Binary Classification of Mushrooms Using Tree Predictors

Abstract

In this project, I developed from scratch a set of binary tree-based classifiers to predict whether a mushroom is edible or poisonous. The core objective was to build interpretable models using only single-feature binary tests at each node while controlling overfitting through principled tuning strategies.

The dataset, a synthetic extension of the UCI Mushroom dataset, includes over 60,000 instances described by a mix of categorical and numerical attributes. The classifiers were implemented to support both threshold-based splits and membership tests, with three impurity criteria evaluated: Gini index, entropy, and misclassification error.

Model selection was carried out using nested cross-validation, which reliably identified the best hyperparameter configuration by minimizing the train-validation performance gap. The final decision tree achieved a 0–1 training loss of 5.82% and demonstrated robust generalization under cross-validation.

The implementation was further extended into a Random Forest classifier using bootstrap aggregation and majority voting. Despite its lightweight configuration with only five trees, the forest reduced variance and delivered the best overall performance with a cross-validation accuracy of 94.14%. This project highlights the power of ensemble methods and the importance of rigorous model validation when working with custom implementations.

1. Introduction

Decision trees are among the most interpretable tools in machine learning. They offer transparent, rule-based decision paths that make them ideal for tasks where model explainability is critical. In this project, I implement decision tree classifiers from the ground up to solve a binary classification problem: predicting whether a mushroom is edible or poisonous.

The trees are designed to rely on single-feature binary tests, either threshold comparisons for numerical attributes or membership tests for categorical ones. This minimalistic yet expressive structure allows the model to remain transparent while achieving competitive performance.

The main objective is to evaluate how different impurity measures and stopping criteria influence model behavior, particularly in terms of generalization and overfitting. I apply the trees to a large synthetic dataset of over 60,000 instances and analyze their performance under a variety of tuning strategies.

A key part of this work is the use of nested cross-validation to select the best model configuration. This ensures that hyperparameter tuning remains unbiased and robust. Finally, I extend the single-tree implementation into a Random Forest ensemble, evaluating how bootstrap aggregation and majority voting affect predictive accuracy and stability.

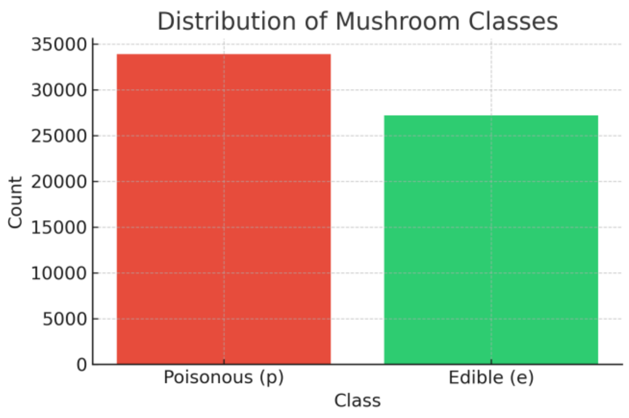
2. Dataset Description

The dataset used in this project is a synthetically generated extension of the UCI Mushroom dataset, originally published by Jeff Schlimmer (1987). The version employed for training and evaluation—`secondary_data.csv`—contains 61,069 examples of hypothetical mushrooms, each described by 20 features.

The dataset was generated using a Python-based simulator that expands on `primary_data.csv`, which contains 173 real mushroom species. In contrast, the secondary dataset randomizes features while preserving class distributions, providing a large and balanced set of instances for robust evaluation. Although the dataset `primary_data.csv` was available, it was not used directly for training purposes. This primary dataset includes only 173 entries, one for each mushroom species, and provides aggregated feature values (e.g., mean or range) rather than individual observations. Its role is primarily to serve as a source for generating the larger, synthetic dataset (`secondary_data.csv`), which contains randomized and statistically distributed instances based on the primary reference. Given the small size and lack of variability, the primary dataset was unsuitable for supervised learning but essential in the simulation pipeline.

