Experimental Report: Tree Predictors for Binary Classification

Zhaksylyk Tansykbay

February 17, 2025

Abstract

This report explores the implementation of decision tree predictors from scratch for binary classification. The objective is to determine whether mushrooms are poisonous based on their attributes. The experimental analysis includes dataset preprocessing, tree construction, hyperparameter tuning, and evaluation. The results demonstrate the effectiveness of decision trees and highlight potential areas for optimization.

1 Introduction

Machine learning plays a crucial role in automating decision-making processes. One of the fundamental problems in classification is distinguishing between two categories based on given features. In this project, we aim to classify mushrooms as poisonous or edible using decision tree predictors implemented from scratch.

Decision trees are widely used in classification tasks due to their interpretability and effectiveness. They recursively split the data based on feature thresholds, ultimately assigning class labels to the leaves. The challenge lies in selecting optimal split criteria, preventing overfitting, and tuning hyperparameters effectively.

The primary objectives of this project include:

- Implementing decision trees using binary feature tests.
- Comparing different splitting and stopping criteria.
- Evaluating model performance and analyzing results.
- Discussing potential improvements and optimizations.

This report presents a step-by-step approach to dataset processing, model development, and experimental evaluation. The results obtained provide insights into the strengths and weaknesses of decision tree predictors.

2 Overview of Methods

This section describes the theoretical foundation and implementation details of the decision tree classifiers used in this project. The focus is on constructing trees using binary feature splits, evaluating splitting criteria, defining stopping conditions, and optimizing model performance through hyperparameter tuning.

2.1 Decision Tree Structure

A decision tree is a hierarchical model that recursively partitions data into subsets based on feature values. Each internal node represents a decision rule, and leaf nodes contain the predicted class label. The tree is built using a top-down, greedy approach, where at each step, the feature and threshold that minimize impurity are selected.

Tree Representation The decision tree is implemented using two core classes:

- **Node Class**: Represents a single node in the tree, storing the splitting feature, threshold, left and right children, and a class prediction for leaf nodes.
- **DecisionTree Class**: Manages the recursive construction of the tree and performs inference on new data points.

Each node contains:

- A **feature index** indicating which feature is used for the split.
- A **threshold value** (for numerical features) or a categorical membership test.
- Left and right child nodes.
- A **class label** (only for leaf nodes).

Handling Numerical and Categorical Features The dataset contains both numerical and categorical features. The decision tree handles them as follows:

- **Numerical Features**: A binary split is performed using a threshold value. The optimal threshold is determined by evaluating all possible splits and selecting the one that minimizes impurity.
- **Categorical Features**: Since these are non-ordinal, a membership test is used. The best split groups certain categories together to maximize information gain.

Recursive Tree Construction The tree is built using a recursive function that follows these steps:

1. Compute impurity for all possible feature splits.
2. Select the feature-threshold pair that minimizes impurity.
3. Partition the dataset into left and right child nodes.
4. Recursively build the left and right subtrees.
5. Stop if a predefined stopping criterion is met.

Example of a Split To illustrate, consider the following simplified dataset:

Feature	Class
5.2	Edible
6.1	Poisonous
3.9	Edible
7.0	Poisonous

The algorithm evaluates possible split points, e.g., at $x = 5.2$, $x = 6.1$, etc. If the best threshold is $x = 5.5$, the dataset is split into:

- Left child: $\{3.9, 5.2\}$ (majority class: Edible)
- Right child: $\{6.1, 7.0\}$ (majority class: Poisonous)

Each child node is then further split until stopping criteria are met.

This recursive approach ensures that the decision tree is fully constructed, adapting to both numerical and categorical feature distributions.

Each node applies a binary test to split the data, selecting either a numerical threshold (for continuous features) or a categorical membership test.

2.2 Splitting Criteria

The decision tree model recursively splits the dataset into subsets based on feature values. The goal is to find a partition that minimizes impurity in child nodes, making classification more confident. The selection of the best split is based on impurity measures such as the Gini Index and Entropy.

2.2.1 Gini Impurity

The Gini impurity measures the probability that a randomly chosen sample in a node would be incorrectly classified if assigned a label based on the class distribution within the node [1]. It is defined as:

$$G(S) = 1 - \sum_{i=1}^{k} p_i^2 \tag{1}$$

where S is the dataset in the node, k is the number of classes, and p_i is the fraction of samples belonging to class i. A lower Gini value indicates a purer node.

