Università degli studi di Milano

Machine Learning
Project 2

Student :
Fabio Casalingo

Teacher :
Niccolò Cesa-Bianchi

October 16, 2024

Contents

[0]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.

1 Introduction

1.1 Data semantic

Within the "secondary_data" dataset, we find information on 61.069 entries referring to mushrooms. Specifically, there are 21 columns, each representing a characteristic of the mushrooms (see table 1). In the dataset, we find only 3 variables of the float type, while all the others are categorical (18).

Name	Description	Type	Example
class	Classification of mushroom (e.g., edible, poisonous)	object	p
cap-diameter	Size of mushroom cap in centimeters	float64	15.26
cap-shape	Shape of the mushroom cap	object	x
cap-surface	Texture of the mushroom cap surface	object	g
cap-color	Color of the mushroom cap	object	o
does-bruise-or-bleed	Whether the mushroom bruises or bleeds when cut	object	f
gill-attachment	How the gills are attached to the stem	object	e
gill-spacing	Spacing between the gills	object	f
gill-color	Color of the gills	object	w
stem-height	Height of the mushroom stem in centimeters	float64	16.95
stem-width	Width of the mushroom stem in centimeters	float64	17.09
stem-root	Type of stem base or root	object	s
stem-surface	Texture of the stem surface	object	y
stem-color	Color of the stem	object	w
veil-type	Type of veil on the mushroom	object	u
veil-color	Color of the veil	object	w
has-ring	Presence of a ring on the stem	object	t
ring-type	Type of ring on the stem	object	g
spore-print-color	Color of the spore print	object	k
habitat	Natural habitat of the mushroom	object	d
season	Season when the mushroom typically grows	object	w

Table 1: Dataset

1.2 Data distribution

To analyze the distribution of the 'class' feature (see Figure 1), a bar plot was used. This allowed us to observe that the dataset contains more poisonous mushrooms (33.888) than non-poisonous ones (27.181).

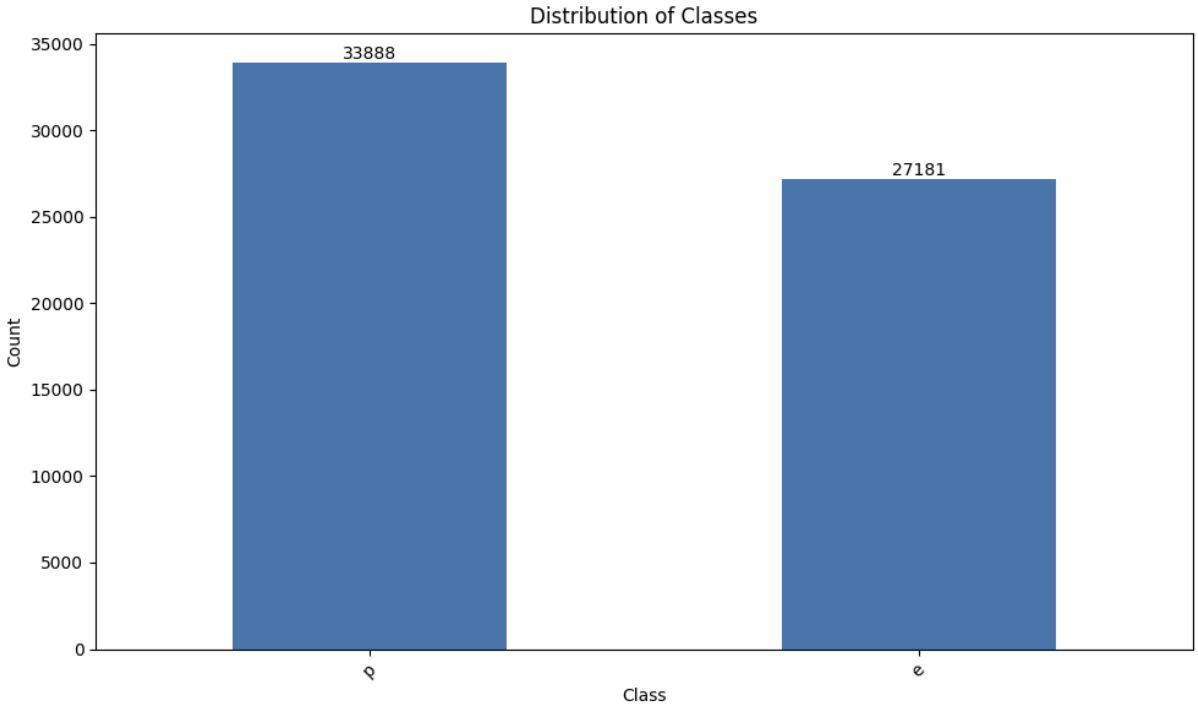

Figure 1: Class distribution

1.3 Data management

The data management process involved several steps to address missing values and ensure the dataset's integrity for model training.

