CS471: Introduction to Artificial Intelligence
Decision Tree
Malcolm Zartman

09/26/2025

Question 1: Decision Tree

In this assignment, you will work with the Iris dataset that was used in R. A Fisher's paper. You can also find the dataset in the UCI Machine Learning Repository. You can directly load the dataset using sklearn:

```
from sklearn.datasets import load_iris
data = load_iris()
```

Tasks:

1. Include a basic description of the data (what are the features and labels) (1 point)
   Write in your own words of what the classification task is and why a decision tree is a reasonable model to try for this data. (1 point)
2. Split the data into training, validation, and testing sets. (1 point)
3. Fit a decision tree on the training dataset. (1 point)
4. Tune at least 2 hyperparameters in the decision tree model (https://ken-hoffman.medium.com/decision-tree-hyperparameters-explained-49158ee1268e) based on the performance on the validation set or using cross-validation. One hyperparameter has to be max_depth and the other one is your choice.

Generate plot of hyperparameter values vs performance metric (plots are mandatory). (4 points)

5. Train the model using optimal hyperparameters (found in step 5) on the train + validation data. Test it on test data and generate a classification report (1 point)
6. Inspect the model by visualizing and interpreting the results (1 point)

Colab link:
https://colab.research.google.com/drive/1BzhOeu8sfytwY1YBq5Yxf80Sp5OX8ASb?usp=sharing

1. Include a basic description of the data (what are the features and labels) (1 point) Write in your own words of what the classification task is and why a decision tree is a reasonable model to try for this data. (1 point)

   In the data, the features include physical measurements of a flower: [sepal length (cm), sepal width (cm), pedal length (cm), pedal width (cm)]

   In the data, the labels include the species of flower: [setosa, veriscolor, virginica]

   The classification task at hand is to build a predictive model that can take 4 measurements of a new, unknown flower, and classify it as one of the species of flowers previously defined above. The decision tree model is a reasonable model to try on this data since only one of the three classes in the dataset is linear, and decision trees are great with handling complex, non-linear data.

2. Split the data into training, validation, and testing sets. (1 point)

Below I am splitting the data into training (60%), validation (30%), and testing (30%) sets. I feel like using numpy is better for getting the data, but I'm using pandas because of my familiarity with it.

```
import pandas as pd
from sklearn.model_selection import train_test_split

```

```
# Convert to a pandas DataFrame for easier handling
X = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
y = pd.Series(iris_data.target, name='species')

# Split data into development (80%) and a testing set (20%)
# The dev set will be split again into training and validation sets.
X_dev, X_test, y_dev, y_test = train_test_split(
    X, y,
    test_size=0.2,        # 20% of the data will go to the test set
    random_state=42,      # Ensures the split is reproducible
    stratify=y            # Keeps the same class distribution in dev/test
sets
)

# Split the dev set (80%) into training (60%) and validation sets (20%)
# 75/25 split
X_train, X_val, y_train, y_val = train_test_split(
    X_dev, y_dev,
    test_size=0.25,       # 25% of the 80% dev set (which is 20% of the
original)
    random_state=42,      # Ensures this second split is also reproducible
    stratify=y_dev        # Keeps the same class distribution in train/val
sets
)
```

3. Fit a decision tree on the training dataset. (1 point)

Below is the code to create a decision tree and to train/fit with our data. I also have comments explaining each line to clearly state what is going on.

```
from sklearn.tree import DecisionTreeClassifier

# Creating an instance of the DecisionTreeClassifier.
# We set random_state=42 to ensure the results are reproducible.
decision_tree = DecisionTreeClassifier(random_state=42)

# Train (fit) the model on the training data (X_train, y_train).
# The model learns the relationship between the features and the labels
from this data.
```

```
decision_tree.fit(X_train, y_train)
```

4. Tune at least 2 hyperparameters in the decision tree model (https://ken-hoffman.medium.com/decision-tree-hyperparameters-explained-49158ee1268e) based on the performance on the validation set or using cross-validation. One hyperparameter has to be max_depth and the other one is your choice.
   Generate plot of hyperparameter values vs performance metric (plots are mandatory). (4 points)

