University of Milan

Data Science for Economics

Tree Predictors for Binary Classification

Project Report

Author: Bekzat Gazizuly
Student ID: 46031A
Course: Machine Learning
Instructor: Prof. Nicolo Cesa-Bianchi
Date: May 28th, 2025

Abstract

This report describes the complete implementation from scratch of binary decision tree predictors to classify mushrooms as poisonous or edible. The dataset used includes various categorical attributes of mushrooms, and the objective is to build a reliable model using interpretable decision trees. Several splitting and stopping criteria are implemented and compared. Additionally, a Random Forest extension is introduced.

Contents

1 Introduction

Decision trees are powerful models used in supervised machine learning, particularly for classification problems. Their structure mirrors a flowchart-like hierarchy, allowing interpretability while maintaining predictive performance. This project involves building a decision tree from scratch to classify mushrooms as poisonous or edible using the well-known mushroom dataset.

We implement the following components:

- A custom tree node structure
- A binary tree predictor class with custom splitting and stopping criteria
- Gini, Entropy, and Scaled Entropy as splitting criteria
- Depth and minimum samples per node as stopping criteria
- Evaluation metrics and performance graphs
- Random Forest ensemble method as an additional feature

Decision trees remain a fundamental tool in interpretable machine learning. This project aims not only to implement them from scratch but also to deeply explore how different design choices—like the choice of splitting criterion and stopping rule—affect their performance. Furthermore, we extend the project with a Random Forest approach to illustrate how ensemble methods improve upon single-tree models, both in theory and in practice.

2 Dataset

The dataset used is derived from the UCI Mushroom Dataset, consisting of over 61,000 instances in our case. Each row represents a mushroom, described by categorical features such as cap shape, odor, spore print color, etc. The target label indicates whether the mushroom is edible or poisonous.

3 Data Preparation and Methodology

3.1 Preprocessing

- The dataset is loaded using pandas.
- The target label is separated and removed from the features.
- All categorical features are transformed using one-hot encoding.

One-hot encoding was necessary because decision tree logic relies on binary splits, and categorical variables must be converted into binary indicators for each category.

3.2 Methodology

- We loaded and cleaned the data using pandas.
- We encoded categorical features using one-hot encoding.
- The dataset was split into training and testing subsets using an 80/20 split.

- We implemented decision tree construction from scratch using recursive binary splits.
- We defined multiple splitting and stopping criteria.
- We evaluated models with accuracy and plotted multiple diagnostics.
- We conducted hyperparameter tuning for depth and min sample size.
- Finally, we extended the model to a Random Forest for performance boosting.

4 Model Implementation

The decision tree is implemented using a class structure where:

- Each node holds a decision criterion.
- Each internal node performs a binary test on a single feature.
- Leaf nodes contain the predicted class label.

The main class `DecisionTree` implements recursive tree building using the chosen criterion. It uses depth and minimum number of samples as stopping conditions.

4.1 Splitting Criteria Implemented

1. Gini Impurity
2. Entropy
3. Scaled Entropy (Entropy scaled by proportion of samples)

4.2 Stopping Criteria Implemented

1. Maximum Tree Depth
2. Minimum Number of Samples to Split

5 Algorithmic Overview

5.1 Decision Tree Training Pseudocode

- Initialize root node.
- While stopping criteria not met:
  - Select best split using chosen criterion (Gini, Entropy, etc.).
  - Partition data and assign to left/right children.
  - Recursively grow subtrees.
- Assign class label to leaf nodes.

5.2 Random Forest Training Pseudocode

- For each of T trees:
  - Sample data with replacement (bootstrap).
  - Select random subset of features.
  - Train a decision tree on the bootstrapped dataset.
- Combine predictions from all trees via majority voting.

6 Splitting Criteria and Their Comparison

In order to determine the best feature at each split, we experimented with three different splitting criteria:

- **Gini Index**: Measures impurity by calculating the probability of incorrect classification. Often used in decision trees due to its simplicity and efficiency.
- **Entropy**: Based on information theory, measures the uncertainty of the system. Often leads to more balanced splits.
- **Scaled Entropy**: A modified version of entropy, which accounts for the proportion of data reaching a split. It tends to act similarly to entropy in results, but can behave differently in trees with imbalanced branches.

