Sardar Patel Institute of Technology, Mumbai
Department of Electronics and Telecommunication Engineering
T.E.-CSE (2023-2024)
PC 307 - Machine Learning

Name: Deepali Daga **Date:19/3/2024**

Experiment 5: DECISION TREE

AIM: Write Python program to demonstrate the working of the decision tree based ID3 algorithm by using appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Outcomes:

1. Find entropy of data and follow steps of the algorithm to construct a tree.
2. Representation of hypothesis using decision tree.
3. Apply Decision Tree algorithm to classify the given data.
4. Interpret the output of Decision Tree.

System Requirements:

Linux OS with Python and libraries or R or windows with MATLAB

Theory:

The decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Entropy
A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

E(S) is the Entropy of the entire set, while the second term E(S, A) relates to an Entropy of an attribute A.

$$E(S) = \sum_{x \in X} -P(x) \log_2 P(x)$$

$$E(S, A) = \sum_{x \in X} [P(x) * E(S)]$$

Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

$$IG(S, A) = E(S) - E(S, A)$$

Dataset:

```
In [4]: data
```

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rainy	Mild	High	Weak	Yes
4	Rainy	Cool	Normal	Weak	Yes
5	Rainy	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rainy	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rainy	Mild	High	Strong	No

Code:

Part 1 : Entropy

```
import numpy as np

import pandas as pd

def entropy(column):

    values, counts = np.unique(column, return_counts=True)

    probabilities = counts / np.sum(counts)

    entropy_val = -np.sum(probabilities * np.log2(probabilities))

    return entropy_val


data = pd.read_csv("exp5.csv")

data

# Calculate entropy for the target variable (PlayTennis)
```

```
target_entropy = entropy(data['PlayTennis'])
print("Entropy of target variable (PlayTennis):", target_entropy)
def conditional_entropy(data, attribute, target_attribute):
    # Calculate the conditional entropy of an attribute given the target attribute
    conditional_entropy_val = 0
    for value in data[attribute].unique():
        subset = data[data[attribute] == value]
        subset_entropy = entropy(subset[target_attribute])
        conditional_probability = len(subset) / len(data)
        conditional_entropy_val += conditional_probability * subset_entropy
        print("Conditional entropy of", attribute, "|", target_attribute, "=", value, ":", subset_entropy)
    return conditional_entropy_val

def information_gain(data, attribute, target_attribute):
    # Calculate the information gain of an attribute
    attribute_entropy = entropy(data[target_attribute])
    conditional_entropy_val = conditional_entropy(data, attribute, target_attribute)
    information_gain_val = attribute_entropy - conditional_entropy_val
    return information_gain_val
# Calculate information gain for all predictor attributes
target_attribute = 'PlayTennis'
information_gains = {}
for column in data.columns[:-1]:  # Exclude the last column (target variable)
    information_gain_val = information_gain(data, column, target_attribute)
    information_gains[column] = information_gain_val
    print("Information gain of", column, ":", information_gain_val)
root_attribute = max(data.columns[:-1], key=lambda col: information_gain(data, col, target_attribute))
print("Root Node Attribute:", root_attribute)

def decision_tree(data, target_attribute):
```

```
    root_node = {}

    if len(data[target_attribute].unique()) == 1:
        return data[target_attribute].iloc[0]

    # Find the attribute with the highest information gain
    best_attribute = max(data.columns[:-1], key=lambda col: information_gain(data, col, target_attribute))
    root_node['attribute'] = best_attribute
    root_node['branches'] = {}

    # Split the dataset based on the chosen attribute
    for value in data[best_attribute].unique():
        subset = data[data[best_attribute] == value]
        root_node['branches'][value] = decision_tree(subset.drop(columns=[best_attribute]), target_attribute)

    return root_node

def print_decision_tree(decision_tree, indent=''):
    if 'attribute' in decision_tree:
        print(indent + decision_tree['attribute'])
        for value, subtree in decision_tree['branches'].items():
            print(indent + '  ' + value + ':')
            print_decision_tree(subtree, indent + '    ')
    else:
        print(indent + decision_tree)

# Print all information gains
print("Information Gain for each attribute:")
for column in data.columns[:-1]:
    ig = information_gain(data, column, target_attribute)
    print(f"{column}: {ig}")
```

```
# Iterate over each unique value of the root node attribute
for root_node_value in data[root_attribute].unique():
    # Reduce the dataset based on the root node attribute value
    reduced_data = data[data[root_attribute] == root_node_value]

#     # Build the decision tree for the reduced dataset
#     decision_tree_root = decision_tree(reduced_data, target_attribute)

#     # Print the decision tree for the current node value of the root attribute
#     print(f"Decision Tree for {root_attribute} = {root_node_value}:")
#     print_decision_tree(decision_tree_root)
    print()
```