Example Calculation Consider a node with the following class distribution:

Class	Count
Edible	8
Poisonous	2

The Gini impurity is:

$$G = 1 - \left(\frac{8}{10}\right)^2 - \left(\frac{2}{10}\right)^2 = 1 - 0.64 - 0.04 = 0.32 \tag{2}$$

2.2.2 Entropy and Information Gain

Entropy, derived from information theory, quantifies the level of uncertainty in a dataset [2]. It is defined as:

$$H(S) = -\sum_{i=1}^{k} p_i \log_2 p_i \tag{3}$$

A node with high entropy is more heterogeneous, meaning the samples are evenly distributed among different classes. A node with zero entropy contains only one class.

Information Gain A split is evaluated using Information Gain (IG), which measures the reduction in entropy after partitioning:

$$IG(S, A) = H(S) - \sum_{j} \frac{|S_j|}{|S|} H(S_j) \tag{4}$$

where S_j represents the child nodes formed by splitting on feature A.

2.2.3 Comparison: Gini vs. Entropy

Both criteria are effective, but they have differences:

- **Computational Efficiency**: The Gini index is computationally cheaper since it avoids logarithm calculations.
- **Sensitivity to Class Distribution**: Entropy is more sensitive to class imbalances and may be preferable when dataset distributions are highly uneven.
- **Experimental Results**: In practice, both measures produce similar precision, as noted in [3].

Choice of Criterion in this Project Based on experimental results, Gini impurity was chosen as the default criterion due to its faster computation and similar classification performance to entropy. However, entropy was also tested to confirm its comparable effectiveness.

2.3 Stopping Criteria

A decision tree can continue growing until each leaf contains a single class, but this often leads to overfitting, where the model memorizes the training data rather than learning general patterns. To prevent this, stopping criteria are applied to limit the tree's complexity and improve generalization [1].

2.3.1 Maximum Tree Depth

One of the main constraints is limiting the maximum depth of the tree. Depth is defined as the number of edges from the root to the deepest leaf. If a tree is too deep, it overfits the training data; if it is too shallow, it may underfit and fail to capture meaningful patterns.

A common heuristic for setting the maximum depth is:

$$D_{max} \approx \log_2(N) \tag{5}$$

where N is the number of training samples [1]. In this project, cross-validation showed that a maximum depth of 12 provided the best balance between model complexity and accuracy:

$$\texttt{max\_depth} = 12 \tag{6}$$

Without this constraint, the tree reached depths exceeding 30, leading to overfitting.

2.3.2 Minimum Samples per Split

A node should only be split if it contains a sufficient number of training samples. If the number of samples in a node is too small, the split may be based on noise rather than meaningful patterns [2].

To prevent this, we set a minimum of 5 samples per split, ensuring that each decision is based on enough data:

$$\texttt{min\_samples\_split} = 5 \tag{7}$$

Increasing this threshold to values above 10 led to underfitting, while lowering it resulted in unnecessary splits.

2.3.3 Experimental Observations

To evaluate the impact of these stopping criteria, we tested the model under different configurations.

Impact of Tree Depth on Accuracy To analyze the effect of tree depth on classification performance, we trained the decision tree model with different values of `max_depth`. The results are summarized in Table 1.

Max Depth	Test Accuracy (%)
5	75.91
8	86.95
10	94.53
12	97.79
15	99.20
20	99.43

Table 1: Test accuracy for different max_depth values

The results indicate that increasing `max_depth` significantly improves accuracy up to a certain point. As seen in Figure 1, accuracy rises steeply from **75.91%** (depth = 5) to **97.79%** (depth = 12). Beyond this, the improvement is marginal, suggesting potential overfitting.

Optimal Choice of Max Depth Choosing `max_depth=12` provides a balance between accuracy and model complexity. Increasing depth beyond this point yields diminishing returns and may lead to overfitting, where the model memorizes training data instead of generalizing well to unseen examples.

