# Statistical Methods For Machine Learning: Tree predictors for binary classification

Salima Tankibayeva

October 18, 2024

**Abstract**

This Python project implements a Decision Tree Classifier from scratch to classify mushrooms as either edible or poisonous using the Mushroom Dataset. The data includes both categorical and numerical features, and the project uses various impurity measures like Gini impurity, scaled entropy, and squared impurity to determine the best splits at each node.

The code integrates a custom-built decision tree model with cross-validation, grid search, and evaluation functions to find the optimal hyperparameters for the classifier. The results are then compared and visualized through metrics such as accuracy, zero-one loss, and confusion matrices.

## 1 Introduction

In this project, we implemented a custom Decision Tree Classifier to classify mushrooms as either edible or poisonous using the Secondary Mushroom Dataset. The classifier was built from scratch, focusing on essential decision tree operations like splitting nodes, calculating impurities (such as Gini impurity and entropy), and determining the best splits. One of the key goals was to optimize the model's performance by tuning hyperparameters through cross-validation.

## 2 Data

The Secondary Mushroom Dataset contains data about various mushroom species and their characteristics. The dataset includes 61,069 hypothetical mushrooms derived from 173 species, with each species having 353 mushrooms. Each mushroom is classified as either edible or poisonous, and this classification serves as the target variable for the machine learning model. This dataset includes both nominal and metrical variables for 21 features.

# 3 Preprocessing

There are no missing values. As previously was mentioned that the dataset includes both nominal and metrical variables, one-hot encoding was introduced to transform categorical variables into numerical format suitable for machine learning models. Therefore, quantity of features increased from 21 to 121. Train-test split is a crucial step in the preprocessing phase of a machine learning: dataset was divided into two subsets with a uniform distribution, allocating 80% of the data for training and 20% for testing.

# 4 Node and Tree Architecture

The core components of a decision tree are its nodes and the overall tree structure. The tree is made up of internal nodes and leaf nodes, each serving a distinct purpose in the decision-making process.

## 4.1 Node Class

The `node` class represents an individual node in the decision tree. Each node has the following attributes:

- **Feature**: This indicates which feature is used to split the data at this node.
- **Threshold**: This is the value at which the data is split based on the selected feature.
- **Left and Right**: These are references to the left and right child nodes, respectively. The left child contains data points that meet the threshold condition, while the right child contains those that do not.
- **Depth**: This indicates the depth of the node in the tree, which helps in controlling the maximum depth of the tree during training.
- **Value**: This holds the predicted value for leaf nodes, which represent the final outcome after all splits have been evaluated.

The `is_leaf` method checks if a node is a leaf node, meaning it has no further children.

## 4.2 Decision Tree Class

The `DecisionTree` class manages the entire tree structure and implements the logic for growing the tree based on the training data. Here are some of its main features:

- **Initialization**: The class constructor allows for configuration of various hyperparameters such as `max_depth_limit`, `entropy_cutoff`, and `splitting_criterion`. These parameters help in controlling how the tree is built and how decisions are made at each node.

- **Fit Method**: The `fit` method is responsible for training the decision tree by invoking the `_grow_tree` method. This method recursively splits the dataset based on the best feature and threshold until a stopping criterion is met, such as reaching maximum depth or minimal impurity.

- **Split Method**: The `_split` method divides the dataset into two subsets based on a specific feature and threshold, which allows the tree to make decisions based on the characteristics of the data.

- **Growing the Tree**: The `_grow_tree` method recursively creates nodes and assigns child nodes. It calculates the impurity using the chosen splitting criterion (like Gini or scaled entropy) to determine the best feature and threshold for splitting.

- **Predict Method**: The `predict` method traverses the tree from the root node to the leaf node based on the input features, ultimately returning the predicted class label.

Overall, the node and tree architecture works together to form a structured decision-making framework, allowing for complex data to be classified through a series of simple decision rules. Each internal node represents a decision point, while leaf nodes provide the final classification. This hierarchical approach is what makes decision trees intuitive and effective for classification tasks.