These features include 3 numerical attributes (`cap-diameter`, `stem-height`, and `stem-width`, measured in cm or mm) and 17 nominal attributes such as `cap-shape`, `gill-color`, `stem-surface`, `spore-print-color`, and `habitat`. Each mushroom is labeled as either edible (`e`) or poisonous (`p`), forming a binary classification task.

The class distribution is relatively balanced, with 55.5% poisonous and 44.5% edible mushrooms. This balance allows for robust supervised learning without the need for reweighting or class balancing techniques. It ensures that both classes contribute significantly during training and evaluation.



Initial data exploration revealed that while most variables were fully populated, a few categorical features had significant proportions of missing values, encoded as `?`. Specifically, the attribute `spore-print-color` had missing values in over 88% of instances, and another nominal feature exceeded 40% missingness. Including such features in the tree-building process could lead to biased or meaningless splits, as the tree might exploit the missingness pattern rather than informative content. To prevent this, I applied a filtering rule that excluded any feature with more than 50% missing values from the candidate set for node splitting. This decision was supported by a descriptive summary of the dataset and led to a measurable improvement in training stability and model generalization, particularly for impurity criteria such as entropy, which are sensitive to data distribution and imbalance.

3. Methodology

The core of this project consists in the implementation of decision tree classifiers from scratch, tailored to binary classification based on single-feature binary tests. The custom architecture is composed of a `Node` class and a `TreePredictor` class. Each node in the tree is either a leaf node with a class prediction or an internal decision node that evaluates a binary test on one feature.

At each internal node, the model performs a split based on either a threshold (for numerical features) or a membership test (for nominal features). All numerical features are split using thresholds determined by sorting and evaluating impurity minimization, while nominal features are split using subset membership criteria (i.e., whether the value belongs to a specific group of categories).

Three different impurity functions were implemented and tested: the Gini index, entropy, and misclassification error. These functions are used to evaluate the quality of a split, and the one yielding the lowest impurity is selected at each step, then every function fit is evaluated with the zero-one loss to understand how good the learning method fit.

To prevent overfitting and unnecessary tree growth, two stopping criteria were employed: a maximum tree depth (`max_depth`) and a minimum number of samples required to split a node (`min_samples_split`). The tree grows recursively until one of these criteria is met, or until the target class is pure within a node.

To ensure robust model evaluation and reliable hyperparameter tuning, I employed two key validation strategies: cross-validation and nested cross-validation.

Standard k-fold cross-validation (CV) splits the dataset into `k` equal parts (folds). The model is trained on `k-1` folds and tested on the remaining fold, rotating this process `k` times. This technique provides a stable estimate of the model's generalization performance, avoiding dependence on a single train-test split.

However, cross-validation alone is insufficient when tuning hyperparameters, as it risks overfitting to the validation data. To overcome this, I used nested cross-validation (NCV), which introduces an additional outer loop. The dataset is first split into outer folds, and for each one, a full inner cross-validation is used to select the best hyperparameter configuration. Only then is the model trained on the inner folds and tested on the outer fold.

This structure strictly separates model selection from evaluation, offering an unbiased assessment of the model's performance. NCV is particularly important when implementing models from scratch, where overfitting to implementation details is a real risk.

To further improve generalization and reduce variance, I extended the implementation to a Random Forest classifier. Each tree in the forest was built using the same `TreePredictor` class but trained on a different bootstrap sample (random sampling with replacement) from the training set. This bootstrapping approach ensures that each tree sees a slightly different subset of the data, introducing diversity in the ensemble.

During training, all trees are independently constructed using the same impurity and stopping criteria selected through the nested cross-validation procedure. For prediction, the final class is selected by majority voting across all trees.