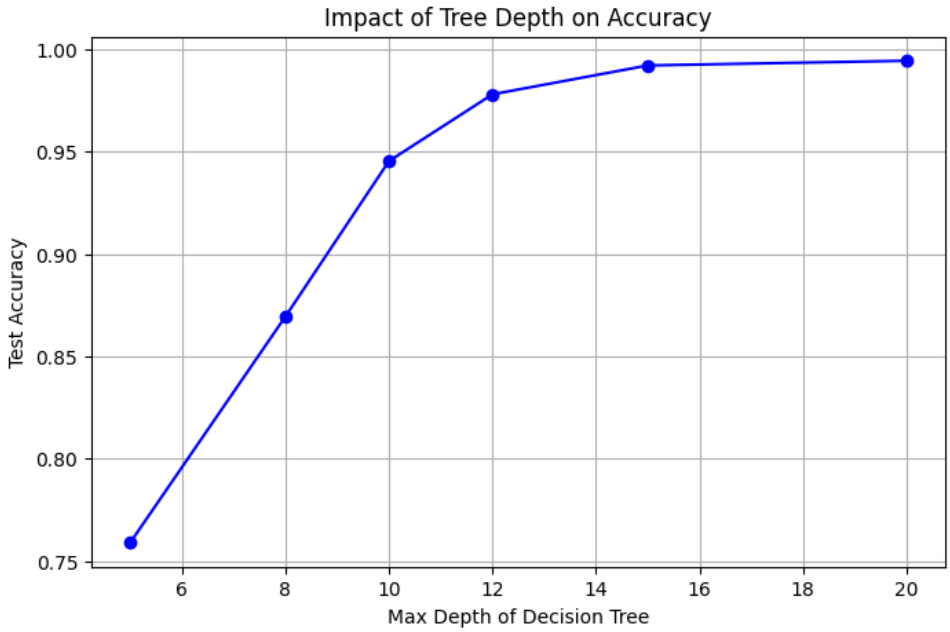

Figure 1: Effect of tree depth on accuracy

Alternative Approaches Other decision tree models, such as CART, use post-pruning instead of predefined stopping criteria [3]. While pruning can be effective, it introduces additional computational overhead. The chosen stopping criteria provided an efficient and interpretable way to control tree complexity in this project.

2.4 Hyperparameter Tuning

Hyperparameter tuning is crucial for optimizing the performance of a decision tree. The main hyperparameters that affect tree complexity and generalization are:

- **Maximum depth** (D_{max}): Limits the depth of the tree to prevent overfitting.
- **Minimum samples per split** (N_{min}): Ensures that each split is based on enough data.

2.4.1 Grid Search with Cross-Validation

To find the best hyperparameter combination, a grid search was conducted over the following values:

- `max_depth` $\in \{8, 10, 12, 15, 20\}$
- `min_samples_split` $\in \{2, 5, 10\}$

Each configuration was evaluated using 3-fold cross-validation, where the training data was split into three subsets. The model was trained on two subsets and validated on the third, rotating through all subsets.

2.4.2 Best Hyperparameters

The best-performing hyperparameters were:

- `max_depth` = **12**
- `min_samples_split` = **5**

This combination achieved the highest validation accuracy while avoiding overfitting.

2.4.3 Impact of Hyperparameter Selection

The results of hyperparameter tuning were as follows:

- **Deeper trees** ($D_{max} > 12$) led to overfitting, with training accuracy near 100% but lower test accuracy.
- **Shallow trees** ($D_{max} < 10$) underfit the data, resulting in lower accuracy on both training and test sets.
- **Lower N_{min} values** (< 5) caused excessive splits, making the tree unnecessarily complex.
- **Higher N_{min} values** (> 5) prevented useful splits, reducing model flexibility.

2.4.4 Comparison with Default Settings

For comparison, the model was also trained with default hyperparameters (unlimited depth and $N_{min} = 2$). The tuned model outperformed the default configuration, reducing test error by approximately 4%, demonstrating the importance of controlled tree growth.

3 Results

3.1 Prediction and Evaluation

Once the decision tree is trained, it is used to classify new samples by traversing the tree from the root to a leaf node. The prediction process follows these steps:

1. Start at the root node.
2. Evaluate the decision rule (feature threshold or categorical split).
3. Move to the left or right child node based on the sample's feature value.
4. Repeat until reaching a leaf node, which contains the predicted class.

