紀老師程式教學網
https://www.facebook.com/teacherchi

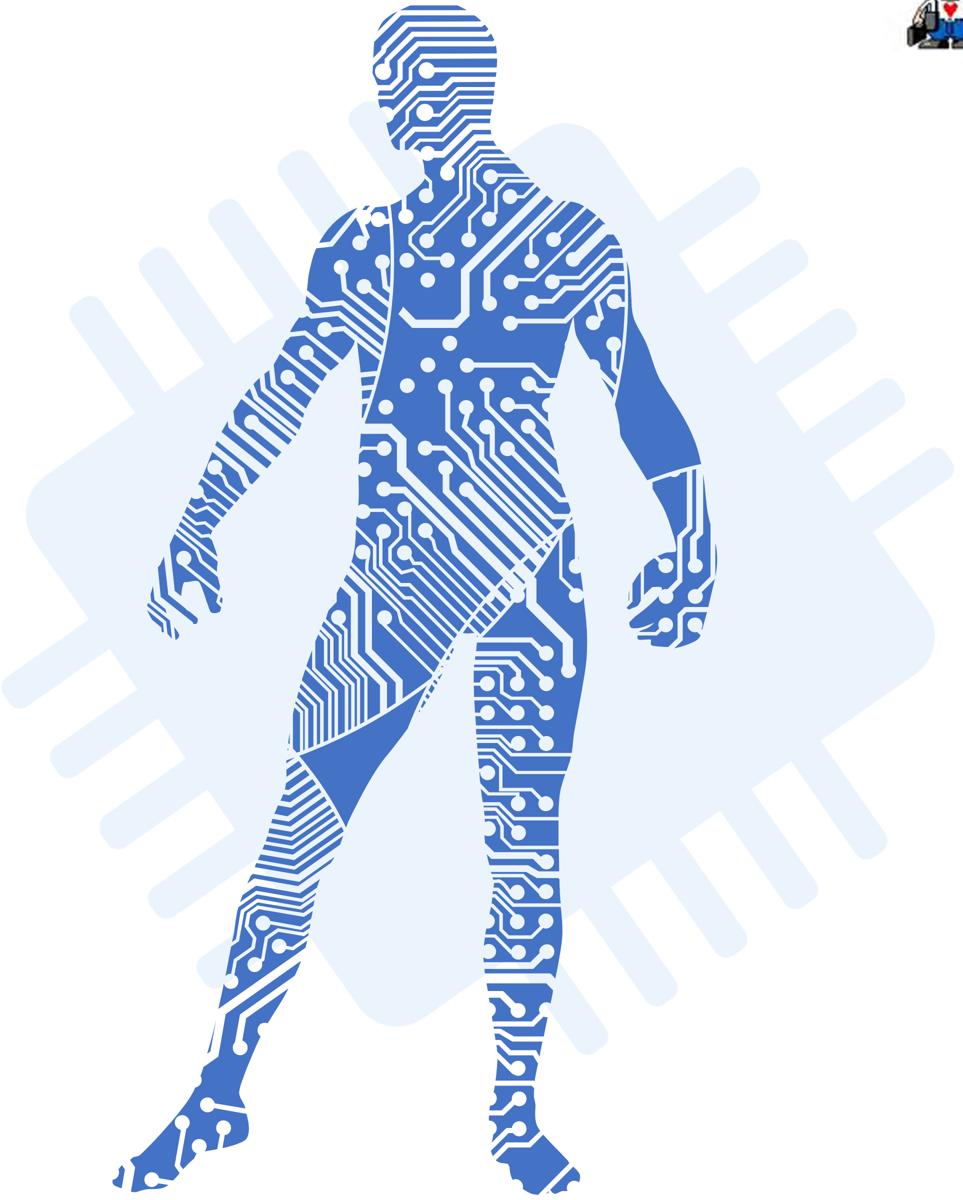
機器學習

第 13 章 隨機森林
(Random Forest)

