Decision tree classifier

Ai project

▼ Import Required Libraries

```
import warnings
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
```

`import warnings`

- **Purpose**:
  - The `warnings` module in Python is used to control the warning messages generated by code.
- **Use in this code**:
  - Warnings might occur due to deprecated functions or minor issues in external libraries.
  - Instead of showing these warnings in the output, they are suppressed.

`import pandas as pd`

  - **Purpose**:
    - `pandas` is a powerful library for data manipulation and analysis.
    - It allows you to work with structured data formats such as DataFrames and Series.
  - **Alias** `pd`:

- The alias `pd` is a convention to simplify usage (e.g., `pd.DataFrame()` instead of `pandas.DataFrame()`).

```
from sklearn.datasets import load_iris
```

- **Purpose**:
  - Imports the `load_iris` function from `scikit-learn` (a machine learning library).
  - The `load_iris` function loads the Iris dataset, a well-known dataset used for classification tasks.
- **What is the Iris dataset?**
  - A small dataset containing 150 samples of Iris flowers, categorized into 3 species:
    - Setosa (class 0)
    - Versicolor (class 1)
    - Virginica (class 2)
  - Each sample has 4 features: sepal length, sepal width, petal length, petal width.

```
from sklearn.model_selection import train_test_split
```

- **Purpose**:
  - Imports the `train_test_split` function, used to split a dataset into training and testing subsets.
- **Why split data?**
  - Splitting data helps evaluate the model's performance on unseen data, ensuring it generalizes well to new inputs.
- **How it works**:
  - The function randomly splits the input features (`X`) and target labels (`y`) into two subsets:
    - Training data: Used to train the model.
    - Testing data: Used to evaluate the trained model.

`from sklearn.tree import DecisionTreeClassifier, plot_tree`

- **Purpose**:
  - Imports two tools from `scikit-learn`'s tree module:
    - `DecisionTreeClassifier`:
      - A machine learning algorithm that uses a tree-like structure to make decisions.
      - It splits data based on feature values to classify inputs into target categories.
    - `plot_tree`:
      - A function to visualize the structure of a decision tree.
      - Provides insights into the model's decision-making process.

`import matplotlib.pyplot as plt`

- **Purpose**:
  - Imports the `pyplot` module from `matplotlib`, a popular visualization library.
  - The alias `plt` is a convention to simplify its usage.
- **What it does**:
  - Enables the creation of static, interactive, and visually appealing plots like line graphs, scatter plots, and more.

`warnings.filterwarnings("ignore")`

- **Purpose**:
  - Suppresses warning messages to make the output cleaner.
  - Often used during exploratory or demonstration tasks where warnings are non-critical.
- **How it works**:
  - The `"ignore"` argument tells Python to ignore all warning messages.
  - If removed, warnings generated by other libraries or the code itself will be displayed.

▼ Load The Iris Dataset

```
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)  #
y = pd.Series(iris.target)  # Target labels
```

`iris = load_iris()`

- **What it does**:
  - Calls the `load_iris()` function from the `sklearn.datasets` module to load the Iris dataset.
  - Returns a Python object containing the data and metadata of the Iris dataset.
- `iris` **Object Structure**:
  - The returned object is a `Bunch`, similar to a dictionary.
  - Key components:
    - `iris.data`: A 2D array of feature values.
    - `iris.target`: A 1D array of target labels.
    - `iris.feature_names`: Names of the features (sepal length, sepal width, etc.).
    - `iris.target_names`: Names of target classes (Setosa, Versicolor, Virginica).

`X = pd.DataFrame(iris.data, columns=iris.feature_names)`

- **What it does**:
  - Converts the `iris.data` (a NumPy array) into a Pandas DataFrame for better usability.
  - Assigns feature names (`iris.feature_names`) as column headers in the DataFrame.
- **Why use** `DataFrame`**?**

    - Easier to manipulate and visualize than raw NumPy arrays.
    - Provides descriptive headers for better understanding.
- **Structure of `X`**:
    - Rows: Each row represents a sample (150 samples in total).
    - Columns: Each column corresponds to a feature (sepal length, sepal width, etc.).

Example:

```
sepal length (cm)  sepal width (cm)  petal length (cm)
petal width (cm)
0                5.1               3.5                1.
4               0.2
1                4.9               3.0                1.
4               0.2
...
```

`y = pd.Series(iris.target)`

- **What it does**:
    - Converts the `iris.target` array into a Pandas Series for better usability.
- **What is `iris.target`?**
    - A 1D array of integers representing the species of each sample.
    - Classes:
        - `0`: Setosa
        - `1`: Versicolor
        - `2`: Virginica
- **Structure of `y`**:
    - A single column where each value corresponds to the class label for a sample.

