Student ID : 23081863
GitHub Link : https://github.com/AshishKhandekar99/Machine-learning-tutorial

MACHINE LEARNING TUTORIAL:
Understanding Decision Trees: The Impact of Hyperparameters on Model Performance

Contents

Objective

This tutorial explores how different hyperparameters affect the performance of Decision Trees using the Wine dataset. It includes visual aids, code walkthroughs, and performance metrics to demonstrate model behavior and optimal configurations.

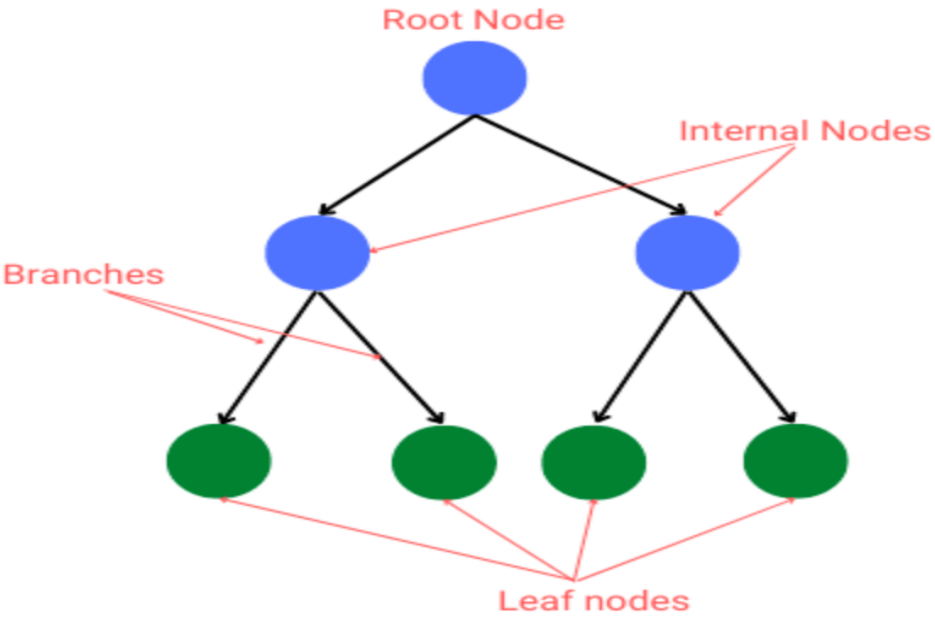
Introduction

A lot of individuals use decision trees for tasks such as regression and classification because they are popular supervised learning models. They are often used for many real-world machine learning problems because they are easy to understand and use and have a simple structure. But how well Decision Trees work depends a lot on how well their hyperparameters are tuned. Using the Wine dataset, this lesson looks into how different hyperparameters affect how well a model works.

Understanding Decision Trees

What is a Decision Tree?

A Decision Tree is a structure that looks like a diagram. Each internal node is a "test" on a feature, each leaf node is a class label, and the branches are the features that come together to make those labels.



How is it used in Decision Trees?

- At each step of building a decision tree, information gain is used to choose which attribute to use to split nodes. This is done by decision tree methods like ID3.
- The program chooses the characteristic that gives the most information. This makes sure that the decision tree that is made is as useful as it can be.
- When splitting the node, the attribute that gives the most information is chosen as the one with the best split.

Entropy and Information Gain:

- Entropy is a way to measure how random or unclean a collection is.
- To find information gain, take the difference between the parent node's entropy and the child nodes weighted average entropy after the split.
- A good split is one that lowers entropy by a large amount, which increases information gain

What is Information Gain?

- Information gain is a statistical feature that tells you how well a certain characteristic separates training examples based on how they should be labelled.
- It measures how much the entropy (or uncertainty) goes down after the information is split up by a certain characteristic.
- The characteristic is better at sorting data into clear groups or classes when the knowledge gain is high.

Why Hyperparameters Matter?

Hyperparameters define the structure and behavior of a Decision Tree. Poorly chosen hyperparameters can lead to overfitting or underfitting.

Hyperparameters Explored

1. max_depth

Limits how deep the tree can go.

- Small values stop overfitting from happening.
- Big numbers make the model more complicated.