講師：紀俊男

本章大綱

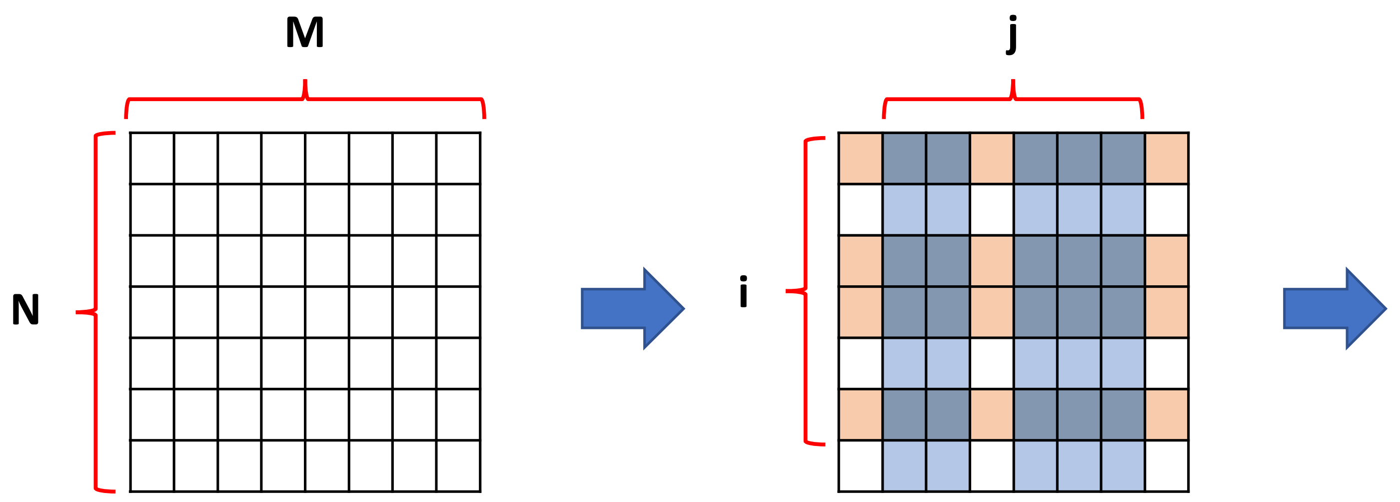
- 原理解說
- 資料前處理
- 實作隨機森林
- PCA 降維
- 本章總結



AI 原理解説

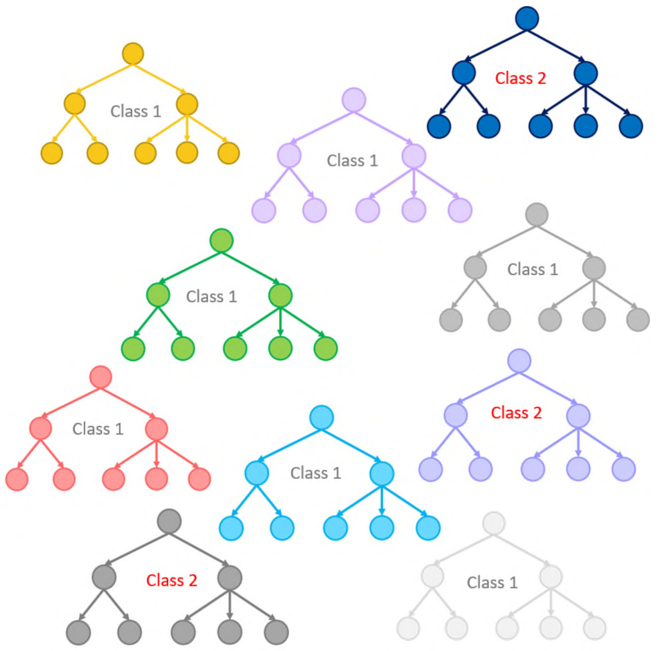
使用多棵決策樹「投票」

- N: 樣本點個數 M: 自變數個數
- 對於每棵樹
  - C_i^N ：以 i 個樣本點建立一棵決策樹（取後放回）
  - C_j^M ：以 j 個自變數建立一棵決策樹（取後放回）
- 對於預測結果，投票決定