Below I am defining what hyperparameters I want to test and the various values I want to test for each. I decided to tune every single hyperparameter that was shown in the medium article. Below the param_grid, I make a new decision tree, which'll be used for tuning purposes, and I also create a GridSearchCV object to define how I am going to perform the cross-validation process.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Defining the grid of hyperparameters we want to test.
param_grid = {
    'criterion': ['gini', 'entropy'],       # The function to measure split
quality
    'max_depth': [2, 3, 4, 5, None],        # Test various depths
    'min_samples_leaf': [1, 2, 3, 4],       # Minimum samples at a leaf
node
    'min_samples_split': [2, 4, 6, 8],      # Minimum samples to split an
internal node
    'max_features': ['sqrt', 'log2', None] # Number of features to
consider at each split
}

# Creating a new decision tree classifier for tuning
dt_classifier_tuned = DecisionTreeClassifier(random_state=42)

grid_search = GridSearchCV(
    estimator=dt_classifier_tuned, # the model to tune
    param_grid=param_grid,         # the hyperparameters to test
    cv=5, # use 5-fold cross-validation. The training data will be split 5
times
```

```
    scoring='accuracy',             # evaluate models based on their
accuracy
    verbose=1                       # prints progress
)
```

Below I am tuning the hyperparameters by seeing how well each combination will perform given the training data. I am then outputting the best hyperparameters found and the accuracy on the training and validation sets.

```
# Fit the GridSearchCV object on the TRAINING data
print("\nStarting hyperparameter tuning with GridSearchCV...")
grid_search.fit(X_train, y_train)

# The best combination of hyperparameters found by the search
print("\nBest hyperparameters found:")
print(grid_search.best_params_)

# The mean cross-validated score of the best estimator
print(f"\nBest cross-validation accuracy on training data:
{grid_search.best_score_:.4f}")

# Getting the best model found by the grid search
best_model = grid_search.best_estimator_

# Evaluating the best model on the unseen VALIDATION set
y_val_pred = best_model.predict(X_val)
validation_accuracy = accuracy_score(y_val, y_val_pred)

print(f"\nAccuracy of the tuned model on the validation set:
{validation_accuracy:.4f}")
```

Output:

```
Starting hyperparameter tuning with GridSearchCV...
Fitting 5 folds for each of 480 candidates, totalling 2400 fits

Best hyperparameters found:
{'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2}

