Report on Tree Predictors for Mushroom Classification

Damir Uvayev

1. Introduction

Decision trees are one of the most widely used machine learning algorithms for classification tasks due to their interpretability, simplicity, and ability to model complex decision boundaries using a series of simple, single-feature tests. In this project, I implement tree predictors from scratch to classify mushrooms as either poisonous or edible based solely on their observed characteristics. This task is of practical importance as misclassifying a poisonous mushroom as edible can have severe consequences.

The Mushroom dataset, which I use in this study, is a well-known benchmark in machine learning. It consists of two variants: a primary dataset with 173 examples (each corresponding to a unique mushroom species) and a secondary, simulated dataset with over 61,000 instances. The primary dataset presents a challenge due to its limited size, which can lead to high variance and difficulties in generalizing from the training data. Conversely, the secondary dataset provides a large and well-separated sample set, enabling a clearer analysis of the model's behavior and performance.

In my implementation, the decision tree uses single-feature binary tests. For numerical features, the tests are based on threshold comparisons, while for categorical features, the tests check for membership in a specified category. To determine the best splits, I employ several impurity measures: the Gini index, standard entropy, and a scaled entropy measure (where the entropy is normalized by the logarithm of the number of classes). This flexibility allows me to examine how different criteria impact the tree's performance and structure.

Furthermore, the model incorporates multiple stopping criteria such as maximum tree depth, minimum number of samples in a leaf, and purity of the node. These criteria help prevent overfitting, particularly in the case of the secondary dataset where the large number of examples could lead to an excessively deep and complex tree if left unrestricted.

Overall, this project aims to provide a comprehensive understanding of tree-based classifiers by:

- Implementing a decision tree predictor from scratch.
- Experimenting with different impurity measures and stopping criteria.

- Evaluating model performance on both a small and a large version of the Mushroom dataset.
- Analyzing the results in terms of accuracy, model complexity, and feature importance.

The insights gained from this project not only illustrate the strengths and weaknesses of decision trees in different scenarios but also lay the groundwork for potential future extensions such as ensemble methods (e.g., random forests) and pruning strategies.

2. Dataset Description and Preprocessing

In this project, I worked with two versions of the Mushroom dataset: a primary dataset and a secondary dataset. These datasets differ significantly in size and complexity, which allowed me to analyze the behavior of my decision tree implementation under different data regimes.

2.1 Primary Dataset

The primary dataset contains 173 examples, each corresponding to a distinct mushroom species. For each species, 20 attributes are recorded, of which 17 are categorical and 3 are metrical. Due to its small size, the primary dataset poses challenges such as high variance and limited generalizability. Moreover, some of the metrical features in this dataset are represented as ranges (for example, `[10, 20]`), rather than as single numerical values. To address these issues, I designed specific data cleaning functions.

2.2 Secondary Dataset

The secondary dataset is a simulated version generated from the primary data. It contains over 61,000 examples (with approximately 353 instances per species). This larger dataset provides a much richer sample, allowing the decision tree to capture fine-grained patterns. In addition, the secondary dataset typically has metrical features already represented as numerical values. The high volume and clearer class separations in this dataset result in significantly higher classification performance.

2.3 Data Preprocessing

Before training the decision tree, I performed several preprocessing steps to ensure the data were in a suitable format:

1. **Dropping Non-informative Columns:** The original dataset includes columns such as `family` and `name`, which are descriptive but do not contribute to the predictive task. These columns were removed from the dataset.

2. **Cleaning Metrical Features:** For numerical attributes such as `cap-diameter`, `stem-height`, and `stem-width`, I applied the `clean_metrical_value` function. In the primary dataset, many values appear as ranges (e.g., "`[10, 20]`"). This function extracts the numbers from such strings and computes their average, yielding a single representative value. In cases where the value is already numeric (as in the secondary dataset), it is left unchanged.

3. **Cleaning Categorical Features:** For categorical attributes, I used the `clean_nominal_value` function. Often, categorical values in the dataset are stored as a list-like string (for instance, "`[x, f]`"). This function removes the surrounding brackets and returns the first element, ensuring that each attribute is represented by a single category.

4. **Handling Missing Data:** If any value is missing (i.e., it is `NaN`), the cleaning functions return a default value: numerical features become `None` and categorical features become `"unknown"`. This approach ensures consistency across the dataset.

