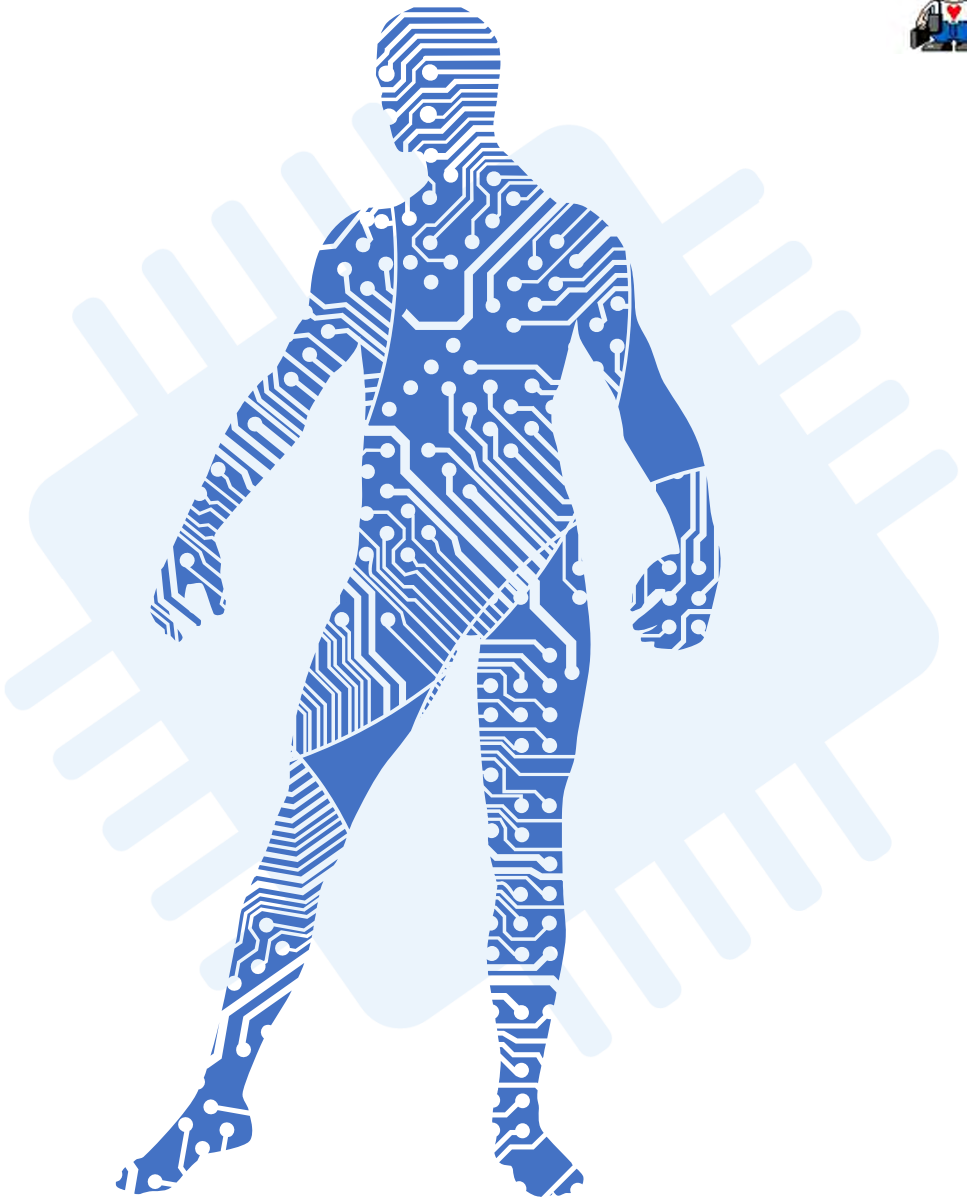
機器學習

第 12 章 決策樹
(Decision Tree)