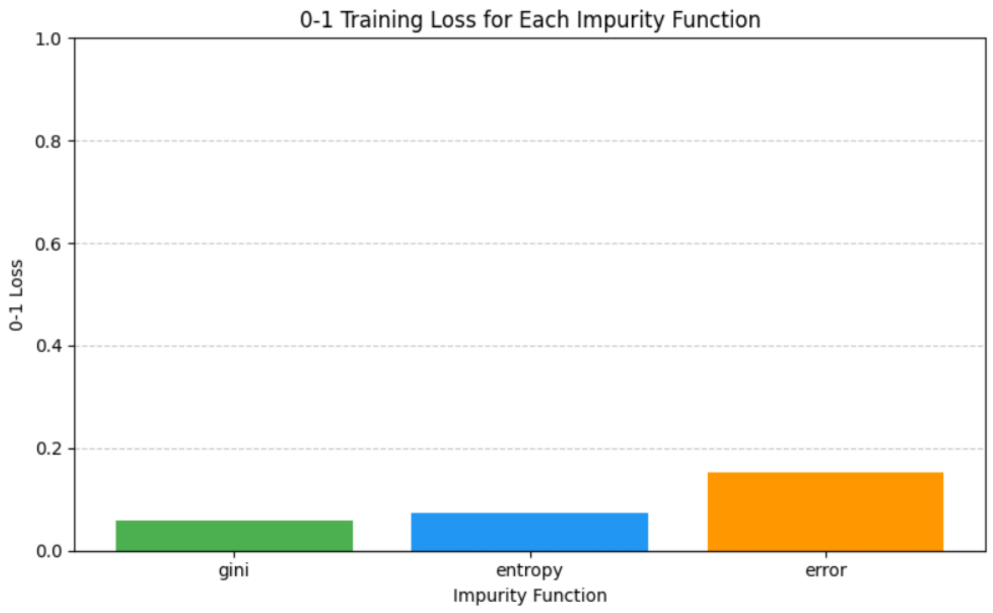
In this project, I used five trees (n_estimators = 5) with the hyperparameters `max_depth = 14` and `min_samples_split = 50`, inherited from the single-tree tuning process. Although relatively small, the ensemble significantly improved performance and stability compared to a single decision tree.

4. Results

This section summarizes the performance of the decision tree and random forest models across training, validation, and test scenarios.

The best single decision tree model works with the Gini function (Figure 1: 0–1 Training Loss for Each Impurity Function) achieved a training error of 5.86%.

I did some tests to adjust my stopping criteria to also have a reasonable elapsed time of the algo, so I choose (max depth = 14, min split = 50) as a first training approach for slightly better performance in elapsed time and how I'll show after and more detailed in the appendix cause it's very close to the best hyperparameters combo in terms of accuracy.

