for data partitioning at each node. The primary criteria include Gini impurity, entropy, and classification error.

Gini Impurity Gini impurity gauges the probability of misclassifying a randomly chosen element if it were labeled according to the class distribution in the subset. It is defined as:

$$Gini(D) = 1 - \sum_{i=1}^{C} p_i^2$$

where p_i is the probability of class i in dataset D, and C is the total number of classes. A lower Gini impurity indicates a purer node.

Entropy (Information Gain) Entropy measures the uncertainty or impurity in the dataset, defined as:

$$Entropy(D) = -\sum_{i=1}^{C} p_i \log_2 p_i$$

Information Gain is the reduction in entropy achieved by partitioning the data based on a feature. The feature yielding the highest information gain is chosen for splitting.

Classification Error Classification error represents the proportion of misclassified samples in a subset, defined as:

$$Error(D) = 1 - \max_i p_i$$

where p_i is the probability of the most frequent class in dataset D. Compared to Gini impurity and entropy, classification error is less sensitive to changes in class distribution.

4.4 Advantages of Each Criterion

- **Gini Impurity**: Fast to compute and tends to favor splits with more classes.
- **Entropy**: Provides a more nuanced measure of impurity, often leading to better performance on certain datasets.
- **Classification Error**: Simple and less computationally demanding, though it may result in less optimal splits.

4.4.1 Extensions and Interpretability

Decision trees can handle multiclass classification by assigning the most frequent label to each leaf. For regression tasks, leaves typically store the mean of target values. One major benefit of decision trees is their interpretability, allowing easy visualization and understanding of the decision-making process through logical rules derived from feature tests.

4.5 Implementation

The implementation comprises two custom classes: `TreeNode` and `DecisionTree`. The `TreeNode` class represents each node in the tree, while the `DecisionTree` class manages the tree structure, training, and prediction functionalities.

4.5.1 TreeNode Class Design

The `TreeNode` class encapsulates all necessary information for decision-making and prediction at each node:

- **Feature Selection**: Stores the index of the feature used for splitting, chosen based on the splitting criterion to maximize impurity reduction.
- **Threshold Value**: For numerical features, holds the threshold value determining data partitioning.

- **Child Nodes**: References to left and right child nodes facilitate recursive tree traversal during training and prediction.
- **Predicted Class**: Leaf nodes store the predicted class label, typically the most frequent class among training samples in that leaf.
- **Is Leaf**: Indicates whether the node is a leaf, determining if it provides a prediction or continues the decision process.

This design ensures each node contains all necessary attributes for efficient tree traversal, splitting, and prediction, promoting scalability and ease of maintenance.

4.5.2 DecisionTree Class Design

The `DecisionTree` class oversees the overall tree structure and functionality:

- **Node Structure**: Utilizes `TreeNode` instances to represent each node, allowing dynamic tree growth based on data and splitting criteria.
- **Fitting the Model**: The `fit` method builds the tree by recursively selecting the best feature and threshold to split the data, minimizing the chosen impurity measure.
- **Splitting Data**: Evaluates all possible splits across features to identify the one that offers the maximum impurity reduction.
- **Stopping Criteria**: Controls tree complexity using parameters like maximum depth and minimum samples required to split a node.
- **Prediction**: The `predict` method traverses the tree for each input sample, making decisions at internal nodes until reaching a leaf node for the final prediction.
- **Handling Features**: Capable of managing both numerical and categorical features, ensuring versatility with diverse datasets.

This comprehensive design facilitates efficient construction, training, and utilization of the decision tree model, ensuring robustness and interpretability aligned with machine learning principles.

4.5.3 Process of Building the Tree

Building the decision tree involves:

1. **Root Node Initialization**: Starts with the entire training dataset at the root.
2. **Evaluating Splits**: Assesses all possible splits across features to find the one that minimizes impurity.
3. **Selecting the Optimal Split**: Chooses the split with the lowest impurity measure.
4. **Creating Child Nodes**: Divides the data into subsets based on the best split and creates corresponding child nodes.
5. **Recursive Splitting**: Repeats the process for each child node until stopping criteria are met.

4.5.4 Handling Numerical and Categorical Features

The implementation addresses both numerical and categorical features with tailored splitting strategies:

- **Numerical Features**: Determines potential split points by sorting unique values and identifying midpoints between consecutive values. Splits data based on these threshold values.
- **Categorical Features**: Creates binary splits by grouping categories into two distinct sets based on membership tests.