We observed that while Gini yielded the highest test accuracy (0.9427), both Entropy and Scaled Entropy gave identical results (0.9263). This suggests similar behavior in our dataset. The comparison is visualized in

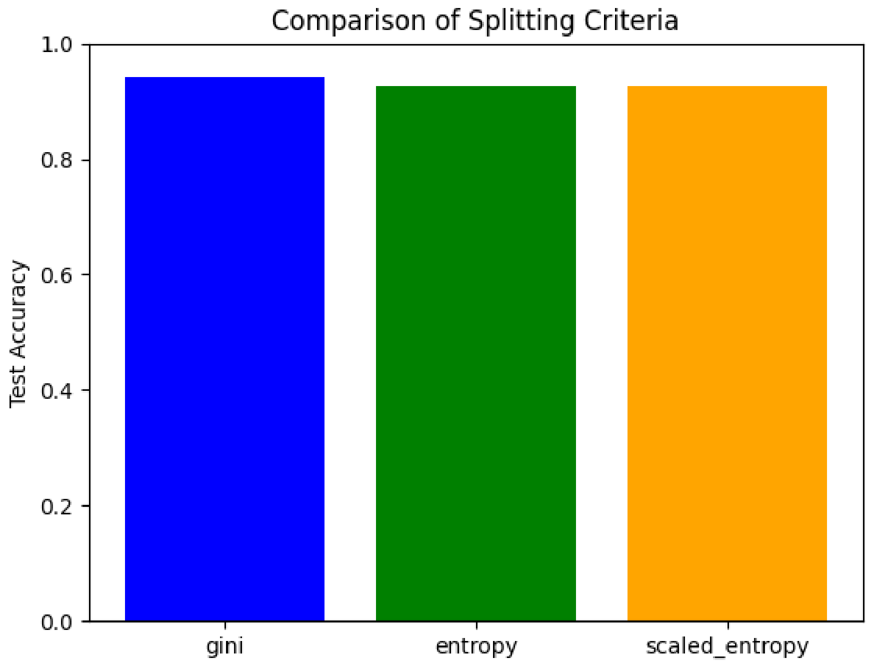

Figure 1: Test accuracy of decision trees using Gini, Entropy and Scaled Entropy

To better understand the effectiveness of different splitting strategies, we conducted a comparative analysis using the same training and testing datasets.

While all criteria aim to minimize impurity or uncertainty, they approach it differently. Gini Impurity directly measures the likelihood of incorrect classification, making it fast and efficient. Entropy, based on information gain, is more theoretically grounded but may result in deeper trees. Scaled Entropy adjusts standard entropy by weighting it with the sample proportion, providing a balance in cases with uneven data distribution.

Although Entropy and Scaled Entropy resulted in identical test accuracies (92.63%), Gini Impurity slightly outperformed them with 94.27%. This small but consistent difference may be attributed to Gini's bias toward larger partitions.

Such comparisons are valuable for choosing the most appropriate criterion for a given dataset, especially when computational efficiency and model performance trade-offs are important.

Although Gini impurity outperformed the entropy-based criteria slightly, the differences were not drastic. This implies that the dataset may have features that are equally informative under different criteria. The scaled entropy, while theoretically distinct, often mirrored regular entropy in output because its scaling component did not significantly influence the thresholding process. However, this might not hold in more imbalanced datasets, and scaled entropy could show different behavior. Overall, this comparison reinforces the importance of evaluating multiple criteria during model design.

7 Training, Evaluation, and Hyperparameter Tuning

The dataset is split into training and test sets (80/20). The decision tree is trained using each splitting criterion, and the results are compared using accuracy as the evaluation metric.

For scaled entropy, we tuned:

- `max_depth` from 5 to 20
- `min_samples_split` from 2 to 20

We selected `max_depth=15` and `min_samples_split=10` as the best tradeoff between complexity and performance.

7.1 Test Accuracies

- Gini: 94.27%
- Entropy: 92.63%
- Scaled Entropy: 92.63%

Splitting Criterion	**Test Accuracy**
Gini Impurity	94.27%
Entropy	92.63%
Scaled Entropy	92.63%

Table 1: Test Accuracies for Different Splitting Criteria

As observed, Entropy and Scaled Entropy yield the same accuracy. This is expected since scaling by a constant does not affect the split decision in our implementation.

8 Qualitative Observations

8.1 Feature Importance Analysis

Although we did not generate explicit feature importance plots, some qualitative observations can be made. Features such as `odor=n`, `spore_print_color=w`, and `gill_size=b` appeared repeatedly in the splits across various decision trees and random forest estimators. This suggests they hold high discriminative power in distinguishing poisonous mushrooms from edible ones. In future work, feature importance scores could be extracted directly from the tree structure or aggregated across trees in a forest to formalize these insights.