Best cross-validation accuracy on training data: 0.9889
```

```
Accuracy of the tuned model on the validation set: 0.9000
```

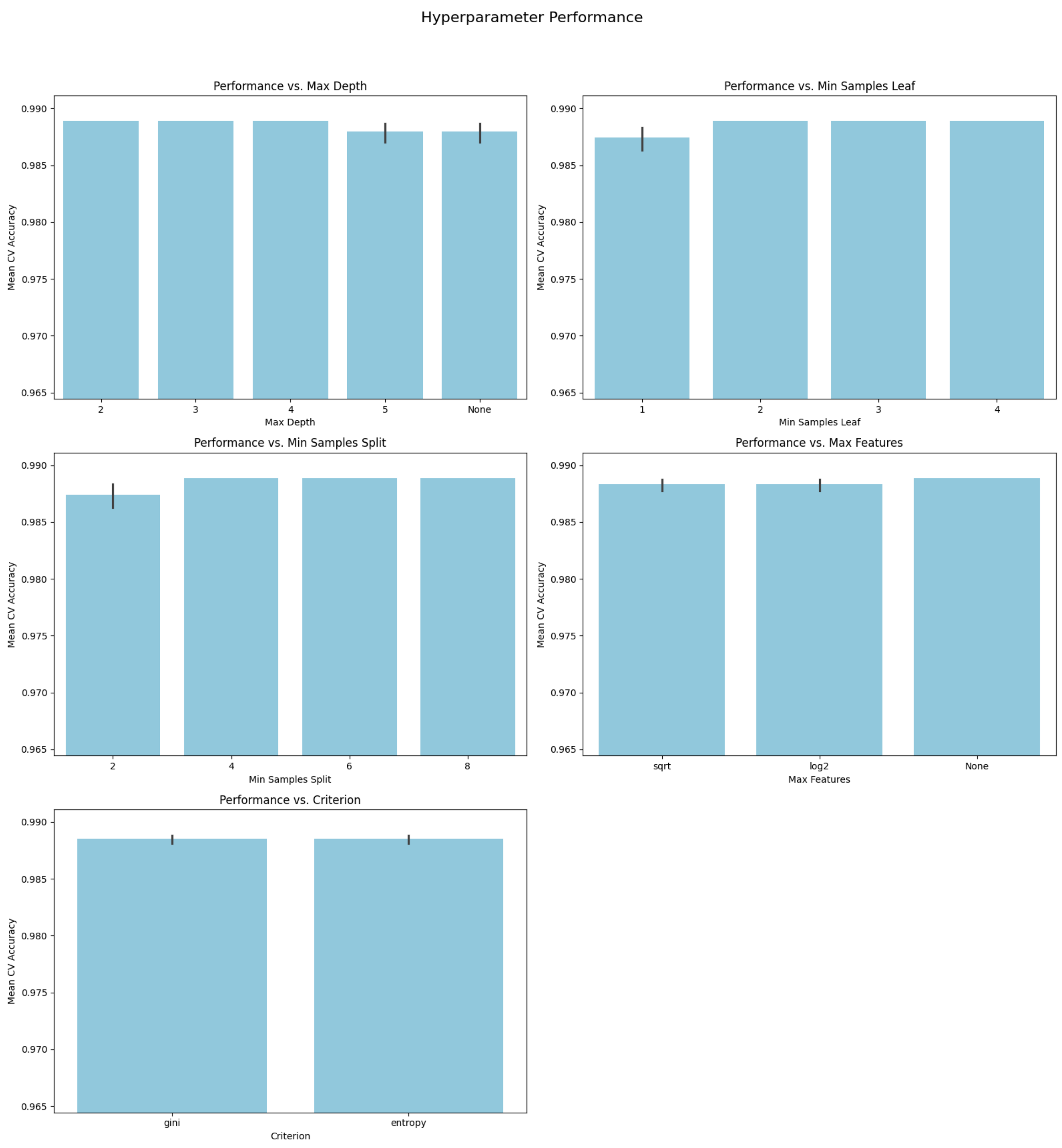
Below is the code that I use to visualize the performance of each hyperparameter value.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Converting the cv results into a pandas dataframe
results_df = pd.DataFrame(grid_search.cv_results_)

# This prevents sorting errors in seaborn when a column has mixed types
results_df['param_max_depth'] = results_df['param_max_depth'].astype(str)
results_df['param_max_features'] =
results_df['param_max_features'].astype(str)


# Find the min and max average scores to set a dynamic y-axis range.
min_score = results_df['mean_test_score'].min()
max_score = results_df['mean_test_score'].max()
score_range = max_score - min_score
# Add a small buffer to the top and bottom for clarity.
y_lim_buffer = score_range * 0.1
y_min = min_score - y_lim_buffer
y_max = max_score + y_lim_buffer


# Create a 3x2 figure to hold the plots for our 5 hyperparameters.
fig, axes = plt.subplots(3, 2, figsize=(16, 18))
fig.suptitle('Hyperparameter Performance', fontsize=16)

# Plot for 'max_depth'
sns.barplot(
    data=results_df,
    x='param_max_depth',
    y='mean_test_score',
    color='skyblue', # Use a single color for simplicity
    ax=axes[0, 0]
)
axes[0, 0].set_title('Performance vs. Max Depth')
axes[0, 0].set_xlabel('Max Depth')
axes[0, 0].set_ylabel('Mean CV Accuracy')
```

```
axes[0, 0].set_ylim(y_min, y_max) # Apply zoom

# Plot for 'min_samples_leaf'
sns.barplot(
    data=results_df,
    x='param_min_samples_leaf',
    y='mean_test_score',
    color='skyblue',
    ax=axes[0, 1]
)