This approach allows the decision tree to effectively handle continuous and discrete data, capturing complex relationships between features and the target variable.

4.5.5 Hyperparameters and Configurations

The `DecisionTree` class offers customization through hyperparameters:

- **Criterion**: Specifies the splitting criterion (e.g., 'gini', 'entropy').
- **Max Depth**: Sets the maximum depth of the tree to control complexity.
- **Min Samples Split**: Defines the minimum number of samples required to split an internal node.

4.6 Hyperparameter Tuning

Hyperparameter optimization was conducted to enhance the decision tree's performance. The tuned hyperparameters include:

- **Maximum Depth (`max_depth`)**: Limits the tree's depth to prevent overfitting.
- **Minimum Samples Split (`min_samples_split`)**: Sets the minimum number of samples needed to split a node, promoting generalization.
- **Splitting Criterion (`criterion`)**: Determines the function for assessing split quality, such as Gini impurity or entropy.

Cross-validation was utilized to evaluate different hyperparameter combinations, selecting the configuration with the highest mean validation score.

4.7 Model Evaluation and Overfitting Prevention

To assess the decision tree model and guard against overfitting, the following strategies were implemented:

- **Cross-Validation**: Employed five-fold cross-validation to tune hyperparameters and ensure consistent performance, providing a reliable estimate of the model's generalization ability.
- **Training and Test Errors**: Calculated to evaluate generalization. A minimal gap between training and test errors indicates good generalization, while a significant gap suggests overfitting.
- **Confusion Matrices**: Constructed for both training and test sets to analyze true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), offering a detailed view of classification performance.
- **Performance Metrics**: Computed precision, recall, F1-score, and accuracy to provide a comprehensive evaluation of the model's effectiveness.
- **Stopping Criteria**: Applied maximum depth and minimum samples split to control tree complexity and prevent overfitting.

4.7.1 Cross-Validation Results

The five-fold cross-validation results are detailed in Appendix **??**. Key findings include:

- **Optimal Hyperparameters**: A maximum depth of 20, a minimum sample split of 2, and the Gini impurity criterion yielded the best performance with a mean cross-validation accuracy of 99.83%.
- **Model Stability**: Low variance across cross-validation folds indicates stable performance and low sensitivity to specific data subsets.
- **Balanced Performance**: The chosen hyperparameters strike a balance between bias and variance, avoiding both underfitting and overfitting.

These results informed the final hyperparameter settings used for training the decision tree on the entire training dataset, ensuring optimal performance.

4.7.2 Training and Test Errors

Both training and test errors were evaluated to gauge the model's generalization:

- **Training Error**: Reflects the model's fit to the training data. A low training error indicates effective pattern learning but may also suggest overfitting if excessively low.
- **Test Error**: Assesses performance on unseen data, providing an unbiased estimate of generalization capability. A small gap between training and test errors signifies good generalization.

Evaluating both errors offers a comprehensive understanding of the model's ability to learn from training data while maintaining robustness on new data.

4.7.3 Confusion Matrices

Confusion matrices for the decision tree model are presented in Tables 5 and 6 for the training and test sets, respectively. These matrices detail the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), providing insights into the model's classification performance.

4.7.4 Performance Metrics

Key performance metrics for the decision tree model include:

- **Accuracy**: The ratio of correct predictions to total predictions, calculated as $\frac{TP+TN}{TP+TN+FP+FN}$.
- **Precision**: The ratio of true positive predictions to the total positive predictions, defined as $\frac{TP}{TP+FP}$.
- **Recall**: The ratio of true positive predictions to the actual positive instances, calculated as $\frac{TP}{TP+FN}$.
- **F1-Score**: The harmonic mean of precision and recall, given by $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.

4.7.5 Overfitting Prevention Techniques

To mitigate overfitting, the following methods were employed:

- **Limiting Tree Depth**: Setting a maximum depth restricts the tree's complexity.
- **Minimum Samples Split**: Ensuring a minimum number of samples required to split a node promotes generalization.
- **Cross-Validation**: Validating the model across multiple data splits during hyperparameter tuning ensures performance generalization.

5 Decision Tree Results

Post-training with optimized hyperparameters, the decision tree's performance was evaluated on both training and test datasets.