Example:

```
0    0
1    0
2    0
...
149  2
```

▼ Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, t
```

1. `train_test_split()` Function

- **Purpose**:
  - Splits the dataset into **training** and **testing** subsets.
  - Ensures that the model is trained on one portion of the data and evaluated on a separate, unseen portion to test generalization.

2. Inputs to `train_test_split`

- `X`:
  - The feature data (input variables).
  - In this case, it's the `DataFrame` created earlier with columns:
    - `sepal length (cm)`
    - `sepal width (cm)`
    - `petal length (cm)`
    - `petal width (cm)`
- `y`:
  - The target data (output labels).
  - This is the `Series` created earlier containing the class labels:
    - `0` for Setosa
    - `1` for Versicolor

        - `2` for Virginica
- `test_size=0.3`:
    - Specifies the proportion of the data to allocate for the testing set.
    - `0.3` means 30% of the total data is reserved for testing, while the remaining 70% is used for training.
- `random_state=42`:
    - A seed for the random number generator to ensure the split is reproducible.
    - With the same `random_state`, the function will generate the same training and testing splits every time it's run.

---

3. Outputs from `train_test_split`

The function returns four subsets:

1. `X_train`:
    - A subset of `X` (features) used for training.
    - Contains 70% of the rows from the original dataset (if `test_size=0.3`).
2. `X_test`:
    - A subset of `X` used for testing.
    - Contains 30% of the rows from the original dataset.
3. `y_train`:
    - A subset of `y` (labels) corresponding to the rows in `X_train`.
4. `y_test`:
    - A subset of `y` corresponding to the rows in `X_test`.

---

4. Why Split the Data?

- **Training Set (`X_train` and `y_train`)**:
    - Used to train the machine learning model.
    - The model learns patterns and relationships between features (`X`) and labels (`y`).

- **Testing Set (`X_test` and `y_test`):**
  - Used to evaluate the trained model's performance.
  - Ensures that the model generalizes well to new, unseen data.

---

5. Example of the Split

If the dataset (`X`, `y`) has 150 samples:

- **Training Set (70%)**:
  - `X_train`: Contains 105 samples (70% of 150).
  - `y_train`: Corresponding labels for the 105 samples.
- **Testing Set (30%)**:
  - `X_test`: Contains 45 samples (30% of 150).
  - `y_test`: Corresponding labels for the 45 samples.

Why is `random_state` Important?

- Without `random_state`, the split is random and different each time the code runs.
- Setting `random_state=42` ensures consistent results, which is useful for debugging and reproducibility.

▼ Initialize and train the decision tree

```
clf = DecisionTreeClassifier(criterion='gini', max_depth=5,
clf.fit(X_train, y_train)
```

1. `clf = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)`

- **What it does**:
  - Creates an instance of the `DecisionTreeClassifier` model from `scikit-learn`.
  - The model is initialized with specific hyperparameters:

- `criterion='gini'`:
  - Determines the metric used to split the nodes of the decision tree.
  - **Gini Impurity**:
    - Measures the impurity (or disorder) of a node.
    - Formula:
      where pi is the probability of class i.

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

    - A lower Gini value indicates a purer node.
- `max_depth=5`:
  - Limits the maximum depth of the tree to 5 levels.
  - Prevents overfitting by restricting the tree from growing too complex.
- `random_state=42`:
  - Sets the seed for the random number generator to ensure consistent results across runs.

- **Why use a decision tree?**
  - It models data by splitting it into subsets based on feature values, forming a tree-like structure.
  - The splits are chosen to maximize information gain or minimize impurity (depending on the criterion).

---

2. `clf.fit(X_train, y_train)`

- **What it does**:
  - Trains the `DecisionTreeClassifier` (`clf`) on the training dataset.
  - The `.fit()` method:
    1. Takes `X_train` (features) and `y_train` (labels) as inputs.

2. Builds a decision tree by recursively splitting the training data based on the selected `criterion`.
3. Adjusts the model's parameters to learn patterns and relationships in the data.

- **Input Parameters**:
  - `X_train`:
    - A DataFrame containing the training features (e.g., sepal length, petal width).
  - `y_train`:
    - A Series containing the corresponding labels (e.g., 0 for Setosa, 1 for Versicolor, etc.).
- **Output**:
  - A trained model stored in `clf`, ready to make predictions.

What Happens Internally?

1. **Tree Initialization**:
   - The tree starts with a single root node containing all the training data.
2. **Recursive Splitting**:
   - At each step, the algorithm:
     - Evaluates all possible splits of the features.
     - Selects the split that minimizes Gini impurity.
   - This process continues until:
     - All leaves are pure (contain only one class).
     - The tree reaches `max_depth`.
     - No further splits can improve the impurity significantly.
3. **Stopping Criteria**:
   - `max_depth=5` restricts the tree to a maximum of 5 levels.

Example:

If `max_depth` is not specified, the tree might grow excessively deep, resulting in overfitting. By limiting it to 5:

- The tree is simpler.
- The model is less likely to overfit but might underfit slightly if `max_depth` is too small.

Why is `random_state` Important?

- Controls the randomness in:
  - Splitting data at nodes when multiple splits have the same Gini value.
  - Handling ties in decision-making during tree construction.
- Ensures reproducibility of the same decision tree structure on repeated runs.

▼ Evaluate the model

