Malika Makhmudova
Matriculation Number: 34003A

Decision Tree Predictors for Mushroom Classification

1. Introduction

The objective of this project is to build a decision tree predictor from scratch to determine whether mushrooms are poisonous or edible based on their features. The decision tree will use single-feature binary tests at each internal node and be evaluated using different splitting and stopping criteria. The project involves implementing the decision tree model, training it, evaluating its performance, and tuning its hyperparameters.

2. Dataset Overview

The dataset contains various features of mushrooms, such as their shape, color, odor, etc. The target variable is binary, indicating whether a mushroom is poisonous (`class_p`) or edible (`class_e`). The dataset was one-hot encoded, converting categorical features into numerical binary variables, so the decision tree can process them.

Data Preprocessing

1. **Loading the Data:** The mushroom dataset was loaded using `pandas`.
2. **Encoding the Data:** Since decision trees work well with numerical data, categorical features were converted into binary columns using one-hot encoding.
3. **Splitting the Data:** The dataset was split into training and testing sets using an 80/20 split to evaluate the model's performance on unseen data.

3. Decision Tree Implementation

3.1 TreeNode Class

The `TreeNode` class represents a single node in the decision tree. Each node can either be an internal node, which decides on a split, or a leaf node, which holds the final prediction.

- **Attributes:**
  - `feature`: The feature used to split the data at this node. (what the question is about, like the color of the mushroom)
  - `threshold`: The value used to decide how to split the data. (the value we compare the feature to. For example, "Is the color red?".)
  - `left`: The left child node (samples that satisfy the split condition).
  - `right`: The right child node (samples that do not satisfy the split condition).

    - `value`: The class label if this node is a leaf. (the final decision if the node is a leaf (end of the branch).)
- **Methods:**
    - `is_leaf()`: Checks if the node is a leaf node (i.e., it holds a class label). (checks if we've reached the end of a branch and can make a decision)

3.2 TreeModel Class

The `TreeModel` class is the main structure of the decision tree, containing all the logic for training, splitting, and predicting.

- **Attributes:**
    - `max_depth`: The maximum depth of the tree. (how many questions it can ask before deciding)
    - `max_leaf_nodes`: The maximum number of leaf nodes. (limits how many end points (decisions) the tree can have)
    - `entropy_threshold`: A threshold for entropy to decide when to stop splitting. (helps decide when to stop growing the tree)
    - `split_function`: The criterion for evaluating splits (`gini`, `scaled_entropy`, `squared`).
    - `min_samples_split`: The minimum number of samples required to split a node.
    - `root`: The root node of the tree.
    - `leaf_count`: The number of leaf nodes in the tree.
    - `depth`: The current depth of the tree.
    - `feature_names`: The names of the features used in the dataset.
- **Methods:**
    - `fit(X, y)`: Trains the decision tree on the provided dataset (`X`) and labels (`y`). (where the tree learns from the data, growing by asking the best questions)
    - `_grow_tree(X, y, depth)`: Recursively grows the tree by finding the best splits.
    - `_gain(y, left_mask, right_mask, parent_criterion)`: Computes the information gain from a potential split.
    - `_scaled_entropy(y)`, `_gini_impurity(y)`, `_squared_impurity(y)`: Methods for calculating different impurity criteria.
    - `predict(X)`: Predicts the class labels for new data. (after learning, the tree can now predict the safety of new mushroom)
    - `visualize_tree()`: Visualises the decision tree using Graphviz.

4. Training and Evaluation

4.1 Training the Model

The decision tree was trained using the `TreeModel` class with different parameters to observe how the tree behaves under various conditions.

