Instruction Manual 8

Question 1: Solution:

Attribute subset selection using the ID3 algorithm. Let take following data to compute maximum information gain.

Attribute A1 (Outlooks)	Attribute A2 (Temperature)	Attribute A3 (Humidity)	Class
Sunny	Hot	High	No
Sunny	Hot	High	No
Overcast	Hot	High	Yes
Rainy	Mild	High	Yes
Rainy	Cool	Normal	Yes
Overcast	Cool	Normal	Yes
Sunny	Mild	Normal	Yes
Sunny	Cool	High	No
Rainy	Mild	Normal	Yes
Sunny	Mild	Normal	Yes
Overcast	Mild	High	Yes
Overcast	Hot	Normal	Yes
Rainy	Mild	High	No

With ID3 algorithms , select an attribute subset that best predicts the class labels.

Step 1: Calculate the Entropy of the Whole Dataset

To calculate information gain for attribute selection, we first calculate the entropy of the whole dataset:

```
Entropy(S) = -p(Yes) * log2(p(Yes)) - p(No) * log2(p(No))
           = -9/13 * log2(9/13) - 4/13 * log2(4/13)
           ≈ 0.940
```

Step 2: Calculate Information Gain for Each Attribute

Information Gain for A1 (Outlook):

Split the dataset based on "Outlook" attribute:

- Outlook = Sunny: 5 examples (2 Yes, 3 No)
- Outlook = Overcast: 4 examples (4 Yes)
- Outlook = Rainy: 4 examples (3 Yes, 1 No)

calculate the information gain for each attribute by partitioning the dataset based on each attribute's values and then calculating the weighted entropy.

```
Entropy(Outlook = Sunny) = -2/5 * log2(2/5) - 3/5 * log2(3/5) ≈ 0.971
Entropy(Outlook = Overcast) = 0 (pure subset)
Entropy(Outlook = Rainy) = -3/4 * log2(3/4) - 1/4 * log2(1/4) ≈ 0.811

Weighted Entropy(Outlook) = (5/13) * Entropy(Outlook = Sunny) + (4/13) *
Entropy(Outlook = Overcast) + (4/13) * Entropy(Outlook = Rainy) ≈ 0.892
```

`Information Gain for A1 = Entropy(S) - Weighted Entropy(Outlook) ≈ 0.048`

Information Gain for A2 (Temperature):

Split the dataset based on "Temperature" attribute:

Temperature = Hot: 4 examples (2 Yes, 2 No)

Temperature = Mild: 6 examples (4 Yes, 2 No)

Temperature = Cool: 3 examples (3 Yes)

```
Entropy(Temperature = Hot) = 1 (impure subset)
Entropy(Temperature = Mild) = -2/6 * log2(2/6) - 4/6 * log2(4/6) ≈ 0.918
Entropy(Temperature = Cool) = 0 (pure subset)

Weighted Entropy(Temperature) = (4/13) * 1 + (6/13) * 0.918 + (3/13) * 0 ≈
0.654
```

`Information Gain for A2(Temperature) =`

`Entropy(S) - Weighted Entropy(Temperature)`

`=0.940-0.654`

`≈ 0.286`

Information Gain for A3 (Humidity):

Split the dataset based on "Humidity" attribute:

Humidity = High: 7 examples (3 Yes, 4 No)

Humidity = Normal: 6 examples (6 Yes)

```
Weighted Entropy(Outlook) = (5/13) * Entropy(Outlook = Sunny) + (4/13) *
Entropy(Outlook = Overcast) + (4/13) * Entropy(Outlook = Rainy) ≈ 0.892

Entropy(Humidity = High) = -3/7 * log2(3/7) - 4/7 * log2(4/7) ≈ 0.985
Entropy(Humidity = Normal) = 0 (pure subset)
```

```
Weighted Entropy(Humidity) = (7/13) * 0.985 + (6/13) * 0 ≈ 0.452
```

Information Gain for A3 = Entropy(S) - Weighted Entropy(Humidity) ≈ 0.4882

Step 3: Attribute Selection

The attribute with the highest information gain is A3 (Humidity), so we select it as the root node of our decision tree.

