UNIVERSITÀ
DEGLI STUDI
DI MILANO

Machine learning Module
Final Project

Tree Predictors for Binary Classification

Amirali Davary
May 2025

Contents

Figures and Tables

Introduction

Mushroom classification is a classic problem in supervised machine learning, particularly relevant due to its real-world implications in foraging and food safety. The goal is to accurately distinguish between edible and poisonous mushroom species based on a set of observable characteristics. In this project, we implement and evaluate custom-built Decision Tree and a Random Forest machine learning models from scratch, without relying on pre-built classifiers from readily available libraries such as scikit-learn. Using the secondary version of the mushroom dataset, which contains categorical attributes for various physical features, we preprocess the data, encode it appropriately, and conduct model training, hyperparameter tuning, and performance evaluation. This approach not only demonstrates practical machine learning skills but also offers deeper insight into the underlying mechanics of decision-based classification algorithms.

Data

The dataset used in this project is the secondary mushroom dataset, which provides detailed categorical information about various mushroom samples. Each row represents an individual mushroom, and the dataset includes a wide array of features that describe physical characteristics such as cap shape, surface, and colour, as well as other traits like odor, gill size, spore print colour, and habitat. The primary target variable is the "class" column, which indicates whether a mushroom is edible (e) or poisonous (p).

The dataset consists of multiple categorical columns with discrete values, each encoded as short text labels (e.g., "x" for convex cap shape, "s" for smooth surface). These features are nominal in nature, and none of them are numerical or ordinal by default. The dataset may include some missing values, and its structure allows for rich classification logic based on attribute combinations. Overall, the dataset provides a suitable and interpretable foundation for exploring decision-based classification models.

Methodology

Data Preprocessing

The dataset was first loaded from a CSV file with semicolon delimiters. Initial inspection included checking for missing values and duplicate records. As highlighted in Table 1, the total number of entries in this dataset is 61069. Except for three columns with continues values (indicated as float64 in the table), all the other properties are categorical features. Several columns in the table contain a high proportion of missing values. As shown in Figure 1, the columns 'stem-root', 'stem-surface', 'veil-type', 'veil-color', and 'spore-print-color' each have over 50% null entries. To ensure the dataset remains reliable for model training, these columns will be removed. Columns containing more than 50% missing values were removed to maintain data quality and reduce noise. Duplicate rows were also dropped to prevent model bias.

Table 1 - Dataset info table

```
RangeIndex: 61069 entries, 0 to 61068
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   class                  61069 non-null  object
 1   cap-diameter           61069 non-null  float64
 2   cap-shape              61069 non-null  object
 3   cap-surface            46949 non-null  object
 4   cap-color              61069 non-null  object
 5   does-bruise-or-bleed   61069 non-null  object
 6   gill-attachment        51185 non-null  object
 7   gill-spacing           36006 non-null  object
 8   gill-color             61069 non-null  object
 9   stem-height            61069 non-null  float64
 10  stem-width             61069 non-null  float64
 11  stem-root              9531 non-null   object
 12  stem-surface           22945 non-null  object
 13  stem-color             61069 non-null  object
 14  veil-type              3177 non-null   object
 15  veil-color             7413 non-null   object
 16  has-ring               61069 non-null  object
 17  ring-type              58598 non-null  object
 18  spore-print-color      6354 non-null   object
 19  habitat                61069 non-null  object
 20  season                 61069 non-null  object
```