- **Splitting Criteria:**
  - **Gini Impurity, Scaled Entropy, Squared Impurity:** Measures how often a randomly chosen element would be incorrectly classified. Shows how mixed the data is at a node. The goal is to make the groups as pure(lower values) as possible (all safe or all poisonous).
- **Stopping Criteria:**
  - **Maximum Depth:** Limits the depth of the tree. (stops if the tree gets too deep)
  - **Maximum Leaf Nodes:** Stops if there are too many decision points.
  - **Entropy Threshold:** Stops splitting when the entropy is below a certain level.(if the data is pure enough)
  - **Minimum Samples Split:** Prevents further splitting if a node has too few samples.(if there aren't enough samples to justify splitting)

4.2 Evaluation Metrics

The decision tree was evaluated using the following metrics:

- **Training Accuracy:** Measures how well the tree performs on the training data.
- **Validation Accuracy:** Measures the performance on the test data, which was not seen during training.

4.3 Results

The decision tree achieved a training accuracy of **98.03%** and a validation accuracy of **97.88%**, indicating that the tree performs well on both training and unseen data.

Confusion Matrix:

- A confusion matrix was plotted to visualize the performance. It shows the counts of true positives, true negatives, false positives, and false negatives, helping to understand the types of errors the model makes.

5. Hyperparameter Tuning

Hyperparameter tuning was performed using `GridSearchCV` to find the best combination of parameters that yield the highest accuracy. This tool tries out different settings (like how deep the tree should go) and finds the best combination.

- **Parameters Grid:**
  - `max_depth`: The maximum depth of the tree (None, 10, 20, 30).
  - `min_samples_split`: The minimum number of samples to split a node (2, 5, 10).

    - `entropy_threshold`: The threshold for stopping based on entropy (0.1, 0.2, 0.3).

- **Data Splitting:**
    - Split the data into two parts: one for training the tree and one for testing it using train_test_split, ensuring that the class distribution is maintained across both sets (stratify=y)

- **Initial Model Configuration:**
    - The TreeModel is configured with initial or default parameters before hyperparameter tuning begins.

- **Setting Up Grid Search:**
    - GridSearchCV is initialized with the decision tree model, the parameter grid, and other settings like cross-validation (cv=5) and scoring method (accuracy). The .fit() method is called to train and evaluate the model across all parameter combinations.
- **Retrieving the Best Model Parameters:**
    - After training, the best-performing parameters are printed, along with the highest cross-validation score, indicating the most effective model configuration.