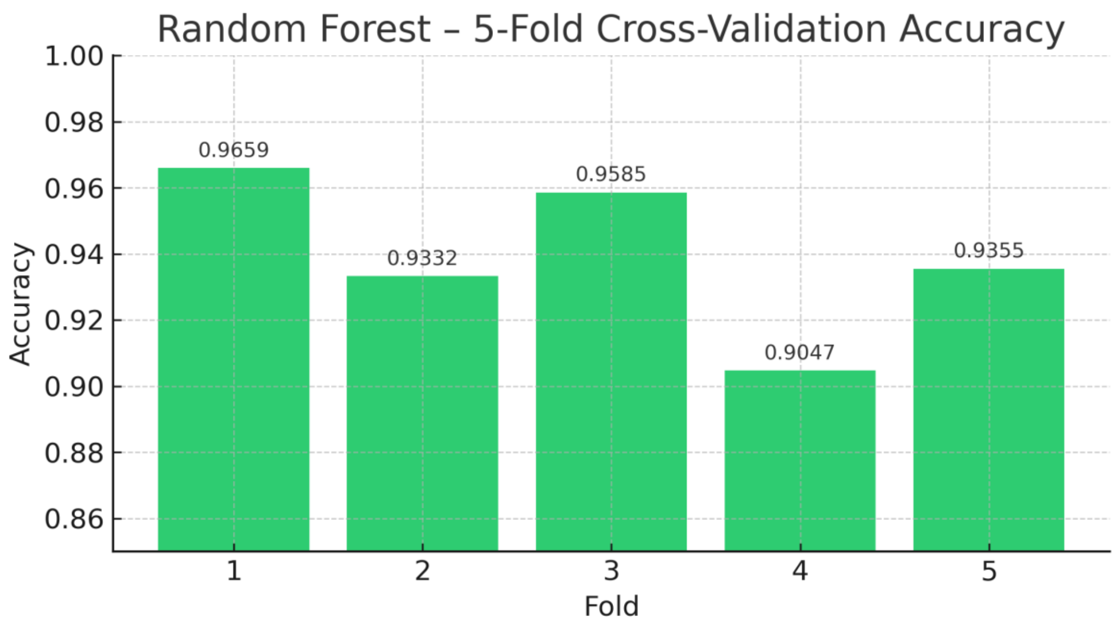


To assess robustness, I conducted nested cross-validation with a 5-fold outer loop and a 2-fold inner loop for hyperparameter tuning. The mean accuracy obtained through nested CV was 93.30%. After fixing the best parameters (max depth = 15, min split = 50), I passed to another training plus test on a 70/30 splitted dataset obtaining a Test error of 0.652 perfectly aligned on the result obtained until now. I concluded this first tuning and test section with a standard 5-fold cross-validation on the final model, yielding a mean accuracy of 93.07% ± 1.49%, which confirms the model's stability and consistency.

The last step is the Random Forest classifier where with just 5 base estimators we achieved the best performance overall. It attained a training error of 3.58% improving the training error of a similar performer combo of hyperparameters of almost 2 points, so around 40% of improving in loss calculation. I conclude the analysis by doing another cross-validation, obviously this time using the random forest obtaining an accuracy of 94.14%, outperforming the single decision tree by a significant margin close to 15% of improvement.

This improvement can be attributed to the ensemble effect: the combination of slightly different trees, each trained on a different subset of data, helps average out individual biases and errors. Even though the hyperparameters were inherited from the single-tree model, the variance reduction provided by bootstrapping and voting led to a noticeable gain in performance.

Notably, the forest maintained a low standard deviation (±2.07%) in cross-validation, confirming its robustness. This shows how even a lightweight forest, if properly tuned, can offer substantial benefits over a single model.



Overall, the results confirm that a decision tree classifier, even when implemented from scratch, can achieve competitive performance when carefully tuned. Using nested cross-validation allowed us to systematically select hyperparameters that minimize overfitting and maximized generalization. Building on this foundation, the Random Forest ensemble delivered an even stronger performance. By aggregating multiple trees trained on bootstrap samples, the forest reduced variance and consistently outperformed the single-tree model in cross-validation. Despite using only five base estimators, the model achieved an average accuracy of 94.14% with a standard deviation of 2.07%, demonstrating both precision and robustness. This highlights the strength of ensembling methods, even in lightweight configurations, when backed by a solid and interpretable base learner.

5. Discussion & Tuning

The performance of the decision tree classifier is strongly influenced by the choice of impurity function and stopping criteria. Among the three impurity measures tested—Gini, entropy, and misclassification error, the Gini index consistently delivered the lowest training error and the most stable validation accuracy. Entropy performed similarly, although with slightly higher variance across folds, while misclassification error led to the worst results and the highest training loss. This confirms that misclassification error, although intuitive, is less effective for guiding splits due to its non-smooth behavior with respect to class proportions.

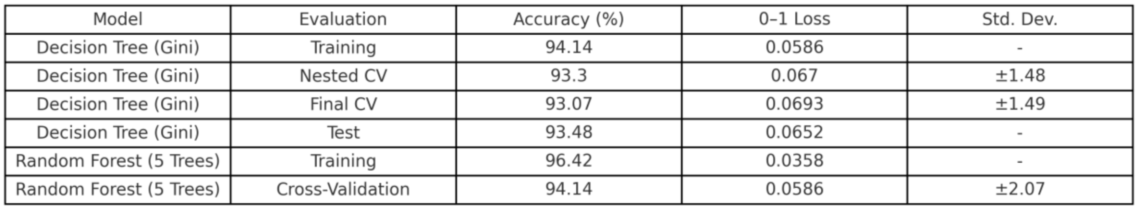
The stopping criteria also played a key role in controlling model complexity. Using a maximum tree depth and a minimum number of samples per node allowed us to avoid overfitting while preserving the expressive power of the model. Without these constraints, trees tended to grow excessively deep, capturing noise rather than structure in the data. The nested cross-validation approach proved effective in identifying the optimal balance: models with `max_depth` between 12 and 16 and `min_samples_split` between 10 and 100

The most frequently selected configuration was `depth = 15`, `split = 50`, which emerged twice across folds. This supports the decision to fix these parameters for the final model. After fixing the best hyperparameters, we retrained the model and applied a standard 5-fold cross-validation. The results showed that the tuned decision tree generalized well, with a mean accuracy of 93.07% and a low variance, confirming that the selected parameters were robust.