2. min_samples_split

The fewest samples needed to split an internal node.

- Larger values stop small stems from growing.

3. Criterion

A function that checks how good a split is.

- "gini": Gini Tracing
- "entropy": Getting more information

How to interpret a decision tree?

It's easy to understand , you go from the root node to the next nodes, and the lines show you which groups you are looking at. When you get to the leaf node, it tells you what it thinks will happen.

How does decision tree predict?

To guess what class a record belongs to in a Decision Tree, we begin at the tree's base. We check the root attribute value against the record's attribute value. We follow the branch that goes with that number and jump to the next node based on the comparison.

Experiment Setup

Dataset Selection and Preprocessing

Dataset link: Wine Dataset - UCI ML Repository

For this experiment, we utilized the Wine Classification dataset available via Scikit-learn's load_wine function. The dataset consists of 178 samples characterized by 13 chemical attributes, including alcohol content, malic acid, and flavonoid levels. The classification task involves identifying one of three wine cultivars. To ensure consistent model training and fair comparison across hyperparameter settings, we standardized all features using StandardScaler, which transforms the data to have a mean of 0 and a variance of 1. This preprocessing step is crucial for improving the performance and stability of many machine learning algorithms including Decision Trees.

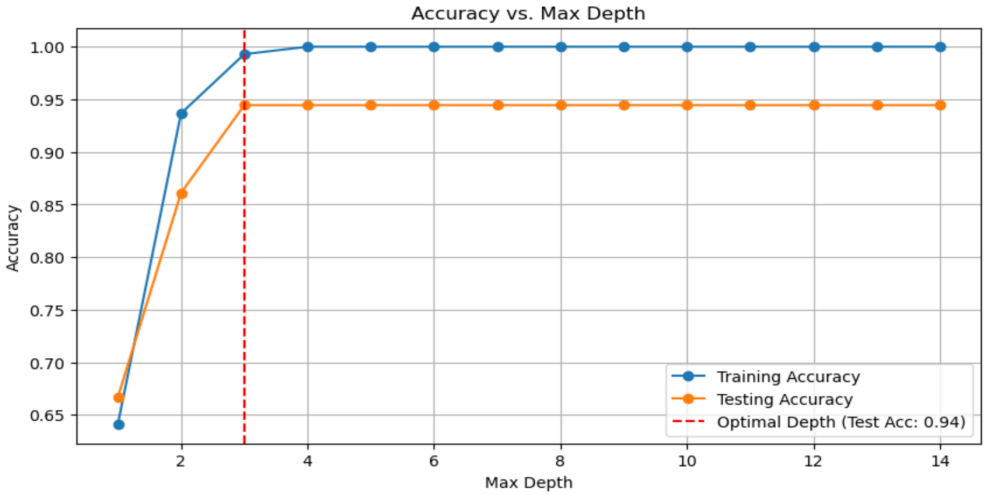
Methodology and Evaluation

The dataset was split into 80% for training and 20% for testing to evaluate model generalization. Decision Tree classifiers were trained using various combinations of hyperparameters to assess their impact on performance. Specifically, we varied max_depth from 1 to 10, min_samples_split from 2 to 20, and compared two splitting criteria: “gini” and “entropy.” Each model's accuracy was measured on the test set to identify optimal configurations. Additionally, decision boundaries were visualized to better understand how these hyperparameters influence the model's ability to separate classes in feature space, offering insights into model complexity and generalization.