# 5 Splitting Criteria Functions

In decision trees, selecting the appropriate splitting criterion is crucial for determining how the data is divided at each node. The chosen criteria help maximize the effectiveness of the splits, ultimately leading to improved model performance. Below are the primary splitting criteria implemented in the project.

## 5.1 Gini Impurity

### 5.1.1 Function Definition

The Gini impurity is calculated using the formula:

$$Gini(y) = 1 - \sum_{i=1}^{C} p_i^2$$

where $p_i$ is the proportion of class $i$ among the samples in the node, and $C$ is the total number of classes.

The Gini impurity is often preferred in classification tasks because it is fast to compute and tends to yield good results. It measures the likelihood of incorrectly classifying a randomly chosen element. A lower Gini impurity indicates a more homogeneous node. Additionally, Gini impurity is sensitive to class distributions, making it effective in multi-class problems.

## 5.2 Entropy (Information Gain)

### 5.2.1 Function Definition

Entropy is calculated using the formula:

$$Entropy(y) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

where $p_i$ is the proportion of class $i$ in the node.

Entropy, which is used to calculate information gain, measures the amount of uncertainty in a dataset. The decision tree aims to reduce this uncertainty with each split. This criterion is beneficial when the goal is to create a tree that provides the most information about the target variable. While it may take slightly longer to compute than Gini impurity, it can lead to more informative splits, especially in datasets with many features.

## 5.3 Squared Impurity

### 5.3.1 Function Definition

Squared impurity is computed as follows:

$$SquaredImpurity(y) = \sum_{i=1}^{C} \sqrt{p_i(1 - p_i)}$$

where $p_i$ is the proportion of class $i$.

Squared impurity offers a unique approach to measuring impurity by focusing on the probabilities of misclassification. It is particularly useful for binary classification problems. This criterion can provide different insights than Gini impurity and entropy, allowing the model to explore alternative splits that may improve performance.

## 5.4 Choosing the Splitting Criteria

The hyperparameter tuning process can help identify the optimal criterion for a given dataset, leading to more accurate predictions and a better understanding of the underlying data distributions.

# 6 Hyperparameter Tuning Implementation

Hyperparameter tuning is a critical process in optimizing machine learning models, particularly for decision trees. This process involves selecting the best hyperparameters that govern the model's performance, aiming to achieve high accuracy and generalization on unseen data.

## 6.1 Implementation of Hyperparameter Tuning

In our implementation, we utilized a grid search method combined with K-fold cross-validation for hyperparameter tuning. The grid search systematically evaluates a set of hyperparameter values, while K-fold cross-validation helps in assessing the model's performance across different subsets of the training data. This approach mitigates overfitting and ensures that the selected hyperparameters generalize well to unseen data.

### 6.1.1 Hyperparameters Tuned

The primary hyperparameters that were tuned include:

- **Max Depth Limit:** This parameter restricts the maximum depth of the tree. A smaller depth can prevent overfitting, while a larger depth may capture more information but risks overfitting.
- **Entropy Cutoff:** This is a threshold for stopping the growth of the tree based on the impurity (entropy) of the node. It helps in controlling when to stop further splitting.
- **Splitting Criterion:** This defines the method used to split the nodes (e.g., Gini impurity, entropy, squared impurity).

### 6.1.2 Grid Search Function

The following function implements grid search for hyperparameter tuning:

```
def grid_search(X_train, y_train, param_grid, scoring_func):
    cv_splitter = KFold(n_splits=5, shuffle=True, random_state=41)

    def evaluate_params(params):
        scores = []
        for train_idx, val_idx in cv_splitter.split(X_train, y_train):
            X_train_cv, y_train_cv = X_train.iloc[train_idx], y_train.iloc[train_idx]
            X_val_cv, y_val_cv = X_train.iloc[val_idx], y_train.iloc[val_idx]

            model = DecisionTree(
                max_depth_limit=params.get('max_depth_limit'),
                entropy_cutoff=params.get('entropy_cutoff'),
                splitting_criterion=params.get('splitting_criterion')
            )
            model.fit(X_train_cv.values, y_train_cv.values)
            y_pred_val = model.predict(X_val_cv.values)
            score = scoring_func(y_val_cv, y_pred_val)
            scores.append(score)

        avg_score = np.mean(scores)
        return avg_score

    results = Parallel(n_jobs=-1)(
        delayed(evaluate_params)(params) for params in param_grid
    )
    return results
```