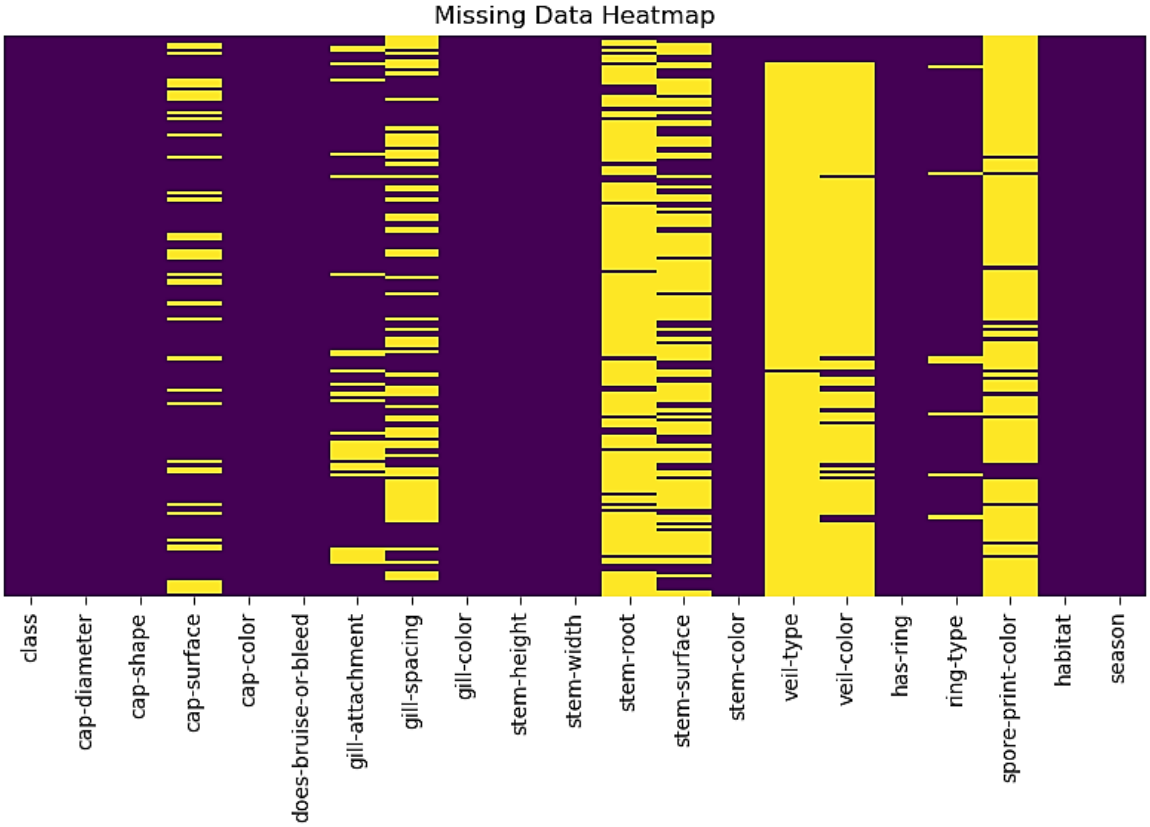

Figure 1 - Heatmap for missing data

The dataset contains only three numerical features. Gaining a clearer understanding of their distributions can aid in more effective data preparation. As illustrated in Figure 2, these features exhibit well-defined distributions and demonstrate a strong correlation with one another.