Problem 2: Python Code:

```python
import numpy as np
import pandas as pd

# Sample dataset
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Overcast',
'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild',
'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',
'High', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Class': ['No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
'Yes', 'Yes', 'Yes', 'No']
})

# Function to calculate entropy
def entropy(class_labels):
    unique_labels, counts = np.unique(class_labels, return_counts=True)
    prob = counts / len(class_labels)
    entropy = -np.sum(prob * np.log2(prob))
    return entropy

# Function to calculate information gain
def information_gain(data, attribute, class_label):
    total_entropy = entropy(data[class_label])
    unique_values = data[attribute].unique()
    weighted_entropy = 0

    for value in unique_values:
        subset = data[data[attribute] == value]
        subset_entropy = entropy(subset[class_label])
        weight = len(subset) / len(data)
        weighted_entropy += weight * subset_entropy

    return total_entropy - weighted_entropy
```

```
# ID3 algorithm for attribute selection
def id3(data, class_label, attributes):
    if len(attributes) == 0:
        # If no attributes are left, return the majority class
        return data[class_label].mode().iloc[0]

    unique_classes = data[class_label].unique()
    if len(unique_classes) == 1:
        # If all examples have the same class, return that class
        return unique_classes[0]

    best_attribute = max(attributes, key=lambda attr: information_gain(data,
attr, class_label))
    tree = {best_attribute: {}}

    for value in data[best_attribute].unique():
        subset = data[data[best_attribute] == value]
        if len(subset) == 0:
            # If the subset is empty, return the majority class
            tree[best_attribute][value] = data[class_label].mode().iloc[0]
        else:
            new_attributes = [attr for attr in attributes if attr !=
best_attribute]
            tree[best_attribute][value] = id3(subset, class_label,
new_attributes)

    return tree

# Define the class label and attributes
class_label = 'Class'
attributes = ['Outlook', 'Temperature', 'Humidity']

# Build the ID3 decision tree
decision_tree = id3(data, class_label, attributes)

# Print the decision tree
import pprint
pprint.pprint(decision_tree)
```

OUTPUT:

```
 PROBLEM  3: Classification of Data(IRIS SAMPLE DATA)with Decision Tree Algo

from sklearn import datasets

import pandas as pd
import numpy as np
from sklearn import metrics
#from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt #if not install then pip install from terninal
-- it
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

iris = datasets.load_iris() #Loading the dataset
iris.keys()

#print(iris.keys)
dict_keys=(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names', 'filename', 'data_module'])
#print(dict_keys)
#convert dataset to pandas df
iris = pd.DataFrame(
    data= np.c_[iris['data'], iris['target']],
    columns= iris['feature_names'] + ['target']
    )
# print(iris)
# he=head(iris)
# print(he)
# print(iris.head())
# To give name to target class as species, Species is a list
species = []

for i in range(len(iris['target'])):
    if iris['target'][i] == 0:
        species.append("setosa")
    elif iris['target'][i] == 1:
        species.append('versicolor')
    else:
        species.append('virginica')
iris['species'] = species
# print("Species have class now", iris['species'])
#To check new data set is taken place or not then used once again tail/head
command
# print(iris.tail())
# Do observe total sample of each species
# print( iris.groupby('species').size())
#Plotting a dataset is a great way to explore its distribution.
#  Plotting the iris dataset can be done using matplotlib,
# a Python library for 2D plotting.
#import library matplotlib
setosa = iris[iris.species == "setosa"]
versicolor = iris[iris.species=='versicolor']
virginica = iris[iris.species=='virginica']
fig, ax = plt.subplots()
```

```
fig.set_size_inches(13, 7) # adjusting the length and width of plot
# lables and scatter points
ax.scatter(setosa['petal length (cm)'], setosa['petal width (cm)'],
label="Setosa", facecolor="blue")
ax.scatter(versicolor['petal length (cm)'], versicolor['petal width (cm)'],
label="Versicolor", facecolor="green")
ax.scatter(virginica['petal length (cm)'], virginica['petal width (cm)'],
label="Virginica", facecolor="red")

ax.set_xlabel("petal length (cm)")
ax.set_ylabel("petal width (cm)")
ax.grid()
ax.set_title("Iris petals")
ax.legend()
# To display image here (optional)
# plt.show()

# Split the data into features (X) and target labels (y)
X = iris.drop('species', axis=1)
y = iris['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Create and Train the Decision Tree Classifier:

# Create an instance of DecisionTreeClassifier and train it on the training
data using the fit() method.
# Create a DecisionTreeClassifier instance

classifier = DecisionTreeClassifier(random_state=42)
# This library need to import--->from sklearn.tree import
DecisionTreeClassifier
# Train the classifier on the training data
classifier.fit(X_train, y_train)

# Now Do Prediction
#Make Predictions:
#Use the trained classifier to make predictions on new,
# unseen data (testing set) using the predict() method.
# Make predictions on the testing data
y_pred = classifier.predict(X_test)
x_pred=classifier.predict(X_train)
'''Evaluate the Model:
Calculate evaluation metrics to assess the performance
of the model. For classification tasks, you can use metrics
like accuracy, precision, recall, F1-score, and confusion matrix.
'''
```

```
''' Compute Training  time Accuracy'''
# Calculate accuracy
accuracy = accuracy_score(y_train, x_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:")
print(classification_report(y_train, x_pred))

# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_train, x_pred))

''' Compute Testing time Accuracy'''


print("Testing Time Accuracy")

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Testing:", accuracy)

# Display classification report
print("Classification Report Testing:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```