CS471: Introduction to Artificial Intelligence
Assignment 4: Decision Tree Classification

In this assignment, you will implement the Decision tree classification method using **Scikit-learn**.

This is an individual assignment. Use Google Colab for your code, plots, and comments. When you finish editing, re-run all the cells to make sure they work.

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-exp

lained-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)

Google Colab Link:
https://colab.research.google.com/drive/1uwhbQPiLedXzWCxIRXIM2ecPhDmZUeRy?usp=sharing
GitHub Link:
https://github.com/jacobalmon/CS-471/blob/main/Homework/Homework%204/dtclass.py

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

These are the following modules used for implementing this assignment. Sklearn is mainly used to create the Machine Learning Model using a Decision Tree Classifier and Splitting Data. The matplot module was used to help plot the graphs for the validation dataset and the hyperparamters.

```
    iris = load_iris() # Load Iris Dataset.
    x = iris.data # Features.
    y = iris.target # Labels.

    # Split Data into Training, Validation, and Testing
Datasets.
    x_train, x_test, y_train, y_test = train_test_split(x,
y, test_size=0.2, random_state=29)
    x_train, x_valid, y_train, y_valid =
train_test_split(x_train, y_train, test_size=0.2,
random_state=29)
```

In this following code segment, we are loading the iris dataset and breaking the dataset into a training, validation, and testing datasets. We just define some random state, in our case its 29.

```
    # Range of Values.
    max_depth_vals = range(1, 15)
    min_samples_leaf_vals = range(1, 11)

    # Creating Lists for the Outcomes of the
Hyperparameters.
    max_depth_outcomes = []
    min_samples_leaf_outcomes = []

    # Evaluating Decision Tree Classifier for Max Depth
using Cross-Validation.
    for max_depth in max_depth_vals:
        dt_classifier =
DecisionTreeClassifier(max_depth=max_depth,
random_state=29)
        scores = cross_val_score(dt_classifier, x_train,
```

```
y_train, cv=5)  # 5-fold cross-validation
        max_depth_outcomes.append(scores.mean())

    # Evaluating Decision Tree Classifier for Min Samples
Leaf using Cross-Validation.
    for min_samples_leaf in min_samples_leaf_vals:
        dt_classifier =
DecisionTreeClassifier(min_samples_leaf=min_samples_leaf,
random_state=29)
        scores = cross_val_score(dt_classifier, x_train,
y_train, cv=5)  # 5-fold cross-validation
        min_samples_leaf_outcomes.append(scores.mean())
```

In this following code segment, we are tuning the hyperparameters which are max depth and min samples leaf. We then evaluate the model with a Decision Tree Classifier with each hyperparameter by storing the outcomes of each value of the ones we define. We find these values, using a 5-fold cross-validation keep the same random state of 29 as well.