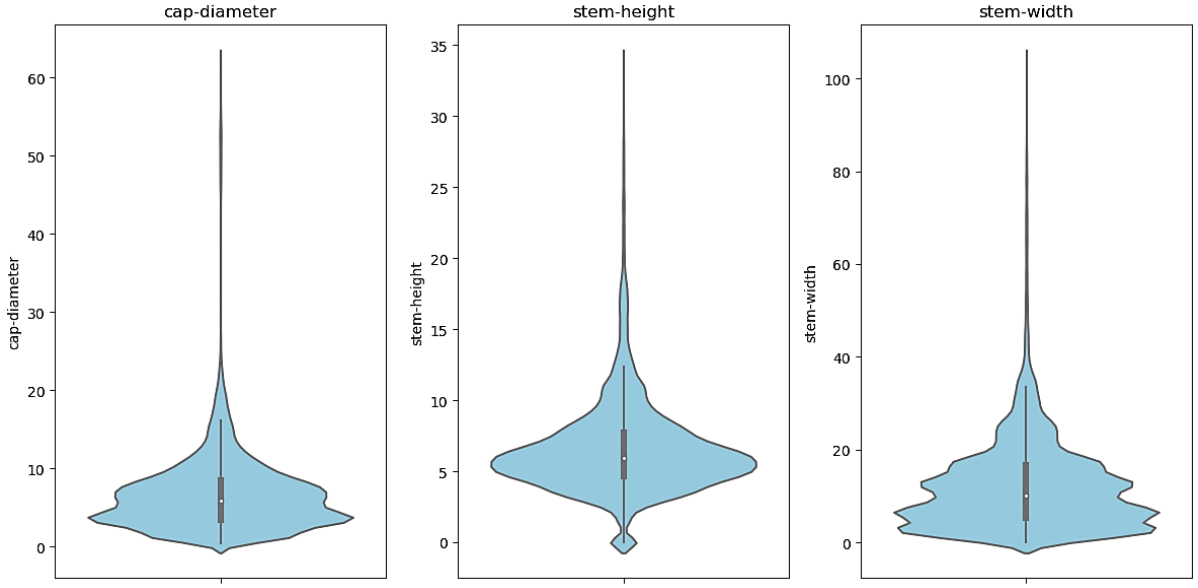

Figure 2 - Violine plot for numerical features

In addition to the target feature, there are two other features with binary values. Figure 3 displays the distribution of all three binary features. The target feature shows a fair distribution, indicating a balanced dataset suitable for this classification task. All categorical features were transformed into numerical representations using Label Encoding from scikit-learn, which assigns each unique category in a feature column a corresponding integer. This encoding approach is particularly well-suited for decision tree algorithms, which can handle non-ordinal numerical labels effectively. The target variable "class" was encoded into binary format: edible mushrooms as 0 and poisonous mushrooms as 1.

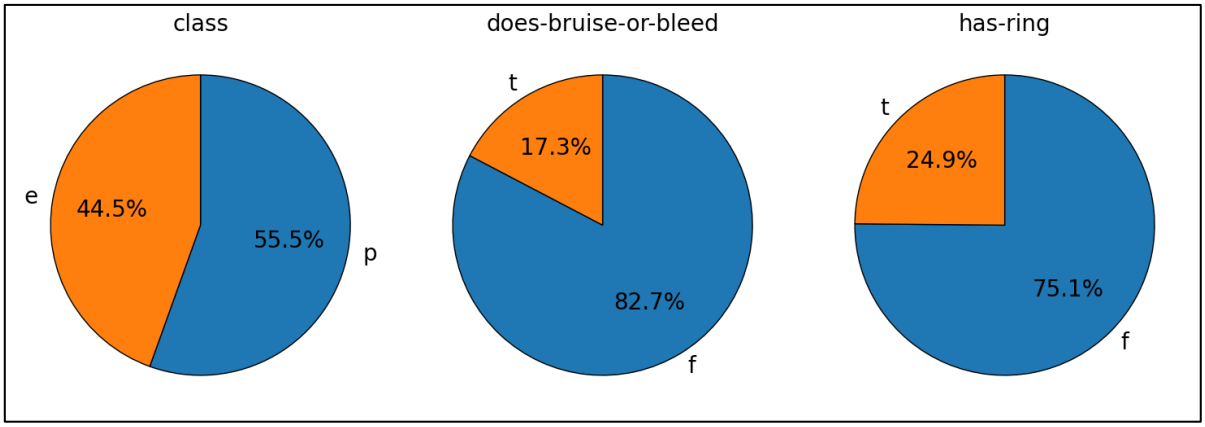

Figure 3 - Binary features distribution

To ensure a fair comparison of results across different modelling approaches, the dataset was divided into training and test sets using the `train_test_split` function from scikit-learn. This function randomly partitions the data, which helps to avoid bias and ensures that both sets are representative of the overall dataset. By default, it also allows for stratified sampling when dealing with classification tasks, preserving the proportion of classes between the training and test sets. This helps in achieving reliable model evaluation by testing the model on data it hasn't seen during training.

Decision Tree

A custom Decision Tree classifier was implemented from scratch. The tree structure consists of TreeNode objects, each representing a decision based on a feature and a threshold value. The algorithm uses a recursive strategy to grow the tree by selecting the best feature and threshold at each node, which minimizes impurity in the resulting child nodes. Three impurity criteria are supported: `Gini index`, `entropy`, and `misclassification error`, allowing flexibility in how splits are evaluated. The tree grows until either a maximum depth is reached, or further splitting does not significantly improve classification, based on a minimum number of samples required for a split.