First, we performed an analysis to check for missing values in each column. The results showed significant amounts of missing data in several features, with columns such as capsurface, gillattachment, gillspacing, stemroot, stemsurface, veiltype, veilcolor, ringtype, and sporeprintcolor having varying degrees of missing values. Notably, stem-root, veil-type, and spore-print-color had over 40,000 missing entries each, making them candidates for exclusion from further analysis.

To manage the missing data, we decided to drop columns with more than 70% missing values to prevent potential biases and inaccuracies in the model. This resulted in the removal of the columns stemroot, veiltype, veilcolor, and sporeprintcolor.

2 Decision Tree

A decision tree classifier is a non-parametric supervised learning algorithm that represents one of the most intuitive and widely used techniques in the

field of machine learning. It models decisions as a tree-like structure where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf represents the class or predicted value.

The primary objective of a decision tree is to construct a predictive model that, given a training dataset, can accurately classify new instances. The tree is constructed in a hierarchical manner, starting from a root node that contains all the data. At each level, the node is split into sub-nodes based on a splitting criterion, such as Gini impurity or information gain, which measures the homogeneity of the classes within each sub-node. This process continues until a stopping criterion is met, such as a minimum number of instances in each leaf or a maximum depth of the tree.

Simplicity and interpretability are among the main advantages of decision trees. The tree structure allows for a clear visualization of the decision rules and a straightforward understanding of the classification process. Furthermore, decision trees can handle both numerical and categorical data, and they do not require any assumptions about the data distribution. However, decision trees are prone to overfitting, meaning they tend to fit the training data too closely, thus losing the ability to generalize to new data. To mitigate this problem, pruning techniques are commonly applied, which involve removing unnecessary branches from the tree.

Decision trees form the basis of many more complex algorithms, such as random forests, which combine multiple decision trees to improve performance and reduce overfitting.

2.1 Node

In a decision tree, the structure is composed of various types of nodes that work together to make classification decisions based on input data. Each node plays a crucial role in guiding the flow of data down the tree, enabling the model to make predictions efficiently. Understanding how these nodes work is essential for grasping the mechanics of a decision tree classifier.

2.1.1 Definition of a Node

A node in a decision tree represents a decision point or a test on an attribute of the dataset. Depending on the result of this test, data instances are directed down different branches of the tree. There are three main types of nodes in a decision tree:

- Root Node: The root node is the topmost node of the tree and contains the entire dataset. It serves as the first decision point, where the model applies a splitting criterion to divide the data into two or more subsets
- Internal Nodes: These nodes appear between the root and the leaves and represent decision points at intermediate levels of the tree. Each internal node tests a specific feature or attribute, using a criterion like Gini impurity or information gain, to determine how the data should be split into more homogeneous subsets.
- Leaf Nodes: Leaf nodes, also known as terminal nodes, represent the final classification outcomes. Once data reaches a leaf node, no further splitting occurs, and the class label or predicted value is assigned based on the majority class of the instances present in that leaf.

2.2 How Nodes Make Decisions

At each internal node, the decision tree evaluates a specific feature or attribute from the input data. The splitting process is based on the evaluation of that feature against a criterion designed to reduce impurity or maximize the separation between classes. This decision-making process involves several steps:

2.2.1 Selecting the Feature for Splitting

The decision on which feature to split the data is crucial for building an effective model. The decision tree algorithm scans through all possible features and chooses the one that best separates the data into homogeneous subsets. This is done using a splitting criterion.

- Entropy is a measure from information theory that quantifies the amount of uncertainty or randomness in a system. In the context of decision trees, entropy measures the impurity or disorder of a node. The formula for entropy is:

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

- Where:
  - S is the current dataset (or subset) at a node,

  - c is the number of classes,
  - p_i is the proportion of instances in class i.