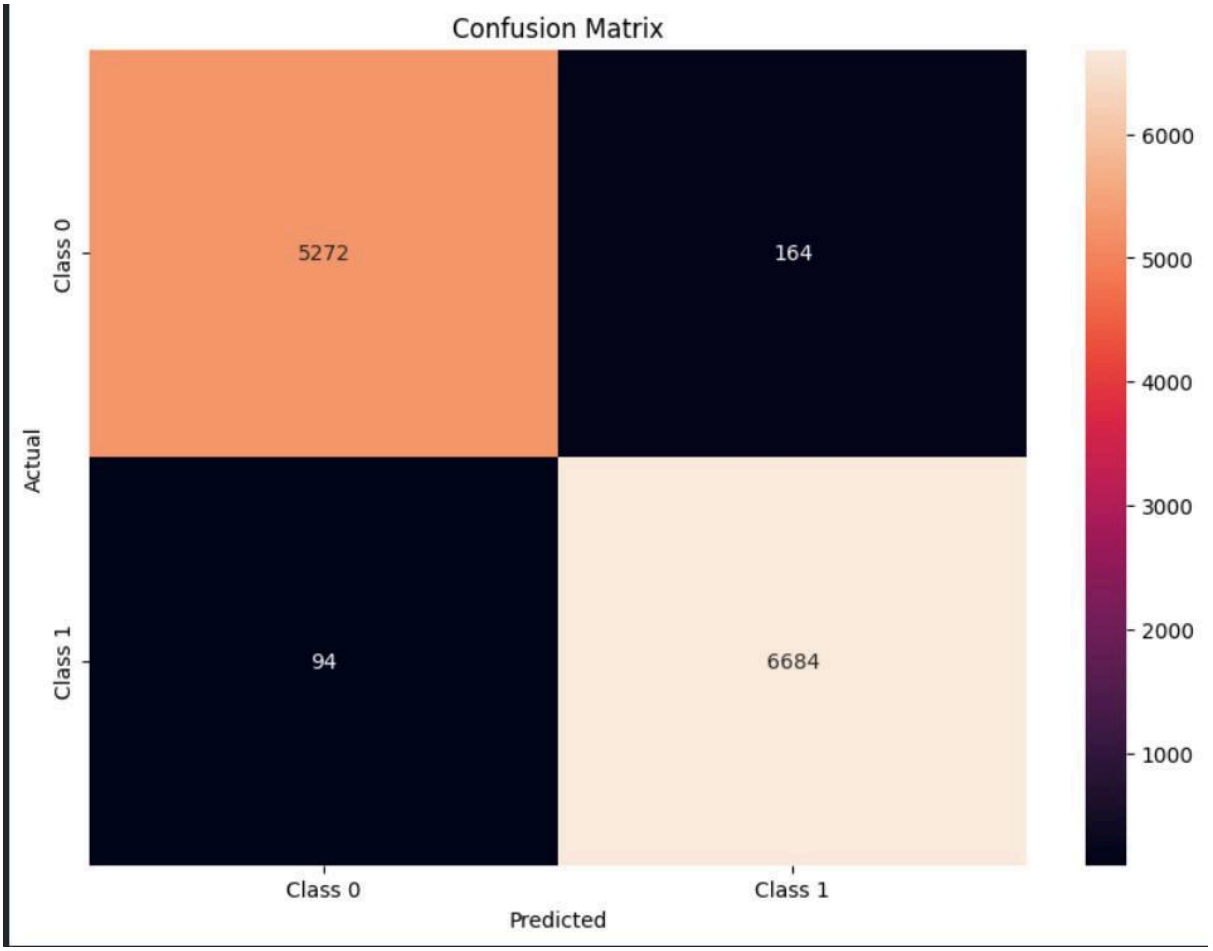
Best Hyperparameters:

The grid search identified that the best parameters for this dataset are:

- `entropy_threshold`: 0.1
- `max_depth`: None
- `min_samples_split`: 2

The best cross-validation score achieved was **98%**.