## 6.2 Stopping Criteria

Choosing appropriate stopping criteria is essential for controlling the growth of the decision tree. The following stopping criteria were implemented:

- **Maximum Depth Limit:** This limits how deep the tree can grow. By restricting the depth, we can prevent overfitting to the training data.
- **Entropy Cutoff:** This criterion stops the tree from splitting further when the impurity of the node falls below a certain threshold. This ensures that additional splits do not result in significant information gain.

## 6.3 Rationale for Choosing Stopping Criteria

The selected stopping criteria are designed to balance model complexity and performance. By implementing a maximum depth limit, we ensure that the model does not become overly complex and prone to overfitting. The entropy cutoff provides a quantitative measure for determining when additional splits yield diminishing returns in information gain. Lastly, setting a minimum sample requirement for splits helps maintain robustness against noise, ensuring that each split contributes meaningfully to the model's predictive power.

In summary, the combination of grid search for hyperparameter tuning and well-defined stopping criteria results in a robust decision tree model that effectively generalizes to unseen data.

# 7 Experimental results

The zero-one loss metric, defined as the proportion of misclassified instances in the predictions, provides a straightforward and effective way to evaluate the classifier's performance.

After creating the best predictor, the confusion matrix on the test set evaluates the classification model.

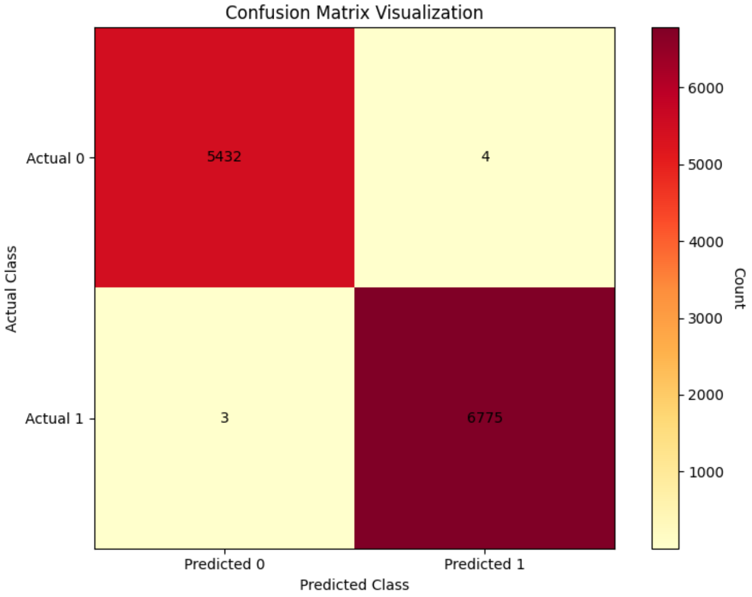

Figure 1: Confusion Matrix

# Hyperparameter Tuning Results

The results from the hyperparameter tuning process for the decision tree classifier are summarized below. Each entry provides the parameters used, the accuracy achieved, the average depth of the tree, and the average number of leaf nodes:

