# Programming Assignment: Decision Tree Classifier

Submission Deadline: 20th November, 2024, 2:59 PM

## Objective

The goal of this assignment is to build and evaluate a Decision Tree classifier for binary classification. You will implement the core components of a Decision Tree from scratch, using a modified version of the Iris dataset.

## Dataset

Use the Iris dataset for binary classification:

- **Dataset Source**: Available directly through `sklearn.datasets`.
- **Features**: This dataset contains 4 numerical features (sepal length, sepal width, petal length, petal width).
- **Target**: For simplicity, we'll create a binary classification problem by selecting only two classes, "Iris-setosa" and "Iris-versicolor".

**Instructions**: Load the Iris dataset using `sklearn.datasets`.load_iris. Filter the data to include only "Iris-setosa" and "Iris-versicolor" classes and set the target labels to 1 for "Iris-setosa" and 0 for "Iris-versicolor".

## Tasks

### 1. Data Preprocessing

- **Load the Dataset**: Use `sklearn.datasets` to load the Iris dataset.
- **Binary Filter**: Filter the dataset to include only "Iris-setosa" and "Iris-versicolor" instances.
- **Encoding**: Encode the target column as binary labels (1 for "Iris-setosa" and 0 for "Iris-versicolor").
- **Train-Test Split**: Split the data into training and testing sets (e.g., 80% training and 20% testing).

## 2. Decision Tree Implementation

Implement the following functions to build a Decision Tree classifier:

- `calculate_entropy(y)`: Calculate the entropy of a set of labels `y`.
- `information_gain(y, y_left, y_right)`: Calculate the information gain from a split, given the parent set `y` and two child sets `y_left` and `y_right`.
- `best_split(X, y)`: Find the best feature and threshold to split the data, maximizing information gain. Return the feature index and threshold value.
- `build_tree(X, y, max_depth)`: Recursively build a Decision Tree, splitting nodes up to a maximum depth `max_depth`. Each node should store its decision (feature index and threshold) and references to left and right child nodes.
- `predict_single_instance(x, tree)`: Use the trained Decision Tree to predict the label of a single instance `x`.
- `predict_decision_tree(X_test, tree)`: Predict binary labels for the test set using the trained Decision Tree.

**Evaluation Metric**:

- **Accuracy**: Measure and print the accuracy of the Decision Tree on both the train and test sets.

## 3. Comparison with Scikit-Learn

Train a Decision Tree classifier on the same dataset using `scikit-learn`'s `DecisionTreeClassifier` with the same `max_depth` parameter:

- Compare the results from your implementation with `scikit-learn`'s version.
- Discuss any differences in accuracy or model performance, if applicable.

## 4. Visualization and Analysis

- **Tree Visualization**: Use `scikit-learn` to visualize the Decision Tree structure for the trained model on a test example.
- **Analysis**: Examine the performance of the Decision Tree classifier:
  - How does the maximum depth `max_depth` affect the accuracy and overfitting of the model?
  - What are the strengths and limitations of using a Decision Tree for this binary classification problem?

## Requirements

- Implement all functions from scratch, without using any pre-built machine learning libraries (e.g., scikit-learn), except for the comparison section.
- Use `Numpy` for numerical operations and `pandas` for data handling.
- Optional: Use `matplotlib` for visualizations.

## Submission

Submit a Jupyter Notebook (`.ipynb`) containing:

- Your implemented code for all functions.
- Accuracy results for both the train and test sets.
- Analysis as discussed in Task 4.
- Experiments with different values of `max_depth` (e.g., `max_depth=3, 5, 10`).