# Decision Tree Classification

## ID3 Algorithm

Name: Debatreya Das Roll No.: 12212070 CS A4 Lab 10

### Importing Necessary Libraries

import pandas as pd  
import numpy as np

### Loading the Dataset

df = pd.read\_csv('./id3.csv')  
  
df

Outlook Temperature Humidity Wind Answer  
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

### HELPER FUNCTIONS

#### 1. Entropy(S)

Considering the last column as the target. Calculate the Entropy of the Example Set S

def Entropy(data):  
 # Count positive and negative examples in the target column  
 target = data.iloc[:, -1] # Assuming target is the last column  
 values, counts = np.unique(target, return\_counts=True)  
 probabilities = counts / counts.sum()  
   
 # Calculate entropy  
 entropy = -np.sum(probabilities \* np.log2(probabilities))  
 return entropy

#### 2. Gain(S, A)

Calcualte the Information gain when feature A is selected in dataset S

def Gain(data, feature):  
 # Calculate the entropy of the whole dataset  
 total\_entropy = Entropy(data)  
   
 # Get the values and the counts of the split for the given feature  
 values, counts = np.unique(data[feature], return\_counts=True)  
   
 # Calculate weighted entropy after the split  
 weighted\_entropy = 0  
 for i, value in enumerate(values):  
 subset = data[data[feature] == value]  
 subset\_entropy = Entropy(subset)  
 weighted\_entropy += (counts[i] / counts.sum()) \* subset\_entropy  
   
 # Information gain is the reduction in entropy  
 info\_gain = total\_entropy - weighted\_entropy  
 return info\_gain

#### 3. Count Positive and Negative Examples

def count\_positive\_negative(data):  
 target = data.iloc[:, -1] # Assuming target is the last column  
 positive\_count = (target == "Yes").sum()  
 negative\_count = (target == "No").sum()  
 return positive\_count, negative\_count

### DecisionTreeID3 CLASS

The decision tree recursively selects the attribute with the highest information gain at each step and continues to split the dataset until a stopping condition is met (e.g., all examples are classified or no attributes are left).

class DecisionTreeID3:  
 def \_\_init\_\_(self):  
 self.tree = {}  
  
 def fit(self, data, original\_data=None, features=None, parent\_node\_class=None):  
 if features is None:  
 features = data.columns[:-1] # All features except the target column  
   
 if original\_data is None:  
 original\_data = data  
   
 # If all examples have the same label, return this label (leaf node)  
 if len(np.unique(data.iloc[:, -1])) <= 1:  
 return np.unique(data.iloc[:, -1])[0]  
   
 # If no more features to split, return the majority class of the parent node  
 elif len(features) == 0:  
 return parent\_node\_class  
   
 # Otherwise, grow the tree  
 else:  
 # Count positive and negative examples  
 positive\_count, negative\_count = count\_positive\_negative(data)  
   
 # Select the majority class as the default class  
 parent\_node\_class = "Yes" if positive\_count >= negative\_count else "No"  
   
 # Calculate the information gain for each feature  
 gains = {feature: Gain(data, feature) for feature in features}  
   
 # Select the feature with the highest information gain  
 best\_feature = max(gains, key=gains.get)  
   
 # Build the tree  
 tree = {best\_feature: {}}  
   
 # Remove the feature from the list of available features  
 remaining\_features = [feat for feat in features if feat != best\_feature]  
   
 # Split the data based on the best feature  
 for value in np.unique(data[best\_feature]):  
 subset = data[data[best\_feature] == value]  
   
 # Recursively build the subtree  
 subtree = self.fit(subset, original\_data, remaining\_features, parent\_node\_class)  
   
 # Assign the subtree to the current tree node  
 tree[best\_feature][value] = subtree  
   
 self.tree = tree  
 return tree  
  
 def predict(self, query):  
 tree = self.tree  
 while isinstance(tree, dict):  
 feature = list(tree.keys())[0]  
 value = query[feature]  
 tree = tree[feature].get(value, "Unknown") # Default to "Unknown" if the value is not found  
 return tree

import json  
  
def print\_tree(tree):  
 # Use json.dumps to format the dictionary with indentation  
 formatted\_tree = json.dumps(tree, indent=4)  
 # Print the formatted tree with single quotes instead of double quotes  
 formatted\_tree = formatted\_tree.replace('"', "'")  
 print(formatted\_tree)

### Instantiate and train the ID3 decision tree

tree = DecisionTreeID3()  
decision\_tree = tree.fit(df)  
  
# Example usage of print\_tree  
print("Decision Tree:")  
print\_tree(decision\_tree)

Decision Tree:  
{  
 'Outlook': {  
 'overcast': 'yes',  
 'rain': {  
 'Wind': {  
 'strong': 'no',  
 'weak': 'yes'  
 }  
 },  
 'sunny': {  
 'Humidity': {  
 'high': 'no',  
 'normal': 'yes'  
 }  
 }  
 }  
}

### Make Prediction on a given SAMPLE

# Example query to predict the outcome  
query = {  
 'Outlook': 'sunny',  
 'Temperature': 'cool',  
 'Humidity': 'high',  
 'Wind': 'strong'  
}  
  
# Make prediction  
prediction = tree.predict(query)  
print(f"Prediction for {query}: {prediction}")

Prediction for {'Outlook': 'sunny', 'Temperature': 'cool', 'Humidity': 'high', 'Wind': 'strong'}: no