During prediction, the model traverses the tree from root to leaf based on the feature values of the input sample. At each node, it evaluates whether the sample's value satisfies the decision threshold and proceeds down the corresponding branch. The prediction returned is the majority class at the leaf node reached. This implementation enables detailed control over the tree-building process, which is valuable for understanding and visualizing decision-making in classification tasks.

Random Forest

A Random Forest classifier was also implemented manually, consisting of an ensemble of decision trees built using the custom tree class. Each tree is trained on a bootstrap sample of the dataset, promoting diversity among the learners. To further increase variation, a random subset of features is selected for each tree, controlled by the max_features parameter (options include `'sqrt'`, `'log2'`, or a fixed integer). The forest's final prediction is obtained through majority voting across the predictions of all individual trees, enhancing stability and generalization over a single decision tree.

This ensemble approach reduces variance and improves robustness, particularly when dealing with noisy or complex datasets. By combining predictions from multiple diverse models, the random forest can often outperform a single decision tree in both accuracy and resistance to overfitting. The implementation includes functionality for computing feature importance based on the frequency of splits involving each feature, giving insight into which variables most influence the model's decisions.

K-Fold Cross Validation

To ensure reliable evaluation of model performance, k-fold cross-validation was used. This technique divides the dataset into k equally sized folds and iteratively trains the model on k-1 folds while testing it on the remaining fold. This process is repeated k times so that each fold is used once as the test set. Performance metrics such as accuracy, precision, recall, F1 score, specificity, and AUC are computed for each fold and averaged to give an overall assessment.

This method helps mitigate the risk of biased evaluations that can result from a single train-test split. In this project, k=5 was chosen to strike a balance between computation time and robustness. Additionally, ROC curves were plotted to visualize the trade-off between true positive rate and false positive rate across different thresholds. The consistent use of cross-validation ensures that results are statistically reliable and not dependent on a particular split of the data.

Hyperparameter Tuning

To enhance model performance, a grid search strategy was employed from scratch same as the two models, to systematically explore combinations of hyperparameters for both the custom decision tree and random forest classifiers. For the decision tree, the tuning process evaluated variations in `max_depth` (5, 10, 20), `min_samples_split` (2, 5, 10), and `criterion` (gini, entropy, misclassification).

These parameters control the tree's depth, the minimum number of samples required to further split a node, and the measure of impurity used during node evaluation, respectively.

For the random forest, additional layers of complexity were introduced in the search space. The grid included different values for `n_estimators` (30, 50), `max_depth` (5, 10), `min_samples_split` (2, 5), `max_features` (sqrt, log2), and `criterion` (gini, entropy). This configuration controls not only the size and depth of each tree in the forest, but also how features are sampled and evaluated during training, which directly affects the diversity and robustness of the ensemble.

Each hyperparameter combination was assessed using 5-fold cross-validation, and the performance was evaluated based on a specified scoring metric, typically accuracy. The configuration that yielded the highest average performance across all folds was selected as the optimal setting. This automated tuning process eliminated the need for manual parameter guessing, ensured fair model comparisons, and improved the likelihood that the models would generalize well on unseen data.

Results

As outlined in the previous section, four different modelling and combination methods were explored: starting with a single decision tree, followed by a random forest, and then performing an extensive grid search to optimize each of these two models. To compare the performance of these models, it is important to first define the evaluation metrics. Table 2 provides a summary of these metrics.

Table 2 - performance metrics definitions

Metric	Definition
Accuracy	The proportion of total correct predictions (both true positives and true negatives) out of all predictions made.
Precision	The proportion of true positive predictions out of all positive predictions made.
Recall	Sensitivity or True Positive Rate: The proportion of actual positives correctly identified.
F1 Score	The harmonic mean of precision and recall, balancing both metrics.
Specificity	True Negative Rate: The proportion of actual negatives correctly identified.
AUC	Area Under the Curve: Represents the area under the Receiver Operating Characteristic (ROC) curve, measuring the model's ability to distinguish between classes across all thresholds. A higher AUC indicates better performance.