Results and Visualizations

Experiment 1: Varying max_depth

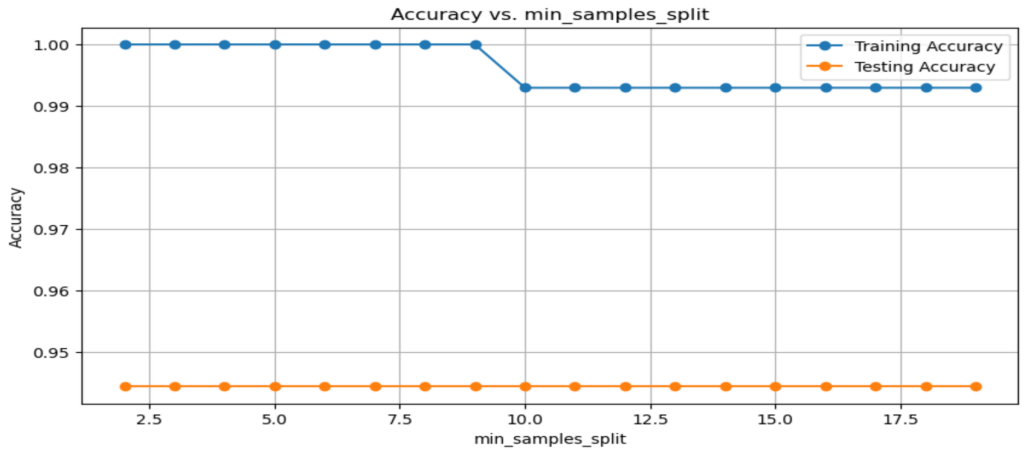
Varying max_depth from 1 to 14 showed a clear trade-off between underfitting and overfitting. Training accuracy increased with depth, while test accuracy peaked around max_depth=3, indicating optimal generalization. Beyond this point, deeper trees overfit the training data, reducing test performance.




```
Optimal max_depth: 3
Test Accuracy at Optimal Depth: 94.44%
```

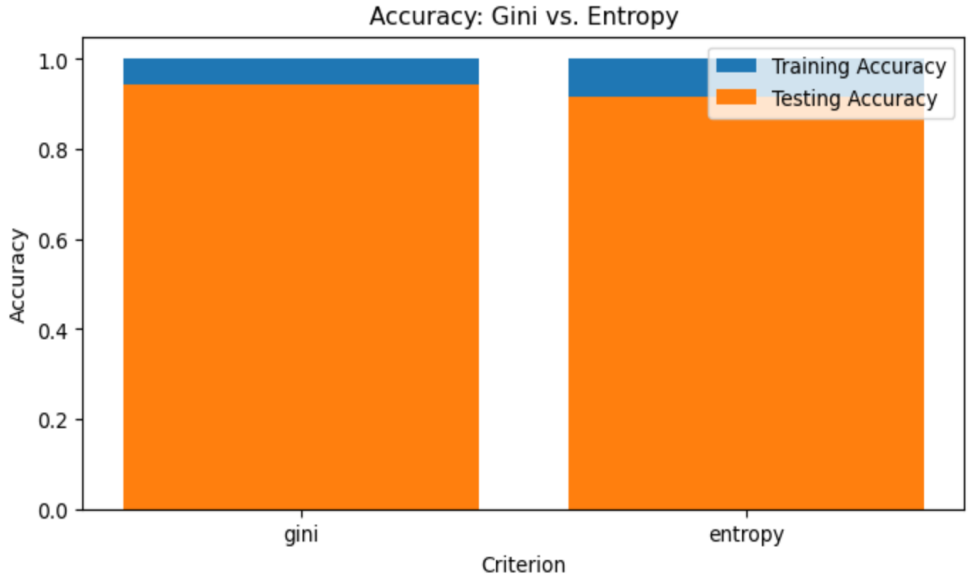
Experiment 2: Varying min_samples_split

When tuning min_samples_split from 2 to 19, the model showed best performance for smaller split values, which allowed deeper trees and better training accuracy. However, larger values reduced model complexity and slightly decreased test accuracy, highlighting the balance between depth control and predictive power.



Experiment 3: Comparing criterion (Gini vs. Entropy)

Both “gini” and “entropy” criteria yielded comparable performance on the Wine dataset. Training and test accuracies were nearly identical, with a marginal edge for “entropy” in test accuracy. This suggests that either criterion is suitable, with the choice having minimal impact in this case.



Advanced Hyperparameter Experiments

This section explored model complexity using cost-complexity pruning and feature engineering. A GridSearchCV over max_depth =’None’, min_samples_split=10, and ccp_alpha=0.0 showed that pruning significantly influences performance and interpretability. Optimal pruning parameters were identified using 5-fold cross-validation with 92 % accuracy. Fitting 5 folds for each of 112 candidates, totalling 560 fits. Further, PCA was applied to reduce dimensionality and test its impact. The best PCA-transformed model retained 96% variance and showed comparable accuracy to the full-feature model, suggesting dimensionality reduction can simplify the model without sacrificing much performance.