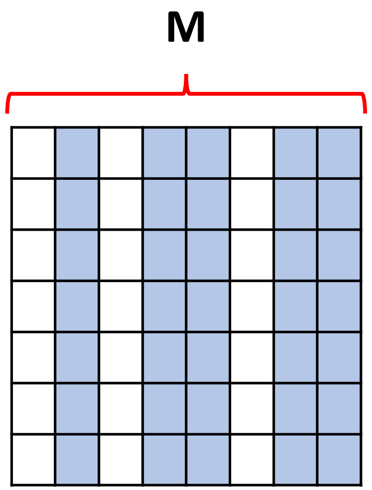


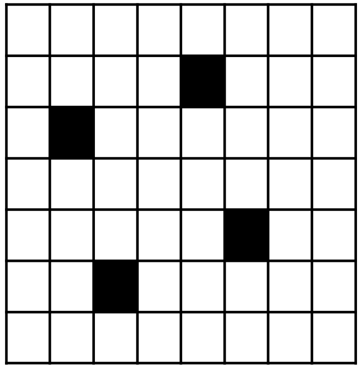
- Class 1 : 7 票
- Class 2 : 3 票

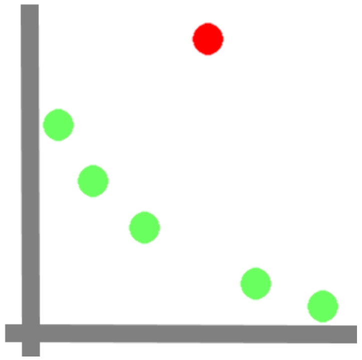
→ Class 1



隨機森林的優點







能處理大量、
無相關性的
自變數

對「缺失資料」
（Missing Data）
耐受度良好

能產生較為
「不偏（Un-bias）」
的訓練集

對於「離群值」
（Outliers）
抗性較佳

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

Wine.csv

	K	L	M	N
1	Hue	OD280	Proline	Type
2	1.04	3.92	1065	1
3	1.05	3.4	1050	1
4	1.03	3.17	1185	1
5	0.86	3.45	1480	1
6	1.04	2.93	735	1
7	1.05	2.85	1450	1
8	1.02	3.58	1290	1
9	1.06	3.58	1295	1
10	1.08	2.85	1045	1

目的：利用「酒精濃度、蘋果酸...」
--> 推算「紅酒等級」

- Alcohol：酒精濃度
- Malic_Acid：蘋果酸濃度
- Ash：粉煤灰濃度
- Ash_Alcanity：粉煤灰鹼度
- Magnesium：鎂濃度
- Total_Phenols：總酚濃度
- Flavanoids：黃烷類濃度
- Nonflavanoid phenols：非黃酮酚濃度
- Proanthocyanins：前花青素濃度
- Color intensity：透光度
- Hue：色澤
- OD280/OD315 of diluted wines：蛋白質吸光度...等指標
- Proline：脯氨酸
- Type：等級
- 原始來源：https://is.gd/Kbmb9h

資料前處理

撰寫程式碼

```python
import HappyML.preprocessor as pp

# Load Data, also can be loaded by sklearn.datasets.load_wine()
dataset = pp.dataset(file="Wine.csv")

# Decomposition
X, Y = pp.decomposition(dataset, x_columns=[i for i in range(13)], y_columns=[13])

# By KBestSelector
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)
```

<u>資料前處理流程</u>：

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
   （無類別資料）
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, also can be loaded by sklearn.datasets.load_wine()
dataset = pp.dataset(file="Wine.csv")

# Decomposition
X, Y = pp.decomposition(dataset, x_columns=[i for i in range(13)], y_columns=[13])

# By KBestSelector
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)
```

實作隨機森林

使用「標準函式庫」實作

程式碼

載入必要套件

```
from sklearn.ensemble import RandomForestClassifier
import time

classifier = RandomForestClassifier(n_estimators=10, criterion="entropy", random_state=int(time.time()))
classifier.fit(X_train, Y_train.values.ravel())
Y_pred = classifier.predict(X_test)
```

產生物件本身 → classifier = RandomForestClassifier(...)

訓練 → classifier.fit(...)

預測 → Y_pred = classifier.predict(X_test)

n_estimators=10：決策樹個數（可用網格搜尋推估）

criterion="entropy"：可選 “entropy” (ID3) 或者 “gini” (CART)

執行結果

Y_test

Index	Type
15	1
150	3
9	1
129	2
36	1
131	3
134	3
100	2
62	2
96	2

Y_pred

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

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：