講師：紀俊男

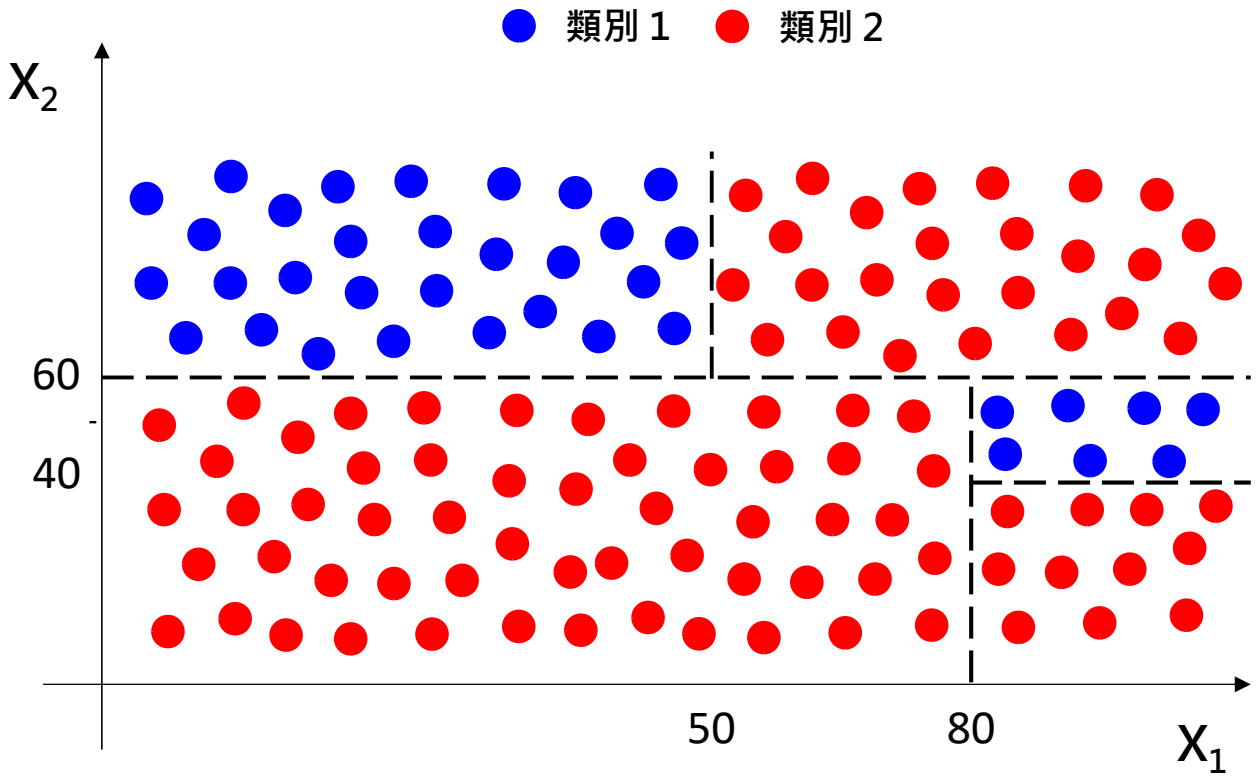
本章大綱

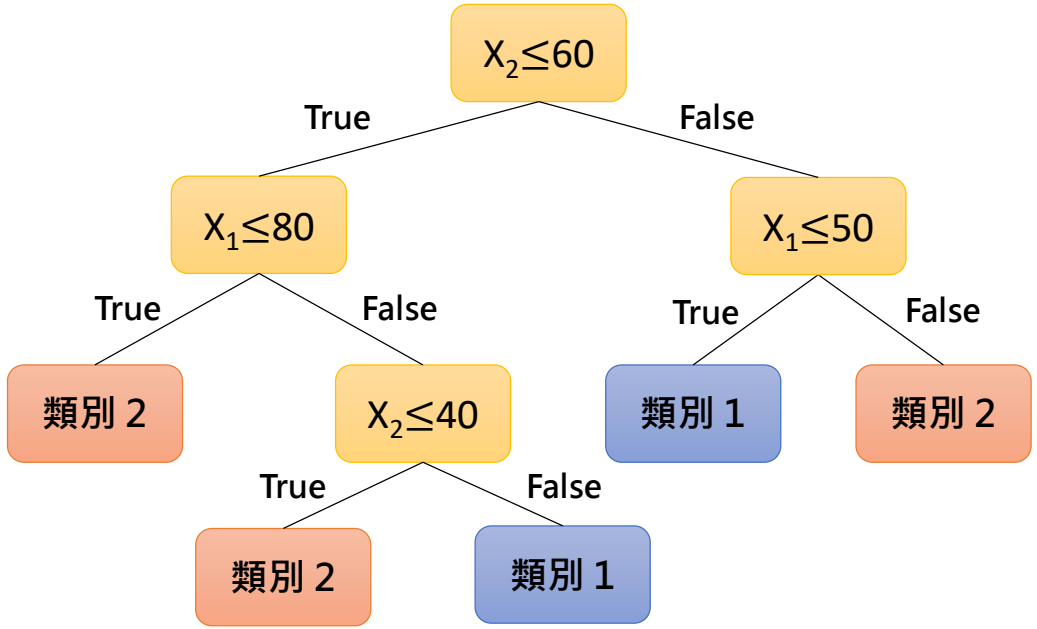
- 原理解說
- 資料前處理
- 實作決策樹
- 將決策樹視覺化
- 本章總結



AI 原理解説

決策樹如何分類？


類別 1
類別 2
X_2
60
40
50
80
X_1


$X_2 \leq 60$
True
False
$X_1 \leq 80$
$X_1 \leq 50$
True
False
True
False
類別 2
$X_2 \leq 40$
類別 1
類別 2
True
False
類別 2
類別 1

決策樹演算法（1）：ID3

- 迭代二分器 3.0 版（Iterative Dichotomiser 3）
  - 利用「資訊熵（Entropy）」計算資訊量
  - 每一刀切下去，都要得到「最大資訊增益（Information Gain）」

資訊越稀有→資訊量越大

$$Info = -logP_i$$

$$\begin{cases} \text{東京下雪機率} = \dfrac{1}{10} \\ \text{台北下雪機率} = \dfrac{1}{1000} \end{cases}$$

$$\begin{cases} \text{東京下雪資訊量} = -log\dfrac{1}{10} = 1 \\ \text{台北下雪資訊量} = -log\dfrac{1}{1000} = 3 \end{cases}$$

$$\text{資訊熵} = \sum_{i=1}^{n}(\text{資訊 } i \text{ 發生機率}) \times (\text{資訊 } i \text{ 資訊量})$$

$$= \sum_{i=1}^{n} Entropy_i = \sum_{i=1}^{n} P_i \times (-logP_i)$$

下雪的「資訊熵」

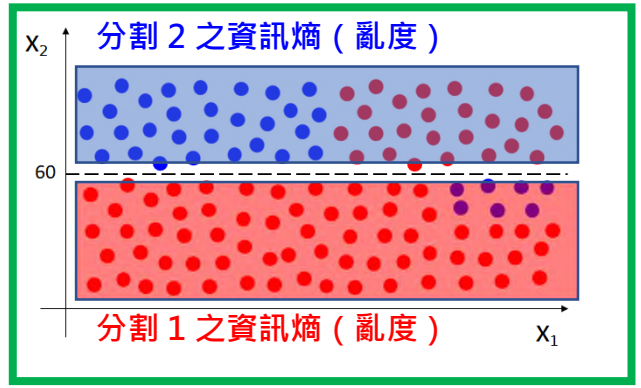
$$= Entropy(\text{東京}) + Entropy(\text{台北}) + \cdots$$

$$= \frac{1}{10} \times \left(-log\frac{1}{10}\right) + \frac{1}{1000} \times \left(-log\frac{1}{1000}\right) + \cdots$$

資訊增益（I.G.）

$$= \text{總資訊熵} - \sum_{i=1}^{n} \text{分割 } i \text{ 之資訊熵}$$

總資訊熵（亂度）



決策樹演算法（1）：ID3

- ID3 範例：婚友社「相親配對」

配對 12 人。7 位見面、5 位不想見面

網站 ID	年齡（歲）	身高（cm）	年收入（萬元）	學歷	有否相親
XXXXXXX	25	179	75	大專	N
XXXXXXX	33	190	95	大專	Y
XXXXXXX	28	180	90	碩士	Y
XXXXXXX	25	178	90	碩士	Y
XXXXXXX	46	177	500	碩士	N
XXXXXXX	40	170	350	大學	N
XXXXXXX	34	174	100	碩士	Y
XXXXXXX	36	181	220	大學	N
XXXXXXX	35	170	125	碩士	Y
XXXXXXX	30	180	175	大學	Y
XXXXXXX	28	174	150	大學	N
XXXXXXX	29	176	180	大學	Y

找「資訊增益」最大的

第一刀用「學歷」來切

$$Entropy = Entropy(\text{大專}) + Entropy(\text{大學}) + Entropy(\text{碩士})$$

$$= -\frac{2}{12}\left(\frac{1}{2}log_2\frac{1}{2}+\frac{1}{2}log_2\frac{1}{2}\right)-\frac{5}{12}\left(\frac{3}{5}log_2\frac{3}{5}+\frac{2}{5}log_2\frac{2}{5}\right)-\frac{5}{12}\left(\frac{4}{5}log_2\frac{4}{5}+\frac{1}{5}log_2\frac{1}{5}\right)$$

$= 0.872$　　資訊增益（I.G.）= 0.98 - 0.872 = 0.108

第一刀用「年收入」來切

……

第一刀用「身高」來切

……

總資訊熵（亂度）$= \frac{7}{12}\left(-log_2\frac{7}{12}\right)+\frac{5}{12}\left(-log_2\frac{5}{12}\right) = 0.98$