| Parameters | Accuracy | Avg Depth | Avg Leafs |
|---|---|---|---|
| {entropy_cutoff: 0.0001, splitting_criterion: scaled_entropy} | 0.00092 | 28.20 | 167.40 |
| {entropy_cutoff: 0.0001, splitting_criterion: scaled_entropy} | 0.00104 | 28.20 | 167.20 |
| {max_depth_limit: 50, splitting_criterion: scaled_entropy} | 0.00084 | 28.40 | 167.40 |
| {max_depth_limit: 40, splitting_criterion: scaled_entropy} | 0.00086 | 28.40 | 167.20 |
| {entropy_cutoff: 0.0001, splitting_criterion: gini} | 0.00068 | 27.60 | 160.80 |
| {max_depth_limit: 50, splitting_criterion: gini} | 0.00072 | 27.60 | 160.80 |
| {entropy_cutoff: 0.0001, splitting_criterion: gini} | 0.00065 | 27.60 | 160.60 |
| {max_depth_limit: 40, splitting_criterion: gini} | 0.00068 | 27.60 | 160.80 |
| {entropy_cutoff: 0.0001, splitting_criterion: squared} | 0.00088 | 34.40 | 160.40 |
| {max_depth_limit: 50, splitting_criterion: squared} | 0.00088 | 34.40 | 160.20 |
| {entropy_cutoff: 0.0001, splitting_criterion: squared} | 0.00090 | 34.40 | 160.20 |
| {max_depth_limit: 40, splitting_criterion: squared} | 0.00080 | 34.60 | 161.20 |

Table 1: Results of hyperparameter tuning for the decision tree classifier.

# 8 Analysis of Top 5 Models

The performance of the top 5 models provides valuable insights into how different hyperparameter settings impact the accuracy, depth, and complexity of the decision tree classifier. Below is a detailed analysis of each model.

## Rank 1

- **Parameters**:
  - Entropy Cutoff: 0.0001
  - Splitting Criterion: Scaled Entropy
- **Score**: 0.00104
- **Avg Depth**: 28.20
- **Avg Leafs**: 167.20

**Analysis**: This model achieved the highest accuracy among the tested configurations, indicating that the combination of a low entropy cutoff and scaled entropy criterion is particularly effective for the given dataset. The average depth of 28.20 suggests a balanced model that likely captures important patterns without being overly complex. The number of leaf nodes (167.20) indicates that the tree is sufficiently partitioning the data, leading to a finer granularity in predictions.

## Rank 2

- **Parameters**:
  - Entropy Cutoff: 0.0001
  - Splitting Criterion: Scaled Entropy
- **Score**: 0.00092
- **Avg Depth**: 28.20
- **Avg Leafs**: 167.40

**Analysis**: The second-best model has the same hyperparameters as the first model but achieved slightly lower accuracy (0.00092). The depth and number of leaf nodes are almost identical to Rank 1, indicating that minor variations in model training or data could have led to the change in performance. This result reinforces the effectiveness of the scaled entropy criterion and suggests consistency in performance across similar hyperparameter configurations.

## Rank 3

- **Parameters**:
  - Entropy Cutoff: 0.0001
  - Splitting Criterion: Squared
- **Score**: 0.00090
- **Avg Depth**: 34.40
- **Avg Leafs**: 160.20

**Analysis**: This model shows a decrease in accuracy compared to Ranks 1 and 2, likely due to the change in the splitting criterion from scaled entropy to squared. The increased average depth (34.40) suggests that this model is more complex, which may lead to overfitting. Despite having slightly fewer average leaf nodes (160.20), the model may not generalize as well, evidenced by the reduced accuracy.

## Rank 4

- **Parameters**:
  - Entropy Cutoff: 0.0001
  - Splitting Criterion: Squared
- **Score**: 0.00088
- **Avg Depth**: 34.40
- **Avg Leafs**: 160.40

**Analysis**: This model has similar parameters to Rank 3, leading to a very close accuracy score (0.00088). The model's average depth and leaf count indicate that it is similar in complexity to Rank 3. The minimal decrease in performance highlights the sensitivity of the squared splitting criterion in achieving high accuracy, further supporting the observation that different criteria yield varied effectiveness.