5.1 Performance Evaluation

Table 4 summarizes the performance metrics of the decision tree model.

Metric	Training Set	Test Set
Accuracy	100.00%	99.80%
Precision	100.00%	99.76%
Recall	99.99%	99.79%
F1-Score	100.00%	99.78%

Table 4: Decision Tree Performance Metrics

5.2 Confusion Matrix

Confusion matrices for the training and test sets are shown in Tables 5 and 6.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP = 13,555	FN = 1
Actual Negative	FP = 0	TN = 16,004

Table 5: Decision Tree Confusion Matrix - Training Set

	Predicted Positive	**Predicted Negative**
Actual Positive	TP = 3,381	FN = 7
Actual Negative	FP = 8	TN = 3,992

Table 6: Decision Tree Confusion Matrix - Test Set

5.3 Interpretation of Results

The decision tree achieved perfect accuracy on the training set, indicating a strong fit to the training data. On the test set, accuracy slightly decreased to 99.80%, suggesting excellent generalization with minimal overfitting. The few misclassifications on the test set (7 FN and 8 FP) demonstrate the model's high efficacy in distinguishing between edible and poisonous mushrooms.

6 Random Forest

This section examines the Random Forest model, covering its theoretical basis, implementation, hyperparameter tuning, evaluation, and results.

Random Forests are ensemble learning methods that construct multiple decision trees and aggregate their predictions to enhance accuracy and stability. They are particularly adept at reducing overfitting and improving generalization compared to individual decision trees.

6.1 Structure and Functionality

A random forest comprises numerous decision trees, each trained on a bootstrap sample of the training data. During training, each tree considers a random subset of features for splitting, introducing diversity among the trees. The final prediction is obtained by aggregating the individual trees' predictions, typically through majority voting for classification tasks.

6.2 Building and Growing the Forest

Constructing a random forest involves:

- **Bootstrap Sampling**: Drawing random samples with replacement from the training dataset for each tree.
- **Random Feature Selection**: Selecting a random subset of features at each split to reduce correlation among trees.
- **Tree Construction**: Growing each decision tree to its maximum depth without pruning, allowing each tree to capture complex data patterns.

6.2.1 Advantages of Random Forests

- **Overfitting Reduction**: Aggregating multiple trees diminishes the overfitting risk inherent in single trees.
- **Enhanced Accuracy**: The ensemble approach typically results in higher predictive accuracy.
- **Robustness**: Random forests are resilient to noise and outliers in the data.
- **Feature Importance**: They provide estimates of feature importance, aiding in feature selection and model interpretation.

While random forests offer improved accuracy and robustness, they require more computational resources and are less interpretable compared to single decision trees.

6.3 Implementation

The random forest classifier was developed using a custom `RandomForest` class, building upon the previously implemented `DecisionTree` class. This ensemble method involves creating multiple decision trees and combining their predictions.

6.3.1 RandomForest Class Design

The `RandomForest` class includes:

- **Ensemble of Trees**: Maintains a list of decision tree instances, each trained on a different bootstrap sample.
- **Bootstrap Sampling**: Generates diverse training subsets for each tree by sampling with replacement.
- **Random Feature Selection**: Selects a random subset of features at each split to reduce tree correlation.
- **Prediction Aggregation**: Combines individual tree predictions through majority voting for classification.
- **Hyperparameters**: Allows customization of parameters like the number of estimators, maximum depth, minimum samples split, splitting criterion, and maximum features considered for splitting.

6.3.2 Process of Building the Forest

Building the random forest involves:

1. **Initializing Trees**: Creating a specified number of decision tree instances.
2. **Training Trees**: Training each tree on a unique bootstrap sample and selecting random feature subsets at each split.
3. **Prediction**: Aggregating predictions from all trees to determine the final output through majority voting.

6.4 Hyperparameter Tuning

Hyperparameter optimization was performed to enhance the random forest's performance. The tuned hyperparameters include:

- **Number of Estimators (`n_estimators`)**: Determines the number of trees in the forest. More trees can improve performance but increase computational cost.
- **Maximum Depth (`max_depth`)**: Sets the maximum depth of each tree to control complexity.

- **Minimum Samples Split** (`min_samples_split`): Defines the minimum number of samples required to split a node, promoting generalization.
- **Splitting Criterion** (`criterion`): Specifies the function to measure split quality, such as Gini impurity or entropy.
- **Maximum Features** (`max_features`): Determines the number of features considered for splitting at each node, reducing tree correlation.