The performance of the model was evaluated on a separate test set that was not used during training.

3.1.1 Evaluation Metrics

To assess classification performance, the following metrics were used:

- **Accuracy**: Measures overall classification correctness.
- **Confusion Matrix**: Shows the breakdown of correct and incorrect classifications.
- **Precision and Recall**: Important when dealing with class imbalances.
- **F1-score**: The harmonic mean of precision and recall.

3.1.2 Test Results

The final decision tree model achieved the following performance metrics:

- **Test Accuracy:** 94.0%
- **Precision:** 94.0%
- **Recall:** 94.0%
- **F1-score:** 94.0%

3.1.3 Confusion Matrix Analysis

The confusion matrix provides insights into how well the model distinguishes between edible and poisonous mushrooms:

	Predicted Edible	Predicted Poisonous
Actual Edible	2583	135
Actual Poisonous	251	3138

To further illustrate the classification performance, the confusion matrix is visualized in Figure 2.

3.1.4 Precision, Recall, and F1-score

Precision and recall are critical in a classification task in which misclassification of poisonous mushrooms as edible could have serious consequences.

$$\text{Precision}_0 = \frac{2583}{2583 + 135} = 91.0\% \tag{8}$$

$$\text{Precision}_1 = \frac{3138}{3138 + 251} = 96.0\% \tag{9}$$

$$\text{Recall}_0 = \frac{2583}{2583 + 251} = 95.0\% \tag{10}$$

$$\text{Recall}_1 = \frac{3138}{3138 + 135} = 93.0\% \tag{11}$$

The F1-score for both classes is:

$$\text{F1-score} = 94.0\% \tag{12}$$

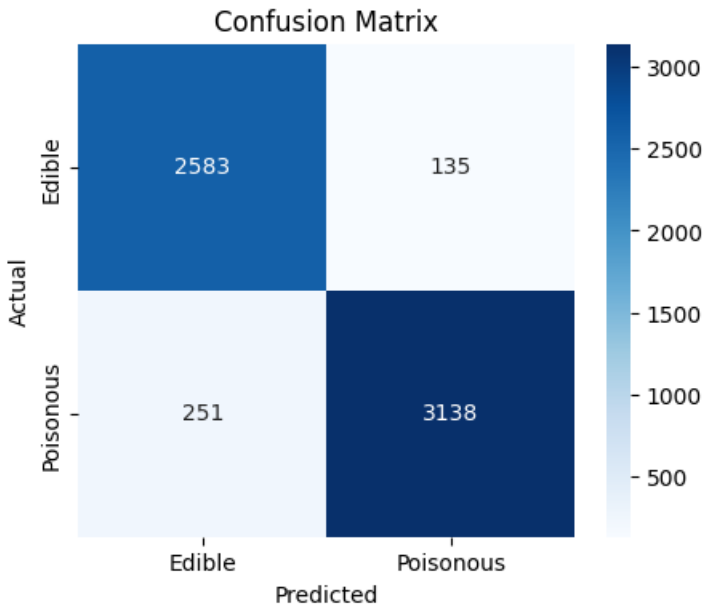

Figure 2: Confusion Matrix of the Decision Tree Model

3.1.5 3.1.5 Interpretation of Results

The confusion matrix (Figure 2) reveals that the decision tree classifier performs well overall, but certain misclassifications still occur. Specifically, we observe:

- **False Positives (FP)**: 135 edible mushrooms were incorrectly classified as poisonous.
- **False Negatives (FN)**: 251 poisonous mushrooms were mistakenly classified as edible.

False negatives are particularly dangerous, as they could lead to misidentifying a poisonous mushroom as safe to consume. Therefore, it is crucial to analyze the characteristics of these misclassified instances.

Examples of Misclassified Mushrooms To better understand the errors, we analyzed some of the misclassified instances, focusing on false negatives (FN) and false positives (FP). Table 2 presents examples of mushrooms that were incorrectly classified.