- The entropy value ranges between 0 and 1:
  - **Entropy = 0**: The node is pure (all instances belong to the same class).
  - **Entropy = 1**: The node is impure (the instances are equally distributed across all classes).

- **Information Gain**
  - When splitting a dataset on a particular feature, the decision tree algorithm computes the **Information Gain (IG)**, which measures how much entropy is reduced after the split. Information Gain is given by:

$$\text{IG}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

  - Where:
    * A is a feature being split,
    * S_v is the subset of data for which feature A has value v.
  - The goal of the algorithm is to maximize the Information Gain by selecting the feature that results in the largest reduction of entropy, meaning it provides the most "information" for making the classification.

- **Gini Index**
  - The Gini Index is another measure of impurity or disorder, but it is slightly different from entropy. The formula for the Gini Index is:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} p_i^2$$

  - Where:
    * p_i is the proportion of instances belonging to class i.

  - Similar to entropy, the Gini Index measures how impure a node is, but with a slightly different mathematical formulation:
    * **Gini = 0**: The node is pure (all instances belong to the same class).
    * **Gini = 0.5**: The node is maximally impure with a binary classification (the instances are evenly distributed between the two classes).
  - In a binary classification problem, the Gini Index tends to favor features that lead to larger, more balanced splits, unlike entropy, which is more sensitive to small changes in class distribution.

- **Gini Gain**
  - Just like Information Gain is used to measure the effectiveness of a split with entropy, Gini Gain is used with the Gini Index. The Gini Gain represents the reduction in Gini impurity after splitting a node. It is computed as:

$$\text{Gini Gain}(S, A) = \text{Gini}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Gini}(S_v)$$

2.3 Test Condition at the Node

Once the feature is selected, a test condition is applied to decide how the data will be split. For numerical features, the test might compare the feature's value to a threshold. For categorical features, the test might check if a value belongs to a particular set or category.

For example:

- **Numerical Feature (e.g., Age)**: A node might test if `Age < 30`, splitting the data into two groups: those younger than 30 and those 30 or older.
- **Categorical Feature (e.g., Gender)**: A node might test if `Gender = Male`, splitting the data into two groups: one with males and the other with non-males.

These tests are performed recursively at each internal node until a stopping condition is met, such as reaching a leaf node or fulfilling a predetermined stopping criterion like maximum tree depth.

2.4 Recursive Splitting Process

The decision tree grows in a recursive manner. Starting at the root node, the dataset is progressively divided at each internal node until the leaf nodes are reached. Each split aims to make the resulting child nodes more homogeneous (i.e., containing instances of mostly one class), thereby improving the predictive power of the model.

The recursion can be described in the following steps:

- **Initial Split**: At the root node, the entire dataset is divided based on the best feature selected according to the splitting criterion.
- **Recursive Splitting**: For each internal node, the subset of data is further split by testing the next most appropriate feature.
- **Termination at Leaf Nodes**: The recursive splitting continues until one of the following stopping conditions is met:
  - The node becomes **pure**, meaning all instances belong to the same class.
  - A predefined depth limit is reached.
  - The number of instances in the node is too small to justify further splits.

2.5 Hyperparameters in Decision Trees

Hyperparameters are configuration settings that govern the training process of a decision tree and affect its performance, complexity, and generalization ability. By tuning these hyperparameters, we can control how the decision tree grows and how well it performs on new, unseen data. Key hyperparameters for decision trees include:

- Maximum Depth (max_depth): Controls the maximum depth of the tree. Deeper trees can capture more complex patterns but may lead to overfitting.
- Minimum Samples Split (min_samples_split): The minimum number of samples required to split an internal node. Higher values prevent creating nodes with few samples, which can help reduce overfitting.

- Maximum Features (max_features): The number of features to consider when looking for the best split. It can be an integer, float, or one of 'auto', 'sqrt', 'log2'.
- Criterion: The function used to measure the quality of a split. Options typically include 'gini' for Gini impurity and 'entropy' for information gain.

2.6 Precision, Recall, F1 Score

- **Precision:**

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Precision measures the accuracy of positive predictions. It answers the question: "Of all the instances the model labeled as positive, what fraction was actually positive?" High precision indicates a low false positive rate.

- **Recall:**

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Recall measures the completeness of positive predictions. It answers the question: "Of all the actual positive instances, what fraction did the model correctly identify?" High recall indicates a low false negative rate.