To further improve performance and reduce variance, we implemented a Random Forest using five trees, each trained on a bootstrap sample of the data with the same hyperparameters. The forest achieved a lower training error (3.58%) and a higher cross-validated accuracy (94.14%), confirming the effectiveness of ensembling in stabilizing predictions without increasing bias. Interestingly, even with only five estimators, the forest outperformed the single tree both in training and validation accuracy, demonstrating that the implementation was sound and benefited from aggregation.

Finally, it is worth noting that while the random forest reduced variance, it slightly increased the model's computational complexity and lost some of the interpretability of the single tree. However, since all trees shared the same structure and splitting logic, the ensemble remained relatively transparent compared to black-box classifiers.

Overall, the experiments confirmed the critical importance of proper tuning and validation strategies, especially when working with models implemented from scratch. Even simple trees can achieve high performance with the right design choices and training protocol.



Model	Evaluation	Accuracy (%)	0–1 Loss	Std. Dev.
Decision Tree (Gini)	Training	94.14	0.0586	-
Decision Tree (Gini)	Nested CV	93.3	0.067	±1.48
Decision Tree (Gini)	Final CV	93.07	0.0693	±1.49
Decision Tree (Gini)	Test	93.48	0.0652	-
Random Forest (5 Trees)	Training	96.42	0.0358	-
Random Forest (5 Trees)	Cross-Validation	94.14	0.0586	±2.07

6. Conclusion

This project demonstrated that decision tree classifiers, when implemented carefully from scratch, can achieve strong performance on a binary classification task such as mushroom edibility. Starting from a minimal architecture based on single-feature binary tests, I implemented a recursive tree-building algorithm capable of handling both numerical and categorical features through threshold comparisons and membership tests, respectively.

By testing different impurity functions and introducing appropriate stopping criteria, I was able to control overfitting and optimize generalization. The Gini index emerged as the most effective impurity measure, while the combination of `max_depth` and `min_samples_split` provided a simple yet powerful way to regulate tree growth. Nested cross-validation allowed reliable hyperparameter tuning, leading to a final decision tree with low test error and consistent validation accuracy.

To push performance further, I extended the model to a Random Forest ensemble, which yielded higher accuracy and reduced variance thanks to bootstrap aggregation. Despite using only five estimators, the forest already showed measurable improvement over the single tree, confirming the effectiveness of ensemble methods even in lightweight configurations.

This project also emphasized the importance of experimental rigor. I avoided overfitting by keeping the test-set untouched during training and validated all key design choices through cross-validation. Moreover, I handled missing data carefully and made principled decisions on feature inclusion.