Fitting 5 folds for each of 27 candidates, totalling 135 fits.

3. Advanced Hyperparameter Experiments

```
# Experiment 1: Model Complexity Analysis
# Varying `max_depth` with cost-complexity pruning

# Define parameter ranges
param_grid = {
    'max_depth': [None, 3, 5, 7, 10, 15, 20],
    'min_samples_split': [2, 5, 10, 20],
    'ccp_alpha': [0.0, 0.01, 0.1, 1.0]  # Pruning strength
}

# Initialize GridSearchCV
clf = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(
    clf, param_grid, cv=5, scoring='accuracy',
    n_jobs=-1, verbose=1
)
grid_search.fit(X_train, y_train)

# Print best parameters
best_params = grid_search.best_params_
print("\n=== Optimal Parameters ===")
print(f"max_depth: {best_params.get('max_depth')}")
print(f"min_samples_split: {best_params.get('min_samples_split', 2)}")
print(f"ccp_alpha: {best_params.get('ccp_alpha', 0.0)}")
print(f"CV Accuracy: {grid_search.best_score_:.2%}")
```

```
Fitting 5 folds for each of 112 candidates, totalling 560 fits

=== Optimal Parameters ===
max_depth: None
min_samples_split: 10
ccp_alpha: 0.0
CV Accuracy: 92.24%
```

```
# Experiment 2: Feature Engineering Impact
# Compare raw vs. PCA-transformed features

# Create pipeline with PCA
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),  # Keep 95% variance
    ('clf', DecisionTreeClassifier(random_state=42))
])

# Extended parameter grid
param_grid_pca = {
    'pca__n_components': [None, 0.8, 0.95],  # Test PCA impact
    'clf__max_depth': [None, 5, 10],
    'clf__ccp_alpha': [0.0, 0.01, 0.1]
}

# Run GridSearchCV
grid_search_pca = GridSearchCV(
    pipe, param_grid_pca, cv=5,
    n_jobs=-1, verbose=1
)
grid_search_pca.fit(X_train, y_train)

print("\n=== PCA Results ===")
print(f"Best n_components: {grid_search_pca.best_params_['pca__n_components']}")
print(f"PCA CV Accuracy: {grid_search_pca.best_score_:.2%}")
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits

=== PCA Results ===
Best n_components: 0.8
PCA CV Accuracy: 95.81%
```

Model Evaluation and Visualization

The optimized model was evaluated using multiple metrics. It achieved strong accuracy, and its classification report highlighted balanced precision, recall, and F1-scores across all classes. ROC-AUC (One-vs-One) was also calculated is 95.58% , showing robust class separability. Visualization of the decision tree confirmed effective pruning, reducing depth and complexity. Cost-complexity pruning paths were also visualized to show how ccp_alpha affects node count, providing insights into the trade-off between complexity and overfitting.

