Lab 5: 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 using existing libraries.

In [ ]:
```
# Run this cell if you use Colab
from google.colab import drive
drive.mount('/content/drive')
```

Code it yourself

In [2]:
```
import pandas as pd

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

Out[2]:

	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

In [3]:
```
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: 692.0+ bytes
None
```

TODO: Write functions to compute Gain and Entropy, as discussed in the lecture.

In [12]:

```python
# Write your code here
import numpy as np
import pandas as pd

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

def information_gain(df, target, feature):
    """คำนวณค่า Information Gain"""
    total_entropy = entropy(df[target])
    values, counts = np.unique(df[feature], return_counts=True)
    weighted_entropy = 0
    for i in range(len(values)):
        subset_entropy = entropy(df[df[feature] == values[i]][target])
        weighted_entropy += (counts[i] / len(df)) * subset_entropy
    return total_entropy - weighted_entropy

print('entropy : ', entropy(df['buys computer']))
print('gain of age : ', information_gain(df, 'buys computer', 'age'))
print('gain of income : ', information_gain(df, 'buys computer', 'income'))
print('gain of student : ', information_gain(df, 'buys computer', 'student'))
print('gain of credit rating : ', information_gain(df, 'buys computer', 'credit
```

```
entropy :  0.9402859586706311
gain of age :  0.24674981977443933
gain of income :  0.02922256565895487
gain of student :  0.15183550136234159
gain of credit rating :  0.04812703040826949
```

Using Libraries

Now that you know how to compute these values by yourselfs, now let's use some libraries.

Steps:

- Split the Data → Divide dataset into training (80%) and testing (20%).

- Train the Model → Fit a Decision Tree using the training data.
- Test the Model → Use the trained model to predict on test data.
- Evaluate Performance → Compare predictions with actual values (e.g., Accuracy Score).

Prepare features and labels.

In [14]:
```
# Features
features = df.drop('buys computer', axis=1)
features

# Alternatively, you can use this:
# features = df.iloc[:, :-1]
```

Out[14]:

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

In [15]:
```
# Labels (or Target)
labels = df['buys computer']
labels

# # Alternatively, you can use this:
# labels = df.iloc[:, [-1]]
```

Out[15]:
```
0      no
1      no
2     yes
3     yes
4     yes
5      no
6     yes
7      no
8     yes
9     yes
10    yes
11    yes
12    yes
13     no
Name: buys computer, dtype: object
```

In [16]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree

# 1. Load the dataset
X = features.values  # Features
y = labels.values  # Target labels

# 2. Split the dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 3. Create and train a Decision Tree model with entropy criterion
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
clf.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 15
     13 # 3. Create and train a Decision Tree model with entropy criterion
     14 clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_sta
te=42)
---> 15 clf.fit(X_train, y_train)

File c:\Users\punch\anaconda3\Lib\site-packages\sklearn\base.py:1473, in _fit_con
text.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or global_skip_validation
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)

File c:\Users\punch\anaconda3\Lib\site-packages\sklearn\tree\_classes.py:1009, in
DecisionTreeClassifier.fit(self, X, y, sample_weight, check_input)
    978 @_fit_context(prefer_skip_nested_validation=True)
    979 def fit(self, X, y, sample_weight=None, check_input=True):
    980     """Build a decision tree classifier from the training set (X, y).
    981 
    982     Parameters
   (...)
   1006         Fitted estimator.
   1007     """
-> 1009     super()._fit(
   1010         X,
   1011         y,
   1012         sample_weight=sample_weight,
   1013         check_input=check_input,
   1014     )
   1015     return self

File c:\Users\punch\anaconda3\Lib\site-packages\sklearn\tree\_classes.py:252, in
BaseDecisionTree._fit(self, X, y, sample_weight, check_input, missing_values_in_f
eature_mask)
    248 check_X_params = dict(
    249     dtype=DTYPE, accept_sparse="csc", force_all_finite=False
    250 )
    251 check_y_params = dict(ensure_2d=False, dtype=None)
--> 252 X, y = self._validate_data(
    253     X, y, validate_separately=(check_X_params, check_y_params)
    254 )
    256 missing_values_in_feature_mask = (
    257     self._compute_missing_values_in_feature_mask(X)
    258 )
    259 if issparse(X):

File c:\Users\punch\anaconda3\Lib\site-packages\sklearn\base.py:645, in BaseEstim
ator._validate_data(self, X, y, reset, validate_separately, cast_to_ndarray, **ch
eck_params)
    643 if "estimator" not in check_X_params:
    644     check_X_params = {**default_check_params, **check_X_params}
--> 645 X = check_array(X, input_name="X", **check_X_params)
    646 if "estimator" not in check_y_params:
    647     check_y_params = {**default_check_params, **check_y_params}
```

```
File c:\Users\punch\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1012,
in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, for
ce_writeable, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_m
in_features, estimator, input_name)
   1010         array = xp.astype(array, dtype, copy=False)
   1011     else:
-> 1012         array = _asarray_with_order(array, order=order, dtype=dtype, xp=x
p)
   1013 except ComplexWarning as complex_warning:
   1014     raise ValueError(
   1015         "Complex data not supported\n{}\n".format(array)
   1016     ) from complex_warning

File c:\Users\punch\anaconda3\Lib\site-packages\sklearn\utils\_array_api.py:751,
in _asarray_with_order(array, dtype, order, copy, xp, device)
    749     array = numpy.array(array, order=order, dtype=dtype)
    750 else:
--> 751     array = numpy.asarray(array, order=order, dtype=dtype)
    753 # At this point array is a NumPy ndarray. We convert it to an array
    754 # container that is consistent with the input's namespace.
    755 return xp.asarray(array)

ValueError: could not convert string to float: '31-40'
```