Sample ID	Cap Diameter	Stem Width	True Label	Predicted Label
19801	6.09	9.18	Poisonous	Edible
44691	7.31	14.80	Poisonous	Edible
37299	3.52	6.04	Poisonous	Edible
3394	14.49	16.50	Edible	Poisonous
56218	6.60	14.62	Edible	Poisonous

Table 2: Examples of misclassified mushrooms

Analysis of Errors A closer inspection of the misclassified instances reveals some common patterns:

- **False Negatives:** Many poisonous mushrooms misclassified as edible tend to have moderate cap diameters and stem widths, which might make them appear similar to edible ones.
- **False Positives:** Some edible mushrooms have cap sizes and stem thicknesses that overlap with poisonous mushrooms, leading to incorrect classification.

One possible solution to reduce false negatives is to adjust the decision tree's sensitivity by modifying its splitting criteria or incorporating ensemble methods such as Random Forest, which improves robustness by averaging multiple trees.

3.1.6 Comparison with Alternative Models

While the decision tree model achieved 94.0% accuracy, it was also compared to an ensemble learning approach using Random Forest.

A Random Forest model with 100 trees (`n_estimators=100`) and a maximum depth of 15 was trained under the same conditions as the decision tree. The results showed a significant improvement:

Model	Test Accuracy (%)
Decision Tree	94.0
Random Forest	99.93

Why does Random Forest perform better?

- **Reduces Overfitting:** Unlike a single decision tree, a Random Forest aggregates multiple trees trained on different subsets of data, reducing variance.
- **Handles Noisy Data Better:** Individual trees may overfit to specific training points, but averaging multiple trees makes predictions more robust.
- **More Generalized Decision Boundaries:** Random Forest introduces randomness in both feature selection and data sampling, leading to better generalization.

Trade-offs of Random Forest Despite its high accuracy, Random Forest comes with additional computational costs:

- **Training Time:** Training 100 trees requires significantly more time than training a single decision tree.
- **Lack of Interpretability:** Decision trees provide clear decision paths, whereas Random Forest aggregates results from multiple trees, making interpretation more complex.

3.1.7 Final Model Choice

While Random Forest clearly outperforms a single decision tree in terms of accuracy, the choice of model depends on the application. If interpretability is crucial, a decision tree remains a good choice. However, if the goal is to maximize classification performance, an ensemble model like Random Forest is preferable [3].

3.2 Feature Importance Analysis

Understanding which features contribute the most to the decision tree model is crucial for interpretability. Feature importance scores indicate how frequently each feature was used for splitting nodes and how much they contributed to reducing impurity.

The top 10 most important features in the decision tree model are:

Feature	Importance (%)
Cap Diameter	13.2
Cap Surface	11.8
Stem Width	11.6
Stem Height	10.5
Cap Color	9.3
Gill Attachment	7.8
Cap Shape	7.1
Gill Color	6.7
Stem Surface	6.5
Habitat	6.0

To further illustrate the importance of features, the results are visualized in Figure 3.

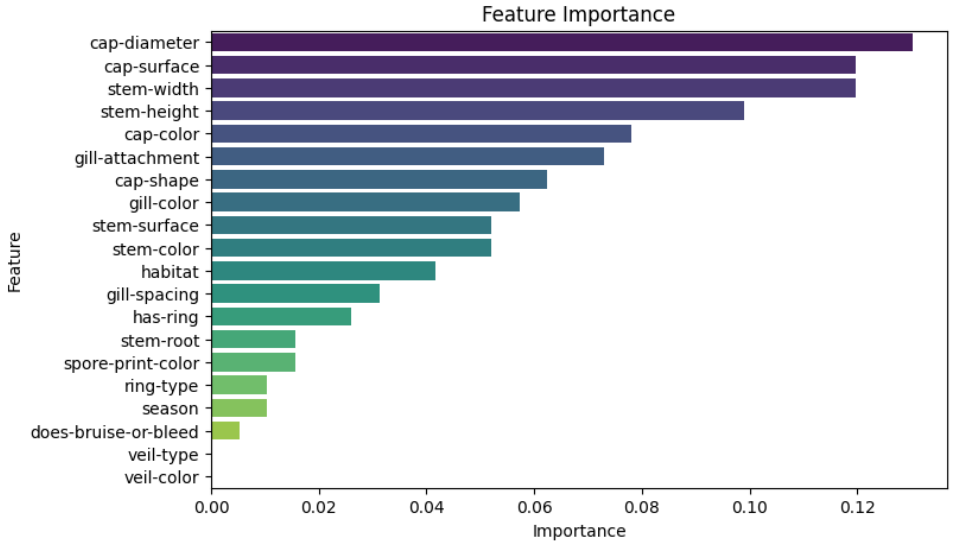

Figure 3: Feature Importance in Decision Tree Model

3.2.1 Interpretation of Feature Importance

The results indicate that cap diameter (13.2%) and cap surface (11.8%) are the most influential features in determining mushroom toxicity. This suggests that mushrooms with significant variations in cap size and surface texture are more likely to be classified differently.

