Tree Predictors for Binary Classification

Damir Uvayev

Abstract

This project implements a decision tree predictor for classifying mushrooms as poisonous or edible. Two datasets were used: a primary dataset with 173 examples and a secondary dataset with over 61,000 examples. The model uses single-feature binary tests and supports three impurity measures: Gini, entropy, and scaled entropy. For the primary dataset, 5-fold cross-validation yielded an average accuracy of approximately 55%, while for the secondary dataset an 80/20 train/test split resulted in a test accuracy of about 95%. The report includes pseudocode, numerical examples, and experimental results to illustrate the model's design and performance.

1 Introduction

Decision trees are widely used for classification due to their interpretability and simplicity. In this project, I implement a decision tree classifier from scratch for the binary task of determining whether mushrooms are poisonous or edible.

Key Points

- **Datasets:**
  - **Primary:** 173 examples, 20 features (17 categorical, 3 metrical).
  - **Secondary:** ≈61,000 examples.
- **Model Features:**
  - Uses single-feature binary tests (e.g., testing if `stem-width` ≤ 7.5).
  - Supports impurity measures: Gini, entropy, and scaled entropy.
  - Stops splitting when a node is pure, the number of samples is too low, or a maximum depth is reached.

Example Pseudocode

Below is a simple pseudocode snippet for the prediction process:

```
function predict(x, node):
    if node.is_leaf:
         return node.prediction
    if x[node.split_feature] <= node.split_value:
         return predict(x, node.left_child)
    else:
         return predict(x, node.right_child)
```

Numerical Highlights

- Primary dataset CV accuracy: ∼55%
- Secondary dataset test accuracy: ∼95%
- Example split: `stem-width` threshold of 7.5

In the sections that follow, I describe the dataset and preprocessing, detail the methodology (including hyperparameter tuning and impurity measure comparisons), present experimental results, and discuss the findings.

2. Dataset Description and Preprocessing

This section provides a concise overview of the datasets used and outlines the preprocessing steps with examples and pseudocode.

2.1 Datasets

- **Primary Dataset:** 173 examples, 20 features (17 categorical, 3 metrical).
  - Example: A record may show `cap-diameter = [10, 20]` and `stem-height = [5, 15]`.
- **Secondary Dataset:** ∼61,000 examples (approx. 353 per species).
  - Example: Most metrical features are given as single numerical values (e.g. `cap-diameter = 15.26`).

2.2 Preprocessing Steps

I applied the following steps to prepare the data:

1. **Drop Non-informative Columns:** Remove columns such as `family` and `name`.
2. **Clean Metrical Features:** For features like `cap-diameter`, `stem-height`, and `stem-width`, I use the `clean_metrical_value` function which:
   - Converts string ranges (e.g., "[10, 20]") into their average (i.e. 15.0).
   - Leaves numerical values unchanged.
3. **Clean Categorical Features:** For features stored as list-like strings (e.g. "[x, f]"), the `clean_nominal_value` function extracts the first element (i.e. `x`).
4. **Handle Missing Data:** Replace missing numerical values with `None` and categorical ones with `"unknown"`.
5. **Convert Format:** Convert the final DataFrame into:
   - A list of dictionaries (`X`) for features.
   - A list (`y`) for labels.

2.3 Preprocessing Pseudocode

```
Input: Raw CSV file with columns [family, name, ..., class]
Output: X (list of dicts), y (list of labels)

1. Read CSV file using specified delimiter (e.g., sep=";").
2. Remove columns "family" and "name".
3. For each column (except "class"):
    if column is metrical:
        Apply clean_metrical_value(value)
    else:
        Apply clean_nominal_value(value)
4. Convert cleaned DataFrame to:
    X = list of records (dict format)
    y = list of class labels
5. Return X and y
```

By following these steps, I ensured that both the primary and secondary datasets are standardized, free from missing or ambiguous values, and ready for model training.

3. Methodology

This section outlines the approach used to build the decision tree classifier, emphasizing concrete examples, numerical details, and pseudocode.

3.1 Tree Predictor Design

The decision tree is constructed using two classes: `Node` and `DecisionTree`. Each `Node` stores:

- **is_leaf**: Boolean flag indicating terminal nodes.
- **prediction**: Class label if the node is a leaf.
- **split_feature** and **split_value**: The feature and threshold (or category) used for splitting.
- **left_child** and **right_child**: Pointers to child nodes.
- **samples_count**: Number of training examples at the node.
- **impurity**: Computed impurity value at the node.