- **F1 Score:**

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. An F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

These metrics are particularly useful in binary classification problems and in cases where class imbalance is present. The choice between prioritizing precision, recall, or a balance of both (F1 score) depends on the specific requirements of the problem at hand.

For example, in a medical diagnosis scenario, high recall might be prioritized to ensure that all potential cases of a serious condition are identified,

even if it means more false positives. In contrast, in a spam email detection system, high precision might be favored to avoid incorrectly labeling important emails as spam.

The F1 score is useful when you want to find an optimal balance between precision and recall, and there is an uneven class distribution.

2.7 Results

Our analysis of the decision tree model yielded promising results. The optimal hyperparameters, determined through cross-validation, were as follows:

- Maximum depth: None (unlimited)
- Minimum samples for split: 5
- Split criterion: Entropy
- Maximum features: None (all features considered)

Using these parameters, we achieved a best cross-validation score of 0.9981, indicating excellent model performance.

The model's performance on both the training and test sets was exceptional:

- Training error: 6.14×10^{-5}
- Test error: 0.0017
- Training accuracy: 99.99%
- Test accuracy: 99.83%
- Precision: 0.998
- Recall: 0.998
- F1 Score: 0.998

These results demonstrate that our decision tree model has learned the patterns in the data extremely well, with near-perfect accuracy on both the training and test sets. The minimal difference between training and test performance suggests that the model generalizes well to unseen data, with only a slight indication of overfitting.

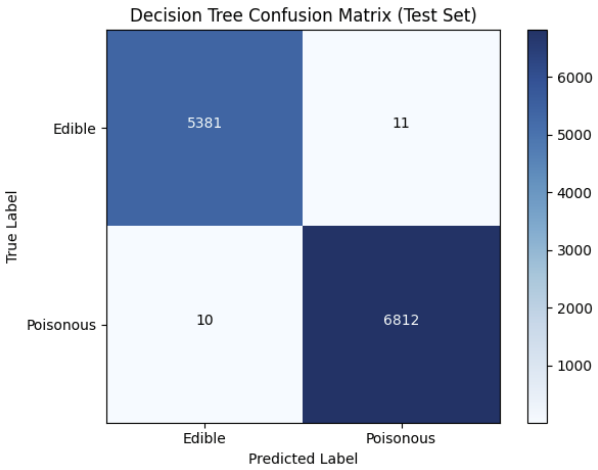

Figure 2: Decision tree confusion matrix

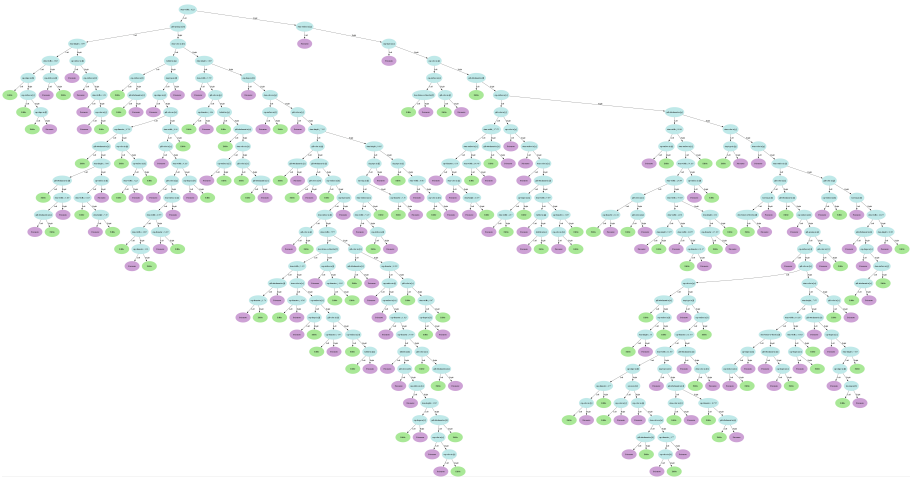

Figure 3: Decision tree

It's worth noting that while these results are impressive, such high accuracy might warrant further investigation to ensure that there are no data leakage issues or overly predictive features that could be skewing the results.

3 Random Forest

Random Forest is an ensemble learning method that builds upon decision trees to improve their accuracy, reduce overfitting, and increase robustness. Instead of relying on a single decision tree, a Random Forest aggregates the predictions of many trees, each built on different random subsets of the data and features. This method capitalizes on the "wisdom of the crowd" to make more reliable predictions.