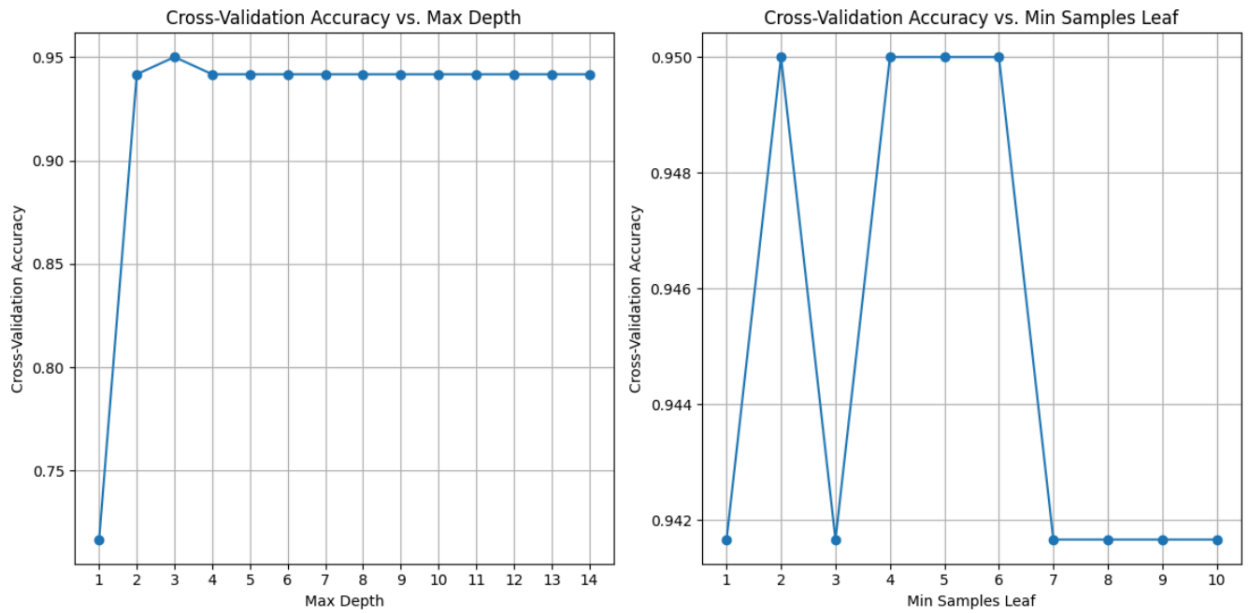
```
    plt.figure(figsize=(12,6))

    # Plotting the Validation Accuracy vs. Max Depth.
    plt.subplot(1,2,1)
    plt.plot(max_depth_vals, max_depth_outcomes, marker='o')
    plt.title('Validation Accuracy vs. Max Depth')
    plt.xlabel('Max Depth')
    plt.ylabel('Validation Accuracy')
    plt.xticks(max_depth_vals)
    plt.grid(True)

    # Plotting the Validation Accuracy vs. Min Samples Leaf.
    plt.subplot(1,2,2)
    plt.plot(min_samples_leaf_vals, min_samples_leaf_outcomes, marker='o')
    plt.title('Validation Accuracy vs. Min Samples Leafs')
    plt.xlabel('Min Samples Leaf')
    plt.ylabel('Validation Accuracy')
    plt.xticks(min_samples_leaf_vals)
    plt.grid(True)
```

```
    # Display Plots onto the Screen.
    plt.tight_layout()
    plt.show()
```

In this following code segment, we plot the two hyperparameters against the cross-validation accuracy. Our results are:



Initially, the max_depth would be low since there is only one datapoint to find the max_depth, so it's not very accurate, but when we go up more nodes, we see the accuracy improves and stays constant above 90 percent. On the other hand, the min sample leaf oscillates from 94 and 95 percents.

```
    # Find the Best Max Depth Outcome.
    best_max_depth =
max_depth_vals[max_depth_outcomes.index(max(max_depth_outcomes))]
    # Find the Best Min Samples Lead Outcome.
    best_min_samples_leaf =
min_samples_leaf_vals[min_samples_leaf_outcomes.index(max(min_samples_leaf_outc
omes))]

    # Fitting the Model with these new Best Max Outcome & Min Samples Leaf
Outcome.
    final_dt_classifier = DecisionTreeClassifier(max_depth=best_max_depth,
min_samples_leaf=best_min_samples_leaf, random_state=42)
```

```
    final_dt_classifier.fit(x_train, y_train)

    # Find the Test Accuracy from our Testing Data.
    y_pred = final_dt_classifier.predict(x_test)
    test_accuracy = accuracy_score(y_test, y_pred)

    # Printing Results.
    print(f'Test Dataset Accuracy: {test_accuracy * 100}')
```

After evaluating our ML model with the validation dataset, we need to pick the best one and we do this in the following code segment to find the best max depth and best min samples leaf. Lastly, we pick the best model and test the model on our testing data which results in the accuracy of:

```
Test Dataset Accuracy: 90.0
```

Now we can visualize the decision tree with the following code segment.

```
    # Visualizing the Decision Tree
    plt.figure(figsize=(5, 5))
    plot_tree(final_dt_classifier,
feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)
    plt.title(f'Decision Tree Visualization (Max Depth:
{best_max_depth}, Min Samples Leaf:
{best_min_samples_leaf})')
    plt.show()
```

And the outcome of our tree would is:

Decision Tree Visualization (Max Depth: 3, Min Samples Leaf: 2)