Exploring Decision Trees in Machine Learning

Student Name: R. Meda

Student ID: 23005003

Introduction

One of the easiest and most comprehensible machine-learning methods is decision trees. They serve as the basis for several sophisticated algorithms, such as Gradient Boosted Trees and Random Forests. To create a tree-like structure for prediction, a decision tree divides the dataset into subgroups according to feature values. The foundations of decision trees are covered in this lesson, along with how they operate and how to adjust them for best results. We'll use code and graphics to demonstrate these ideas using the well-known Iris dataset.

Understanding Decision Trees

Based on feature values, a decision tree divides data into "branches" that eventually lead to judgments at the "leaves." The purpose of these splits is to optimize the purity of the resultant subsets, which is often assessed using metrics such as entropy (used in information gain) or Gini impurity. Decision trees may be used to represent non-linear connections because of their flexibility, but if they are not adequately regularized, they may overfit.

Important settings to adjust:

1. Max Depth: Limits the number of splits, controlling overfitting.

2. Min Samples Split: Minimum number of samples required to split a node.

3. Criterion: Metric to measure the quality of a split (e.g., Gini or Entropy).

Dataset

One of the most well-known and often utilized datasets in statistics and machine learning is the Iris dataset. It was first used in 1936 to illustrate linear discriminant analysis by British statistician and biologist Ronald A. Fisher. It has now evolved into a common dataset for algorithm testing and evaluation in tasks involving pattern recognition and classification.

Dataset Overview

The Iris dataset contains **150 samples** of iris flowers, divided into three distinct classes, with **50 samples per class**. The dataset's classes represent three species of iris flowers:

1. **Setosa**: Easily separable from the other two species.
2. **Versicolour**: Overlaps with Virginica, making it harder to distinguish.
3. **Virginica**: Like Versicolour, presenting a non-linear decision boundary.

Each sample in the dataset is described by **four numerical features**:

1. **Sepal Length** (cm): The length of the sepal (outer part of the flower).
2. **Sepal Width** (cm): The width of the sepal.
3. **Petal Length** (cm): The length of the petal (inner part of the flower).
4. **Petal Width** (cm): The width of the petal.

These features are continuous and measured in centimetres, making them suitable for a variety of algorithms, including decision trees, which can handle both numerical and categorical data.