5. **Conversion to Dictionary Format:** After cleaning, I convert the processed DataFrame into a list of dictionaries for features (`X`) and a list of labels (`y`). This format is well-suited for the custom decision tree implementation.

Through these preprocessing steps, I ensured that the datasets—despite their differences in size and format—were standardized and ready for model training and evaluation. This careful preprocessing is crucial for avoiding data leakage and ensuring that the decision tree learns meaningful patterns from the data.

3. Methodology

This section describes in detail the approach I followed to implement the decision tree predictor from scratch.

3.1 Tree Predictor Design

The decision tree is built using two main classes: a `Node` class and a `DecisionTree` class. Each `Node` represents a single decision point in the tree. At an internal node, a single-feature binary test is performed—using a threshold for numerical features or an equality test for categorical features—to determine the subsequent branch. Each node stores:

- A flag (`is_leaf`) indicating whether it is terminal.
- A prediction value for leaves.
- The feature and threshold (or category) used for splitting.

- Pointers to its left and right children.
- The number of training examples that reach the node.
- The impurity value computed for the node.

The `DecisionTree` class orchestrates the construction of the tree using the following steps:

1. **Initialization:** The constructor accepts hyperparameters including maximum tree depth (`max_depth`), minimum samples per leaf (`min_samples_in_leaf`), the impurity criterion (`criterion`), and the maximum number of candidate thresholds for numeric features (`max_candidates`). The impurity criterion can be set to "gini", "entropy", or "scaled_entropy".
2. **Fitting the Model:** The `fit` method first determines the total number of examples and the set of all class labels. It then infers the type (numeric or categorical) for each feature from the first training example. Using these types, the method initiates a recursive tree-building process.
3. **Recursive Tree Building:** The private method `_build_tree` creates a new node and computes its impurity. The algorithm stops splitting when the node is pure, when the number of samples is below the minimum threshold, or when the maximum depth is reached. Otherwise, it finds the best split by calling the `_find_best_split` method.
4. **Best Split Selection:** In the `_find_best_split` method, each feature is evaluated for potential splits. For numeric features, candidate thresholds are generated either from unique values or from quantiles if there are many unique values. For categorical features, each unique category is considered as a candidate. The method computes the impurity decrease (gain) for each candidate split and selects the split that maximizes the gain.

3.2 Impurity Measures

The decision tree uses impurity measures to determine the quality of splits. I implemented three measures:

- **Gini Index:** Measures the probability of misclassification if a sample is randomly labeled according to the distribution in the node.
- **Entropy:** Captures the disorder within the node and is defined as $-\sum_i p_i \log_2(p_i)$.
- **Scaled Entropy:** Normalizes the standard entropy by dividing by $\log_2(K)$, where K is the total number of classes. In binary classification, scaled entropy is equivalent to standard entropy.

These measures are used in the `_impurity` method and help determine the best split by computing the weighted decrease in impurity after a split.

3.3 Stopping Criteria

The tree construction process stops based on three criteria:

- **Node Purity:** If all training examples in a node belong to the same class.
- **Minimum Samples per Leaf:** If the number of examples in a node is less than a predefined threshold (`min_samples_in_leaf`).
- **Maximum Depth:** If the current depth equals or exceeds the maximum allowed depth (`max_depth`).

These criteria help prevent overfitting by avoiding overly complex trees.

3.4 Hyperparameter Tuning and Evaluation

To assess the performance of the decision tree, I adopt an adaptive evaluation strategy:

- For the **Primary Dataset** (173 examples), I use 5-fold cross-validation. Fixed hyperparameters (e.g., `max_depth = 5` and `min_samples_in_leaf = 5`) are used with the scaled entropy criterion.
- For the **Secondary Dataset** (over 61,000 examples), I use an 80%/20% train/test split. Although grid search might select an unlimited depth (i.e., `max_depth = None`), I override this with a fixed value (e.g., 15) to balance training time and performance.

Evaluation metrics include the 0–1 loss (accuracy), confusion matrix, and detailed classification reports. In addition, I compute feature importances by aggregating the impurity decrease contributions from all splits in the tree.

3.5 Implementation Summary

In summary, the decision tree predictor I implemented:

- Uses single-feature binary tests based on thresholds for numerical features and membership tests for categorical features.
- Employs three impurity measures (Gini, entropy, and scaled entropy) to choose the best splits.
- Incorporates multiple stopping criteria (node purity, minimum samples, maximum depth) to control model complexity.
- Is trained using an adaptive evaluation strategy tailored to dataset size.