Single Decision Tree

To train an initial decision tree classifier, the hyperparameters were set as follows: `max_depth` was limited to 10, `min_samples_split` was set to 5, and the `gini criterion` was used for measuring the quality of splits. The performance metrics, illustrated in Figure 5, show a relatively low error rate, with a training error of 0.0992 and a testing error of 0.1078. This corresponds to training and testing accuracies of 90.08% and 89.22%, respectively. These results suggest that the model generalizes well and maintains consistent performance on unseen data.

Table 3 - Single decision tree performance metrics

	Accuracy	Error (0-1 loss)
Training	0.0739	0.0992
Test	0.0810	0.1087

Random Forest

The Random Forest classifier was trained using 50 decision trees (`n_estimators`=50), with each tree having a maximum depth of 10 (`max_depth`=10). Additionally, the number of features considered for each split was set to the square root of the total number of features (`max_features`='sqrt'), which is a common practice to reduce correlation among trees and enhance diversity within the forest. As shown in Table 4, the model achieved strong performance, with a training error of 0.0739 and a testing error of 0.0810. This corresponds to training and testing accuracies of 92.61% and 91.90%, respectively. These results indicate that the Random Forest model not only fits the training data well but also generalizes effectively to unseen data, improving upon the single decision tree classifier.

Table 4 - Random Forest performance metrics

	Accuracy	Error (0-1 loss)
Training	0.9008	0.0739
Test	0.8922	0.0810

Grid Search Results

As outlined in the methodology section, two independent grid search processes were conducted—one for a decision tree classifier and the other for a random forest classifier. Both models were evaluated using 5-fold cross-validation, with performance measured across six key metrics: accuracy, precision, recall, F1 score, specificity, and AUC. This robust evaluation framework ensured reliable performance estimates by averaging results across multiple folds and highlighting model generalization capabilities.

Decision Tree Grid Search

A comprehensive grid search was performed for the decision tree classifier, exploring variations in max_depth, min_samples_split, and criterion. To ensure reliability, 5-fold cross-validation was employed, and model performance was evaluated using key metrics such as accuracy, precision, recall, F1 score, specificity, and AUC. The search revealed that shallower trees (e.g., max_depth=5) produced moderate performance. Models using a custom misclassification-based criterion achieved average accuracies around 75.6%, F1 scores near 0.79, and AUC values around 0.75. In contrast, trees using traditional criteria like Gini and Entropy at the same depth performed slightly worse, with F1 scores around 0.72 and AUCs closer to 0.66, indicating challenges in balancing sensitivity and specificity with limited tree depth.

As the maximum depth was increased to 10, a notable improvement in classification performance was observed. Models with max_depth=10 and the Gini criterion achieved accuracies around 91%, F1 scores of approximately 0.916, and AUC scores averaging 0.9126. This depth allowed the tree to better capture underlying patterns in the data, resulting in a substantial boost across all evaluation metrics. The best-performing decision tree model was obtained with parameters max_depth=20, min_samples_split=2, and criterion='gini'. This configuration achieved:

Table 5 - Best performing decision tree metrics

Metric	Result
Accuracy	0.9942
Precision	0.9953
Recall	0.9943
F1 Score	0.9948
AUC	0.9940

The ROC curve and confusion matrix of the best performing decision tree are shown in Figure 4.