Output:

```
In [4]: data
```

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rainy	Mild	High	Weak	Yes
4	Rainy	Cool	Normal	Weak	Yes
5	Rainy	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rainy	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rainy	Mild	High	Strong	No

```
In [5]: # Calculate entropy for the target variable (PlayTennis)
        target_entropy = entropy(data['PlayTennis'])
        print("Entropy of target variable (PlayTennis):", target_entropy)

Entropy of target variable (PlayTennis): 0.9402859586706311
```

```
In [7]:
# Calculate information gain for all predictor attributes
target_attribute = 'PlayTennis'
information_gains = {}
for column in data.columns[:-1]:  # Exclude the last column (target variable)
    information_gain_val = information_gain(data, column, target_attribute)
    information_gains[column] = information_gain_val
    print("Information gain of", column, ":", information_gain_val)
```

```
Conditional entropy of Outlook | PlayTennis = Sunny : 0.9709505944546686
Conditional entropy of Outlook | PlayTennis = Overcast : -0.0
Conditional entropy of Outlook | PlayTennis = Rainy : 0.9709505944546686
Information gain of Outlook : 0.24674981977443933
Conditional entropy of Temperature | PlayTennis = Hot : 1.0
Conditional entropy of Temperature | PlayTennis = Mild : 0.9182958340544896
Conditional entropy of Temperature | PlayTennis = Cool : 0.8112781244591328
Information gain of Temperature : 0.02922256565895487
Conditional entropy of Humidity | PlayTennis = High : 0.9852281360342515
Conditional entropy of Humidity | PlayTennis = Normal : 0.5916727785823275
Information gain of Humidity : 0.15183550136234159
Conditional entropy of Wind | PlayTennis = Weak : 0.8112781244591328
Conditional entropy of Wind | PlayTennis = Strong : 1.0
Information gain of Wind : 0.04812703040826949
```

```
In [8]:
root_attribute = max(data.columns[:-1], key=lambda col: information_gain(data, col, target_attribute))
print("Root Node Attribute:", root_attribute)
```

```
Conditional entropy of Outlook | PlayTennis = Sunny : 0.9709505944546686
Conditional entropy of Outlook | PlayTennis = Overcast : -0.0
Conditional entropy of Outlook | PlayTennis = Rainy : 0.9709505944546686
Conditional entropy of Temperature | PlayTennis = Hot : 1.0
Conditional entropy of Temperature | PlayTennis = Mild : 0.9182958340544896
Conditional entropy of Temperature | PlayTennis = Cool : 0.8112781244591328
Conditional entropy of Humidity | PlayTennis = High : 0.9852281360342515
Conditional entropy of Humidity | PlayTennis = Normal : 0.5916727785823275
Conditional entropy of Wind | PlayTennis = Weak : 0.8112781244591328
Conditional entropy of Wind | PlayTennis = Strong : 1.0
Root Node Attribute: Outlook
```

```
Information Gain for each attribute:
Conditional entropy of Outlook | PlayTennis = Sunny : 0.9709505944546686
Conditional entropy of Outlook | PlayTennis = Overcast : -0.0
Conditional entropy of Outlook | PlayTennis = Rainy : 0.9709505944546686
Outlook: 0.24674981977443933
Conditional entropy of Temperature | PlayTennis = Hot : 1.0
Conditional entropy of Temperature | PlayTennis = Mild : 0.9182958340544896
Conditional entropy of Temperature | PlayTennis = Cool : 0.8112781244591328
Temperature: 0.02922256565895487
Conditional entropy of Humidity | PlayTennis = High : 0.9852281360342515
Conditional entropy of Humidity | PlayTennis = Normal : 0.5916727785823275
Humidity: 0.15183550136234159
Conditional entropy of Wind | PlayTennis = Weak : 0.8112781244591328
Conditional entropy of Wind | PlayTennis = Strong : 1.0
Wind: 0.04812703040826949
Conditional entropy of Outlook | PlayTennis = Sunny : 0.9709505944546686
Conditional entropy of Temperature | PlayTennis = Hot : -0.0
Conditional entropy of Temperature | PlayTennis = Mild : 1.0
Conditional entropy of Temperature | PlayTennis = Cool : -0.0
Conditional entropy of Humidity | PlayTennis = High : -0.0
Conditional entropy of Humidity | PlayTennis = Normal : -0.0
Conditional entropy of Wind | PlayTennis = Weak : 0.9182958340544896
Conditional entropy of Wind | PlayTennis = Strong : 1.0
```