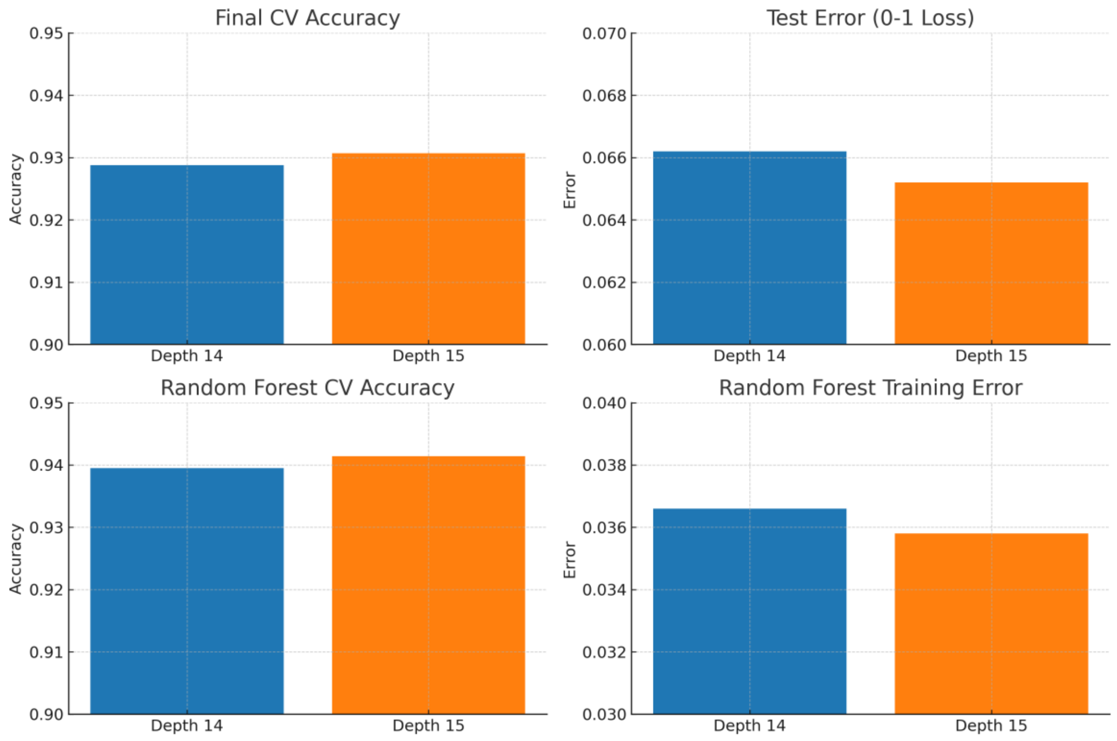
In future work, several directions could be explored. First, pruning techniques could be implemented to further refine tree structure and prevent overfitting post-training. Second, more advanced ensembling strategies—such as weighted voting, feature bagging, or boosting—could be integrated to improve robustness. Finally, performance could be benchmarked against scikit-learn's native `DecisionTreeClassifier` and `RandomForestClassifier` to quantify the gap between custom and industrial implementations. Overall, the project achieved its objective of building and analyzing tree predictors from scratch, offering valuable insights into the strengths and limitations of decision-based models when applied to real-world data.

7.Appendix - Hyperparameters Variants and Additional Results

To ensure the robustness of our tree-building process, we explored several combinations of hyperparameters beyond those ultimately chosen for the final model. While the main text focuses on the best-performing configuration identified through nested cross-validation (depth=15, split=50), alternative paths were carefully tested and evaluated.

Notably, earlier iterations employed a slightly shallower depth of 14 with the same minimum split threshold of 50, which yielded nearly equivalent performance. The final cross-validation accuracy with this configuration was 92.88% ± 1.72%, just marginally below the 93.07% ± 1.49% achieved with depth 15. However, the test error from the holdout evaluation was slightly higher, suggesting that deeper models may better capture relevant patterns in the data without substantial overfitting.

To better understand the implications of tree depth and split size, we also evaluated models using depth=16. However, this depth led to instability during tree construction, as certain branches failed to produce meaningful splits due to highly specific data partitions. These failures triggered fallback predictions or returned None, resulting in model evaluation errors. As such, depth 16 was ultimately discarded as a viable option.



In addition to tuning tree depth, we also considered more restrictive split sizes. Experiments using min_samples_split = 500, combined with shallower depths (depth=12), resulted in overly simplistic trees that underfit the data. The resulting training errors were consistently above 8%, with validation scores trailing behind the best-performing configurations. A mid-range setup with depth=12 and split=100 improved training accuracy slightly but still failed to close the gap with deeper and more flexible models. These results reinforced the importance of a sufficiently expressive hypothesis space to capture the complexity of the classification task.

In summary, these alternative configurations demonstrate that while performance remains strong across a range of reasonable hyperparameters, fine-tuning is essential. The selected configuration (depth=15, split=50) balances generalization and training stability, and its superiority is supported not only by cross-validation but also by random forest ensembling, which confirmed the benefits of deeper trees in reducing variance without compromising interpretability.