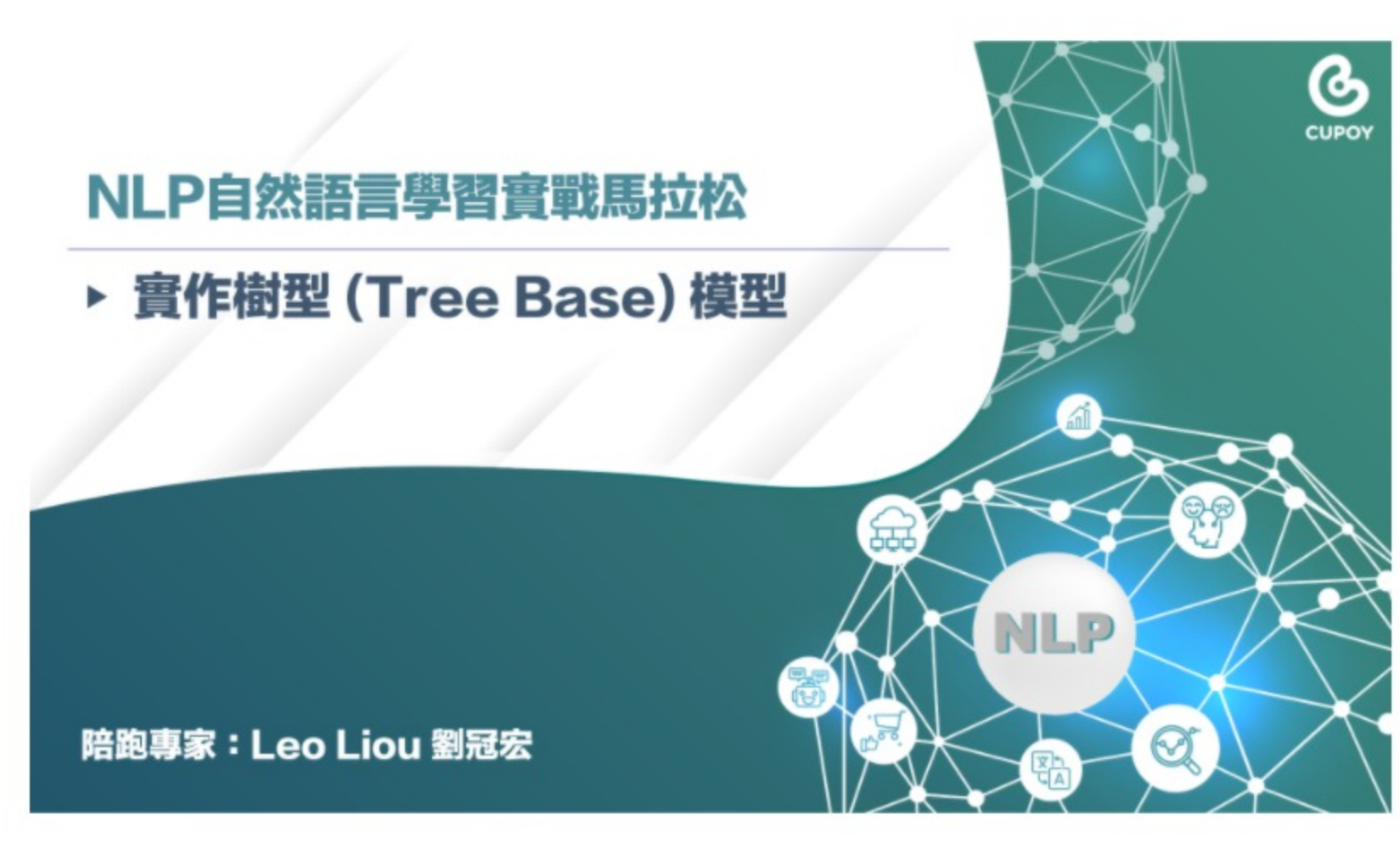