The `DecisionTree` class performs the following steps:

1. **Initialization:** Sets hyperparameters:
   - `max_depth`: e.g., {3, 5, 7, None}.
   - `min_samples_in_leaf`: e.g., {1, 5, 10}.
   - `criterion`: "gini", "entropy", or "scaled_entropy".
   - `max_candidates`: e.g., 20.
2. **Fitting:** Infers feature types, stores total examples, and begins recursive tree building.
3. **Recursive Tree Building:** Stops if:
   - All samples in a node belong to one class.
   - Number of samples $<$ `min_samples_in_leaf`.
   - Current depth $\geq$ `max_depth` (if specified).

   Otherwise, it selects the best split.
4. **Best Split Selection:** For each feature:
   - For numeric features, candidate thresholds are computed as midpoints or quantiles.

- For categorical features, each unique value is considered.
- The split with the highest impurity decrease is chosen.

Pseudocode for Tree Prediction:

```
function predict(x, node):
    if node.is_leaf:
         return node.prediction
    if x[node.split_feature] <= node.split_value:
         return predict(x, node.left_child)
    else:
         return predict(x, node.right_child)
```

3.2 Impurity Measures

I implemented three impurity measures:

- **Gini Index:** $G = 1 - \sum p_i^2$. Example: For labels [e,e,p,p,p], $G = 1 - ((2/5)^2 + (3/5)^2) = 1 - (0.16 + 0.36) = 0.48$.
- **Entropy:** $H = -\sum p_i \log_2 p_i$. Example: For the same labels, $H = -(0.4 \log_2 0.4 + 0.6 \log_2 0.6) \approx 0.971$.
- **Scaled Entropy:** $H_{scaled} = H/\log_2(K)$, where K is the number of classes. For binary classification ($K = 2$, $\log_2(2) = 1$), it is equivalent to standard entropy.

3.3 Stopping Criteria

The tree stops growing when:

- **Node Purity:** All examples in a node have the same label.
- **Minimum Samples per Leaf:** A node has fewer samples than a set threshold.
- **Maximum Depth:** The current depth equals or exceeds `max_depth`.

Example: If `min_samples_in_leaf` is set to 5 and a node contains only 3 samples, splitting stops.

3.4 Hyperparameter Tuning and Evaluation

An adaptive evaluation strategy was adopted:

- **Primary Dataset:** Use 5-fold CV with hyperparameter ranges: `max_depth` = {3, 5, 7, None}, `min_samples_in_leaf` = {1, 5, 10}. Best performance was observed with `max_depth = 5` and `min_samples_in_leaf = 5`.
- **Secondary Dataset:** Use an 80%/20% train/test split. Although grid search may favor `max_depth = None`, I fix it at 15 to reduce training time.

Evaluation Metrics:

- Accuracy (0–1 loss)
- Confusion Matrix
- Classification Report (precision, recall, f1-score)

3.5 Implementation Summary

In summary, the decision tree predictor:

- Uses single-feature binary tests (threshold for numeric and equality for categorical).
- Supports Gini, entropy, and scaled entropy as impurity measures.
- Applies three stopping criteria: node purity, minimum samples, and maximum depth.
- Is tuned using grid search with cross-validation (or train/test split) based on dataset size.

4. Experimental Results

4.1 Primary Dataset Results

- **Dataset Size:** 173 examples.
- **Evaluation:** 5-fold cross-validation.
- **Average Accuracy:** Approximately 54.9%.
- **Example Split:**
  - Root node: splits on `stem-width` with threshold 7.5.
  - Subsequent nodes split on `gill-spacing`, `stem-height`, etc.
- **Learned Tree Structure (Excerpt):**

```
Node: if stem-width <= 7.5? (samples: 173)
  Node: if gill-spacing <= d? (samples: 73)
    Node: if stem-height <= 3.25? (samples: 12)
      Leaf: prediction = p (samples: 2)
      Node: if Cap-surface <= d? (samples: 10)
        Leaf: prediction = p (samples: 1)
        Node: if Cap-surface <= l? (samples: 9)
          Leaf: prediction = p (samples: 1)
          Leaf: prediction = e (samples: 8)
    Node: if gill-color <= n? (samples: 61)
      Leaf: prediction = p (samples: 9)
      Node: if stem-height <= 6.75? (samples: 52)
        Node: if stem-height <= 6.25? (samples: 43)
          Leaf: prediction = p (samples: 40)
          Leaf: prediction = e (samples: 3)
        Leaf: prediction = p (samples: 9)
```