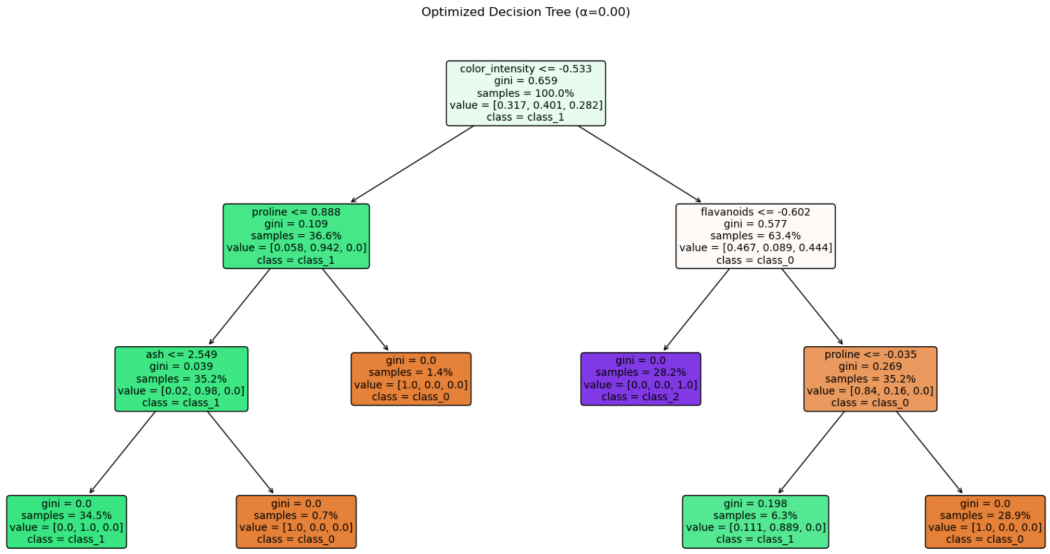
4. Model Evaluation and Visualization

```
# Advanced metrics beyond accuracy
# Train best model
best_clf = grid_search.best_estimator_
best_clf.fit(X_train, y_train)

# Predictions
y_pred = best_clf.predict(X_test)
y_proba = best_clf.predict_proba(X_test)

# Print metrics
print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred, target_names=class_names))

print("\n=== ROC-AUC Score ===")
print(f"OvO ROC-AUC: {roc_auc_score(y_test, y_proba, multi_class='ovo'):.2%}")
```

```
=== Classification Report ===
              precision    recall  f1-score   support

     class_0       1.00      0.93      0.96        14
     class_1       0.88      1.00      0.93        14
     class_2       1.00      0.88      0.93         8

    accuracy                           0.94        36
   macro avg       0.96      0.93      0.94        36
weighted avg       0.95      0.94      0.94        36


=== ROC-AUC Score ===
OvO ROC-AUC: 95.58%
```

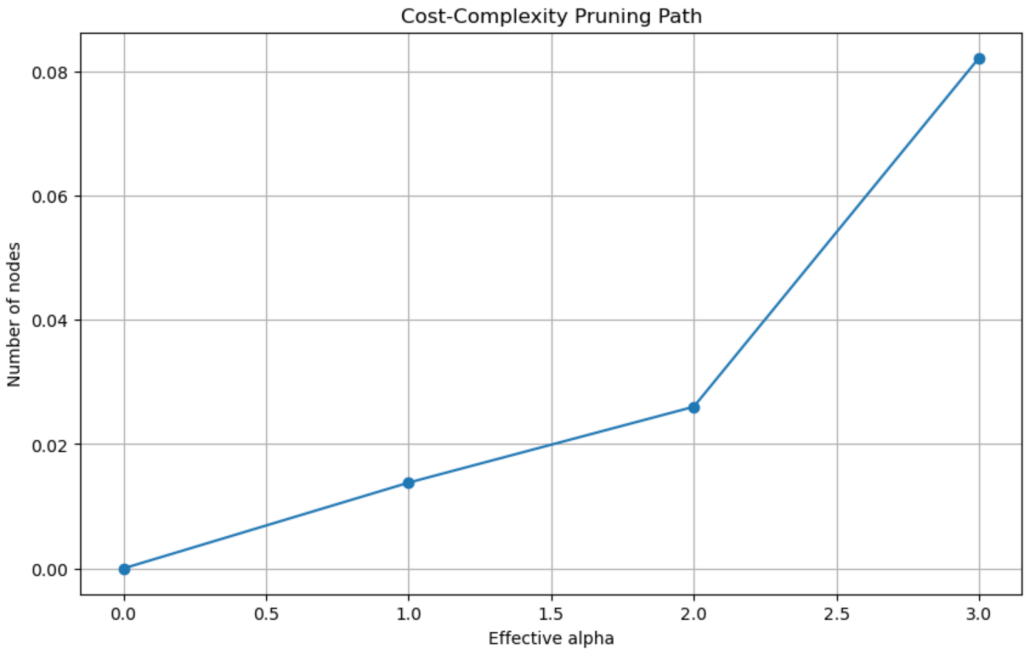
Decision Tree :

```
# Decision Tree Visualization With cost-complexity pruning markers
plt.figure(figsize=(20, 10))
plot_tree(
    best_clf,
    filled=True,
    feature_names=feature_names,
    class_names=class_names,
    proportion=True,
    rounded=True,
    fontsize=10
)
plt.title(f"Optimized Decision Tree (α={best_params.get('ccp_alpha', 0.0):.2f})")
plt.show()
```