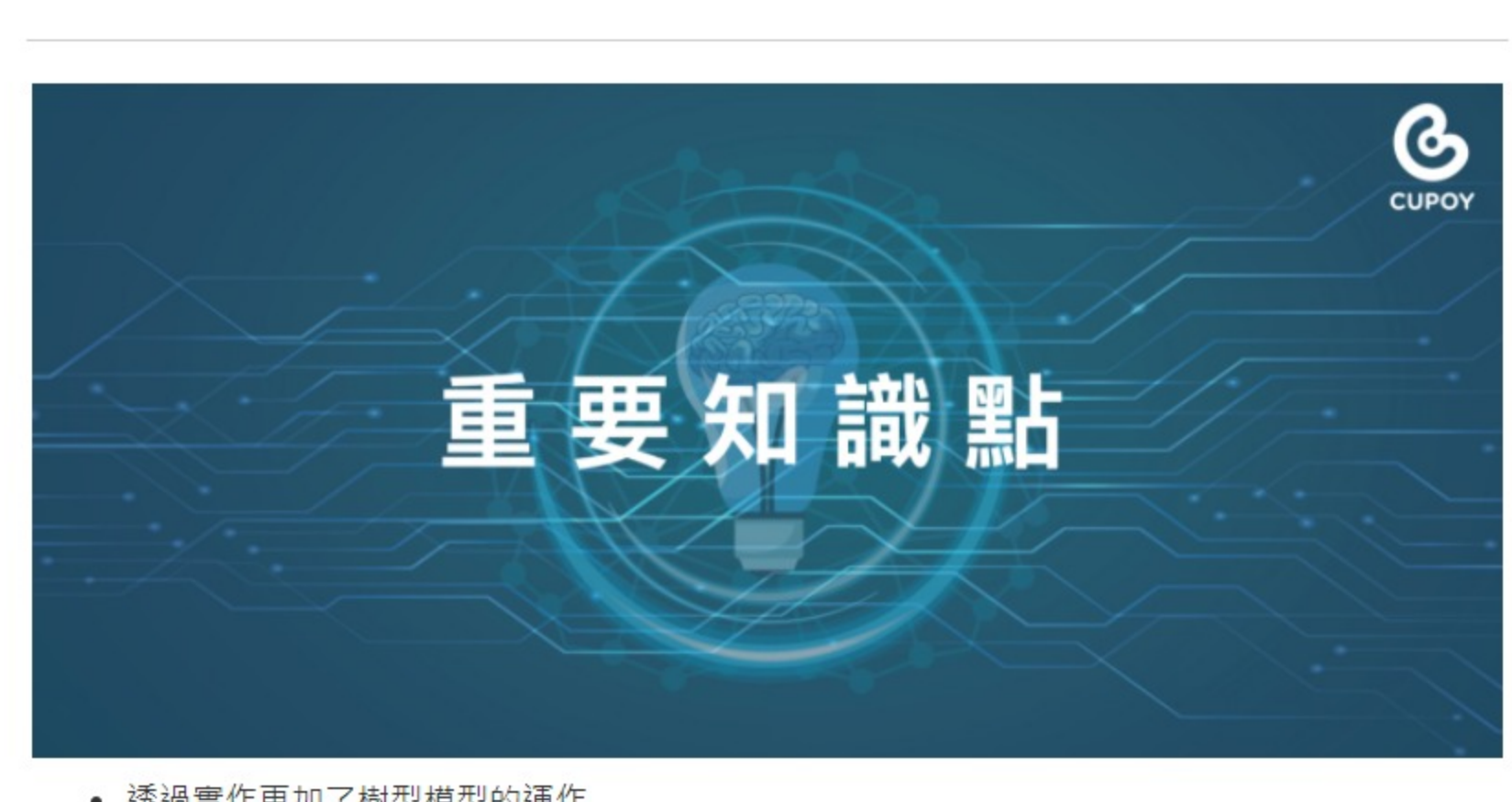
D27：實作樹型(Tree Base)模型