- **Feature Importances (Excerpt):**
  - stem-width: 0.0448
  - gill-spacing: 0.0584
  - stem-height: 0.0784

Pseudocode for Cross-Validation Evaluation:

```
for each fold in 5-fold CV:
    split data into training and validation sets
    train decision tree on training set
    predict on validation set
    compute accuracy for fold
average_accuracy = mean(accuracy_scores)
```

4.2 Secondary Dataset Results

- **Dataset Size:** Approximately 61,000 examples.
- **Evaluation:** 80% train / 20% test split.
- **Test Accuracy:** Approximately 95.2%.
- **Hyperparameter Override:** `max_depth` set to 15 (instead of `None`).
- **Example Split:**
  - Root node: splits on `stem-width` with threshold ∼8.31.

- **Confusion Matrix:**

$$\begin{matrix} 5223 & 151 \\ 430 & 6410 \end{matrix}$$

- **Classification Report (Excerpt):**
  - Class `e`: Precision 0.92, Recall 0.97, F1-score 0.95.
  - Class `p`: Precision 0.98, Recall 0.94, F1-score 0.96.
- **Learned Tree Structure (Excerpt):**

```
Node: if stem-width <= 8.31? (samples: 48855)
  Node: if gill-spacing <= d? (samples: 20946)
    Node: if stem-height <= 3.73? (samples: 3430)
      Node: if cap-surface <= s? (samples: 981)
        Leaf: prediction = e (samples: 190)
        Node: if cap-shape <= b? (samples: 791)
          Leaf: prediction = e (samples: 18)
          Node: if cap-shape <= f? (samples: 773)
            Leaf: prediction = e (samples: 7)
            ...
```

- **Feature Importances (Excerpt):**
  - cap-surface: 0.1411
  - stem-surface: 0.0923
  - stem-width: 0.0657

The secondary dataset results confirm that with a large, clean dataset the decision tree achieves high accuracy and effectively leverages key features for classification.

5. Discussion

- **Data Size Impact:**
  - Primary dataset (173 examples) yielded an average CV accuracy of approximately 55%, indicating high variance and difficulty in learning robust patterns.
  - Secondary dataset (over 61,000 examples) achieved a test accuracy of about 95.2%, demonstrating that ample, high-quality data greatly enhances model performance.
- **Model Complexity and Overfitting:**
  - With limited data, the decision tree tends to overfit; many splits might capture noise rather than true signal.

  - For the secondary dataset, the model naturally builds a deeper tree. To avoid excessive complexity, I capped `max_depth` at 15, which balances model complexity with training efficiency.

- **Impurity Measures:**
  - Gini and entropy produced similar splits in binary classification.
  - Scaled entropy, although equivalent to entropy for binary problems, provides a normalized measure beneficial for multi-class settings.
  - Example: For a node with labels [e,e,p,p,p], entropy ≈ 0.971 and scaled entropy (with $K = 2$) remains 0.971.

- **Stopping Criteria Effectiveness:**
  - The three stopping criteria (node purity, minimum samples, maximum depth) successfully prevent uncontrolled growth.
  - Pseudocode for evaluating a stopping condition:

```
if (all samples have same label) or
   (number of samples < min_samples_in_leaf) or
   (current_depth >= max_depth):
    stop splitting and mark node as leaf
```

- **Feature Importance Insights:**
  - In the primary dataset, features like `stem-height` and `Cap-surface` had higher contributions.
  - In the secondary dataset, `cap-surface` and `stem-surface` were particularly influential.

Overall, the discussion indicates that dataset size significantly influences model performance, and that carefully chosen stopping criteria and impurity measures are crucial for balancing complexity and generalization.

6. Conclusion

- The decision tree predictor was implemented from scratch using single-feature binary tests.
- It supports three impurity measures: Gini, entropy, and scaled entropy, along with multiple stopping criteria (node purity, minimum samples per leaf, and maximum depth).
- Experiments on the primary dataset (173 examples) showed limited performance (CV accuracy $\sim$55%) due to the small sample size, while the secondary dataset (over 61,000 examples) achieved high accuracy (test accuracy $\sim$95.2%).

- Hyperparameter tuning (using grid search and cross-validation) and the use of scaled entropy provided valuable insights into model behavior and feature importance.
- The results emphasize the importance of dataset size and quality, as well as the need for controlling model complexity to prevent overfitting.

In summary, the project demonstrates that even a basic decision tree can perform well on large, clean datasets, while its performance is limited on small, noisy datasets. Future work may include extending the model to ensemble methods, such as random forests, and exploring pruning techniques to further enhance generalization.

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.