Part 2 : Gini Index

Code :

```
import pandas as pd

class Node:
    def __init__(self, attribute=None, value=None, result=None):
        self.attribute = attribute  # Attribute to split on
        self.value = value  # Value of the attribute
        self.result = result  # Result if this is a leaf node
        self.children = {}  # Dictionary to store child nodes

def calculate_gini_index(data, attribute, target):
    gini_index = 0.0
    values = data[attribute].unique()

    # Calculate Gini index for each value of the attribute
    for value in values:
        subset = data[data[attribute] == value]
        prob = len(subset) / len(data)

        # Calculate the probability of each class in the subset
        class_prob = subset[target].value_counts() / len(subset)

        # Calculate the Gini index for the subset
        gini = 1 - sum(class_prob ** 2)

        # Weighted sum of Gini index
```

```
        gini_index += prob * gini

    return gini_index

def build_tree(data, max_depth, depth=0):
    # Check if data is pure or max depth is reached
    if len(data['PlayTennis'].unique()) == 1 or depth == max_depth:
        return Node(result=data['PlayTennis'].iloc[0])

    # Get attributes and calculate Gini index for each
    attributes = data.columns[:-1]
    gini_indices = {}
    for attribute in attributes:
        gini_index = calculate_gini_index(data, attribute, 'PlayTennis')
        gini_indices[attribute] = gini_index

    # Choose attribute with lowest Gini index
    best_split_attribute = min(gini_indices, key=gini_indices.get)
    node = Node(attribute=best_split_attribute)

    # Split data based on chosen attribute
    for value in data[best_split_attribute].unique():
        subset = data[data[best_split_attribute] == value]
        node.children[value] = build_tree(subset.drop(columns=[best_split_attribute]), max_depth, depth+1)

    return node

def print_tree(node, depth=0):
    if node.result is not None:
        print(f"{'  '*depth}Result: {node.result}")
    else:
```

```
        print(f"{'  '*depth}{node.attribute}:")
        for value, child_node in node.children.items():
            print(f"{'  '*(depth+1)}{value}")
            print_tree(child_node, depth+2)

# Load the dataset
data = pd.read_csv('exp5.csv')

# Build the decision tree iteratively
max_depth = 4
for i in range(max_depth):
    print(f"Iteration {i+1}:")

    # Calculate Gini index for each attribute
    attributes = data.columns[:-1]  # Exclude the target variable
    gini_indices = {}
    for attribute in attributes:
        gini_index = calculate_gini_index(data, attribute, 'PlayTennis')
        gini_indices[attribute] = gini_index

    # Print Gini index for each attribute
    for attribute, gini_index in gini_indices.items():
        print(f"Gini index for {attribute}: {gini_index:.3f}")

    # Build decision tree
    root_node = build_tree(data, max_depth=i+1)

    # Print decision tree
    print("Decision Tree:")
    print_tree(root_node)
    print()
```

```
    # Reduce dataset based on the tree
    current_node = root_node
    while current_node.children:
        attribute = current_node.attribute
        value = next(iter(current_node.children))
        data = data[data[attribute] == value]
        current_node = current_node.children[value]

    if len(data['PlayTennis'].unique()) == 1:
        print(f"Reached pure leaf node. Stopping iterations.")
        break

    # Print reduced dataset
    print("Reduced Dataset:")
    print(data)
    print()
```

Output:

```
Iteration 1:
Gini index for Outlook: 0.343
Gini index for Temperature: 0.440
Gini index for Humidity: 0.367
Gini index for Wind: 0.429
Decision Tree:
Outlook:
  Sunny
    Result: No
  Overcast
    Result: Yes
  Rainy
    Result: Yes

Reduced Dataset:
   Outlook Temperature Humidity    Wind PlayTennis
0    Sunny         Hot     High    Weak         No
1    Sunny         Hot     High  Strong         No
7    Sunny        Mild     High    Weak         No
8    Sunny        Cool   Normal    Weak        Yes
10   Sunny        Mild   Normal  Strong        Yes
```

```
Iteration 2:
Gini index for Outlook: 0.480
Gini index for Temperature: 0.200
Gini index for Humidity: 0.000
Gini index for Wind: 0.467
Decision Tree:
Humidity:
  High
    Result: No
  Normal
    Result: Yes

Reached pure leaf node. Stopping iterations.
```

Conclusion:

In this experiment we performed the working of the decision tree based ID3 algorithm by using a data set for building the decision tree.

Decision Tree algorithms, often used for classification tasks, rely on metrics like Gini Index to determine the best attributes for splitting data. Gini Index is computationally more feasible than ID3 algorithm While effective in producing interpretable models, the computational complexity of Decision Trees and Gini Index calculations can grow with larger datasets, impacting both training time and computational resources.