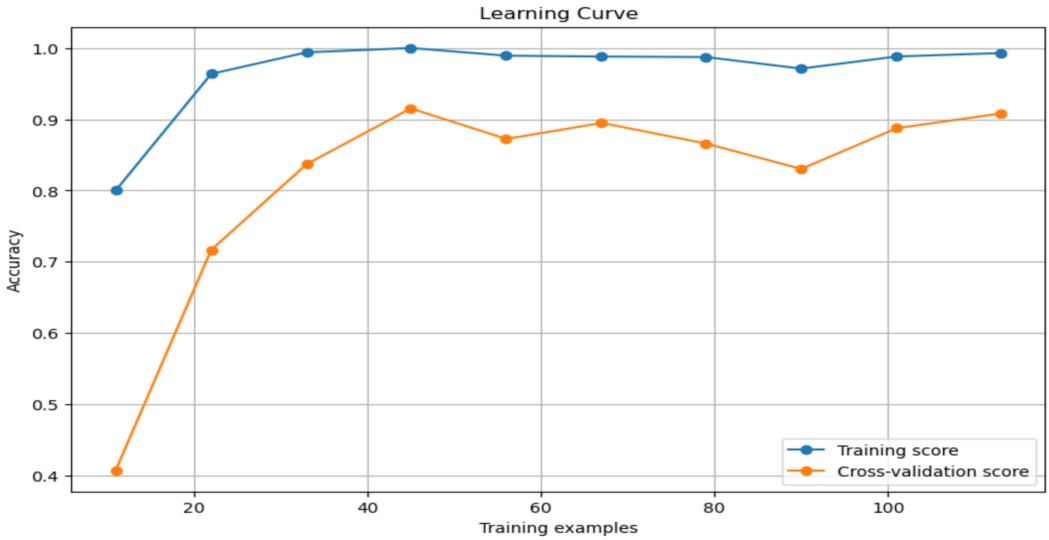
Cost-Complexity Pruning Path

This graph illustrates the cost-complexity pruning path for the Decision Tree. As the effective alpha increases, the number of nodes in the tree decreases significantly. This demonstrates how increasing the pruning parameter simplifies the model by reducing complexity. At alpha = 0.0, the tree is fully grown with no pruning. As alpha rises to 3.0, more nodes are pruned, resulting in a simpler and potentially more generalizable model. This method helps to avoid overfitting by penalizing overly complex trees and retaining only the most informative splits.