→ 依照「資訊增益」最大、次大...依序切割

決策樹演算法（2）：CART

- CART = Classification And Regression Tree
  - 使用「**基尼指數（Gini Index）**」取代「資訊熵」
  - 使用「**基尼增益（Gini Gain）**」取代「資訊增益」
  - 優點：公式簡單，計算量較低
  - 比較：ID3 vs. CART 大部分的情況效能差不多，挑一個習慣用的就可以。

基尼指數（Gini Index）

$$= 1 - \sum_{i=1}^{n} (\text{資訊 } i \text{ 發生的機率})^2$$

基尼增益（Gini Gain）

$$= \text{總基尼指數} - \sum_{i=1}^{n} \text{分割 } i \text{ 之基尼指數}$$

決策樹演算法（2）：CART

- CART 範例：婚友社「相親配對」

配對 12 人。7 位見面、5 位不想見面

網站 ID	年齡（歲）	身高（cm）	年收入（萬元）	學歷	有否相親
XXXXXXX	25	179	75	大專	N
XXXXXXX	33	190	95	大專	Y
XXXXXXX	28	180	90	碩士	Y
XXXXXXX	25	178	90	碩士	Y
XXXXXXX	46	177	500	碩士	N
XXXXXXX	40	170	350	大學	N
XXXXXXX	34	174	100	碩士	Y
XXXXXXX	36	181	220	大學	N
XXXXXXX	35	170	125	碩士	Y
XXXXXXX	30	180	175	大學	Y
XXXXXXX	28	174	150	大學	N
XXXXXXX	29	176	180	大學	Y

總基尼指數

$$= 1 - \left(\left(\frac{7}{12}\right)^2 + \left(\frac{5}{12}\right)^2\right) = 0.4861\ldots$$

學歷切分的基尼指數

$$= \frac{2}{12}\left(1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right)\right) + \frac{5}{12}\left(1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right)\right) + \frac{5}{12}\left(1 - \left(\left(\frac{4}{5}\right)^2 + \left(\frac{1}{5}\right)^2\right)\right)$$

$$= 0.5$$

基尼增益（Gini Gain）

$$= 0.4861 - 0.5$$

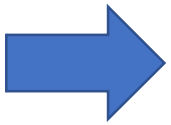
$$= -0.0139$$ ← 變得更差了！不該先由學歷切

資料前處理

下載與瀏覽資料集

- 依照講師指示，**下載**並**瀏覽**資料集

Mushroom.csv



	A	B	C	D
1	class	cap-shape	cap-surface	cap-color
2	p	x	s	n
3	e	x	s	y
4	e	b	s	w
5	p	x	y	w
6	e	x	s	g
7	e	x	y	y
8	e	b	s	w
9	e	b	y	w
10	p	x	y	w

目的：利用「傘帽、表皮、傘色...」
--> 推算「香菇是否有毒」

- **class**：p-有毒、e-可吃
- **cap-shape**：傘帽
  b-鐘形、c-錐形、x-圓形、
  f-扁形、k-把手形、s-凹陷
- **cap-surface**：傘皮
  f-纖維感、g-凹洞、s-鱗片
  s-光滑
- **cap-color**：傘色
  n-棕、b-淺黃、c-肉桂、
  g-灰、r-綠、p-粉紅、u-紫
  e-紅、w-白、y-黃
- **bruises**：瘀傷 (t-有/f-無)
- odor：氣味。a-杏仁、l-茴香
  c-雜酚油味、y-魚腥味、f-臭
  m-霉、n-無、p-辛、s-辣
- .....
- 完整列表：
  https://bit.ly/3kiiCs7

資料前處理

撰寫程式碼

```
1  import HappyML.preprocessor as pp
2
3  # Load Data
4  dataset = pp.dataset(file="Mushrooms.csv")
5
6  # Decomposition
7  X, Y = pp.decomposition(dataset, x_columns=[i for i in range(1, 23)], y_columns=[0])
8
9  # Dummy Variables
10 X = pp.onehot_encoder(X, columns=[i for i in range(22)], remove_trap=True)
11 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
12
13 # Feature Selection
14 from HappyML.preprocessor import KBestSelector
15 selector = KBestSelector(best_k="auto")
16 X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)
17
18 # Split Training / TEsting Set
19 X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)
```

自動移除
虛擬變數陷阱

取得 {0:e, 1:p} 之對照表

資料前處理流程：

1. 載入資料
2. 切分自變數、應變數
3. 處理缺失資料
   （無缺失資料）
4. 類別資料數位化
   （+移除虛擬變數陷阱）
5. 特徵選擇
6. 切分訓練集、測試集
7. 特徵縮放
   （暫無需要）

隨堂練習：資料前處理

- 請撰寫下列程式碼，並予以執行，完成「**資料前處理**」的步驟：

```python
import HappyML.preprocessor as pp

# Load Data
dataset = pp.dataset(file="Mushrooms.csv")

# Decomposition
X, Y = pp.decomposition(dataset, x_columns=[i for i in range(1, 23)], y_columns=[0])

# Dummy Variables
X = pp.onehot_encoder(X, columns=[i for i in range(22)], remove_trap=True)
Y, Y_mapping = pp.label_encoder(Y, mapping=True)

# Feature Selection
from HappyML.preprocessor import KBestSelector
selector = KBestSelector(best_k="auto")
X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)

# Split Training / TEsting Set
X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)
```

實作決策樹

使用「標準函式庫」實作

程式碼

載入必要套件

```
1  from sklearn.tree import DecisionTreeClassifier
2  import time
3
4  classifier = DecisionTreeClassifier(criterion="entropy", random_state=int(time.time()))
5  classifier.fit(X_train, Y_train)
6  Y_pred = classifier.predict(X_test)
```

可選 “entropy” (ID3)
或者 “gini” (CART)

產生物件本身→
訓練→
預測→

執行結果

Y_test

Index	class
1914	0
1930	1
4585	1
6097	1
547	0
7223	1
7789	1
1048	0
7069	1
1719	0

Y_pred

	0
0	0
1	1
2	1
3	1
4	0
5	1
6	1
7	0
8	1
9	0

隨堂練習：使用「標準函式庫」實作

- 請撰寫下列程式碼，並執行之：

```
1  from sklearn.tree import DecisionTreeClassifier
2  import time
3
4  classifier = DecisionTreeClassifier(criterion="entropy", random_state=int(time.time()))
5  classifier.fit(X_train, Y_train)
6  Y_pred = classifier.predict(X_test)
```

- 執行完畢後，請比較 **Y_test（真實值）**與 **Y_pred（預測值）**的差異。

使用「快樂版函式庫」實作

- 程式碼解說（1）：

/HappyML/classification.py

引入必要套件

類別的成員變數

建構函數

classifier 的 getter

演算法

亂數種子

```python
from sklearn.tree import DecisionTreeClassifier
import time

class DecisionTree:
    __classifier = None
    __criterion = None
    __y_columns = None

    def __init__(self, criterion="entropy", random_state=int(time.time())):
        self.__criterion = criterion
        self.__classifier = DecisionTreeClassifier(criterion=self.__criterion, random_state=random_state)

    @property
    def classifier(self):
        return self.__classifier
```