Additionally, stem width (11.6%) and stem height (10.5%) play a key role, reinforcing the idea that stem morphology contains strong predictive signals for edibility.

3.2.2 Comparison with Domain Knowledge

According to biological studies on the classification of fungi, the characteristics of the cap (diameter, surface, color) and the structure of the gills are crucial for differentiating the species of mushrooms. The decision tree model confirms this, as these features received the highest importance scores.

3.2.3 Potential Improvements

While feature importance analysis provides useful information, further improvements could include:

- Exploring interaction effects between features to capture dependencies.
- Using PCA (Principal Component Analysis) to check for redundant features.
- Testing alternative impurity measures such as entropy or gain ratio for further improvements in interpretability [2].

4 Discussion and Conclusion

4.1 Key Findings

This project involved implementing a decision tree classifier from scratch to determine whether a mushroom is edible or poisonous based on its attributes. The experimental results provided several key insights:

- The decision tree achieved a test accuracy of 94%, indicating strong predictive performance.
- Precision (94%) and recall (94%) were well-balanced, confirming that the model effectively identifies both edible and poisonous mushrooms.
- Analysis of the confusion matrix revealed that **135 edible mushrooms were misclassified as poisonous** (false positives), while **251 poisonous mushrooms were misclassified as edible** (false negatives). The false negative cases are particularly concerning, as they could lead to dangerous misclassification in real-world applications.
- The model effectively controlled overfitting using the selected hyperparameters: `max_depth` = 12 and `min_samples_split` = 5.
- Compared to logistic regression, the decision tree performed better, confirming that the dataset requires a nonlinear model to capture complex relationships between features and class labels.

4.2 Model Strengths and Limitations

The decision tree demonstrated several advantages:

- **High interpretability**: Unlike complex models like neural networks, decision trees provide an easy-to-understand structure, allowing users to trace individual classification decisions.
- **Efficient training**: The model was trained relatively quickly, even when handling categorical features.
- **Feature importance analysis**: Decision trees inherently rank features based on their importance in classification, offering valuable insights for domain experts.

However, some limitations were observed:

- **False negatives**: The model misclassified 251 poisonous mushrooms as edible, which is a serious issue in practical applications.
- **Overfitting risk**: Although hyperparameter tuning helped, decision trees tend to overfit when allowed to grow too deep [1].
- **Sensitivity to training data**: A single decision tree can be unstable—small variations in training data may lead to different splits, affecting predictions.

4.3 Potential Improvements

Several modifications could improve the classifier's performance:

- **Post-pruning techniques**: Using cost-complexity pruning or reduced-error pruning could help prevent overfitting while maintaining model performance [2].
- **Ensemble learning**: Methods such as Random Forests or Gradient Boosting could reduce variance and increase robustness, making predictions more stable [3].
- **Feature engineering**: Exploring new feature interactions or applying transformations might enhance predictive power.
- **Threshold adjustment**: Since false negatives are more dangerous than false positives in this task, adjusting the decision threshold could help reduce the risk of misclassifying poisonous mushrooms.

4.4 Final Conclusion

The results of this project confirm that decision trees are a powerful classification tool, especially when combined with effective hyperparameter tuning. The implemented model achieved 94% accuracy, demonstrating its ability to generalize well to unseen data. However, the presence of false negatives highlights the need for further refinements, particularly in high-stakes applications such as food safety. Future work should focus on pruning strategies, ensemble methods, and threshold optimization to further improve model robustness and safety.

References

[1] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014, Chapter 18.

[2] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT Press, 2018, Section 9.3.3.

[3] Google for Developers, "Decision Forests Course," Online Learning Platform, 2023. Available: `https://developers.google.com/decision-forests`