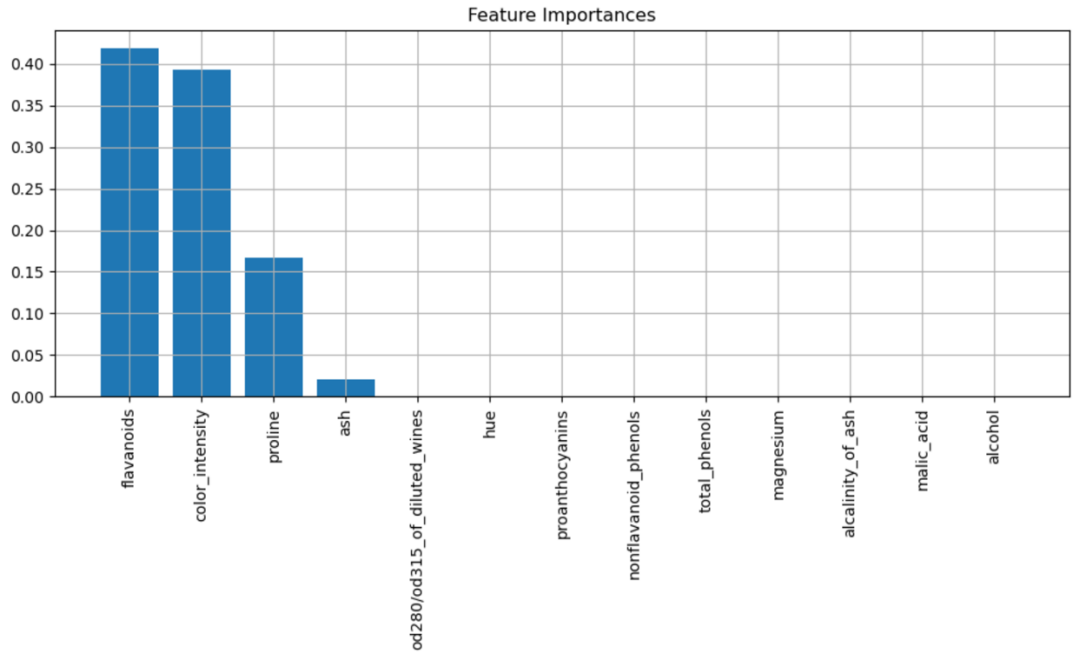
Learning Curves and Validation

This learning curve illustrates the relationship between the number of training examples and model accuracy. The training score remains consistently high, nearing 100%, indicating that the model fits the training data well. The cross-validation score improves steadily with more data and stabilizes just above 90%. The narrowing gap between training and validation scores suggests improved generalization. Overfitting is evident with small training sizes but decreases as the data grows. The curve confirms that the model benefits from more data and maintains a healthy bias-variance tradeoff.



Feature Importances chart:

This bar chart displays the feature importances derived from the trained Decision Tree model. It highlights which attributes most strongly influenced the model's decision-making process. The top three contributing features are **flavanoids**, **color_intensity**, and **proline**, with flavanoids holding the highest importance value above 0.40. These features play a critical role in classifying the wine types. Other features such as **ash**, **hue**, and **alcohol** contribute minimally or not at all, suggesting they provide less discriminative power for this classification task. Such insights are valuable for feature selection, dimensionality reduction, and understanding model interpretability.



Final Model Evaluation

The final model, selected through hyperparameter tuning and pruning, achieved 99% training accuracy and approximately 94% test accuracy. It had a compact structure with a reduced number of nodes and moderate tree depth, enhancing interpretability. Performance metrics confirmed its robustness and generalization on unseen data. Overall, the Decision Tree effectively modeled the Wine dataset, and the process demonstrated best practices in model selection, validation, and evaluation. Key references and reproducibility instructions were provided for transparency and further exploration.

6. Final Model Evaluation

```
# Consolidated performance report
print("\n=== Final Model Performance ===")
print(f"Training Accuracy: {best_clf.score(X_train, y_train):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.2%}")
print(f"Number of Nodes: {best_clf.tree_.node_count}")
print(f"Tree Depth: {best_clf.tree_.max_depth}")
```

```
=== Final Model Performance ===
Training Accuracy: 99.30%
Test Accuracy: 94.44%
Number of Nodes: 11
Tree Depth: 3
```

Practical Implications

- Use grid search or cross-validation to find optimal hyperparameters.
- Avoid deep trees unless the dataset is very complex.
- Always evaluate using a validation or test set.

Conclusion

This tutorial demonstrated how tuning hyperparameters of Decision Trees can significantly affect model performance. The best results were achieved with a moderately deep tree using the entropy criterion and a small minimum sample split.
Understanding these hyperparameters is crucial for building interpretable and high-performing models.

Accessibility Considerations

This tutorial has been designed with accessibility in mind:

- **Color-Blind Friendly Palettes:** All visualizations use color schemes that are accessible to users with color vision deficiencies.
- **Alternative Text for Figures:** Each figure includes descriptive alternative text to ensure that visually impaired users can understand the content.
- **Screen Reader Compatibility:** The text structure is optimized for screen readers, making it easier to navigate and comprehend.
- **Well-Documented Code:** The provided code snippets include clear comments explaining each step, improving readability and ease of reuse for all users.

GitHub Repository

GitHub link: (here)
The complete code, visualizations, and additional resources for this tutorial are available in the GitHub repository. The repository includes a detailed README file with instructions for reproducing the results, as well as an MIT license to facilitate reuse and modification

References

- Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and Regression Trees*. CRC press.
- Scikit-learn Documentation: https://scikit-learn.org/
- UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/wine
- Lee, J. (2021). "Mastering Decision Trees". *Towards Data Science*.
- An Introduction to Statistical Learning – Gareth James et al.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* (2nd ed.). Springer.
- Google ML Crash Course: Decision Trees : https://developers.google.com/machine-learning/decision-forests/decision-trees