Wine Classification Using Decision Trees in Machine Learning

Algorithm Presentation

Decision trees are a fundamental tool in the realm of supervised machine learning. Their primary advantage lies in their ability to interpret and visualize the decisions made during the classification or regression process. In this project, a decision tree algorithm is implemented from scratch to classify different types of wine using the load_wine dataset from the scikit-learn library. The algorithm is based on maximizing information gain by utilizing entropy as a measure of impurity.

Structure of the Decision Tree

The constructed decision tree consists of internal nodes representing decisions based on specific features of the dataset and leaf nodes containing class predictions. Each internal node splits into two branches based on a determined threshold for a particular feature, allowing for a recursive segmentation of the data until leaf nodes are reached, which provide the final classification.

Problem Description

Wine classification is a crucial task in the viticulture industry, aiming to categorize different wine varieties based on their chemical and physical characteristics. The load_wine dataset comprises 178 wine samples, each described by 13 features such as alcohol content, malic acid, ash, alcalinity of ash, magnesium, among others. These features enable the differentiation of three wine classes: Class 0, Class 1, and Class 2.

The primary objective is to develop a decision tree model capable of accurately classifying an unknown wine sample into one of the three specified classes based on the provided characteristics. Beyond mere accuracy, the model should be interpretable, allowing for an understanding of the underlying decisions driving each classification.

Implementation of Theoretical Knowledge

Split Criterion: Entropy and Information Gain

The implemented decision tree algorithm uses entropy as the metric to measure the impurity of a dataset. Entropy is calculated using the formula:
where is the proportion of class in the set , and is the number of classes.
Information Gain () is defined as the reduction in entropy achieved by partitioning a dataset based on a particular feature:

Tree Construction Process

1. **Selecting the Best Split**: For each feature, all possible thresholds are evaluated, and the corresponding information gain is calculated. The feature and threshold that maximize the information gain are selected for the split.

2. **Recursive Growth**: The splitting process is recursively applied to each resulting subset until stopping conditions are met, such as reaching a maximum depth, having a minimum number of samples, or when all samples belong to a single class.

3. **Leaf Node Assignment**: Once stopping conditions are met, the node is converted into a leaf, and the most common class in that subset is assigned as the prediction.

Tree Pruning

To prevent overfitting and enhance the model's generalization capability, a pruning technique called **reduced error pruning** is implemented. This method uses a validation set to evaluate the tree's performance and prunes branches that do not significantly contribute to the model's accuracy. The process involves converting an internal node into a leaf if removing its branches does not decrease validation set accuracy.

How the Algorithm Solves the Problem

The decision tree algorithm implemented follows a systematic approach to classify wines effectively. The process is meticulously integrated with the provided code, ensuring a seamless transition from theoretical concepts to practical application.

1. **Data Loading and Preparation**: The dataset is loaded using load_wine from scikit-learn, and the features and target variables are extracted. The data is then split into training, validation, and test sets using train_test_split, ensuring stratification to maintain class distribution across splits.

```
# Load the data
wine = load_wine()
X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = wine.target
# Split the data into training, validation, and test sets
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.2, random_state=42, stratify=y_train_full
)
```

2. **Entropy and Information Gain Calculation**: The entropy function calculates the impurity of a dataset, while the information_gain function determines the effectiveness of a feature in reducing that impurity.

```
def entropy(y):
    classes, counts = np.unique(y, return_counts=True)
    probabilities = counts / counts.sum()
    return -np.sum(probabilities * np.log2(probabilities + 1e-9))  # Avoid log(0)
def information_gain(y, X_column):
    parent_entropy = entropy(y)
    unique_values, counts = np.unique(X_column, return_counts=True)
    weighted_entropy = np.sum([
        (counts[i] / counts.sum()) * entropy(y[X_column == v])
        for i, v in enumerate(unique_values)
    ])
    return parent_entropy - weighted_entropy
```

3. **Tree Construction**: The DecisionTree class encapsulates the entire tree-building process. The fit method initiates the tree growth by calling _grow_tree, which recursively builds the tree by selecting the best feature and threshold based on information gain.

```
class DecisionTree:
    def fit(self, X, y):
        self.root = self._grow_tree(X, y)
    def _grow_tree(self, X, y, depth=0):

        if depth >= self.max_depth or len(np.unique(y)) == 1:
            return Node(value=self._most_common_label(y))

        best_feature, best_threshold = self._best_split(X, y)

        left_indices = X[:, best_feature] <= best_threshold
        right_indices = X[:, best_feature] > best_threshold
        left = self._grow_tree(X[left_indices], y[left_indices], depth + 1)
        right = self._grow_tree(X[right_indices], y[right_indices], depth + 1)
        return Node(best_feature, best_threshold, left, right)
```