Dataset Structure

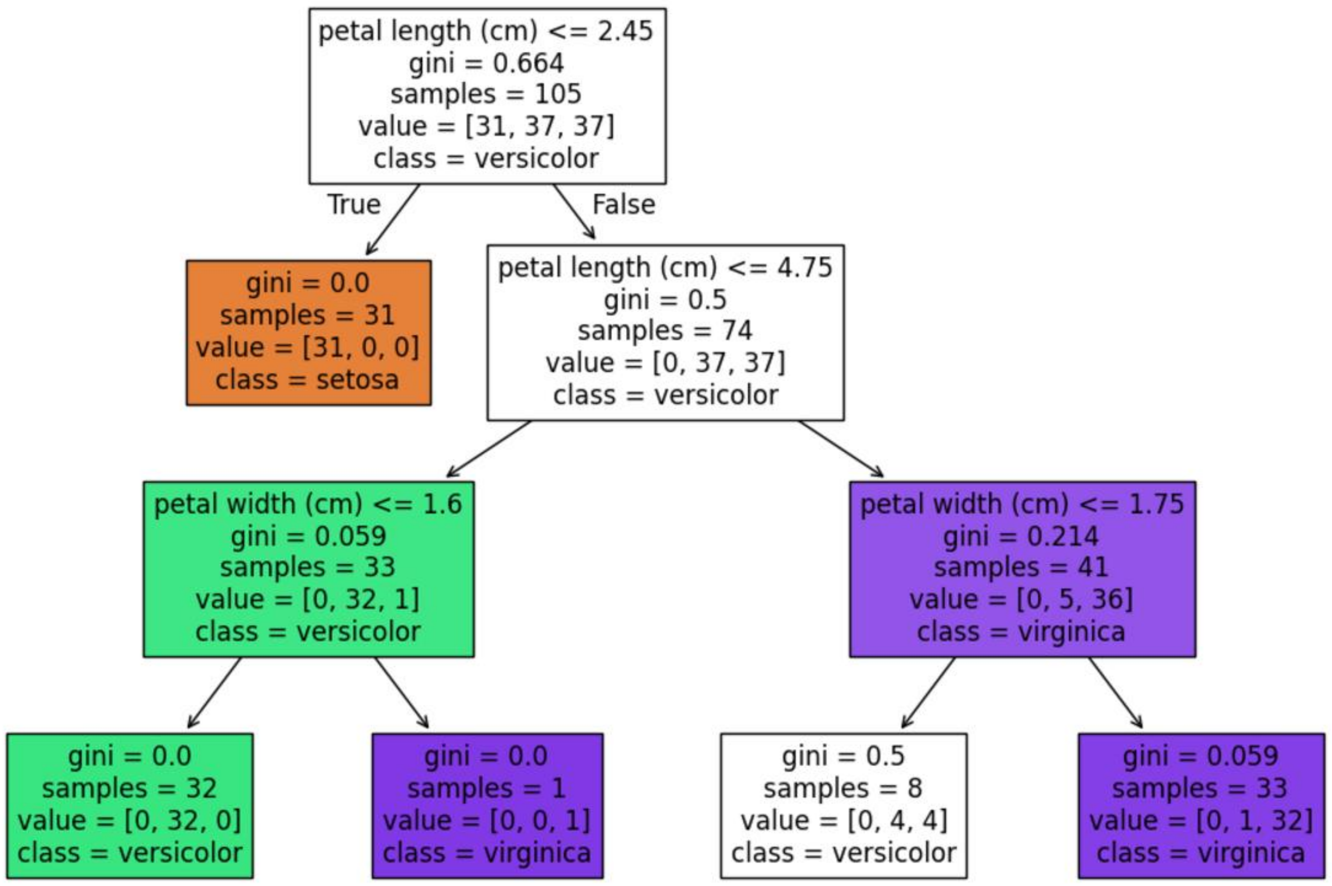
The dataset is organized as follows:

Feature	Description	Example Value
Sepal Length (cm)	Length of the sepal	5.1
Sepal Width (cm)	Width of the sepal	3.5
Petal Length (cm)	Length of the petal	1.4
Petal Width (cm)	Width of the petal	0.2
Species	Class label (Setosa, etc.)	Setosa

Code Implementation

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# Predict on test set
y_pred = clf.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```

This code uses the Iris dataset to create a Decision Tree Classifier. To prevent overfitting, it restricts the tree depth to three and trains the model using the Gini impurity criteria. It uses accuracy, a classification report, and a confusion matrix to assess the model's performance after making predictions on the test set. After that, a visualization of the decision tree shows how it divides the data according to characteristics in order to categorize iris species.



Analysis

1. **Accuracy**: The decision tree achieves high accuracy due to the simplicity of the Iris dataset. For real-world problems, more tuning might be required.
2. **Classification Report**: Precision, recall, and F1-score provide insights into the performance for each class.
3. **Tree Visualization**: The plot highlights the decision-making process, with splits based on features like petal length.

Advantages of Decision Trees

- **Interpretability**: Easy to visualize and understand.
- **Non-linear Decision Boundaries**: Handles complex datasets without requiring feature scaling.
- **Versatility**: Can be used for classification and regression tasks.

Limitations

- **Overfitting**: Tends to overfit on small datasets without proper tuning.
- **Instability**: Small changes in data can significantly alter the tree structure.

Conclusion

Using the Iris dataset, this lesson illustrates the principles of decision trees and how to apply them. A strong yet understandable method for machine learning is provided by decision trees. To avoid overfitting, regularization strategies are essential, such as restricting tree depth. You may use decision trees for a variety of jobs or expand your knowledge to more complex tree-based techniques like Random Forests by comprehending these ideas.