重要知識點



- 透過實作更加了解樹型模型的運作
- 實作與使用決策樹模型

(本日課程為實作導向課程，同學可參考附件"實作 TreeBase 模型.ipynb")

生成假數據

為了要驗證與使用所做的模型，這邊我們自行製作少量的假數據。此數據為類別是水果而特徵有顏色與半徑兩個。下述的介紹都會基於此製作的假數據進行。

```python
training_data = [
    ['Green', 3.1, 'Apple'],
    ['Yellow', 3.2, 'Apple'],
    ['Red', 1.2, 'Grape'],
    ['Red', 1, 'Grape'],
    ['Yellow', 3.3, 'Lemon'],
    ['Yellow', 3.1, 'Lemon'],
    ['Green', 3, 'Apple'],
    ['Red', 1.1, 'Grape'],
    ['Yellow', 3, 'Lemon'],
    ['Red', 1.2, 'Grape'],
]

header = ["color", "diameter", "label"]

df = pd.DataFrame(data=training_data, columns=header)
df.head()
```

特徵　標籤

	color	diameter	label
0	Green	3	Apple
1	Yellow	3	Apple
2	Red	1	Grape
3	Red	1	Grape
4	Yellow	3	Lemon

```python
training_data = [
    ['Green', 3.1, 'Apple'],
    ['Yellow', 3.2, 'Apple'],
    ['Red', 1.2, 'Grape'],
    ['Red', 1, 'Grape'],
    ['Yellow', 3.3, 'Lemon'],
    ['Yellow', 3.1, 'Lemon'],
    ['Green', 3, 'Apple'],
    ['Red', 1.1, 'Grape'],
    ['Yellow', 3, 'Lemon'],
    ['Red', 1.2, 'Grape'],
]
```

數據分割

一般在進行訓練前會將數據集，會將數據集切分成訓練集(training set) 與驗證集(validation set)，這裡我們實作數據切分函式。

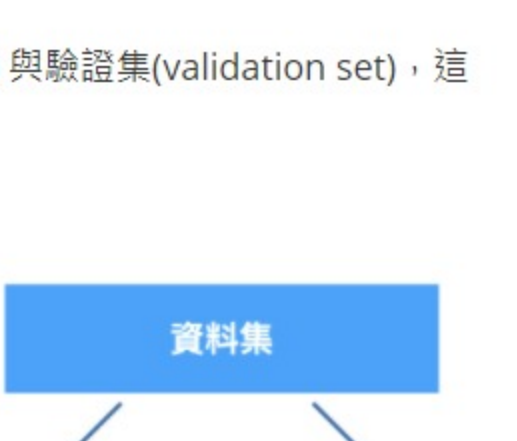
```python
# 分割測試集與訓練集
def train_test_split(df, test_size=0.1):

    if isinstance(test_size, float):
        test_size = round(test_size * len(df))

    #以隨機的方式取的測試集資料點的index
    indices = list(df.index)
    test_indices = random.sample(population=indices, k=test_size)

    #分割測試集與訓練集
    test_df = df.loc[test_indices]
    train_df = df.drop(test_indices)

    return train_df, test_df
```




```python
# 分割測試集與訓練集
def train_test_split(df, test_size=0.1):


    if isinstance(test_size, float):
        test_size = round(test_size * len(df))

    #以隨機的方式取的測試集資料點的index
    indices = list(df.index)
    test_indices = random.sample(population=indices, k=test_size)

    #分割測試集與訓練集
    test_df = df.loc[test_indices]
```

檢測節點函式

在樹型模型中，當節點中的資料都屬於同一類別時即不須再往下分割，此節點即為終端節點(leaf node)，因此這部分實作檢測節點資料是否為同一類別的函式。

```python
# 檢查資料是否都為同一類別
def check_purity(data):
    '''Function to check if input data all belong to the same class
    Parameter
    ---------
    data: list
        Input data
    '''
    #取的資料的label訊息
    labels = data[:, -1]

    #檢查是否所有的label都為同一種
    unique_classes = np.unique(labels)

    if len(unique_classes) == 1:
        return True
    else:
        return False
```

```python
check_purity(df.values)
```

```
False
```

```python
# 檢查資料是否都為同一類別
def check_purity(data):
    '''Function to check if input data all belong to the same class
    Parameter
    ---------
    data: list
        Input data
    '''
    #取的資料的label訊息
    labels = data[:, -1]

    #檢查是否所有的label都為同一種
```