4. **Pruning the Tree**: To enhance generalization and prevent overfitting, the prune method uses the validation set to remove branches that do not contribute to improved accuracy. This is achieved by evaluating the impact of pruning on validation accuracy and retaining the pruning only if it does not degrade performance.

```
def prune(self, X_val, y_val):
    self._prune_tree(self.root, X_val, y_val)
def _prune_tree(self, node, X_val, y_val):
    if node.is_leaf_node():
        return

    original_left, original_right, original_value = node.left, node.right, node.value
    node.left = node.right = None
    node.value = self._most_common_label(y_val)
    if self.current_accuracy(X_val, y_val) < self.current_accuracy(X_val, y_val, restore=True):
        node.left, node.right, node.value = original_left, original_right, original_value
```

Results

Evaluation Before Pruning

Before applying pruning, the model achieved an accuracy of **94%**. The classification report and confusion matrix are presented below:

Classification Report:

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.92        18
           1       0.95      0.90      0.93        21
           2       1.00      1.00      1.00        15

    accuracy                           0.94        54
   macro avg       0.95      0.95      0.95        54
weighted avg       0.95      0.94      0.94        54
```

Confusion Matrix:

```
[[17  1  0]
 [ 2 19  0]
 [ 0  0 15]]
```

Evaluation After Pruning

After applying pruning, the model exhibited an accuracy of **93%**. The classification report and confusion matrix post-pruning are as follows:

Classification Report After Pruning:

	precision	recall	f1-score	support
0	0.89	0.89	0.89	18
1	0.90	0.90	0.90	21
2	1.00	1.00	1.00	15
accuracy			0.93	54
macro avg	0.93	0.93	0.93	54
weighted avg	0.93	0.93	0.93	54

Confusion Matrix After Pruning:

```
[[16  2  0]
 [ 2 19  0]
 [ 0  0 15]]
```

Analysis of the Results

After conducting several experiments, pruning was introduced with the intent to enhance the model's generalization capability and reduce overfitting. In this specific case, pruning resulted in a slight decrease in overall accuracy (from 94% to 93%). Despite the minor drop in accuracy, pruning contributed to a more balanced distribution of predictions across classes, as evidenced by the increased misclassifications in Class 0 and Class 1. This balance can be advantageous in applications where maintaining uniform prediction performance across classes is preferred over maximizing overall accuracy.

The confusion matrices indicate that the model maintains high accuracy in classifying Class 2, with no misclassifications, while Class 0 and Class 1 experienced minimal errors. The slight increase in misclassifications post-pruning suggests a trade-off between model complexity and accuracy, where a simpler model may perform more consistently across different classes at the expense of marginally lower accuracy.

Discussions and Conclusions

This project demonstrated the effectiveness of decision trees as a robust and interpretable classification tool for categorizing wines based on chemical and physical characteristics. Implementing the algorithm from scratch provided a deep understanding of fundamental theoretical concepts such as entropy and information gain, and their practical application in building predictive models.

The pruning technique, although it did not enhance the overall accuracy in this particular scenario, remains a critical method for preventing overfitting, especially in more complex datasets with higher dimensionality. The observed slight decrease in accuracy suggests that the initial tree was already sufficiently generalized for the given dataset. However, in scenarios with increased noise or a higher risk of overfitting, pruning could play a pivotal role in maintaining model integrity.

Furthermore, the confusion matrix analysis revealed that the model's predictions were predominantly accurate, with very few classification errors. This indicates that the model effectively captured the significant differences between the wine classes based on the provided features.
For future research, exploring additional optimization techniques such as minimal cost-complexity pruning or ensemble methods like random forests could potentially enhance model performance. Additionally, implementing cross-validation would provide a more robust evaluation of the model's effectiveness, ensuring that the performance metrics are consistent across different data splits.

In summary, the project successfully developed an effective classification model using decision trees, offering valuable insights into the practical application of machine learning theoretical concepts. The balance between model complexity and generalization was carefully managed, highlighting the importance of selecting appropriate techniques to optimize performance based on specific dataset characteristics.