3.1 Key Concepts

The Random Forest algorithm is based on two primary techniques: Bagging (Bootstrap Aggregating) and Random Feature Selection. These techniques introduce randomness into the model-building process, which helps to prevent overfitting and improve generalization.

3.1.1 Bagging (Bootstrap Aggregating)

Bagging is an ensemble technique that generates multiple versions of a predictor (in this case, decision trees) by resampling the training data. The process can be described as follows:

- Bootstrap Sampling: The training dataset is sampled with replacement, creating multiple subsets of data, each of which is used to train a different decision tree. This means that each tree in the forest sees a slightly different version of the data, introducing diversity among the trees.
- Aggregation: Once each tree has made a prediction, the Random Forest aggregates these predictions. For classification tasks, this is done by majority voting—each tree "votes" for a class, and the class with the most votes is selected as the final prediction. For regression tasks, the predictions are averaged.

Bagging helps reduce the variance of the model, which is one of the main weaknesses of individual decision trees. By averaging the predictions of multiple trees, the Random Forest creates a more stable model that is less sensitive to fluctuations in the training data.

3.1.2 Random Feature Selection

In addition to sampling the data, Random Forests introduce an extra layer of randomness by selecting a random subset of features at each node when growing a tree. This ensures that each tree explores a different feature space, further promoting diversity and reducing correlation between the trees. The steps involved in random feature selection are:

- Random Subset of Features: At each node, a random subset of the features is selected, and only this subset is considered for the best split. This process prevents any one feature from dominating the trees and encourages a wider exploration of the feature space.
- Reduction in Overfitting: By limiting the number of features considered at each node, Random Forest reduces the risk of overfitting, which can occur when trees grow too deep or focus too narrowly on particular patterns in the data.

3.2 Advantages of Random Forest

Random Forests have several advantages over individual decision trees:

- Improved Accuracy: By combining multiple trees, Random Forests typically achieve better predictive performance than a single decision tree, especially on large and complex datasets.
- Robustness to Noise: The use of bagging and random feature selection makes Random Forests less sensitive to noise in the data. Outliers and misclassifications in the training data have a limited impact on the overall model.
- Reduced Overfitting: Decision trees are prone to overfitting, especially when they are grown deep. Random Forest mitigates this issue by averaging the predictions of multiple trees, reducing the variance and producing a more generalized model.
- Handles Missing Data: Random Forests can handle missing values by calculating predictions based on the majority vote of trees that can make a prediction with the available data.
- Feature Importance: Random Forests provide a mechanism for evaluating feature importance, which can help with feature selection. By observing how much each feature contributes to reducing impurity across all trees, the model can rank the relative importance of each feature.

3.3 How Random Forest Works

The process of training and using a Random Forest model can be broken down into the following steps:

- Step 1: Bootstrapping the Data: The algorithm first generates multiple training datasets by sampling the original training data with replacement. Each dataset is then used to grow a different decision tree.
- Step 2: Growing Decision Trees: For each tree, a random subset of features is chosen at each node, and the best split is determined based on the selected features (e.g., using Gini Index or Information Gain). The tree is grown until a stopping criterion is met (e.g., maximum depth or minimum number of instances in a node).
- Step 3: Aggregating Predictions: Once all the trees have been grown, the Random Forest aggregates their predictions. For classification, the majority class across all trees is selected; for regression, the average of all tree predictions is taken.

3.4 Random Forest Hyperparameters

Several key hyperparameters influence the performance of a Random Forest model. The most important ones include:

- Number of Trees (n_estimators): This defines how many trees will be grown in the forest. A higher number of trees typically leads to better performance but at the cost of increased computational resources.
- Maximum Depth (max_depth): This controls how deep each tree can grow. Deeper trees can capture more complex patterns but may lead to overfitting.
- Minimum Samples Split (min_samples_split): This parameter defines the minimum number of samples required to split an internal node. Increasing this value can help prevent overfitting by making splits less frequent.
- Number of Features (max_features): This controls how many features are considered for the best split at each node. Smaller subsets reduce correlation among trees and can help improve generalization.
- Criterion: This parameter determines the function used to measure the quality of a split. Common options are Gini impurity and 'entropy' for information gain. The choice can affect how the trees are grown and, consequently, the model's performance.