Cross-validation was utilized to evaluate various hyperparameter combinations, selecting the configuration with the highest mean validation score.

6.5 Model Evaluation and Overfitting Prevention

To assess the random forest model and prevent overfitting, the following strategies were employed:

- **Cross-Validation**: Conducted five-fold cross-validation to tune hyperparameters and ensure consistent performance, providing a reliable estimate of generalization ability.
- **Training and Test Errors**: Calculated to evaluate generalization, with a small gap indicating good performance.
- **Confusion Matrices**: Created for both training and test sets to analyze TP, TN, FP, and FN, offering a detailed view of classification performance.
- **Performance Metrics**: Computed precision, recall, F1-score, and accuracy for a comprehensive evaluation.
- **Stopping Criteria**: Applied maximum depth and minimum samples split to control tree complexity and prevent overfitting.

6.5.1 Cross-Validation Results

The five-fold cross-validation outcomes are detailed in Appendix **??**. Key observations include:

- **Optimal Hyperparameters**: A maximum depth of 20, a minimum sample split of 2, and the Gini impurity criterion achieved the highest mean cross-validation accuracy of 99.93%.
- **Model Stability**: Low variance across folds indicates stable and reliable performance.
- **Balanced Performance**: The selected hyperparameters balance bias and variance, avoiding underfitting and overfitting.

These results guided the final hyperparameter settings for training the random forest on the entire training dataset, ensuring optimal performance.

6.5.2 Training and Test Errors

Both training and test errors were assessed to determine the model's generalization:

- **Training Error**: Measures the proportion of incorrect predictions on the training data. A low training error indicates effective learning but may suggest overfitting if excessively low.
- **Test Error**: Measures the proportion of incorrect predictions on unseen data, providing an unbiased estimate of generalization capability. A small gap between training and test errors indicates good generalization.

Evaluating both errors offers a comprehensive view of the model's ability to learn from training data while maintaining robustness on new data.

6.5.3 Confusion Matrices

Confusion matrices for the random forest model are presented in Tables 8 and 9 for the training and test sets, respectively. These matrices detail TP, TN, FP, and FN counts, providing insights into the model's classification performance.

6.5.4 Performance Metrics

Key performance metrics for the random forest model include:

- **Accuracy**: The ratio of correct predictions to total predictions, calculated as $\frac{TP+TN}{TP+TN+FP+FN}$.
- **Precision**: The ratio of true positive predictions to the total positive predictions, defined as $\frac{TP}{TP+FP}$.
- **Recall**: The ratio of true positive predictions to the actual positive instances, calculated as $\frac{TP}{TP+FN}$.
- **F1-Score**: The harmonic mean of precision and recall, given by $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.

6.5.5 Overfitting Prevention Techniques

To avoid overfitting in the random forest model, the following methods were applied:

- **Limiting Tree Depth**: Setting a maximum depth restricts each tree's complexity.
- **Minimum Samples Split**: Ensuring a minimum number of samples required to split a node promotes generalization.
- **Cross-Validation**: Validating the model across multiple data splits during hyperparameter tuning ensures performance generalization.

7 Random Forest Results

After training the random forest model with optimized hyperparameters, its performance was evaluated on both the training and test datasets.

7.1 Performance Evaluation

Table 7 summarizes the random forest model's performance metrics.

Metric	Training Set	Test Set
Accuracy	99.99%	99.96%
Precision	99.99%	99.97%
Recall	99.99%	99.94%
F1-Score	99.99%	99.96%

Table 7: Random Forest Performance Metrics

7.2 Confusion Matrix

Confusion matrices for the random forest model are shown in Tables 8 and 9.

	Predicted Positive	Predicted Negative
Actual Positive	TP = 13,554	FN = 2
Actual Negative	FP = 2	TN = 16,002

Table 8: Random Forest Confusion Matrix - Training Set

	Predicted Positive	Predicted Negative
Actual Positive	TP = 3,386	FN = 2
Actual Negative	FP = 1	TN = 3,999

Table 9: Random Forest Confusion Matrix - Test Set

7.3 Interpretation of Results

The random forest model demonstrated superior generalization on the test set with an accuracy of 99.96%. The minimal misclassifications (2 FN and 1 FP) indicate exceptional performance in distinguishing between edible and poisonous mushrooms, outperforming the decision tree model.