```
accuracy = clf.score(X_test, y_test)
print(f"Model Accuracy on Test Data: {accuracy * 100:.2f}%"
```

1. `accuracy = clf.score(X_test, y_test)`

- **What it does**:
  - Calculates the accuracy of the trained model (`clf`) on the testing dataset (`X_test` and `y_test`).
  - Uses the `score` method provided by `scikit-learn` for the `DecisionTreeClassifier`.
- **How `score` Works**:
  - For a classifier, `score` computes the fraction of correctly classified samples:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}}$$

- Internally, it:
    1. Makes predictions for `X_test` using `clf.predict(X_test)`.
    2. Compares these predictions with the actual labels in `y_test`.

- `clf`:
    - The trained `DecisionTreeClassifier` model.
    - It was trained earlier on `X_train` and `y_train`.
- `X_test` **and** `y_test`:
    - These are the test datasets, containing:
        - `X_test`: Features of the testing data.
        - `y_test`: True labels for the testing data.

---

2. `print(f"Model Accuracy on Test Data: {accuracy * 100:.2f}%")`

- **What it does**:
    - Prints the accuracy as a percentage, formatted to two decimal places.
- **String Interpolation (`f-strings`)**:
    - The `f` in `f"..."` allows embedding variables and expressions directly inside strings.
    - `{accuracy * 100:.2f}`:
        - `accuracy * 100`: Converts accuracy (a fraction) to a percentage.
        - `:.2f`: Formats the number to two decimal places.

Example of Output:

If the accuracy is `0.9333333333333333` (fraction of correct predictions), the output will be:

```
Model Accuracy on Test Data: 93.33%
```

Why is Accuracy Important?

- **Measures Model Performance**:
  - Accuracy quantifies how well the model is predicting the correct labels on unseen data.
  - A high accuracy suggests the model generalizes well to new data.
- **Indicates Generalization**:
  - Ensures the model is not overfitting or underfitting the training data.

▼ Plot the decision tree

