Nithin S
221IT085

IT350 Lab Assignment 3

Q1. For the below dataset, construct a decision tree for classifying a new test record using
information gain measure.
Also, classify the below test records using the constructed decision tree:
a. Outlook = Rainy, Temp = Cool, Humidity = High, Windy = True
b. Outlook = Sunny, Temp = Mild, Humidity = Normal, Windy = False

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

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

X = data.drop(columns=["Play Golf"])
y = data["Play Golf"]

def entropy(y):
    values, counts = np.unique(y, return_counts=True)
    probabilities = counts / counts.sum()
    return -np.sum(probabilities * np.log2(probabilities))

def information_gain(X, y, feature):
    total_entropy = entropy(y)
    values, counts = np.unique(X[feature], return_counts=True)
    weighted_entropy = sum(
        (counts[i] / np.sum(counts)) * entropy(y[X[feature] == values[i]])
        for i in range(len(values))
    )
    return total_entropy - weighted_entropy

def best_split(X, y):
    return max(X.columns, key=lambda col: information_gain(X, y, col))

def build_tree(X, y):
    if len(np.unique(y)) == 1:
        return y.iloc[0]

    if X.empty:
        return y.mode()[0]

    best_feature = best_split(X, y)
    tree = {best_feature: {}}

    for value in np.unique(X[best_feature]):
        subset_X = X[X[best_feature] == value].drop(columns=[best_feature])
        subset_y = y[X[best_feature] == value]
        tree[best_feature][value] = build_tree(subset_X, subset_y)

    return tree
```

```python
decision_tree = build_tree(X, y)


def predict(tree, sample, default_class="Yes"):
    if not isinstance(tree, dict):
        return tree
    feature = next(iter(tree))
    value = sample.get(feature)
    if value in tree[feature]:
        return predict(tree[feature][value], sample, default_class)
    return default_class  # Default to majority class


test_records = [
    {"Outlook": "Rainy", "Temp": "Cool", "Humidity": "High", "Windy": "True"},
    {"Outlook": "Sunny", "Temp": "Mild", "Humidity": "Normal", "Windy": "False"}
]


majority_class = y.mode()[0]  # Get the majority class in the dataset
predictions = [predict(decision_tree, record, majority_class) for record in test_records]

for i, test in enumerate(test_records):
    print(f"Test Record {chr(97 + i)}: {test} -> {predictions[i]}")

y_pred = [predict(decision_tree, X.iloc[i].to_dict(), y.mode()[0]) for i in range(len(X))]

unique_classes = np.unique(y)
confusion_matrix = {c1: {c2: 0 for c2 in unique_classes} for c1 in unique_classes}

for actual, predicted in zip(y, y_pred):
    confusion_matrix[actual][predicted] += 1

print("Confusion Matrix:")
print("\t" + "\t".join(unique_classes))
for actual in unique_classes:
    print(f"{actual}\t" + "\t".join(str(confusion_matrix[actual][pred]) for pred in unique_classes))

# Compute accuracy
accuracy = sum(confusion_matrix[c][c] for c in unique_classes) / len(y)
print(f"Accuracy: {accuracy:.2f}")
```

```
Test Record a: {'Outlook': 'Rainy', 'Temp': 'Cool', 'Humidity': 'High', 'Windy': 'True'} -> No
Test Record b: {'Outlook': 'Sunny', 'Temp': 'Mild', 'Humidity': 'Normal', 'Windy': 'False'} -> Yes
Confusion Matrix:
        No      Yes
No      5       0
Yes     0       9
Accuracy: 1.00
```

Q2. 2. For the below dataset, construct a decision tree for classifying a new test record using
information gain measure.
Also, classify the below test records using the constructed decision tree:
a. Industry = Urban, JobType = Sales, Income = Low, Previous Customer = Yes
b. Industry = Electronics, JobType = Engineering, Income = High, Previous Customer = No

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

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

X = data.drop(columns=["Class"])
y = data["Class"]

def entropy(y):
    values, counts = np.unique(y, return_counts=True)
    probabilities = counts / counts.sum()
    return -np.sum(probabilities * np.log2(probabilities))

def information_gain(X, y, feature):
    total_entropy = entropy(y)
    values, counts = np.unique(X[feature], return_counts=True)
    weighted_entropy = sum(
        (counts[i] / np.sum(counts)) * entropy(y[X[feature] == values[i]])
        for i in range(len(values))
    )
    return total_entropy - weighted_entropy

def best_split(X, y):
    return max(X.columns, key=lambda col: information_gain(X, y, col))

def build_tree(X, y):
    if len(np.unique(y)) == 1:
        return y.iloc[0]

    if X.empty:
        return y.mode()[0]

    best_feature = best_split(X, y)
    tree = {best_feature: {}}

    for value in np.unique(X[best_feature]):
        subset_X = X[X[best_feature] == value].drop(columns=[best_feature])
        subset_y = y[X[best_feature] == value]
        tree[best_feature][value] = build_tree(subset_X, subset_y)

    return tree
```

```python
decision_tree = build_tree(X, y)

def predict(tree, sample, default_class="YES"):
    if not isinstance(tree, dict):
        return tree
    feature = next(iter(tree))
    value = sample.get(feature)
    if value in tree[feature]:
        return predict(tree[feature][value], sample, default_class)
    return default_class  # Default to majority class

test_records = [
    {"Industry": "Urban", "JobType": "Sales", "Income": "Low", "PreviousCustomer": "Yes"},
    {"Industry": "Electronics", "JobType": "Engineering", "Income": "High", "PreviousCustomer": "No"},
]

majority_class = y.mode()[0]  # Get the majority class in the dataset
predictions = [predict(decision_tree, record, majority_class) for record in test_records]

for i, test in enumerate(test_records):
    print(f"Test Record {chr(97 + i)}: {test} -> {predictions[i]}")

y_pred = [predict(decision_tree, X.iloc[i].to_dict(), y.mode()[0]) for i in range(len(X))]

unique_classes = np.unique(y)
confusion_matrix = {c1: {c2: 0 for c2 in unique_classes} for c1 in unique_classes}

for actual, predicted in zip(y, y_pred):
    confusion_matrix[actual][predicted] += 1

print("Confusion Matrix:")
print("\t" + "\t".join(unique_classes))
for actual in unique_classes:
    print(f"{actual}\t" + "\t".join(str(confusion_matrix[actual][pred]) for pred in unique_classes))

accuracy = sum(confusion_matrix[c][c] for c in unique_classes) / len(y)
print(f"Accuracy: {accuracy:.2f}")
```

```
Test Record a: {'Industry': 'Urban', 'JobType': 'Sales', 'Income': 'Low', 'PreviousCustomer': 'Yes'} -> NO
Test Record b: {'Industry': 'Electronics', 'JobType': 'Engineering', 'Income': 'High', 'PreviousCustomer': 'No'} -> NO
Confusion Matrix:
        NO      YES
NO      7       0
YES     0       7
Accuracy: 1.00
```