There's an error:

ValueError: could not convert string to float: '31-40'

In [18]:
```python
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply Label Encoding for all categorical columns
df['age'] = label_encoder.fit_transform(df['age'])
df['income'] = label_encoder.fit_transform(df['income'])
df['student'] = label_encoder.fit_transform(df['student'])
df['credit rating'] = label_encoder.fit_transform(df['credit rating'])
df['buys computer'] = label_encoder.fit_transform(df['buys computer'])

# Display the encoded DataFrame
print(df)
```

```
    age  income  student  credit rating  buys computer
0     1       0        0              1              0
1     1       0        0              0              0
2     0       0        0              1              1
3     2       2        0              1              1
4     2       1        1              1              1
5     2       1        1              0              0
6     0       1        1              0              1
7     1       2        0              1              0
8     1       1        1              1              1
9     2       2        1              1              1
10    1       2        1              0              1
11    0       2        0              0              1
12    0       0        1              1              1
13    2       2        0              0              0
```

Let's check out an updated dataframe.

In [19]:
```
df
```

Out[19]:

	age	income	student	credit rating	buys computer
0	1	0	0	1	0
1	1	0	0	0	0
2	0	0	0	1	1
3	2	2	0	1	1
4	2	1	1	1	1
5	2	1	1	0	0
6	0	1	1	0	1
7	1	2	0	1	0
8	1	1	1	1	1
9	2	2	1	1	1
10	1	2	1	0	1
11	0	2	0	0	1
12	0	0	1	1	1
13	2	2	0	0	0

In [20]:
```
X = df.drop('buys computer', axis=1)  # Features
y = df['buys computer']  # Target
```

Let's continue where we left off!