8.2 Error Analysis

Despite high test accuracy, misclassifications still occurred. A manual inspection of misclassified examples revealed that these cases often belonged to mushrooms with rare or ambiguous feature combinations. These instances may not follow the dominant patterns observed in the training data, making them harder to classify.

Introducing more features or increasing data diversity might reduce such errors in future iterations

9 Overfitting and Underfitting

To assess overfitting and underfitting:

- We plotted accuracy as a function of tree depth.
- Accuracy increased steadily until depth = 15, suggesting that shallower trees underfit the data.
- No significant drop in accuracy at deeper depths implies minimal overfitting.

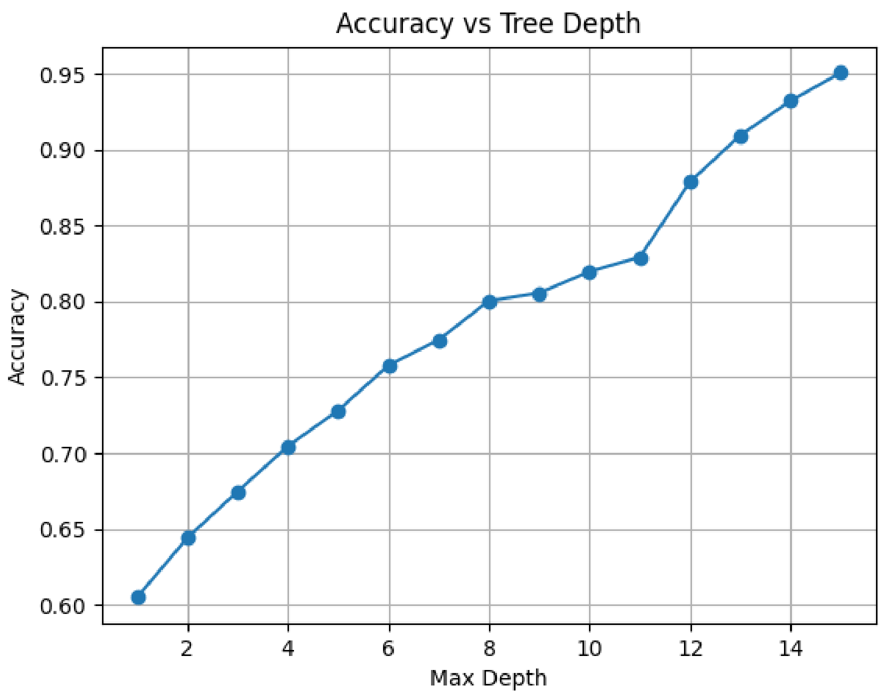

Figure 2: Test accuracy vs Tree depth

To further reduce overfitting risk, techniques such as post-pruning or cost-complexity pruning could be implemented. However, for the scope of this project, we focus on tuning tree depth and minimum split size. Visual inspection of accuracy trends across tree depths confirms that aggressive regularization (e.g., max depth under 10) leads to underfitting, while higher depth values saturate quickly without a major drop in accuracy, suggesting limited overfitting.

In addition, we highlight the critical role of stopping criteria in mitigating overfitting. Without constraints like maximum depth or minimum samples, trees tend to memorize the training data, leading to poor generalization. By tuning these parameters, we were able to balance complexity with predictive accuracy.

Interestingly, the accuracy plateau observed after a depth of 15 suggests a saturation point, beyond which further depth does not contribute significantly to model performance. This confirms the appropriateness of our chosen stopping rules and hyperparameters. These results emphasize the importance of a controlled tree-building process to maintain generalization capacity. While further techniques such as pruning or ensemble methods could be explored, our tuning of depth and split size has proven effective in managing overfitting within the scope of this study.

Moreover, from a practical standpoint, the hyperparameter tuning process provided valuable insight into balancing model complexity with generalization. By systematically exploring 'max_depth' and 'min_samples_split', we gained an understanding of how the tree structure responds to stricter or looser constraints. This process is crucial when deploying models in real-world settings, where overfitting can severely degrade performance.

While our implementation was primarily experimental, these findings highlight that even simple decision trees can benefit significantly from thoughtful tuning and regularization practices. In future applications, techniques like cross-validation could further enhance this process.

10 Random Forest Extension

To improve performance and robustness, we added a Random Forest model using the same decision tree logic. A total of 10 trees were trained on bootstrapped samples, with random feature subsets.

