Lab 3: Introducing Classification

Objectives:

- To gain hands-on experience classifying small dataset
- To implement concepts related to Decision Tree classifier (i.e. Entropy, Information Gain), along with the Decision Tree algorithm

```
# Run this cell if you use Colab
#df = drive.mount('/content/drive/MyDrive/toy_data.csv')
```

```
import pandas as pd

# Read the data
df = pd.read_csv('toy_data.csv')
df
```

	age	income	student	credit rating	buys computer
0	<=30	high	no	fair	no
1	<=30	high	no	excellent	no
2	31-40	high	no	fair	yes
3	>40	medium	no	fair	yes
4	>40	low	yes	fair	yes
5	>40	low	yes	excellent	no
6	31-40	low	yes	excellent	yes
7	<=30	medium	no	fair	no
8	<=30	low	yes	fair	yes
9	>40	medium	yes	fair	yes
10	<=30	medium	yes	excellent	yes
11	31-40	medium	no	excellent	yes
12	31-40	high	yes	fair	yes
13	>40	medium	no	excellent	no

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   age            14 non-null     object
 1   income         14 non-null     object
 2   student        14 non-null     object
 3   credit rating  14 non-null     object
 4   buys computer  14 non-null     object
dtypes: object(5)
memory usage: 688.0+ bytes
None
```

1. Calculate Income Attribute

```
#65070503428 Pitchayapat Wareevanich
import numpy as np
import pandas as pd

#define the function to calculate the entropy
def entropy(p):
    return -p * np.log2(p) - (1 - p) * np.log2(1 - p)

#calculate the information gain
def information_gain(parent, splits):
    parent_entropy = entropy(parent['buys computer'].value_counts(normalize=True).values[0])
    weighted_child_entropy = 0

    for split in splits:
        split_entropy = entropy(split['buys computer'].value_counts(normalize=True).values[0])
        weight = len(split) / len(parent)
        weighted_child_entropy = weighted_child_entropy + (weight * split_entropy)

    result = parent_entropy - weighted_child_entropy
    return result

parent_node = df

child_nodes = [df[df['income'] == value] for value in df['income'].unique()]
information_gain_income = information_gain(parent_node, child_nodes)
print("Information Gain of Income:", information_gain_income)
```

```
Information Gain of Income: 0.02922256565895487
```

2.STUDENT

```
#define the formula to calculate the entropy
def entropy(p):
    return -p * np.log2(p) - (1 - p) * np.log2(1 - p)

#find the information gain
def information_gain(parent, splits):
    parent_entropy = entropy(parent['buys computer'].value_counts(normalize=True).values[0])
    weighted_child_entropy = 0

    #sum of entropy
    for split in splits:
        split_entropy = entropy(split['buys computer'].value_counts(normalize=True).values[0])
        weight = len(split) / len(parent)
        weighted_child_entropy = weighted_child_entropy + (weight * split_entropy)

    result = parent_entropy - weighted_child_entropy
    return result

parent_node = df
#to split data in student
child_nodes = [df[df['student'] == value] for value in df['student'].unique()]

#print the result
infogain_student = information_gain(parent_node, child_nodes)
print("Information Gain of Student:", infogain_student)
```

```
Information Gain of Student: 0.15183550136234159
```

3.Credit rating

```
#define the formula to calculate the entropy
def entropy(p):
    return -p * np.log2(p) - (1 - p) * np.log2(1 - p)

#find the information gain
def information_gain(parent, splits):
    parent_entropy = entropy(parent['buys computer'].value_counts(normalize=True).values[0])
    weighted_child_entropy = 0

    #sum of entropy
    for split in splits:
        split_entropy = entropy(split['buys computer'].value_counts(normalize=True).values[0])
        weight = len(split) / len(parent)
        weighted_child_entropy = weighted_child_entropy + (weight * split_entropy)

    result = parent_entropy - weighted_child_entropy
    return result

parent_node = df
#to split data in credit rating
child_nodes = [df[df['credit rating'] == value] for value in df['credit rating'].unique()]

#print the result
infogain_credit = information_gain(parent_node, child_nodes)
print("Information Gain of Credit Rating:", infogain_credit)
```

```
Information Gain of Credit Rating: 0.04812703040826949
```

4.TREE

```
import numpy as np
import pandas as pd

class Node:
    def __init__(self, attr=None, lab=None):
        self.attr = attr
        self.lab = lab
        self.ch = {}

def ent(data):
    labs = data.iloc[:, -1]
    uniq_labs = labs.unique()
    ent_val = 0

    for lab in uniq_labs:
        p_lab = (labs == lab).sum() / len(labs)
        ent_val -= p_lab * np.log2(p_lab)

    return ent_val

def info_gain(data, attr):
    tot_ent = ent(data)
    uniq_vals = data[attr].unique()
    split_ent = 0

    for val in uniq_vals:
        sub = data[data[attr] == val]
        split_ent += len(sub) / len(data) * ent(sub)

    ig = tot_ent - split_ent
    return ig

def construct_tree(data, attrs):
    labs = data.iloc[:, -1]

    if len(labs.unique()) == 1:
        return Node(lab=labs.iloc[0])

    if len(attrs) == 0:
        maj_lab = labs.mode()[0]
```