8 Comparison between Decision Tree and Random Forest

This section compares the Decision Tree and Random Forest models based on their performance metrics and inherent characteristics.

8.1 Performance Comparison

Metric	Decision Tree	Random Forest
Training Accuracy	100.00%	99.99%
Test Accuracy	99.80%	99.96%
Training F1-Score	100.00%	99.99%
Test F1-Score	99.78%	99.96%

Table 10: Comparison of Model Performance

8.2 Interpretation of Results

- **Accuracy**: Both models exhibit high accuracy, with the random forest slightly outperforming the decision tree on the test set.
- **Overfitting**: The decision tree achieved perfect training accuracy, suggesting potential overfitting, whereas the random forest demonstrated better generalization.
- **Robustness**: The random forest's ensemble nature provides higher robustness against data variations.
- **Interpretability**: The decision tree offers greater interpretability, allowing for straightforward visualization and understanding, while the random forest, comprising multiple trees, is less transparent.

While both models perform exceptionally well, the random forest offers enhanced generalization and robustness, making it preferable when overfitting is a concern. However, if model interpretability is paramount, the decision tree may be more suitable.

9 Conclusion

This project involved the implementation of decision tree and random forest classifiers to classify mushrooms as poisonous or edible. Both models achieved near-perfect performance, effectively capturing the necessary patterns in the data. Through enhancements and meticulous hyperparameter tuning, the random forest model slightly outperformed the decision tree on unseen data, demonstrating superior generalization and robustness.

A Appendix

A.1 Cross-Validation Results for Decision Tree

Combination	Max Depth	Min Samples Split	Criterion	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean CV Score
1	10	2	gini	0.9452	0.9440	0.9447	0.9452	0.9374	0.9433
2	10	2	entropy	0.9306	0.8676	0.8831	0.8833	0.8853	0.8900
3	10	2	error	0.8698	0.8615	0.8490	0.8461	0.8706	0.8594
4	10	5	gini	0.9452	0.9440	0.9447	0.9452	0.9374	0.9433
5	10	5	entropy	0.9306	0.8676	0.8831	0.8833	0.8853	0.8900
6	10	5	error	0.8698	0.8615	0.8490	0.8461	0.8706	0.8594
7	20	2	gini	0.9978	0.9985	0.9983	0.9980	0.9988	0.9983
8	20	2	entropy	0.9937	0.9983	0.9976	0.9971	0.9981	0.9970
9	20	2	error	0.9494	0.9543	0.9362	0.9487	0.9582	0.9494
10	20	5	gini	0.9980	0.9983	0.9978	0.9980	0.9988	0.9982
11	20	5	entropy	0.9931	0.9985	0.9976	0.9964	0.9983	0.9968
12	20	5	error	0.9494	0.9543	0.9357	0.9487	0.9582	0.9493