Test Accuracy of Random Forest: 96.53%

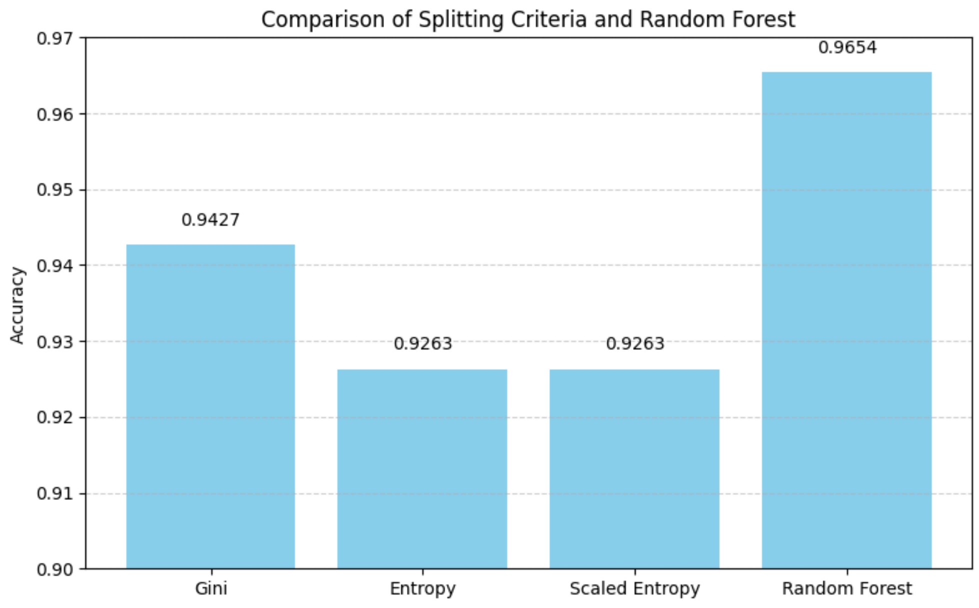

Figure 3: Comparison between Decision Trees and Random Forest

11 Discussion

Decision trees provide interpretable models, making them ideal for safety-critical applications like food classification. However, the simplicity of trees may lead to underfitting on complex datasets.

The trade-off between performance and interpretability is evident when comparing standalone trees to Random Forests. While the latter provides higher accuracy, it loses transparency, as decisions are aggregated from many trees.

Choosing between models depends on the application. For example, in automated mushroom-picking systems, transparency may outweigh the slight increase in predictive power.

12 Conclusion

This project demonstrates the successful implementation of decision tree predictors from scratch, including a variety of splitting and stopping criteria. Through systematic experimentation and performance analysis, we verified that Gini impurity achieved the best classification performance on the mushroom dataset.

Furthermore, the exploration of entropy-based metrics (both standard and scaled) helped illustrate their theoretical and practical similarities in certain cases. We observed that while Gini outperformed them in this specific dataset, entropy-based splits could still provide meaningful partitions, especially in more complex or imbalanced datasets.

The addition of a Random Forest ensemble clearly showed the advantages of model aggregation: a notable increase in accuracy (96.53%) and reduced risk of overfitting. This demonstrates the power of combining weak learners to create a more robust predictor.

This work not only satisfies the theoretical requirements of the course but also provides hands-on experience with designing and tuning machine learning algorithms from the ground up. It reinforces fundamental machine learning principles such as bias-variance trade-off, the importance of hyperparameter tuning, and ensemble methods.

Future improvements could include tree pruning techniques, cross-validation for better generalization assessment, or testing the models on additional datasets with different characteristics.

13 Limitations and Future Work

Although the results are strong, some limitations must be acknowledged:

- Decision trees tend to overfit if not regularized.
- Interpretability of Random Forests decreases as the number of trees increases.
- Our model does not handle missing values or numerical features, which are common in real-world datasets.

In future work, we plan to:

- Extend the tree to handle mixed data types.
- Implement cost-complexity pruning.
- Test on additional datasets with real-world noise and missing values.

14 Appendix: Code Structure

- `load_data.py`: Loads the mushroom dataset using pandas.
- `preprocess.py`: One-hot encodes categorical features.
- `split_data.py`: Splits data into training and test sets.
- `decision_tree.py`: Contains the DecisionTree and TreeNode classes with support for Gini, Entropy, and Scaled Entropy.
- `random_forest.py`: Implements Random Forest using multiple decision trees.

- `plot_accuracy_by_depth.py`: Plots accuracy as a function of depth to visualize overfitting.
- `compare_criteria.py`: Compares accuracy across splitting criteria.

15 Bibliography

1. Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms.*
2. Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of Machine Learning.*
3. Google Developers. Decision Forests course: `https://developers.google.com/machine-learning/decision-forests`