取得可分割特徵值

樹型模型中，每個節點會根據所選取的特徵設定一個條件值，而這個條件值可以從特徵所有的獨特值中取得。

```python
# 根據給定的資料，取得每個特徵(feature)可能做為樹型模型分割節點的值
# 可能作為分割節點值即為每個特徵的獨特值(unique value)
def get_potential_splits(data):
    '''Function to get all potential split value for tree base model
    Parameter
    ---------
    data: list
        Input data
    '''

    potential_splits = {}
    _, n_columns = data.shape

    #欄位最後一欄為label不須計算
    for column_index in range(n_columns - 1):

        #取得欄位的所有獨特值(unique values)
        values = data[:, column_index]
        unique_values = np.unique(values)

        #將獨特值存放到potential_splits字典中
        potential_splits[column_index] = unique_values

    return potential_splits
```

```python
get_potential_splits(df.values)
```

```
{0: array(['Green', 'Red', 'Yellow'], dtype=object),
 1: array([1.0, 1.1, 1.2, 3.0, 3.1, 3.2, 3.3], dtype=object)}
```

```python
根據給定的資料，取得每個特徵(feature)可能做為樹型模型分割節點的值
# 可能作為分割節點值即為每個特徵的獨特值(unique value)
def get_potential_splits(data):
    '''Function to get all potential split value for tree base model
    Parameter
    ---------
    data: list
        Input data
    '''

    potential_splits = {}
    _, n_columns = data.shape
```

判別特徵值型態

結點再選取條件值時，根據特徵資料型態的不同，會有不同的條件判斷式，如數值型可以用大小於判別，而類別型可以用是否相同類別判斷。

```python
#判別給定的輸入DataFrame每個特徵的型態(數值型特徵或類別型特徵)
def determine_type_of_feature(df):
    '''Function to get features types
    Parameter
    ---------
    df: pd.DataFrame
        Input raw pd.DataFrame data
    '''

    feature_types = []

    #當特徵的獨特值個數較少，及當作類別型特徵資料(就為數值型，獨特值個數還是會很多)
    n_unique_values_treshold = ...

    for feature in df.columns:
        if feature != "label":
            unique_values = df[feature].unique()
            exp_value = unique_values[0]

            if (isinstance(exp_value, str)) or (len(unique_values) <= n_unique_values_treshold):
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

    return feature_types
```

```python
determine_type_of_feature(df)
```

```
['categorical', 'continuous']
```

```python
#判別給定的輸入DataFrame每個特徵的型態(數值型特徵或類別型特徵)
def determine_type_of_feature(df):
    '''Function to get features types
    Parameter
    ---------
    df: pd.DataFrame
        Input raw pd.DataFrame data
    '''

    feature_types = []

    #當特徵的獨特值個數較少，及當作類別型特徵資料(就為數值型，獨特值個數還是會很多)
```

取得分割節點

節點在分割後，會產生左節點(滿足分割條件)與右節點(不滿足分割條件)

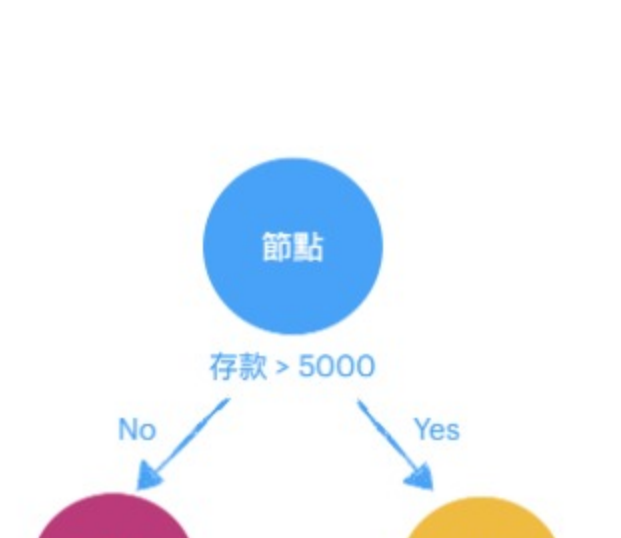
```python
def split_data(data, split_column, split_value):
    '''Function to splitted left and right nodes
    Parameter
    ---------
    data: list
        Input data
    split_column: int
        index for feature column
    split_value: float or int or string
        value to be used as split benchmark
    '''

    #取得用來分割的特徵欄位
    split_column_values = data[:, split_column]

    #依據欄位值的型態(數值型特徵或類別型特徵)來進行節點分割
    type_of_feature = FEATURE_TYPES[split_column]

    if type_of_feature == "continuous":
        #數值型特徵分割
        data_left = data[split_column_values <= split_value]
        data_right = data[split_column_values >  split_value]
    else:
        #類別型特徵分割
        data_left = data[split_column_values == split_value]
        data_right = data[split_column_values != split_value]

    return data_left, data_right
```




```python
def split_data(data, split_column, split_value):
    '''Function to splitted left and right nodes
    Parameter
    ---------
    data: list
        Input data
    split_column: int
        index for feature column
    split_value: float or int or string
        value to be used as split benchmark
    '''

    #取得用來分割的特徵欄位
    split_column_values = data[:, split_column]

    #依據欄位值的型態(數值型特徵或類別型特徵)來進行節點分割
    type_of_feature = FEATURE_TYPES[split_column]

    if type_of_feature == "continuous":
        #數值型特徵分割
        data_left = data[split_column_values <= split_value]
        data_right = data[split_column_values >  split_value]
    else:
        #類別型特徵分割
        data_left = data[split_column_values == split_value]
        data_right = data[split_column_values != split_value]

    return data_left, data_right
```