3.6 Detailed Hyperparameter Tuning Process

To optimize the performance of the decision tree, I conducted a grid search over two key hyperparameters: `max_depth` and `min_samples_in_leaf`. For the primary dataset, I used 5-fold cross-validation to obtain robust performance estimates. The following ranges were explored:

- `max_depth`: {3, 5, 7, None} (with `None` representing no depth restriction).
- `min_samples_in_leaf`: {1, 5, 10}.

For each parameter combination, the average cross-validation accuracy was computed. The best performance on the primary dataset was achieved with `max_depth = 5` and `min_samples_in_leaf = 5`. For the secondary dataset, although grid search initially favored an unbounded tree, I overrode the setting to use `max_depth = 15` to reduce training time while preserving high accuracy.

3.7 Comparative Analysis of Impurity Measures

The decision tree implementation supports three impurity measures:

- **Gini Index:** Calculates the probability of misclassification and tends to favor splits that produce large, pure partitions.
- **Entropy:** Measures the disorder within a node using the formula $-\sum_i p_i \log_2(p_i)$, making it sensitive to the distribution of classes.
- **Scaled Entropy:** Normalizes entropy by dividing it by $\log_2(K)$, where K is the number of classes. This normalization is especially useful when extending the method to multi-class problems. In the binary case, scaled entropy behaves equivalently to standard entropy.

Experimental observations on the Mushroom dataset indicate that while Gini and entropy yield similar splits in a binary setting, scaled entropy offers a normalized measure that could be beneficial for more complex classification tasks.

3.8 Expanded Discussion on Stopping Criteria

The growth of the decision tree is regulated by several stopping criteria to prevent overfitting:

- **Node Purity:** Splitting stops if all training examples in a node belong to the same class, implying that no further decision is necessary.

- **Minimum Samples per Leaf:** To ensure statistical significance in each decision, a node must contain a minimum number of samples. Setting this threshold too low may lead to overfitting, while too high a threshold might result in underfitting.
- **Maximum Depth:** Limiting the depth of the tree prevents the model from becoming overly complex. Although a deeper tree can model more intricate patterns, it also increases the risk of overfitting and computational expense. For the large secondary dataset, I capped the depth at 15 to strike a balance between model complexity and training efficiency.

These criteria collectively ensure that the tree remains both interpretable and generalizable.

This expanded methodology provides a robust and flexible framework for binary classification, addressing both theoretical and practical considerations in tree-based learning.

4. Experimental Results

4.1 Primary Dataset Results

For the primary dataset, which consists of 173 examples, I employed 5-fold cross-validation to evaluate model performance. The key results are summarized as follows:

- **Accuracy:** The average accuracy across the 5 folds was approximately 54.9%. This modest performance is expected given the very limited amount of training data and the inherent ambiguity in some of the features.
- **Learned Tree Structure:** The decision tree constructed on the primary data is relatively shallow. For example, the root node splits on `stem-width` (i.e., whether `stem-width` ≤ 7.5). Subsequent nodes further split on features such as `gill-spacing`, `stem-height`, and `Cap-surface`. The structure indicates that certain features, like `stem-height` and `Cap-surface`, are used to refine the decision boundaries.
- **Feature Importances:** The computed importances reveal that features such as `stem-height` (0.0784) and `Cap-surface` (0.0638) contribute more substantially to the overall impurity reduction than others (e.g., `gill-color` at 0.0214). These values suggest that, for this dataset, the model relies more on certain key features.

The relatively low CV accuracy indicates that the model struggles to generalize with such a small dataset, which is a common challenge when training on limited data.

4.2 Secondary Dataset Results

The secondary dataset, containing over 61,000 examples, provides a much richer training environment. For this dataset, I used an 80%/20% train/test split. Although the grid search initially suggested an unbounded tree (i.e., `max_depth=None`), I overrode this with a fixed depth of 15 to ensure faster training and to limit model complexity. The results obtained are as follows:

- **Test Accuracy:** The final model achieved a test accuracy of approximately 95.2%, demonstrating its ability to capture and generalize the patterns in this large and well-separated dataset.
- **Learned Tree Structure:** The decision tree for the secondary data is significantly deeper and more complex. The root node splits on `stem-width` (with a threshold of approximately 8.31), and subsequent splits involve features such as `gill-spacing`, `stem-height`, `cap-surface`, `cap-shape`, among others. This depth allows the model to capture fine-grained details of the data distribution.
- **Confusion Matrix and Classification Report:** The confusion matrix shows a high number of correctly classified instances for both classes. The classification report confirms that both classes achieve high precision, recall, and f1-scores, which supports the high overall accuracy.
- **Feature Importances:** The feature importances computed from the tree indicate that `cap-surface` (0.1411) and `stem-surface` (0.0923) are particularly influential in reducing impurity. Other features, such as `stem-width` and `gill-spacing`, also contribute significantly, highlighting their importance in the decision-making process.