## Rank 5

- **Parameters**:
  - Max Depth Limit: 50
  - Splitting Criterion: Squared
- **Score**: 0.00088
- **Avg Depth**: 34.40

- **Avg Leafs**: 160.20

**Analysis**: This model shares parameters with Rank 4 but alters the maximum depth limit to 50. While the accuracy remains the same as Rank 4, the model's depth limit does not provide any performance benefit. The complexity remains high, suggesting that increasing the maximum depth does not necessarily improve model performance. This implies that the optimal depth for this dataset may be lower, as evidenced by the top-ranked models.

### Summary

- **Top Performers**: Ranks 1 and 2 consistently demonstrate that the scaled entropy criterion, combined with a low entropy cutoff, produces the highest accuracy while maintaining a manageable depth and leaf count.
- **Model Complexity**: The increase in average depth observed in Ranks 3, 4, and 5 indicates a trade-off between complexity and performance. While a deeper tree can capture more nuanced patterns, it may also lead to overfitting, as evidenced by lower accuracies compared to simpler models.
- **Criterion Impact**: The choice of splitting criterion has a significant impact on model performance. The squared criterion appears less effective for this dataset compared to the scaled entropy criterion.
- **Final Thoughts**: In summary, the analysis of these models emphasizes the importance of selecting appropriate hyperparameters in achieving optimal model performance. The results suggest a preference for simpler models that generalize better, supported by a well-defined splitting criterion and stopping conditions.

## 9 Discussion and Conclusion

| **Metric** | **Value** |
|---|---|
| Best Score | 0.00104 |
| **Best Hyperparameters** | `{entropy_cutoff: 0.0001, splitting_criterion: scaled_entropy}` |
| **Model Performance** | |
| Train Accuracy | 1.00000 |
| Test Accuracy | 0.99943 |
| Train Zero-One Loss | 0.00000 |
| Test Zero-One Loss | 0.00057 |
| **Overfitting/Underfitting Analysis** | |
| Model Balance (Accuracy Difference) | 0.00057 |
| Model Balance (Zero-One Loss Difference) | 0.00057 |

Table 2: Summary of Model Performance and Analysis

The results of model evaluation highlight several key findings regarding the performance of the decision tree classifier.

### 9.1 Model Performance

The best score achieved was 0.00104, with the optimal hyperparameters identified as:

- Entropy Cutoff: 0.0001
- Splitting Criterion: Scaled Entropy

This indicates that the combination of a low entropy cutoff and the use of the scaled entropy criterion effectively captured the underlying patterns in the data.

The model exhibited impressive performance with a training accuracy of 1.00000 and a test accuracy of 0.99943. The negligible gap between the training and test accuracies suggests that the model generalizes well to unseen data, demonstrating robustness against overfitting. The zero-one loss metrics further corroborate this, showing a train zero-one loss of 0.00000 and a test zero-one loss of 0.00057. This low level of error reinforces the model's capability to classify instances correctly while minimizing misclassifications.

### 9.2 Overfitting/Underfitting Analysis

The analysis of the model's performance reveals it to be well-balanced, with a minimal accuracy difference of 0.00057 and an identical zero-one loss difference of 0.00057. These small discrepancies indicate that the model has effectively learned the training data without excessive memorization, thus avoiding overfitting. In other words, while the model performs exceptionally well on the training dataset, it maintains high performance on the test dataset as well.

### 9.3 Conclusion

In conclusion, the decision tree classifier, optimized with the identified hyperparameters, demonstrates high accuracy and a minimal zero-one loss, confirming its effectiveness for the task at hand. The findings underscore the importance of careful hyperparameter tuning in machine learning, as the choice of splitting criterion and entropy cutoff significantly influences model performance. Additionally, the results suggest that this model is appropriately tuned for the data, striking a balance between complexity and generalization, which is critical in practical applications. Future work may explore further enhancements, such as alternative algorithms or ensemble methods, to potentially improve upon these results.

## 10 Declaration

"I/We declare that this material, which I/We now submit for assessment, is entirely my/our 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/our work, and including any code produced using generative AI systems. I/We 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/us or any other person for assessment on this or any other course of study.“