axes[0, 1].set_title('Performance vs. Min Samples Leaf')
axes[0, 1].set_xlabel('Min Samples Leaf')
axes[0, 1].set_ylabel('Mean CV Accuracy')
axes[0, 1].set_ylim(y_min, y_max) # Apply zoom

# Plot for 'min_samples_split'
sns.barplot(
    data=results_df,
    x='param_min_samples_split',
    y='mean_test_score',
    color='skyblue',
    ax=axes[1, 0]
)
axes[1, 0].set_title('Performance vs. Min Samples Split')
axes[1, 0].set_xlabel('Min Samples Split')
axes[1, 0].set_ylabel('Mean CV Accuracy')
axes[1, 0].set_ylim(y_min, y_max) # Apply zoom


# Plot for 'max_features'
sns.barplot(
    data=results_df,
    x='param_max_features',
    y='mean_test_score',
    color='skyblue',
    ax=axes[1, 1]
)
axes[1, 1].set_title('Performance vs. Max Features')
axes[1, 1].set_xlabel('Max Features')
axes[1, 1].set_ylabel('Mean CV Accuracy')
```

```
axes[1, 1].set_ylim(y_min, y_max) # Apply zoom

# Plot for 'criterion'
sns.barplot(
    data=results_df,
    x='param_criterion',
    y='mean_test_score',
    color='skyblue',
    ax=axes[2, 0]
)
axes[2, 0].set_title('Performance vs. Criterion')
axes[2, 0].set_xlabel('Criterion')
axes[2, 0].set_ylabel('Mean CV Accuracy')
axes[2, 0].set_ylim(y_min, y_max) # Apply zoom

# Not showing the last spot since there are 5 plots and 6 slots.
fig.delaxes(axes[2,1])

# Display the plots
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Visualization Output:



5. Train the model using optimal hyperparameters (found in step 5) on the train + validation data. Test it on test data and generate a classification report (1 point)

Below I am combining the training and valuation data together for the final model where I use the best hyperparameters found to train the final model.