Table 11: Cross-Validation Results for Decision Tree

A.2 Cross-Validation Results for Random Forest

Combination	n_estimators	Max Depth	Min Samples Split	Criterion	Max Features	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean CV Score
1	10	10	2	gini	sqrt	0.9628	0.9658	0.9537	0.9459	0.9511	0.9559
2	10	10	2	gini	log2	0.9689	0.9606	0.9712	0.9516	0.9699	0.9644
3	10	10	2	entropy	sqrt	0.9474	0.9268	0.9405	0.9239	0.9293	0.9336
4	10	10	2	entropy	log2	0.9197	0.9362	0.9142	0.9354	0.9506	0.9312
5	10	10	2	error	sqrt	0.9242	0.9320	0.9078	0.9190	0.9071	0.9180
6	10	10	2	error	log2	0.8950	0.9300	0.9381	0.9347	0.8985	0.9192
7	10	10	5	gini	sqrt	0.9242	0.9474	0.9606	0.9699	0.9630	0.9530
8	10	10	5	gini	log2	0.9648	0.9704	0.9525	0.9650	0.9406	0.9587
9	10	10	5	entropy	sqrt	0.9142	0.9283	0.9381	0.9339	0.9501	0.9329
10	10	10	5	entropy	log2	0.9393	0.9352	0.9423	0.9124	0.9423	0.9343
11	10	10	5	error	sqrt	0.9276	0.8792	0.9210	0.9234	0.9200	0.9142
12	10	10	5	error	log2	0.9131	0.9181	0.9266	0.9372	0.9205	0.9231
13	10	20	2	gini	sqrt	0.9988	0.9992	0.9978	0.9995	0.9997	0.9990
14	10	20	2	gini	log2	0.9993	0.9995	0.9993	0.9993	0.9993	0.9993
15	10	20	2	entropy	sqrt	0.9992	0.9983	0.9980	0.9992	0.9990	0.9987
16	10	20	2	entropy	log2	0.9988	0.9988	0.9985	0.9990	0.9995	0.9989
17	10	20	2	error	sqrt	0.9980	0.9958	0.9937	0.9919	0.9939	0.9947
18	10	20	2	error	log2	0.9939	0.9904	0.9949	0.9949	0.9865	0.9921
19	10	20	5	gini	sqrt	0.9998	0.9996	0.9990	0.9993	0.9998	0.9995
20	10	20	5	gini	log2	0.9976	0.9990	0.9983	0.9990	0.9995	0.9987
21	10	20	5	entropy	sqrt	0.9988	0.9986	0.9983	0.9976	0.9988	0.9984
22	10	20	5	entropy	log2	0.9958	0.9985	0.9983	0.9988	0.9976	0.9978
23	10	20	5	error	sqrt	0.9941	0.9905	0.9949	0.9948	0.9937	0.9936
24	10	20	5	error	log2	0.9914	0.9942	0.9926	0.9941	0.9899	0.9924
25	20	10	2	gini	sqrt	0.9569	0.9660	0.9670	0.9557	0.9619	0.9615
26	20	10	2	gini	log2	0.9716	0.9723	0.9675	0.9608	0.9624	0.9669
27	20	10	2	entropy	sqrt	0.9357	0.9393	0.9408	0.9653	0.9560	0.9474
28	20	10	2	entropy	log2	0.9506	0.9254	0.9119	0.9306	0.9415	0.9320
29	20	10	2	error	sqrt	0.9337	0.9285	0.9164	0.9244	0.9078	0.9217
30	20	10	2	error	log2	0.9278	0.9119	0.9369	0.9252	0.9356	0.9275
31	20	10	5	gini	sqrt	0.9501	0.9624	0.9555	0.9750	0.9682	0.9622
32	20	10	5	gini	log2	0.9603	0.9628	0.9718	0.9718	0.9702	0.9674
33	20	10	5	entropy	sqrt	0.9396	0.9369	0.9711	0.9511	0.9427	0.9483
34	20	10	5	entropy	log2	0.9401	0.9403	0.9560	0.9604	0.9139	0.9422
35	20	10	5	error	sqrt	0.9313	0.9285	0.9164	0.9244	0.9078	0.9217
36	20	10	5	error	log2	0.9131	0.9181	0.9266	0.9372	0.9205	0.9231
37	20	20	2	gini	sqrt	0.9995	0.9997	0.9990	0.9993	0.9998	0.9995
38	20	20	2	gini	log2	0.9992	0.9995	0.9993	0.9993	0.9993	0.9993
39	20	20	2	entropy	sqrt	0.9992	0.9993	0.9997	0.9992	0.9992	0.9993
40	20	20	2	entropy	log2	0.9995	0.9993	0.9986	0.9995	0.9993	0.9993
41	20	20	2	error	sqrt	0.9973	0.9954	0.9927	0.9966	0.9953	0.9955
42	20	20	2	error	log2	0.9948	0.9985	0.9942	0.9956	0.9958	0.9958
43	20	20	5	gini	sqrt	0.9995	0.9997	0.9990	0.9986	0.9995	0.9993
44	20	20	5	gini	log2	0.9997	0.9997	0.9992	0.9997	0.9993	0.9995
45	20	20	5	entropy	sqrt	0.9997	0.9988	0.9986	0.9990	0.9993	0.9991
46	20	20	5	entropy	log2	0.9998	0.9997	0.9978	0.9990	0.9992	0.9991
47	20	20	5	error	sqrt	0.9963	0.9973	0.9944	0.9970	0.9963	0.9962
48	20	20	5	error	log2	0.9973	0.9973	0.9956	0.9971	0.9932	0.9961

Table 12: Cross-Validation Results for Random Forest

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.