使用「快樂版函式庫」實作

- 程式碼解說（２）：

/HappyML/classification.py

classifier 的 setter

訓練

預測

```
17      @classifier.setter
18      def classifier(self, classifier):
19          self.__classifier = classifier
20
21      def fit(self, x_train, y_train):
22          self.classifier.fit(x_train, y_train)
23          self.__y_columns = y_train.columns   ← 保存 Y_train 的欄位名稱
24          return self
25
26      def predict(self, x_test):                ↓ 把預測出來的 Y_pred 重新包裝回 DataFrame 再傳回
27          return pd.DataFrame(self.classifier.predict(x_test), index=x_test.index, columns=self.__y_columns)
```

使用「快樂版函式庫」實作

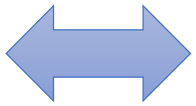
- 呼叫範例

```
1  from HappyML.classification import DecisionTree
2
3  classifier = DecisionTree()
4  Y_pred = classifier.fit(X_train, Y_train).predict(X_test)
```

- 執行結果

Y_test

Index	class
1914	0
1930	1
4585	1
6097	1
547	0
7223	1
7789	1
1048	0
7069	1
1719	0



Y_pred

Index	class
1914	0
1930	1
4585	1
6097	1
547	0
7223	1
7789	1
1048	0
7069	1
1719	0

隨堂練習：使用「快樂版函式庫」實作

- 請撰寫下列程式碼，並執行之：

```
from HappyML.classification import DecisionTree

classifier = DecisionTree()
Y_pred = classifier.fit(X_train, Y_train).predict(X_test)
```

- 執行完畢後，請比較 Y_test（真實值）與 Y_pred（預測值）的差異。

計算決策樹的效能

- 程式碼解說

```
1  from HappyML.performance import KFoldClassificationPerformance
2
3  K = 10
4  kfp = KFoldClassificationPerformance(x_ary=X, y_ary=Y, classifier=classifier.classifier, k_fold=K)
5
6  print("----- Decision Tree Classification -----")
7  print("{} Folds Mean Accuracy: {}".format(K, kfp.accuracy()))
8  print("{} Folds Mean Recall: {}".format(K, kfp.recall()))
9  print("{} Folds Mean Precision: {}".format(K, kfp.precision()))
10 print("{} Folds Mean F1-Score: {}".format(K, kfp.f_score()))
```

- 執行結果

```
----- Decision Tree Classification -----
10 Folds Mean Accuracy: 0.9685116851168513
10 Folds Mean Recall: 0.9673469387755101
10 Folds Mean Precision: 0.9810930576070902
10 Folds Mean F1-Score: 0.9641000165590329
```

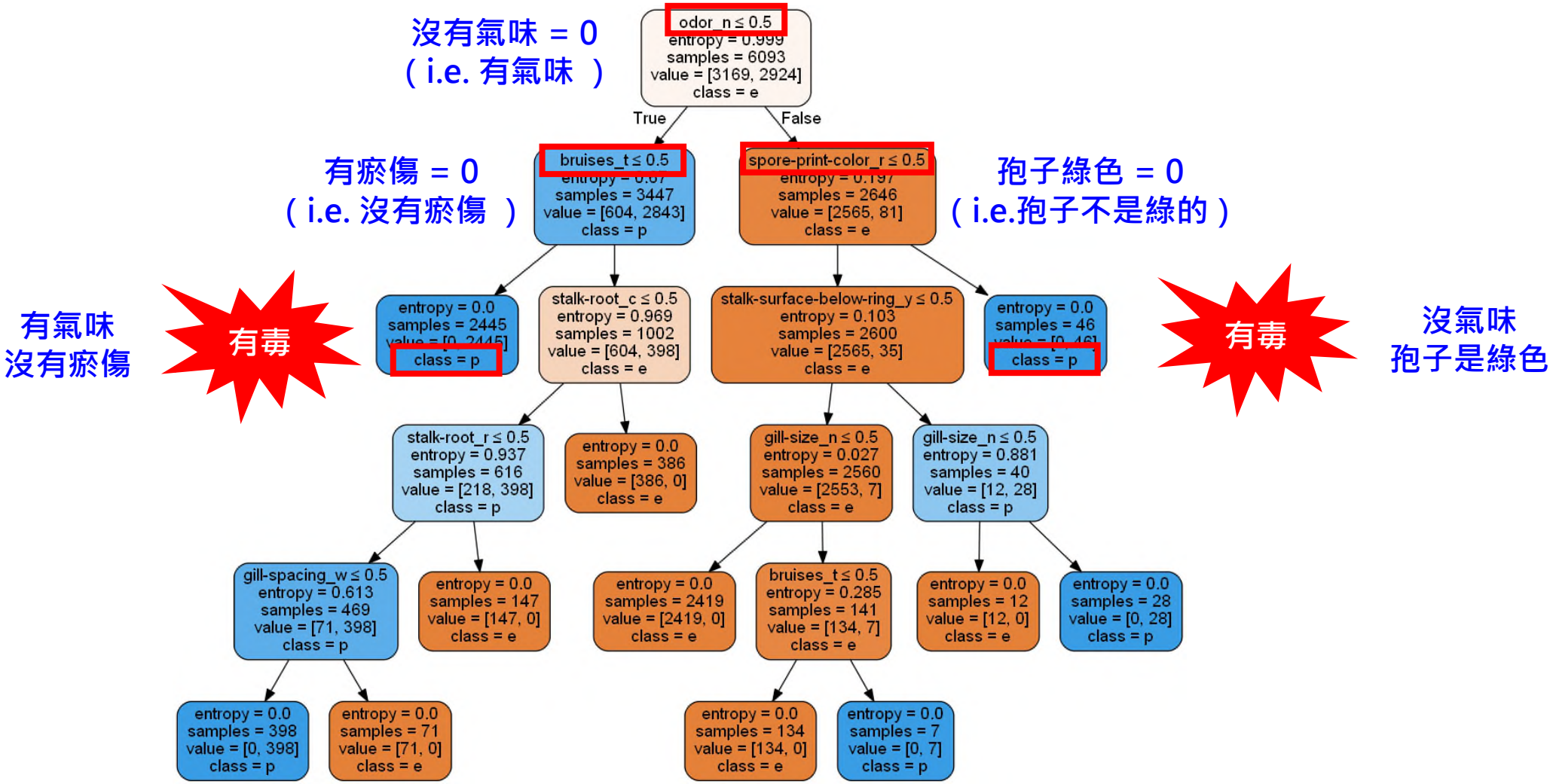
隨堂練習：計算決策樹的效能

- 請撰寫下列程式碼，計算決策樹的效能：

```
1  from HappyML.performance import KFoldClassificationPerformance
2
3  K = 10
4  kfp = KFoldClassificationPerformance(x_ary=X, y_ary=Y, classifier=classifier.classifier, k_fold=K)
5
6  print("----- Decision Tree Classification -----")
7  print("{} Folds Mean Accuracy: {}".format(K, kfp.accuracy()))
8  print("{} Folds Mean Recall: {}".format(K, kfp.recall()))
9  print("{} Folds Mean Precision: {}".format(K, kfp.precision()))
10 print("{} Folds Mean F1-Score: {}".format(K, kfp.f_score()))
```