```python
from sklearn.ensemble import RandomForestClassifier
import time

classifier = RandomForestClassifier(n_estimators=10, criterion="entropy", random_state=int(time.time()))
classifier.fit(X_train, Y_train.values.ravel())
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

幾棵決策樹

演算法

```
from sklearn.ensemble import RandomForestClassifier
import time

class RandomForest:
    __classifier = None
    __n_estimators = None
    __criterion = None
    __y_columns = None

    def __init__(self, n_estimators=10, criterion="entropy"):
        self.__n_estimators = n_estimators
        self.__criterion = criterion
        self.__classifier = RandomForestClassifier(n_estimators=self.__n_estimators,
                                    criterion=self.__criterion, random_state=int(time.time()))
```

使用「快樂版函式庫」實作

- 程式碼解說（2）：

/HappyML/classification.py

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

    @classifier.setter
    def classifier(self, classifier):
        self.__classifier = classifier

    @property
    def n_estimators(self):
        return self.__n_estimators

    def fit(self, x_train, y_train):
        self.classifier.fit(x_train, y_train.values.ravel())
        self.__y_columns = y_train.columns
        return self

    def predict(self, x_test):
        return pd.DataFrame(self.classifier.predict(x_test), index=x_test.index, columns=self.__y_columns)
```

classifier 的 setter & getter

n_estimators 的 getter

訓練

← 保存 Y_train 的欄位名稱

預測

↓ 把預測出來的 Y_pred 重新包裝回 DataFrame 再傳回

使用「快樂版函式庫」實作

- 呼叫範例

```python
from HappyML.classification import RandomForest

classifier = RandomForest(n_estimators=10, criterion="entropy")
Y_pred = classifier.fit(X_train, Y_train).predict(X_test)
```

- 執行結果

Y_test

Index	Type
15	1
150	3
9	1
129	2
36	1
131	3
134	3
100	2
62	2
96	2

Y_pred

Index	Type
15	1
150	3
9	1
129	2
36	1
131	3
134	2
100	2
62	2
96	2

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：

```python
from HappyML.classification import RandomForest

classifier = RandomForest(n_estimators=10, criterion="entropy")
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。

計算隨機森林的效能

- 程式碼解說

```python
from HappyML.performance import KFoldClassificationPerformance

K = 10
kfp = KFoldClassificationPerformance(x_ary=X, y_ary=Y, classifier=classifier.classifier, k_fold=K)

print("----- Random Forest Classification -----")
print("{} Folds Mean Accuracy: {}".format(K, kfp.accuracy()))
print("{} Folds Mean Recall: {}".format(K, kfp.recall()))
print("{} Folds Mean Precision: {}".format(K, kfp.precision()))
print("{} Folds Mean F1-Score: {}".format(K, kfp.f_score()))
```

- 執行結果

```
----- Random Forest Classification -----
10 Folds Mean Accuracy: 0.9502923976608187
10 Folds Mean Recall: 0.9504365079365508
10 Folds Mean Precision: 0.9595899470899472
10 Folds Mean F1-Score: 0.9497772597772597
```

隨堂練習：計算隨機森林的效能

- 請撰寫下列程式碼，計算隨機森林的效能：

```python
from HappyML.performance import KFoldClassificationPerformance

K = 10
kfp = KFoldClassificationPerformance(x_ary=X, y_ary=Y, classifier=classifier.classifier, k_fold=K)

print("----- Random Forest Classification -----")
print("{} Folds Mean Accuracy: {}".format(K, kfp.accuracy()))
print("{} Folds Mean Recall: {}".format(K, kfp.recall()))
print("{} Folds Mean Precision: {}".format(K, kfp.precision()))
print("{} Folds Mean F1-Score: {}".format(K, kfp.f_score()))
```