取得終端節點

節點無法再分割時，此節點即成為終端節點(Leaf node)，在終端節點需要取得此節點最後分類的類別(在此節點的所有樣本皆被分類為同一個類別)

```python
def create_leaf(data, task_type):
    '''Function to create leaf node
    Parameters
    ----------
    data: list
        Input data
    task_type: str
        indicate the type of tree (regression or classification)
    '''

    #取的資料的label欄位
    label_column = data[:, -1]

    if task_type == "regression":
        #回歸任務
        leaf = np.mean(label_column)
    else:
        #分類任務
        #取得所有輸入資料的獨立類別與其個數
        unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)

        #以個數最多的類別，作為此節點的輸出類別
        index = counts_unique_classes.argmax()
        leaf = unique_classes[index]

    return leaf
```

```python
def create_leaf(data, task_type):
    '''Function to create leaf node
    Parameters
    ----------
    data: list
        Input data
    task_type: str
        indicate the type of tree (regression or classification)
    '''

    #取的資料的label欄位
    label_column = data[:, -1]
```

計算訊息增益

節點分割的依據在於取得最大的訊息增益，而計算訊息增益的方法之一就是計算樣本的熵。(同學可到決策樹章節複習訊息增益的計算)

計算熵(Entropy)

$$Entropy = -\sum_{i=1}^{c} p(i)log_2p(i)$$

```python
def calculate_entropy(data):

    #取的資料的label訊息
    label_column = data[:, -1]

    #取得所有輸入資料的獨立類別與其個數
    _, counts = np.unique(label_column, return_counts=True)

    #計算機率
    probabilities = counts / counts.sum()

    #計算entropy
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy
```

```python
def calculate_entropy(data):

    #取的資料的label訊息
    label_column = data[:, -1]

    #取得所有輸入資料的獨立類別與其個數
    _, counts = np.unique(label_column, return_counts=True)

    #計算機率
    probabilities = counts / counts.sum()

    #計算entropy
```

計算總訊息增益

```python
def calculate_overall_metric(data_below, data_above, metric_function):

    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_metric =  (p_data_below * metric_function(data_below)
                     + p_data_above * metric_function(data_above))

    return overall_metric
```

```python
def calculate_overall_metric(data_below, data_above, metric_function):

    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_metric =  (p_data_below * metric_function(data_below)
                     + p_data_above * metric_function(data_above))

    return overall_metric
```

決策樹演算法

將上述介紹的函式組裝成決策樹模型，模型中的參數代表的意義如下：

1. metric_function：代表欲使用的訊息增益計算函式(ex：Entropy)
2. task_type：此決策樹是分類樹(classification)或回歸樹(regression)
3. counter：用來計算樹生成的深度
4. min_sample：代表節點要分割最少需要的樣本數
5. max_depth：樹最深的層數

```python
class decision_tree():
    '''Decision Tree model
    Parameters
    -----------
    metric_function: function
        the metric function used to calculate information gain
    task_type: str
        indicate the type of tree (regression or classification)
    counter: int
        counter for recording number of splits
    min_samples: int
        minimum number of samples for a node to be able to split
    max_depth: int
        Maximum depth for the decision tree
    '''
    def __init__(self, metric_function, task_type='classification', counter=0, min_samples=2, max_depth=5):

        self.metric_function = metric_function
        self.task_type = task_type
        self.counter = counter
        self.min_samples = min_samples
        self.max_depth = max_depth
```

```python
class decision_tree():
    '''Decision Tree model
    Parameters
    ----------
    metric_function: function
        the metric function used to calculate information gain
    task_type: str
        indicate the type of tree (regression or classification)
    counter: int
        counter for recording number of splits
    min_samples: int
        minimum number of samples for a node to be able to split
```