將決策樹視覺化

先看執行結果


沒有氣味 = 0
(i.e. 有氣味)
odor_n ≤ 0.5
entropy = 0.999
samples = 6093
value = [3169, 2924]
class = e
True
False
有瘀傷 = 0
(i.e. 沒有瘀傷)
bruises_t ≤ 0.5
samples = 3447
value = [604, 2843]
class = p
spore-print-color_r ≤ 0.5
samples = 2646
value = [2565, 81]
class = e
孢子綠色 = 0
(i.e.孢子不是綠的)
有氣味
沒有瘀傷
有毒
entropy = 0.0
samples = 2445
class = p
stalk-root_c ≤ 0.5
entropy = 0.969
samples = 1002
value = [604, 398]
class = e
stalk-surface-below-ring_y ≤ 0.5
entropy = 0.103
samples = 2600
value = [2565, 35]
class = e
entropy = 0.0
samples = 46
class = p
有毒
沒氣味
孢子是綠色
stalk-root_r ≤ 0.5
entropy = 0.937
samples = 616
value = [218, 398]
class = p
entropy = 0.0
samples = 386
value = [386, 0]
class = e
gill-size_n ≤ 0.5
entropy = 0.027
samples = 2560
value = [2553, 7]
class = e
gill-size_n ≤ 0.5
entropy = 0.881
samples = 40
value = [12, 28]
class = p
gill-spacing_w ≤ 0.5
entropy = 0.613
samples = 469
value = [71, 398]
class = p
entropy = 0.0
samples = 147
value = [147, 0]
class = e
entropy = 0.0
samples = 2419
value = [2419, 0]
class = e
bruises_t ≤ 0.5
entropy = 0.285
samples = 141
value = [134, 7]
class = e
entropy = 0.0
samples = 12
value = [12, 0]
class = e
entropy = 0.0
samples = 28
value = [0, 28]
class = p
entropy = 0.0
samples = 398
value = [0, 398]
class = p
entropy = 0.0
samples = 71
value = [71, 0]
class = e
entropy = 0.0
samples = 134
value = [134, 0]
class = e
entropy = 0.0
samples = 7
value = [0, 7]
class = p

原理簡介

- Step 1：將決策樹模型**輸出成 DOT 格式**

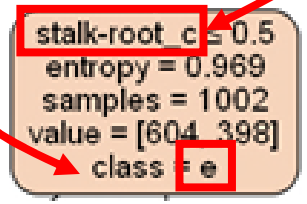
```
1  from sklearn import tree
2
3  dot_data = tree.export_graphviz(
4      classifier.classifier,
5      filled=True,
6      feature_names=X_test.columns,
7      class_names=["e", "p"],
8      rounded=True,
9      special_characters=True )
```

決策樹模型本身 →

方框是否填滿顏色 →

是否使用圓角矩形 →

是否使用特殊符號 →




```
digraph Tree {
node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;
edge [fontname=helvetica] ;
0 [label=<odor_n &le; 0.5<br/>entropy = 0.999<br/>samples = 6093<br/>value = [3169, 2924]<br/>class = e>, fillcolor="#e5813914"] ;
1 [label=<bruises_t &le; 0.5<br/>entropy = 0.67<br/>samples = 3447<br/>value = [604, 2843]<br/>class = p>, fillcolor="#399de5c9"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label=<entropy = 0.0<br/>samples = 2445<br/>value = [0, 2445]<br/>class = p>, fillcolor="#399de5ff"] ;
1 -> 2 ;
3 [label=<stalk-root_c &le; 0.5<br/>entropy = 0.969<br/>samples = 1002<br/>value = [604, 398]<br/>class = e>, fillcolor="#e5813957"] ;
1 -> 3 ;
4 [label=<stalk-root_r &le; 0.5<br/>entropy = 0.937<br/>samples = 616<br/>value = [218, 398]<br/>class = p>, fillcolor="#399de573"] ;
3 -> 4 ;
5 [label=<gill-spacing_w &le; 0.5<br/>entropy = 0.613<br/>samples = 469<br/>value = [71, 398]<br/>class = p>, fillcolor="#399de5d2"] ;
4 -> 5 ;
6 [label=<entropy = 0.0<br/>samples = 398<br/>value = [0, 398]<br/>class = p>, fillcolor="#399de5ff"] ;
5 -> 6 ;
7 [label=<entropy = 0.0<br/>samples = 71<br/>value = [71, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
5 -> 7 ;
8 [label=<entropy = 0.0<br/>samples = 147<br/>value = [147, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
4 -> 8 ;
9 [label=<entropy = 0.0<br/>samples = 386<br/>value = [386, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
3 -> 9 ;
10 [label=<spore-print-color_r &le; 0.5<br/>entropy = 0.197<br/>samples = 2646<br/>value = [2565, 81]<br/>class = e>, fillcolor="#e58139f7"] ;
0 -> 10 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
11 [label=<stalk-surface-below-ring_y &le; 0.5<br/>entropy = 0.103<br/>samples = 2600<br/>value = [2565, 35]<br/>class = e>, fillcolor="#e58139fc"] ;
10 -> 11 ;
12 [label=<gill-size_n &le; 0.5<br/>entropy = 0.027<br/>samples = 2560<br/>value = [2553, 7]<br/>class = e>, fillcolor="#e58139fe"] ;
11 -> 12 ;
13 [label=<entropy = 0.0<br/>samples = 2419<br/>value = [2419, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
12 -> 13 ;
14 [label=<bruises_t &le; 0.5<br/>entropy = 0.285<br/>samples = 141<br/>value = [134, 7]<br/>class = e>, fillcolor="#e58139f2"] ;
12 -> 14 ;
15 [label=<entropy = 0.0<br/>samples = 134<br/>value = [134, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
14 -> 15 ;
16 [label=<entropy = 0.0<br/>samples = 7<br/>value = [0, 7]<br/>class = p>, fillcolor="#399de5ff"] ;
14 -> 16 ;
17 [label=<gill-size_n &le; 0.5<br/>entropy = 0.881<br/>samples = 40<br/>value = [12, 28]<br/>class = p>, fillcolor="#399de592"] ;
11 -> 17 ;
18 [label=<entropy = 0.0<br/>samples = 12<br/>value = [12, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
17 -> 18 ;
19 [label=<entropy = 0.0<br/>samples = 28<br/>value = [0, 28]<br/>class = p>, fillcolor="#399de5ff"] ;
17 -> 19 ;
20 [label=<entropy = 0.0<br/>samples = 46<br/>value = [0, 46]<br/>class = p>, fillcolor="#399de5ff"] ;
10 -> 20 ;
}
```