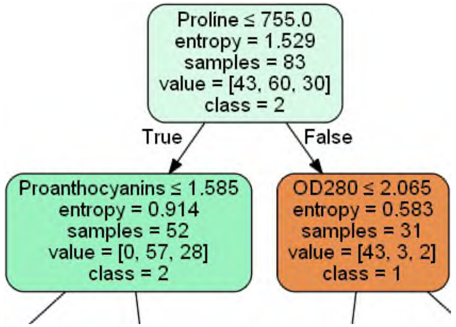
隨機森林視覺化

- 程式碼解說

```
① GRAPHVIZ_INSTALL = "C:/Program Files/Graphviz/bin"

② import HappyML.model_drawer as md
  from IPython.display import Image, display

③ clfr = classifier.classifier.estimators_[0]
④ graph = md.tree_drawer(classifier=clfr, feature_names=X_test.columns,
      target_names="123", graphviz_bin=GRAPHVIZ_INSTALL)
⑤ display(Image(graph.create_png()))
```

範例：只顯示一棵樹
（可以用迴圈顯示所有樹）

- 執行結果



隨堂練習：隨機森林視覺化

- 請撰寫下列程式碼，將隨機森林視覺化：

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

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

clfr = classifier.classifier.estimators_[0]
graph = md.tree_drawer(classifier=clfr, feature_names=X_test.columns,
    target_names="123", graphviz_bin=GRAPHVIZ_INSTALL)
display(Image(graph.create_png()))
```

隨機森林「決定邊界」視覺化

- 程式碼解說

前處理程式碼

```python
import HappyML.preprocessor as pp

...

# By KBestSelector
from HappyML.preprocessor import KBestSelector
selector = KBestSelector(best_k=2)
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)

# Feature Scaling
X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

best_k = 2
壓制在兩個特徵之下

一定要記得做「特徵縮放」
（維持各維度資料的比例尺一致）

圖形程式碼

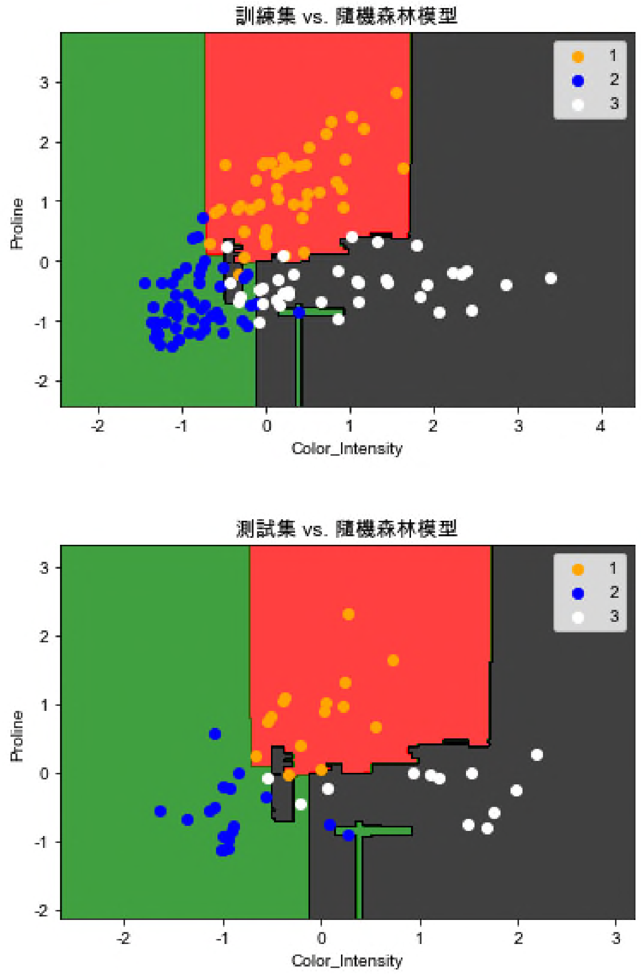
```python
import HappyML.model_drawer as md

md.classify_result(x=X_train, y=Y_train, classifier=classifier.classifier,
                   fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
                   title="訓練集 vs. 隨機森林模型", font="DFKai-sb")
md.classify_result(x=X_test, y=Y_test, classifier=classifier.classifier,
                   fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
                   title="測試集 vs. 隨機森林模型", font="DFKai-sb")
```