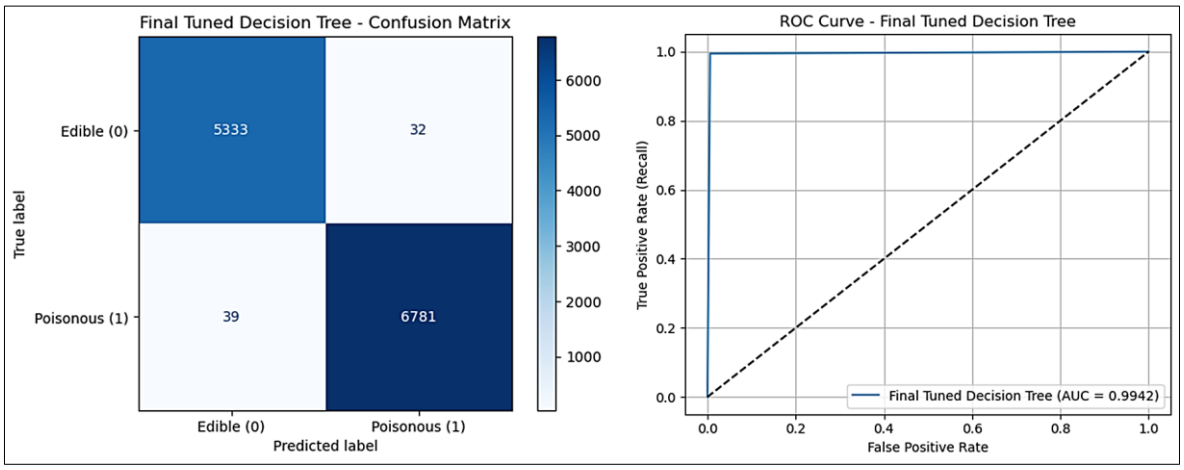

Figure 4 - Best performing decision tree after grid search

The feature importance analysis of the best-performing decision tree model highlighted cap-diameter as the most critical variable, contributing approximately 16.5% to the model's decisions. Other highly influential features included cap-surface (11.6%), stem-width (10.9%), gill-attachment (10.1%), and stem-height (10.1%), underscoring the significant role of both cap and stem characteristics in classification. Additionally, cap-color (9.3%), gill-spacing (5.7%), and gill-color (5.7%) were moderately important. Features such as cap-shape and stem-color showed moderate but notable contributions, while ring-type and habitat had smaller effects. In contrast, variables like does-bruise-or-bleed, has-ring, and season demonstrated relatively minimal influence on the tree's decision-making process.

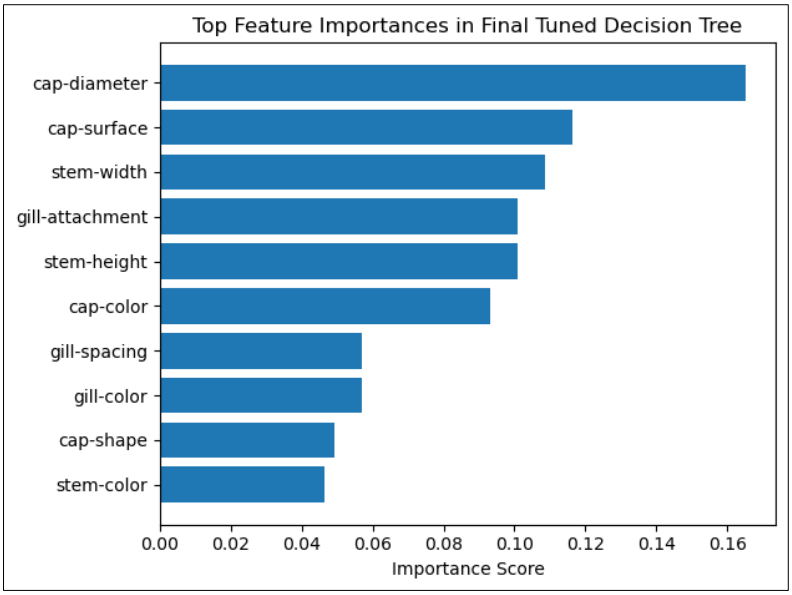

Figure 5 - Best performing decision tree feature importance

Random Forest Grid Search

For the random forest classifier, grid search included variations of `n_estimators`, `max_depth`, `min_samples_split`, `max_features`, and criterion. Even at lower depths (e.g., `max_depth`=5), random forests provided consistent and competitive performance, with F1 scores ranging from 0.75 to 0.83 and AUC scores between 0.74 and 0.78. These outcomes reflect the ensemble method's ability to generalize better than single trees, even when individual trees are shallow. As depth increased to 10, random forest models significantly improved. The most effective model used: `n_estimators`=50, `max_depth`=10, `min_samples_split`=2, `max_features`='log2', and `criterion`='gini'. This configuration achieved the results show in Table 6.