dot_data（DOT 格式）

原理簡介

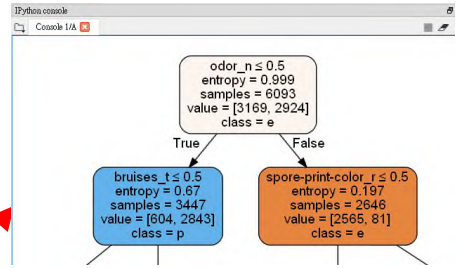
- Step 2：將 **DOT 格式**，轉成 **.PNG 格式**


```
digraph Tree {
node [shape=box, style="filled, rounded", color="black", fontname=helvetica] ;
edge [fontname=helvetica] ;
0 [label=<odor_n &le; 0.5<br/>entropy = 0.999<br/>samples = 6093<br/>value = [3169, 2924]<br/>class = e>, fillcolor="#e5813914"] ;
1 [label=<bruises_t &le; 0.5<br/>entropy = 0.67<br/>samples = 3447<br/>value = [604, 2843]<br/>class = p>, fillcolor="#399de5c9"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label=<entropy = 0.0<br/>samples = 2445<br/>value = [0, 2445]<br/>class = p>, fillcolor="#399de5ff"] ;
1 -> 2 ;
3 [label=<stalk-root_c &le; 0.5<br/>entropy = 0.969<br/>samples = 1002<br/>value = [604, 398]<br/>class = e>, fillcolor="#e5813957"] ;
1 -> 3 ;
4 [label=<stalk-root_r &le; 0.5<br/>entropy = 0.937<br/>samples = 616<br/>value = [218, 398]<br/>class = p>, fillcolor="#399de573"] ;
3 -> 4 ;
5 [label=<gill-spacing_w &le; 0.5<br/>entropy = 0.613<br/>samples = 469<br/>value = [71, 398]<br/>class = p>, fillcolor="#399de5d2"] ;
4 -> 5 ;
6 [label=<entropy = 0.0<br/>samples = 398<br/>value = [0, 398]<br/>class = p>, fillcolor="#399de5ff"] ;
5 -> 6 ;
7 [label=<entropy = 0.0<br/>samples = 71<br/>value = [71, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
5 -> 7 ;
8 [label=<entropy = 0.0<br/>samples = 147<br/>value = [147, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
4 -> 8 ;
9 [label=<entropy = 0.0<br/>samples = 386<br/>value = [386, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
3 -> 9 ;
10 [label=<spore-print-color_r &le; 0.5<br/>entropy = 0.197<br/>samples = 2646<br/>value = [2565, 81]<br/>class = e>, fillcolor="#e58139f7"] ;
0 -> 10 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
11 [label=<stalk-surface-below-ring_y &le; 0.5<br/>entropy = 0.103<br/>samples = 2600<br/>value = [2565, 35]<br/>class = e>, fillcolor="#e58139fc"] ;
10 -> 11 ;
12 [label=<gill-size_n &le; 0.5<br/>entropy = 0.027<br/>samples = 2560<br/>value = [2553, 7]<br/>class = e>, fillcolor="#e58139fe"] ;
11 -> 12 ;
13 [label=<entropy = 0.0<br/>samples = 2419<br/>value = [2419, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
12 -> 13 ;
14 [label=<bruises_t &le; 0.5<br/>entropy = 0.285<br/>samples = 141<br/>value = [134, 7]<br/>class = e>, fillcolor="#e58139f2"] ;
12 -> 14 ;
15 [label=<entropy = 0.0<br/>samples = 134<br/>value = [134, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
14 -> 15 ;
16 [label=<entropy = 0.0<br/>samples = 7<br/>value = [0, 7]<br/>class = p>, fillcolor="#399de5ff"] ;
14 -> 16 ;
17 [label=<gill-size_n &le; 0.5<br/>entropy = 0.881<br/>samples = 40<br/>value = [12, 28]<br/>class = p>, fillcolor="#399de592"] ;
11 -> 17 ;
18 [label=<entropy = 0.0<br/>samples = 12<br/>value = [12, 0]<br/>class = e>, fillcolor="#e58139ff"] ;
17 -> 18 ;
19 [label=<entropy = 0.0<br/>samples = 28<br/>value = [0, 28]<br/>class = p>, fillcolor="#399de5ff"] ;
17 -> 19 ;
20 [label=<entropy = 0.0<br/>samples = 46<br/>value = [0, 46]<br/>class = p>, fillcolor="#399de5ff"] ;
10 -> 20 ;
}
```


使用程式碼

```
1  import pydotplus
2  from IPython.display import Image, display
3
4  graph = pydotplus.graph_from_dot_data(dot_data)
5  display(Image(graph.create_png()))
```

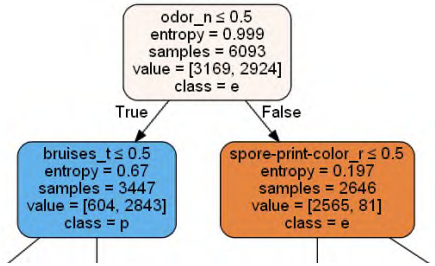
顯示於開發環境中



使用命令列工具


```
dot -Tpng tree.dot -o tree.png
```

顯示於作業系統中



軟體安裝

PyDotPlus 函式庫

```
import pydotplus
from IPython.display import Image

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

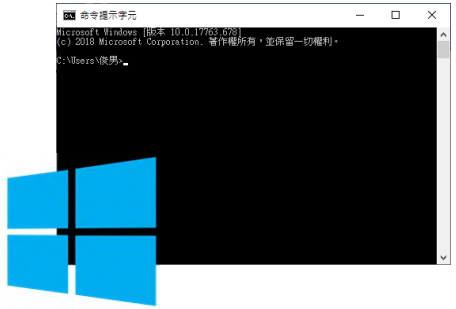
SPYDER

PyDotPlus 函式庫

Anaconda Prompt

conda install **pydotplus**

系統執行軟體



GraphViz 執行環境 ＋

https://graphviz.org/download/

下載執行環境，至作業系統內解壓縮安裝

隨堂練習：安裝相關軟體

- 請打開 **Anaconda Prompt**，執行下列指令：
  - conda install pydotplus

- 請到 https://graphviz.org/download/ 網址，下載與您作業系統相符的 **GraphViz 執行期環境**，並將之解壓縮後，拷貝到一個你喜歡的路徑。

- 安裝時，請牢記 **GraphViz 執行檔路徑**，後續設定會用到。
  （比如說： C:/Program Files (x86)/Graphviz/bin ）

使用「標準函式庫」視覺化

- 程式碼解說

引入必要套件

{0:"e", 1:"p"} 依照關鍵詞排序，將對應值提取成串列。像這樣：["e", "p"]

將「決策樹」匯出成 DOT 格式

將「DOT 格式」轉成 PNG 圖檔後繪出