3.5 Limitations of Random Forest

Despite its many strengths, Random Forest has some limitations:

- Computational Cost: Training a large number of trees can be computationally expensive and memory-intensive, especially for large datasets.
- Interpretability: While individual decision trees are easy to interpret, Random Forest models are more complex. Understanding the contribution of each feature to the final prediction can be difficult, though feature importance metrics can help.
- Overfitting with Noisy Data: While Random Forest is less prone to overfitting than individual trees, it can still overfit if the trees are allowed to grow too deep, or if the training data is noisy.

3.6 Results

Our analysis of the Random Forest model yielded exceptional results. The optimal hyperparameters, determined through cross-validation, were as follows:

- Number of estimators: 30
- Maximum depth: None (unlimited)
- Minimum samples for split: 2
- Split criterion: Gini
- Maximum features: 4

Using these parameters, we achieved outstanding performance metrics:

- Best cross-validation score: 0.9999
- Out-of-Bag (OOB) Score: 0.9996
- Training error: 2.05×10^{-5}
- Test error: 0.0000
- Training accuracy: 99.998%
- Test accuracy: 100.00%
- Precision: 1.0
- Recall: 1.0
- F1 Score: 1.0

These results demonstrate that our Random Forest model has achieved near-perfect performance. The model exhibits exceptional accuracy on both the training and test sets, with the test set showing perfect classification.

The extremely low training error and zero test error, coupled with the high OOB score, suggest that the Random Forest model has captured the underlying patterns in the data exceptionally well. The perfect test accuracy indicates that the model correctly classified all instances in the unseen test data.

Overall, the Random Forest model demonstrates superior performance compared to the previously discussed Decision Tree model, showing improved accuracy and generalization.

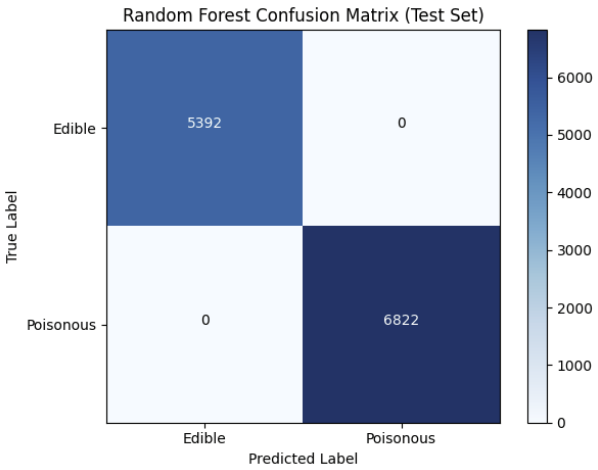

Figure 4: Random forest confusion matrix

4 Conclusion

This project explored the application of two powerful machine learning algorithms Decision Trees and Random Forests to the task of mushroom classification. Both models demonstrated exceptional performance, achieving near-perfect accuracy in distinguishing between edible and poisonous mushrooms based on various morphological features.

4.1 Key Findings

Decision Tree Model

Achieved 99.99% accuracy on the training set and 99.83% on the test set. Demonstrated excellent generalization with only a slight indication of overfitting. Best cross-validation score: 0.9981.

4.1.1 Random Forest Model

Outperformed the Decision Tree with 99.998% accuracy on the training set and a perfect 100% on the test set. Exhibited superior generalization capabilities. Best cross-validation score: 0.9999. Out-of-Bag (OOB) Score: 0.9996.

4.1.2 Feature Importance

Stem width emerged as the most critical feature for classification. Cap surface and gill attachment were also highly influential.

The Random Forest model's superior performance can be attributed to its ensemble nature, which combines multiple decision trees to reduce overfitting and improve overall

accuracy. This approach proved particularly effective for our mushroom classification task.

While these results are impressive, it is important to note that such high accuracy might warrant further investigation. It could indicate either an exceptionally well-suited dataset for these algorithms or potential data leakage issues.

In conclusion, both Decision Trees and Random Forests have demonstrated their effectiveness in mushroom classification. The Random Forest model, in particular, shows promise as a highly accurate tool for this task. However, as with any machine learning application in critical domains like food safety, these models should be used in conjunction with expert knowledge and additional verification methods to ensure the highest level of accuracy and safety in real-world applications.