Table 6 - Best performing random forest metrics

Metric	Result
Accuracy	0.9236
Precision	0.8998
Recall	0.9711
F1 Score	0.9339
AUC	0.9177

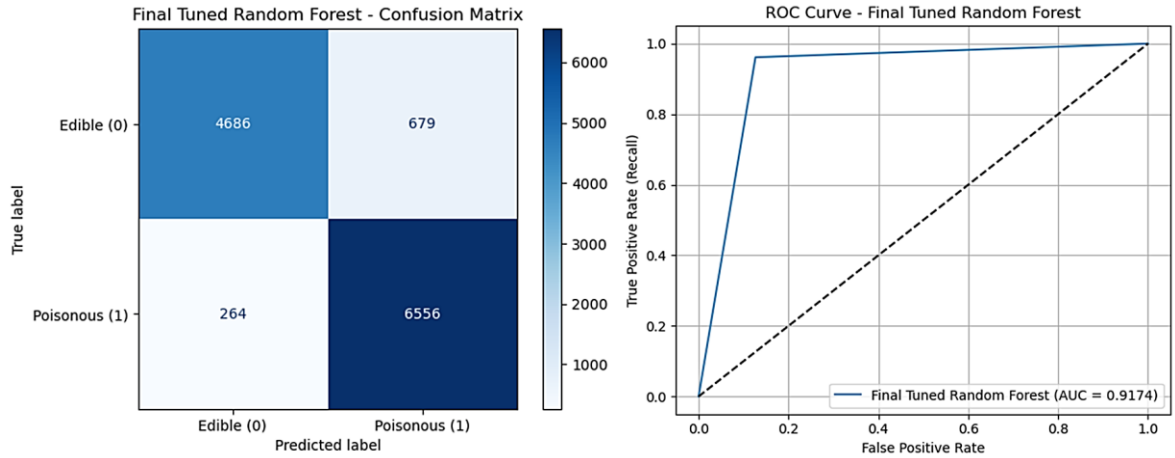

Figure 6 - Best performing random forest after grid search

For the best-performing random forest model, stem-height stood out as the most influential feature, with an importance score of 23.6%. It was followed closely by cap-diameter (19.4%) and stem-width (19.1%), emphasizing the relevance of size-related features. Additional contributing variables included stem-color, cap-color, and cap-shape, reflecting the model's broader use of morphological cues. Unlike the decision tree, the random forest model leveraged ensemble averaging to distribute importance across a wider set of features, capturing more complex patterns and interactions, especially among stem and cap attributes.

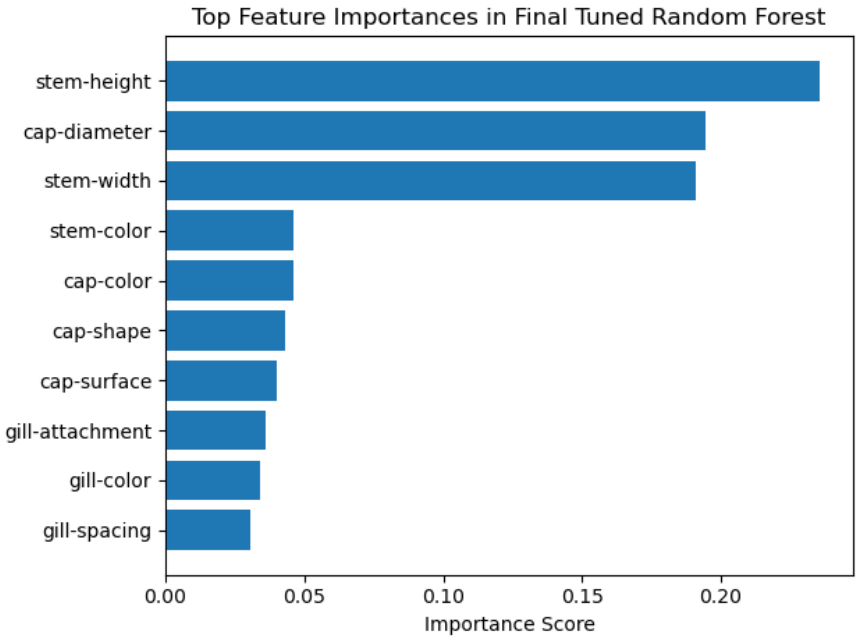

Figure 7 - Best performing random forest feature importance

Declaration

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.