```
1  GRAPHVIZ_INSTALL = "C:/Program Files/Graphviz/bin"
2
3  from sklearn import tree
4  import pydotplus
5  from IPython.display import Image, display
6  import os
7
8  os.environ["PATH"] += os.pathsep + GRAPHVIZ_INSTALL
9  cls_name = [Y_mapping[key] for key in sorted(Y_mapping.keys())]
10 dot_data = tree.export_graphviz(classifier.classifier, filled=True,
11                 feature_names=X_test.columns, class_names=cls_name,
12                 rounded=True, special_characters=True)
13 graph = pydotplus.graph_from_dot_data(dot_data)
14 display(Image(graph.create_png()))
```

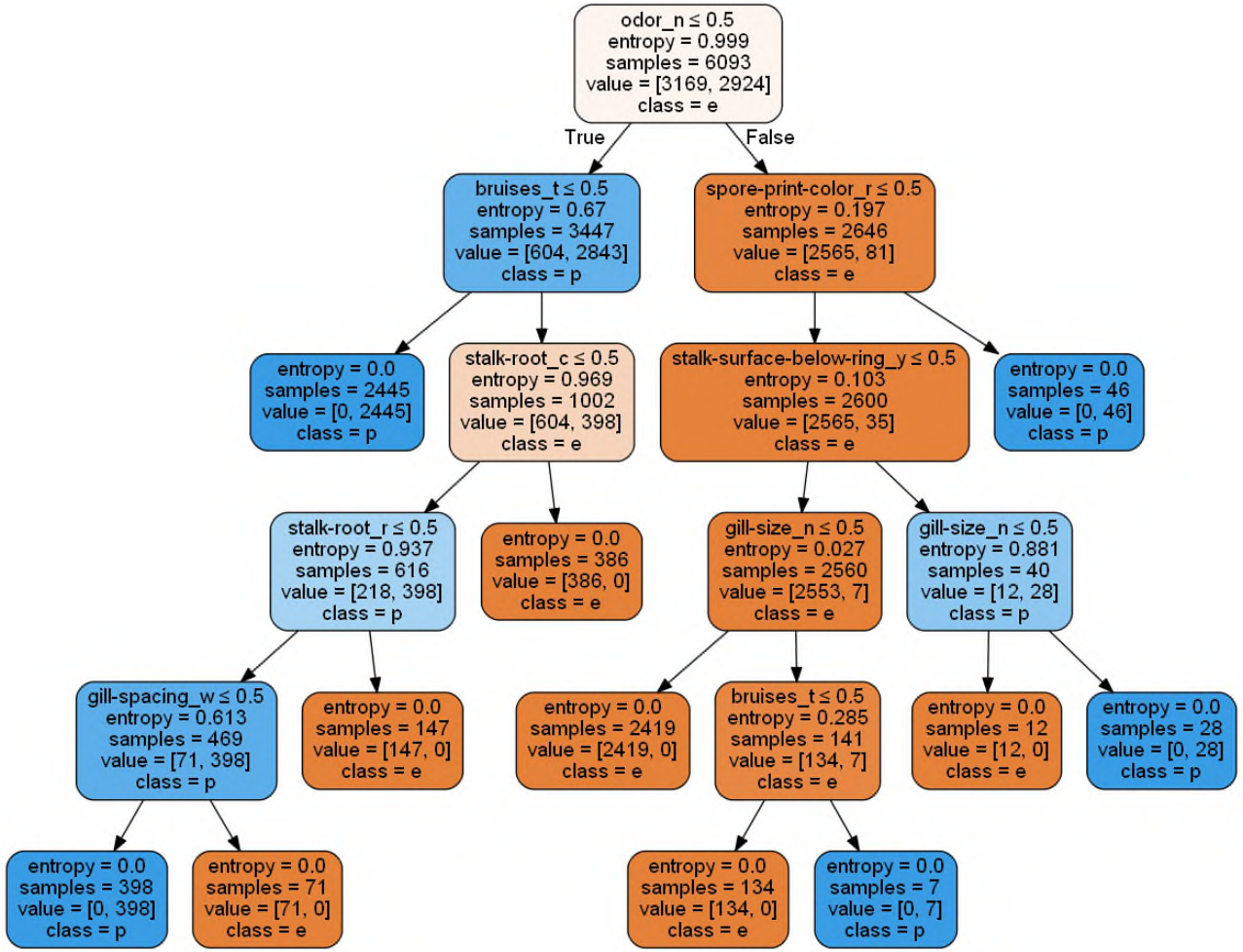
GraphViz 執行檔路徑

原始的軟體搜尋路徑

路徑分隔符號

使用「標準函式庫」視覺化

- 執行結果



隨堂練習：使用「標準函式庫」視覺化

- 請輸入下列程式碼，並執行看看。是否能繪製出**決策樹**：

```
GRAPHVIZ_INSTALL = "C:/Program Files/Graphviz/bin"

from sklearn import tree
import pydotplus
from IPython.display import Image, display
import os

os.environ["PATH"] += os.pathsep + GRAPHVIZ_INSTALL
cls_name = [Y_mapping[key] for key in sorted(Y_mapping.keys())]
dot_data = tree.export_graphviz(classifier.classifier, filled=True,
        feature_names=X_test.columns, class_names=cls_name,
        rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
display(Image(graph.create_png()))
```

使用「快樂版函式庫」視覺化

- 程式碼講解：

/HappyML/model_drawer.py

