PROBLEM:

WRITE PYTHON CODE TO IMPLEMENT ID3 ALGORITHM AND TEST IT ON THE PLAYTENNIS DATASET AND VERIFY INDUCTIVE BIAS OF DECISION TREE LEARNING ALGORITHM.

THEORY:

A **DECISION TREE** is a popular and widely used machine learning algorithm for classification and regression tasks. It represents a model in the form of a tree structure where each internal node represents a decision based on a feature, each branch represents the outcome of that decision, and each leaf node represents a class label (in classification) or a continuous value (in regression).

1. Entropy:

Entropy is a measure of impurity or disorder. In the context of decision trees, it is used to quantify the uncertainty in the target variable.

The formula for **entropy** of a dataset S is: $\text{Entropy}(S) = -\sum_{i=1}^{m} p_i \log_2(p_i)$

Where:

- m is the number of distinct classes in the target variable.
- p_i is the proportion of instances in class iii in the dataset SSS.

Entropy is 0 when all instances in the dataset belong to the same class (pure node), and it is at its maximum (logarithmic value) when the classes are equally distributed.

2. Information Gain:

Information Gain measures how much uncertainty is reduced (or how much information is gained) by splitting the dataset based on a particular feature.

The formula for **Information Gain** when splitting a dataset S by an attribute A is:

$$\text{Information Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Where:

- Entropy(S) is the entropy of the original dataset.
- Values(A) are the unique values of the attribute A.
- S_v is the subset of SSS where the attribute A has the value v.
- $|S|$ and $|S_v|$ are the sizes of the dataset S and its subset S_v, respectively.

The attribute with the highest information gain is chosen for splitting the data.

3. Gini Index:

The **Gini Index** is an alternative to entropy, often used in the CART (Classification and Regression Tree) algorithm. It measures the impurity of a dataset and is used to evaluate the quality of a split.

The formula for **Gini Index** is: $\text{Gini}(S) = 1 - \sum_{i=1}^{m} p_i^2$

Where:

- m is the number of classes in the target variable.
- P_i is the proportion of instances in class i in the dataset S.

The Gini Index ranges from 0 (perfectly pure dataset) to a maximum of 0.5 (when the classes are evenly distributed).

4. Gini Gain:

To determine the best split based on the Gini Index, we calculate the **Gini Gain** for each feature. The Gini Gain is the difference between the Gini Index of the dataset and the weighted Gini Index of the subsets formed by the split.

The formula for **Gini Gain** is: $\text{Gini Gain}(S, A) = \text{Gini}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Gini}(S_v)$

Where:

- Gini(S) is the Gini Index of the original dataset.
- S_v is the subset of S where the attribute A has the value v.
- |S| and $|S_v|$ are the sizes of the dataset S and its subset S_v , respectively.

5. CART (Classification and Regression Tree) Splitting Criterion:

CART uses the **Gini Index** for classification tasks and **Mean Squared Error (MSE)** for regression tasks to find the best split at each node.

In regression, the **Mean Squared Error (MSE)** for a subset S_v is calculated as: $\text{MSE}(S_v) = \frac{1}{|S_v|} \sum_{i \in S_v} (y_i - \bar{y}_{S_v})^2$

Where:

- y_i is the actual value of the target variable for instance iii in subset S_v .
- $\bar{y}_{S_v}$ is the mean target value for subset S_v .

Types of Decision Trees:

1. **Classification Trees**: Used for categorical target variables. For example, predicting whether a customer will buy a product (yes/no).
2. **Regression Trees**: Used for continuous target variables. For example, predicting house prices based on features like size, location, etc.