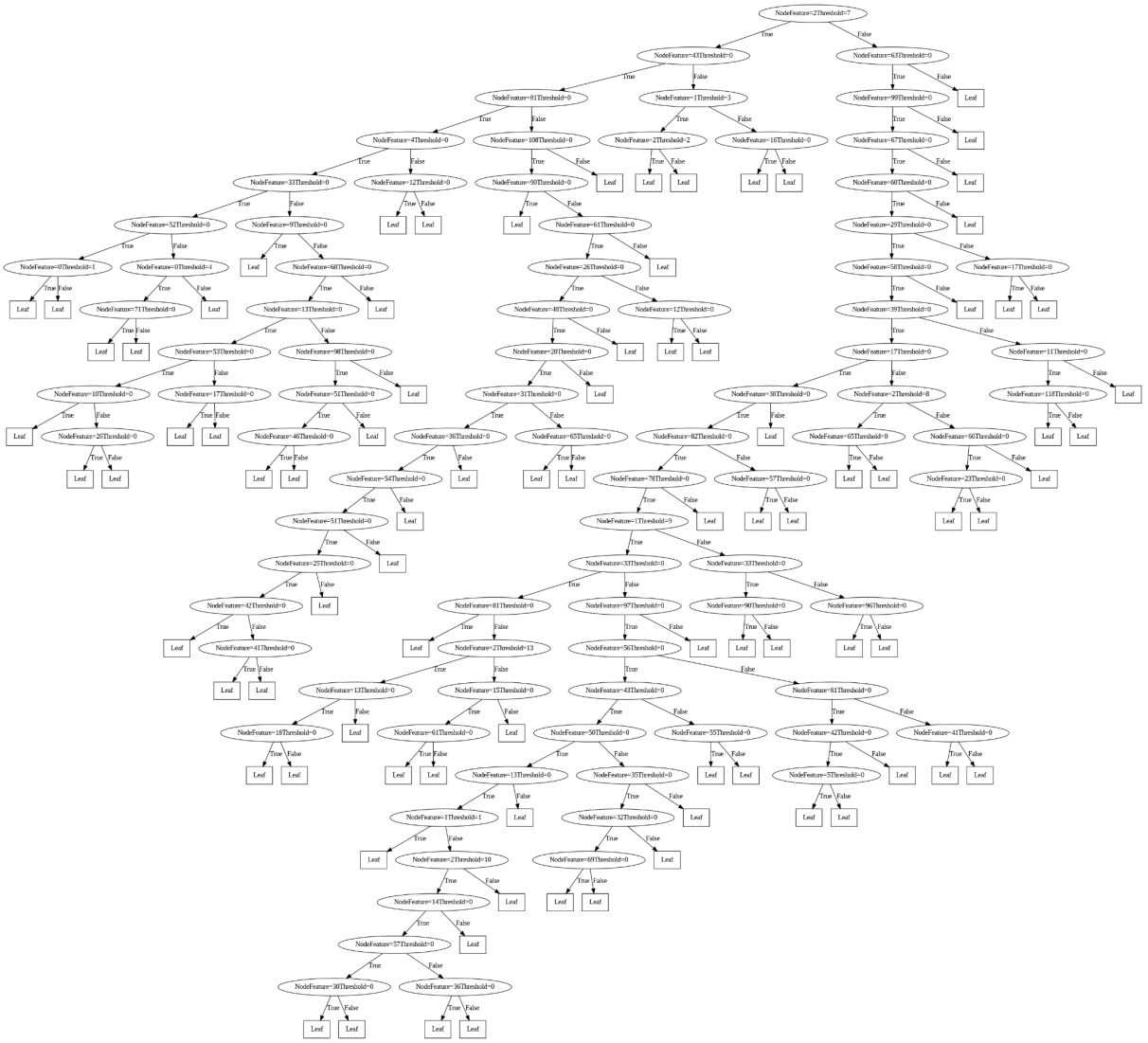
Malika Makhmudova
Matriculation Number: 34003A



* The confusion matrix shows that the tree rarely makes mistakes. The heatmap confirms that our tree is doing very well

- **Final Model Training:**
  - The model is re-trained using the best parameters identified by GridSearchCV, this time on the entire training dataset.
- **Model Evaluation:**
  - The model's performance is evaluated on the test set, with accuracy and a confusion matrix being calculated to assess how well the model predicts the test data.

Malika Makhmudova
Matriculation Number: 34003A



6. Discussion

6.1 Model Performance

The decision tree performed exceptionally well on both the training and test sets, with minimal difference between the two accuracies, suggesting that the model generalizes well to new data. The high accuracy indicates that the model effectively learned the patterns in the data to distinguish between poisonous and edible mushrooms.

6.2 Overfitting and Underfitting

- **Overfitting:** There is no significant overfitting in this model as the training and validation accuracies are very close.
- **Underfitting:** The model does not show signs of underfitting, as it achieved high accuracy on both sets.

6.3 Model Tuning and Optimization

The hyperparameter tuning process helped in fine-tuning the model to achieve the best performance. The use of `GridSearchCV` was critical in systematically exploring different combinations of parameters and selecting the most effective ones.

7. Conclusion

This project involved building a decision tree from scratch to classify mushrooms as poisonous or edible. The tree was implemented with customizable splitting and stopping criteria, and its performance was evaluated using accuracy and confusion matrices. Hyperparameter tuning further optimized the model, achieving a high accuracy of nearly 98%. The project demonstrated the effectiveness of decision trees for binary classification tasks and the importance of careful parameter selection to balance the trade-off between complexity and generalization.

8. Future Work

- **Pruning:** Implementing tree pruning methods could further enhance the model by removing unnecessary branches.
- **Random Forest:** Extending the current implementation to create a random forest could improve performance by combining multiple decision trees.