```
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=iris.feature_names, class_name
plt.title("Decision Tree Visualization")
plt.show()
```

1. `plt.figure(figsize=(12, 8))`

- `plt` is the abbreviation for `matplotlib.pyplot`, a popular plotting library in Python used for creating visualizations.
- `figure()` is a function in `matplotlib` used to initialize a new figure (a blank canvas) for plotting.
- `figsize=(12, 8)` specifies the size of the figure in inches, with a width of 12 inches and a height of 8 inches. The figure size is important for making the plot look proportionally correct and readable, especially for complex visualizations like decision trees.

2. `plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)`

- `plot_tree()` is a function from `sklearn.tree` that visualizes a decision tree classifier.
- `clf` is the trained decision tree model, likely an instance of `DecisionTreeClassifier`. It contains the model's decision tree structure, which will be visualized.
- `feature_names=iris.feature_names` specifies the names of the features (input variables) in the dataset. In this case, `iris.feature_names` refers to the

feature names in the Iris dataset (e.g., "sepal length", "sepal width", "petal length", "petal width").

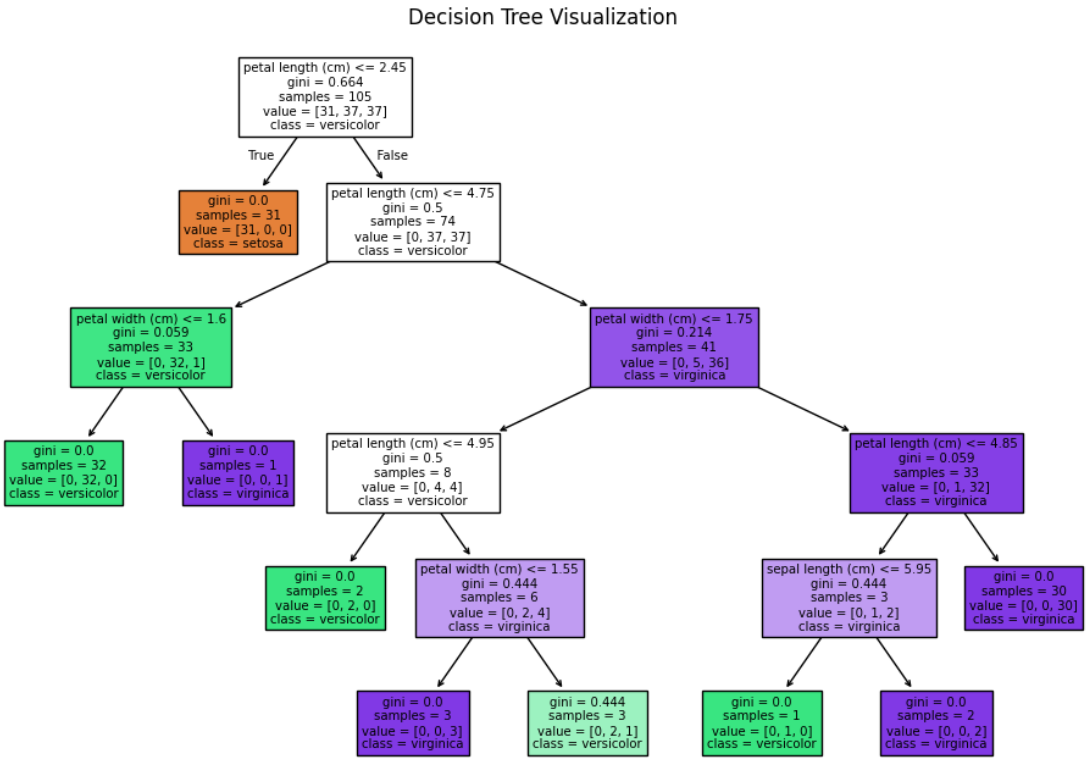
- `class_names=iris.target_names` provides the names of the classes or target categories the model is trying to predict. For the Iris dataset, these are the names of the flower species (e.g., "setosa", "versicolor", "virginica").
- `filled=True` tells the function to fill the nodes of the decision tree with colors representing the class they belong to. The color intensity indicates the confidence of the prediction.

3. `plt.title("Decision Tree Visualization")`

- `title()` is a `matplotlib` function used to add a title to the plot.
- `"Decision Tree Visualization"` is the string that will be displayed as the title of the plot. This helps to provide context for the viewer, indicating that the plot is a visualization of a decision tree.

4. `plt.show()`

- `show()` is another `matplotlib` function used to display the plot. It renders the figure on the screen so the user can see the output.



▼ Predict on new data

```
sample_data = [[5.1, 3.5, 1.4, 0.2]]  # Example input
predicted_class = clf.predict(sample_data)
print(f"Predicted Class: {iris.target_names[predicted_class
```

1. `sample_data = [[5.1, 3.5, 1.4, 0.2]]`

- `sample_data` is a list that contains a single sample (in this case, a flower's attributes). It is a 2D list (a list within a list), which is how input data is often formatted when passed into machine learning models. The sample here represents four features of a flower, corresponding to the Iris dataset:
  - `5.1`: Sepal length (in cm)
  - `3.5`: Sepal width (in cm)
  - `1.4`: Petal length (in cm)
  - `0.2`: Petal width (in cm)
- This data will be fed into the trained decision tree classifier to make a prediction.

2. `predicted_class = clf.predict(sample_data)`

- `clf.predict()` is a method of the trained decision tree model (`clf`). It takes an input (in this case, `sample_data`) and outputs the predicted class label.
- `sample_data` is passed to the `predict()` method, and the model uses the learned decision tree structure to classify the input into one of the target classes.
- The result, `predicted_class`, will be a list or array containing the predicted class index. In this case, it would correspond to the index of one of the Iris species in the `iris.target_names` array.

3. `print(f"Predicted Class: {iris.target_names[predicted_class[0]]}")`

- `f"Predicted Class: {iris.target_names[predicted_class[0]]}"` is an f-string, a feature in Python that allows you to embed expressions inside string

literals. It will evaluate the expression inside `{}` and insert the result into the string.

- `iris.target_names[predicted_class[0]]` accesses the `iris.target_names` array (which contains the names of the Iris species) and selects the predicted class name by indexing it with `predicted_class[0]`. Since `predicted_class` is a list or array with a single element, `predicted_class[0]` gives the class index.
    - For example, if the predicted class is `0`, it will print `"setosa"`.
- `print()` outputs the formatted string, showing the predicted class name based on the model's decision.