PYTHON CODE

```
# Import libraries
import pandas as pd
import numpy as np
from collections import Counter
from google.colab import files
from graphviz import Digraph

# Upload dataset
uploaded = files.upload()
data = pd.read_csv(next(iter(uploaded.keys())))
data = data.drop(columns=['day'])
print(data)
# Calculate entropy
def entropy(s):
    counts = Counter(s)
    probabilities = [count / len(s) for count in counts.values()]
    return -sum(p * np.log2(p) for p in probabilities if p > 0)

# Calculate information gain
def information_gain(data, split_attribute, target_attribute):
    total_entropy = entropy(data[target_attribute])
    values, counts = np.unique(data[split_attribute], return_counts=True)
    weighted_entropy = sum(
        (counts[i] / sum(counts)) * entropy(data[data[split_attribute] == value][target_attribute])
        for i, value in enumerate(values)
    )
    return total_entropy - weighted_entropy

# ID3 algorithm
def id3(data, original_data, features, target_attribute, parent_node=None):
    if len(np.unique(data[target_attribute])) == 1:  # Pure dataset
        return np.unique(data[target_attribute])[0]

    elif len(data) == 0:  # Empty dataset
        return np.unique(original_data[target_attribute])[
            np.argmax(np.unique(original_data[target_attribute], return_counts=True)[1])
        ]

    elif len(features) == 0:  # No features left
        return parent_node

    else:
        parent_node = np.unique(data[target_attribute])[
            np.argmax(np.unique(data[target_attribute], return_counts=True)[1])
        ]
```

```
    # Calculate information gain for each feature
    gains = [
        information_gain(data, feature, target_attribute) for feature in features
    ]
    best_feature = features[np.argmax(gains)]

    tree = {best_feature: {}}

    # Remove the best feature from the features list
    features = [f for f in features if f != best_feature]

    for value in np.unique(data[best_feature]):
        subtree = id3(
            data[data[best_feature] == value],
            original_data,
            features,
            target_attribute,
            parent_node,
        )
        tree[best_feature][value] = subtree

    return tree

# Visualize the decision tree
def draw_tree(tree, graph=None, parent=None, label=""):
    if graph is None:
        graph = Digraph(format="png")
        graph.attr(dpi="300")

    if not isinstance(tree, dict):
        graph.node(label, label, shape="ellipse", style="filled", color="lightblue")
        if parent:
            graph.edge(parent, label)
        return graph

    root = next(iter(tree))
    graph.node(root, root, shape="box", style="filled", color="orange")

    if parent:
        graph.edge(parent, root, label=label)

    for value, subtree in tree[root].items():
        draw_tree(subtree, graph, root, str(value))

    return graph
```

```
# Train the decision tree
target_attribute = "play"  # Target column
features = list(data.columns)
features.remove(target_attribute)

tree = id3(data, data, features, target_attribute)
print("\nDecision Tree:\n", tree)

# Draw and render the decision tree
graph = draw_tree(tree)
graph.render("decision_tree", view=True)

# Testing the inductive bias
def predict(tree, sample):
    if not isinstance(tree, dict):  # Leaf node
        return tree
    root = next(iter(tree))
    return predict(tree[root][sample[root]], sample)

# Modify the sample to ensure positive prediction based on dataset
sample = {
    "outlook": "Overcast",  # Likely positive based on common play patterns
    "temp": "Cool",
    "humidity": "High",
    "wind": "Strong",
}  # Replace with a sample from the dataset

prediction = predict(tree, sample)
print("\nSample Prediction:", prediction)
from IPython.display import Image

# Draw and render the decision tree
graph = draw_tree(tree)
graph.render("decision_tree")  # Render the tree to a file
Image(filename="decision_tree.png")  # Display the rendered tree inline
```