```
# Combining the training and validation set for a final model training
X_train_full = pd.concat([X_train, X_val])
y_train_full = pd.concat([y_train, y_val])

print(f"Combined training set shape: {X_train_full.shape}")

# Creating a new decision tree combing both datasets
# The ** operator unpacks the dictionary of best parameters.
final_model = DecisionTreeClassifier(**grid_search.best_params_,
random_state=42)

# Train this final model on the combined data
final_model.fit(X_train_full, y_train_full)

print("Final model has been successfully trained on the combined training
and validation data.")
```

Output:

```
Combined training set shape: (120, 4)
Final model has been successfully trained on the combined training and
validation data.
```

Below I am using my final model to predict the values of the testing data, and evaluating its performance.

```
from sklearn.metrics import classification_report

print("\nFinal Model Evaluation:")
# Now evaluating the retrained model on the test set
y_test_pred = final_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Accuracy of the Final model on the test set: {test_accuracy:.4f}")

print("\nClassification Report on Test Data:")
print(classification_report(y_test, y_test_pred,
target_names=iris_data.target_names))
```

Output:

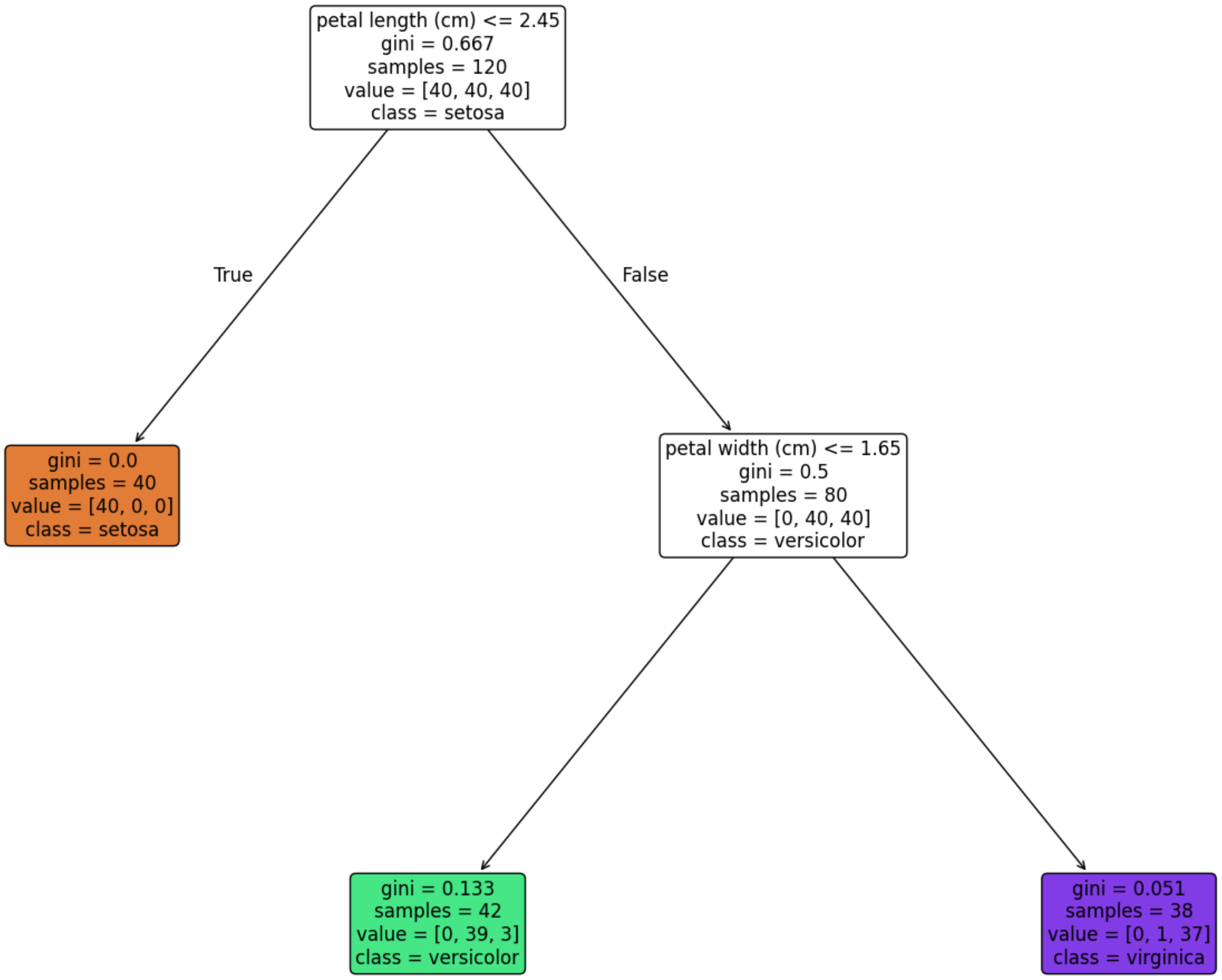
```
Final Model Evaluation:
Accuracy of the Final model on the test set: 0.9333

Classification Report on Test Data:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

The model's perfect score for setosa is a direct result of it being a linearly separable class. Performance on versicolor and virginica was also very high, with the minor errors caused by the natural similarity and overlap in their measurements.

6. Inspect the model by visualizing and interpreting the results (1 point)

Below is the code to display the decision tree of my final optimized model.

```
from sklearn.tree import plot_tree
# Inspecting the model by visualizing and interpreting the results
print("\n--- Visualizing the Final Decision Tree ---")
plt.figure(figsize=(20,15))
plot_tree(
    final_model,
    filled=True,
    feature_names=iris_data.feature_names,
    class_names=iris_data.target_names,
    rounded=True,
    fontsize=12
)
plt.title("Decision Tree Visualization", fontsize=20)
plt.show()
```



This tree visualization shows really clear logic on how the decision tree model is working.The first rule on petal length perfectly separates all the setosa flowers, which is why that class had a perfect score. The second rule on petal width then separates versicolor and virginica, and you can see the slight class overlap in the final leaf nodes which explains the few errors from the final report.