詳細使用操作

請參照使用實做 TreeBase 模型 .ipynb 檔進行更詳細的使用操作

知識點回顧

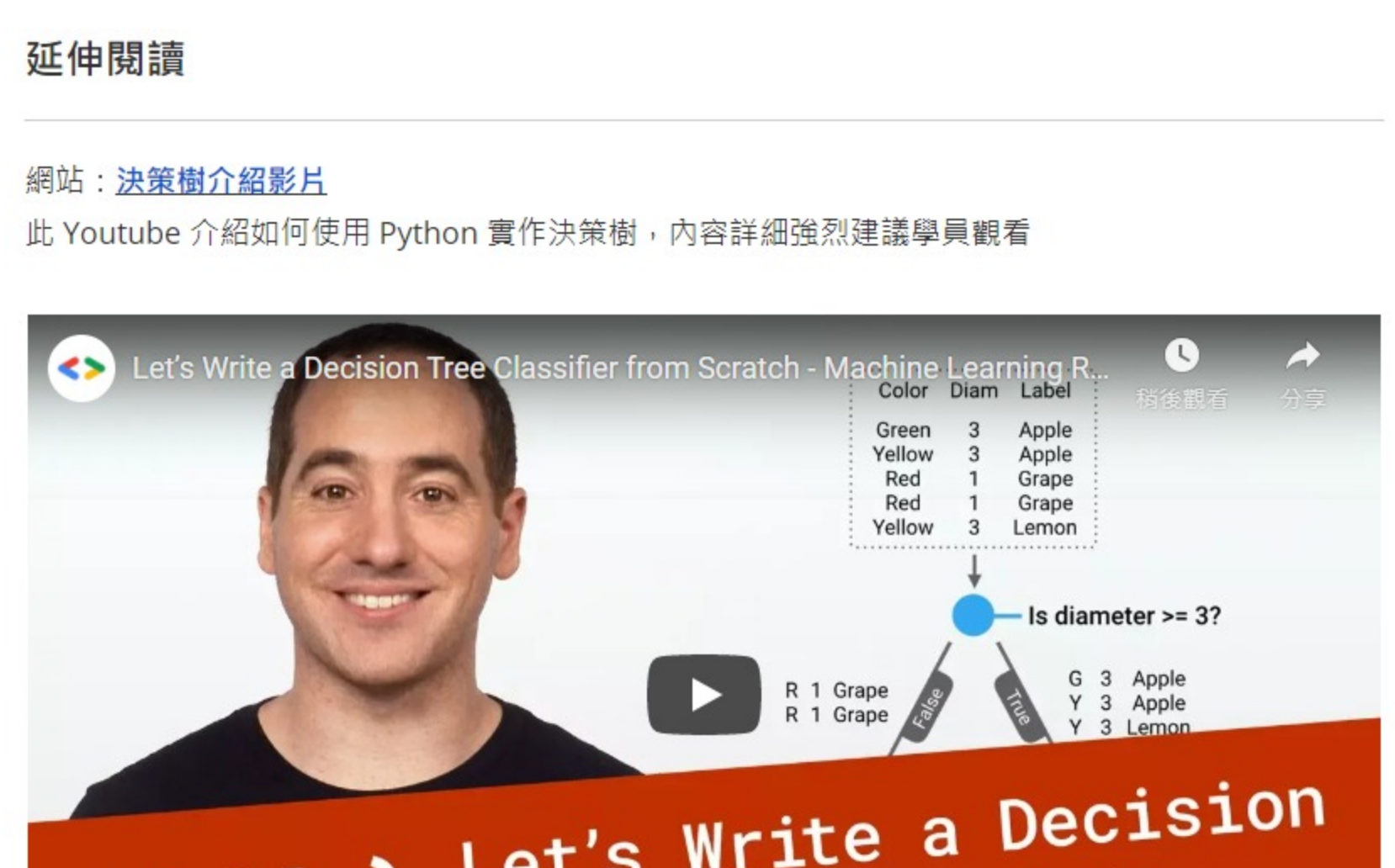
在這章節我們學習到了

- 實作與使用決策樹模型

延伸閱讀

網站：決策樹介紹影片

此 Youtube 介紹如何使用 Python 實作決策樹，內容詳細強烈建議學員觀看



下一步：完成作業