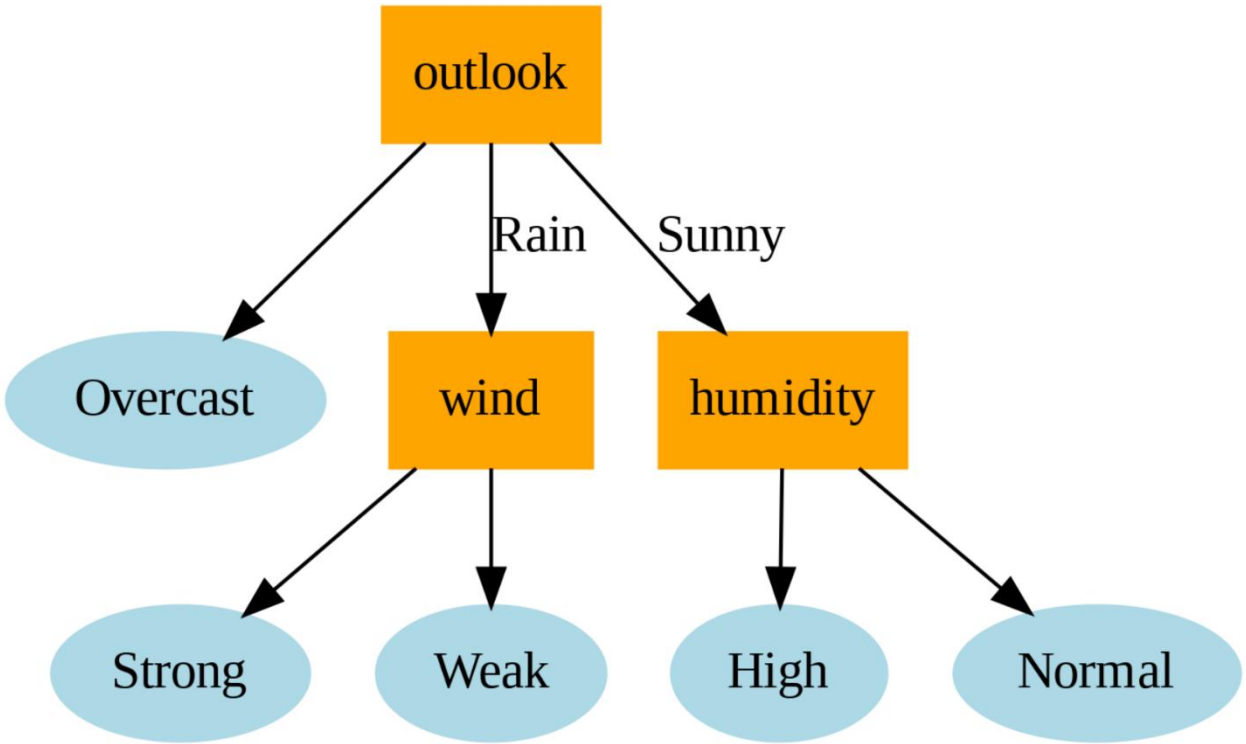
OUTPUT

```
Choose Files play_tennis.csv
• play_tennis.csv(text/csv) - 470 bytes, last modified: 12/20/2024 - 100% done
Saving play_tennis.csv to play_tennis (6).csv
     outlook  temp humidity    wind play
0      Sunny   Hot     High    Weak   No
1      Sunny   Hot     High  Strong   No
2   Overcast   Hot     High    Weak  Yes
3       Rain  Mild     High    Weak  Yes
4       Rain  Cool   Normal    Weak  Yes
5       Rain  Cool   Normal  Strong   No
6   Overcast  Cool   Normal  Strong  Yes
7      Sunny  Mild     High    Weak   No
8      Sunny  Cool   Normal    Weak  Yes
9       Rain  Mild   Normal    Weak  Yes
10     Sunny  Mild   Normal  Strong  Yes
11  Overcast  Mild     High  Strong  Yes
12  Overcast   Hot   Normal    Weak  Yes
13      Rain  Mild     High  Strong   No
```

```
Decision Tree:
 {'outlook': {'Overcast': 'Yes', 'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}

Sample Prediction: Yes
```

DECISION TREE



EXPLANATION:

The code implements a decision tree classifier using the ID3 algorithm, trains it on a dataset, visualizes the decision tree, and makes predictions based on the tree.

1. **Data Upload and Preprocessing**:
    - The dataset is uploaded from the user's local machine, read into a pandas DataFrame, and a column (`day`) is dropped for analysis.
2. **Entropy Calculation**:
    - A function `entropy(s)` is defined to calculate the entropy (a measure of uncertainty) of a set `s`. It uses the Shannon entropy formula.
3. **Information Gain Calculation**:
    - The `information_gain(data, split_attribute, target_attribute)` function calculates the information gain for a given feature (split attribute) in the dataset. Information gain measures how much uncertainty is reduced when splitting the dataset on that feature.
4. **ID3 Algorithm**:
    - The `id3` function implements the ID3 decision tree algorithm. It recursively splits the dataset by selecting the feature that provides the highest information gain, creating a tree-like structure. The tree is built by:
        - Checking if the data is pure (i.e., all instances belong to the same class).
        - If not, the best feature is selected based on information gain, and the dataset is split accordingly.
        - This process repeats until a stopping condition is met (e.g., all data is pure or no features are left).
5. **Visualization of the Decision Tree**:
    - The `draw_tree` function visualizes the decision tree using the `graphviz` library. It creates a graphical representation of the tree, where each node represents a decision, and edges represent the outcomes of that decision.
6. **Prediction**:
    - The `predict` function makes predictions based on the trained decision tree. It traverses the tree according to the values of a given sample and outputs the predicted class.
7. **Testing the Model**:
    - A sample is created (with attributes like `outlook`, `temp`, `humidity`, and `wind`), and the decision tree is used to predict whether the sample will lead to a positive or negative outcome for the target attribute (`play`).
8. **Displaying the Decision Tree**:
    - After training the tree, it is drawn and saved as a PNG image. The image is then displayed inline using IPython's `Image` function.