These results confirm that, when provided with a large, clean dataset, the decision tree model is capable of achieving excellent performance, though the model complexity increases substantially.

5. Discussion

The experimental results highlight several important aspects of the implemented decision tree model and its sensitivity to dataset characteristics.

First, on the primary dataset (173 examples), the 5-fold cross-validation accuracy was around 54.9%. This relatively low accuracy is indicative of the challenges posed by a very small dataset. With only a few examples, the decision tree has limited data to learn robust decision boundaries. As a consequence, the model tends to overfit to the small training set and suffers from high variance. Additionally, some of the features in the primary dataset may be ambiguous or less discriminative at the species level, further hindering performance.

In contrast, the secondary dataset, which contains over 61,000 examples, achieved a test accuracy of approximately 95.2%. The large size and higher quality of the secondary dataset allow the decision tree to capture more complex and reliable patterns. Although the grid search initially suggested an unbounded tree (i.e., `max_depth = None`), I overrode this by setting `max_depth = 15` to control the model complexity and reduce training time. This controlled depth was sufficient to achieve near-perfect classification without incurring excessive computational cost.

The different performance levels between the primary and secondary datasets underscore the critical impact of dataset size and quality on model accuracy. In environments with abundant, well-separated examples—as in the secondary dataset—even a decision tree built from scratch can achieve excellent performance. However, when data are scarce, as with the primary dataset, the model's capacity to generalize is significantly impaired.

The inclusion of multiple impurity measures (Gini, entropy, and scaled entropy) also provided valuable insights. Scaled entropy, which normalizes the standard entropy by $\log_2(K)$ (with K being the number of classes), proved especially useful when the number of classes is small, as in binary classification. Although for binary problems scaled entropy behaves similarly to standard entropy, it offers a framework that could be extended to multi-class scenarios.

Furthermore, the feature importance analysis revealed that certain features, such as `cap-surface` and `stem-surface` in the secondary dataset, contributed more significantly to the reduction of impurity. This suggests that these features play a key role in differentiating between the two classes. Such insights could be beneficial for future work in feature selection and model refinement.

In summary, the results demonstrate that the decision tree predictor is highly sensitive to the amount and quality of data. The model struggles with limited, ambiguous data but performs exceptionally well on large, clean datasets. Future work could explore ensemble methods (e.g., random forests) and pruning strategies to further improve performance, especially in scenarios where overfitting is a concern.

6. Conclusion

This project demonstrates a comprehensive implementation of a decision tree predictor for binary classification from scratch. I developed a tree predictor that uses single-feature binary tests—thresholds for numerical attributes and equality tests for categorical ones—to classify mushrooms as poisonous or edible. The implementation supports multiple impurity measures (Gini, entropy, and scaled entropy) and employs several stopping criteria (purity, minimum samples per leaf, and maximum depth).

The experimental results reveal a stark contrast in performance between the small primary dataset and the large secondary dataset. On the primary dataset, with only 173 examples, the 5-fold cross-validation accuracy was around 55%, indicating the challenges of learning robust decision boundaries from limited data. In contrast, on the secondary dataset comprising over 61,000 examples, the model achieved a test accuracy of approximately 95.2% after constraining the maximum depth to 15 to ensure efficient training. These findings underscore the critical impact of dataset size and quality on model performance.

Moreover, the analysis of feature importances provided insights into which attributes play key roles in reducing impurity. For instance, features such as `cap-surface` and `stem-surface` emerged as particularly influential in the secondary dataset, suggesting avenues for further feature selection and model enhancement.

Future work may include extending this model to ensemble methods like random forests, as well as implementing pruning strategies to combat overfitting in deeper trees. Overall, this project not only validates the theoretical concepts covered in the course but also highlights practical challenges and considerations in constructing machine learning models from scratch.

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. 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 any of these practices. 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.