In [21]:
```
# 2. Split the dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 3. Create and train a Decision Tree model with entropy criterion
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
clf.fit(X_train, y_train)
```

Out[21]:

```
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=4
2)
```

In [22]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(11, 4)
(3, 4)
```

Now we're going build the Decision Tree Classifier

In [23]:
```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree classifier
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)  # Using 'ent

# Train the model
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)
```

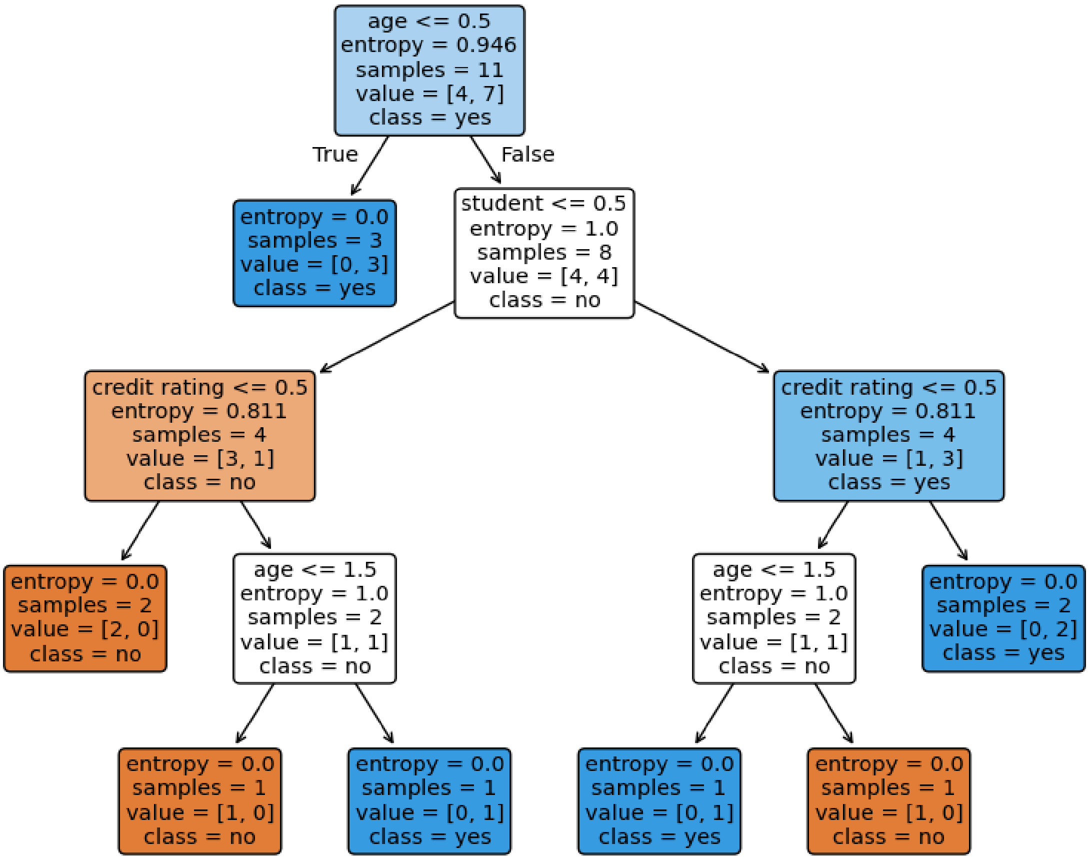
And evaluate our model.

In [33]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_mat

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

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

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

```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

Confusion Matrix:
[[1 0]
 [0 2]]
```

And visualize our tree!

In [25]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Plot the decision tree
plt.figure(figsize=(10, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['no', 'yes'],
plt.show()
```