```
1  from sklearn import tree
2  import os
3
4  try:
5      import pydotplus
6  except ImportError:
7      pass
8
9  def tree_drawer(classifier, feature_names=None, target_names=None, graphviz_bin='C:/Program Files (x86)/Graphviz2.38/bin/'):
10     os.environ["PATH"] += os.pathsep + graphviz_bin
11     dot_data = tree.export_graphviz(classifier, filled=True, feature_names=feature_names,
12                                     class_names=target_names, rounded=True, special_characters=True)
13     return pydotplus.graph_from_dot_data(dot_data)
```

引入必要的標準版套件

可讓使用者在沒安裝 pydotplus 時，
也能安全引用 model_drawer 的方法。

↑ 其它程式碼，幾乎都與「Python 函式庫」寫法相同。講解省略。

使用「快樂版函式庫」視覺化

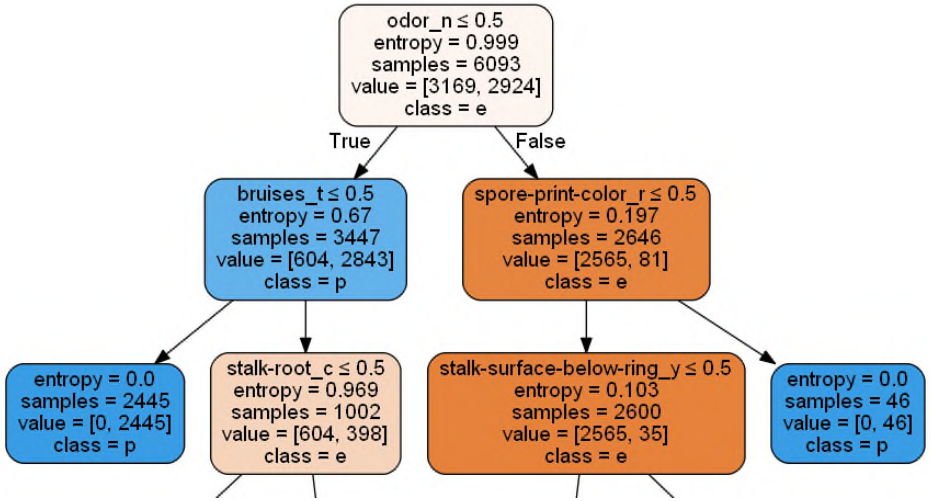
- 呼叫範例

```
GRAPHVIZ_INSTALL = "C:/Program Files/Graphviz/bin"

import HappyML.model_drawer as md
from IPython.display import Image, display

cls_name = [Y_mapping[key] for key in sorted(Y_mapping.keys())]
graph = md.tree_drawer(classifier=classifier.classifier,
        feature_names=X_test.columns, target_names=cls_name,
        graphviz_bin=GRAPHVIZ_INSTALL)
display(Image(graph.create_png()))
```

- 執行結果



隨堂練習：使用「快樂版函式庫」視覺化

- 請將原來的程式碼註解掉，改用下列程式碼繪製**決策樹**：

```
GRAPHVIZ_INSTALL = "C:/Program Files/Graphviz/bin"

import HappyML.model_drawer as md
from IPython.display import Image, display

cls_name = [Y_mapping[key] for key in sorted(Y_mapping.keys())]
graph = md.tree_drawer(classifier=classifier.classifier,
        feature_names=X_test.columns, target_names=cls_name,
        graphviz_bin=GRAPHVIZ_INSTALL)
display(Image(graph.create_png()))
```

課後作業：員工離職原因調查

- 資料集描述
  - IBM 人資部收集了 1470 筆員工留 / 離職資料集，如 **HR-Employee-Attrition.csv** 所示（資料原始出處： https://is.gd/AhAETa ）。
  - 資料集欄位解釋如下：
    - **Age**：年齡。
    - **Attrition**：是否離職。
    - **BusinessTravel**：出差頻率。
    - ……
- 題目要求
  - 請先過濾掉那些「**不顯著**」的特徵。
  - 使用決策樹，以 **10 次交叉驗證**，顯示您模型的**確度、廣度、精度、F1**。
  - **繪製**出決策樹，並用它說明，員工留 / 離職**最主要的一個原因**是什麼？

課後作業：員工離職原因調查

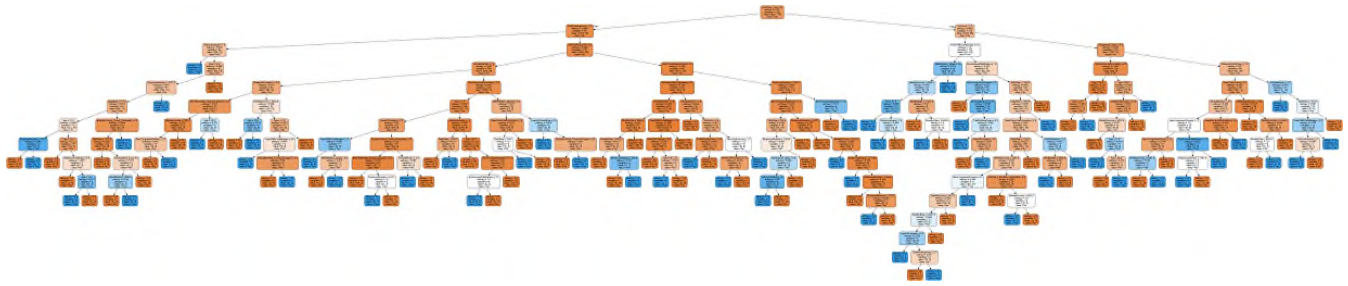
- 提示：
  - **應變數 Y** 在 [1] 欄，其它皆為**自變數 X**。切分 X、Y 可以這麼做：
    - **自變數 X**： x_columns=[i for i in range(35) if i != 1]
    - **應變數 Y**： y_columns=[1]
  - [1, 3, 6, 10, 14, 16, 20, 21] 需要做 One Hot Encoding。
  - 移除「**虛擬變數陷阱**」，只要把 **onehot_encoder**() 的 remove_trap=True 即可。
- 輸出

```
The Significant Level: 0.05

--- The p-values of Feature Importance ---
TRUE  <0.05 0.00000000e+00 (MonthlyIncome)
TRUE  <0.05 3.28793276e-262 (MonthlyRate)
TRUE  <0.05 4.92392441e-210 (DailyRate)
TRUE  <0.05 4.14926182e-52 (TotalWorkingYears)
TRUE  <0.05 9.24706566e-33 (YearsAtCompany)
......
FALSE >0.05 6.27821283e-01 (PercentSalaryHike)
FALSE >0.05 9.82052402e-01 (PerformanceRating)
FALSE >0.05 9.90153390e-01 (JobRole_Research Scientist)
FALSE >0.05 1.00000000e+00 (EmployeeCount)
FALSE >0.05 1.00000000e+00 (StandardHours)

Number of Features Selected: 29
```

```
----- Decision Tree Classification -----
10 Folds Mean Accuracy: 0.778267119509781
10 Folds Mean Recall: 0.6189479783121814
10 Folds Mean Precision: 0.6086799674101879
10 Folds Mean F1-Score: 0.6108189437966041
```



本章總結

- **決策樹演算法**
  - **ID3**：使用「**資訊熵**」 + 「**資訊增益**」
  - **CART**：使用「**基尼指數**」 + 「**基尼增益**」

- **相關函式庫**
  - 決策樹模型： sklearn.tree.**DecisionTreeClassifier**
  - 輸出 DOT 格式： sklearn.tree.**export_graphviz**()
  - 繪製 DOT 成 PNG： **pydotplus**、IPython.display.**Image**

- **何時使用決策樹**
  - 樣本點成「**巧拼板**」分佈，東一塊、西一塊時。
  - **自變數 X** 多為「離散數值」時（離散 = 好分）。
  - 比起預測結果，更想得到模型**如何**分類的**合理解釋**時。