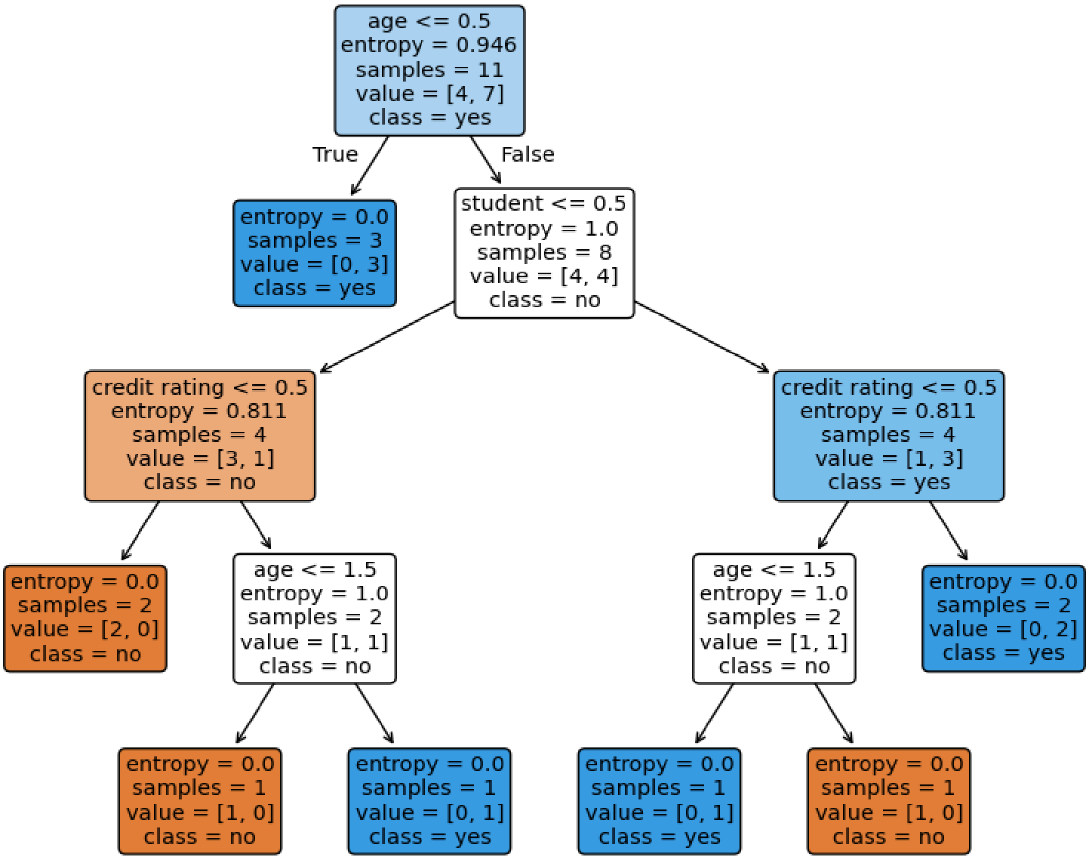


Put them all together.

In [27]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# data
data = pd.read_csv('toy_data.csv')

df = pd.DataFrame(data)

# Encode categorical columns using LabelEncoder
label_encoder = LabelEncoder()
df['age'] = label_encoder.fit_transform(df['age'])
df['income'] = label_encoder.fit_transform(df['income'])
df['student'] = label_encoder.fit_transform(df['student'])
df['credit rating'] = label_encoder.fit_transform(df['credit rating'])
df['buys computer'] = label_encoder.fit_transform(df['buys computer'])

# Separate features (X) and target (y)
X = df.drop('buys computer', axis=1)
y = df['buys computer']

# Split the dataset into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize the Decision Tree classifier
```

```python
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

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

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Plot the decision tree
plt.figure(figsize=(10, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['no', 'yes'],
plt.show()
```

```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

Confusion Matrix:
[[1 0]
 [0 2]]
```



Is the output tree the same as what you calculated yourself? Explain in your own words why they are the same or different.

Ans: จากการคำนวณ entropy ได้ 0.9402859586706311 ซึ่งใกล้เคียงกับสิ่งที่คำนวณใน DecisionTree

---

Another example, another dataset -- Iris

In [28]:

```
from sklearn.datasets import load_iris

# 1. Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

# 2. Split the dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 3. Create and train a Decision Tree model with entropy criterion
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# 4. Make predictions on the test set
y_pred = clf.predict(X_test)

# 5. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```python
# 6. Visualize the Decision Tree
plt.figure(figsize=(10, 6))
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.t
plt.show()
```

```
Model Accuracy: 1.00
```