隨機森林「決定邊界」視覺化

- 執行結果

```
Number of Features Selected: 2
----- Random Forest Classification -----
10 Folds Mean Accuracy: 0.8783281733746129
10 Folds Mean Recall: 0.8668253968253969
10 Folds Mean Precision: 0.890489417989418
10 Folds Mean F1-Score: 0.8635534889946654
```

請各位記住這些數字，
以便稍後與PCA對比之用。



如「巧拼板」般的決定邊界

隨堂練習：「決定邊界」視覺化

- 請先將 best_k 改為 =2：
  - selector = KBestSelector(best_k=2)

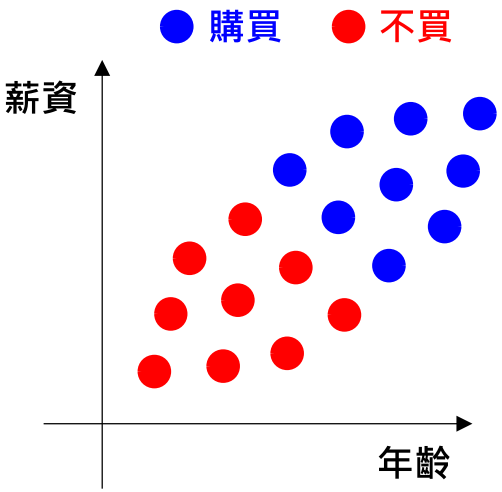
- 再將「特徵縮放」程式碼加上去：
  - X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))

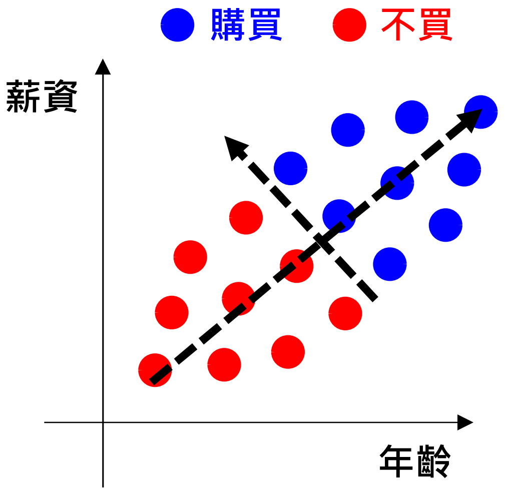
- 撰寫下列「決定邊界」視覺化的程式碼，並將整個程式執行一遍：

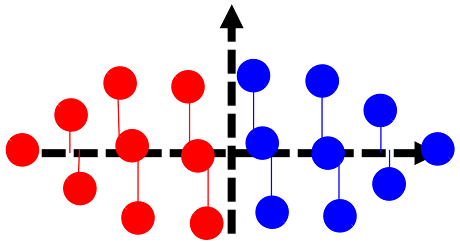
```python
import HappyML.model_drawer as md

md.classify_result(x=X_train, y=Y_train, classifier=classifier.classifier,
                   fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
                   title="訓練集 vs. 隨機森林模型", font="DFKai-sb")
md.classify_result(x=X_test, y=Y_test, classifier=classifier.classifier,
                   fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
                   title="測試集 vs. 隨機森林模型", font="DFKai-sb")
```

PCA 降維

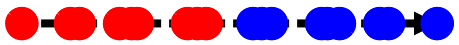
何謂「PCA 降維」？







往新 X 座標投影



僅剩一個維度，資訊量略損

- PCA 降維法
  - 以「**最大方差法**」找最適座標（因此一定要先「**特徵縮放**」）
  - 所有資料點往**新 X 座標**投影（此乃線性代數之「**座標變換**」）
  - 降維成功

使用「標準函式庫」實作

- 程式碼解說

```python
# Feature Scaling
X = pp.feature_scaling(fit_ary=X, transform_arys=X)

# PCA (with Python Libararies)
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

pca = PCA(n_components=None)
pca.fit(X)
info_covered = pca.explained_variance_ratio_
cumulated_sum = np.cumsum(info_covered)
plt.plot(cumulated_sum, color="blue")
```

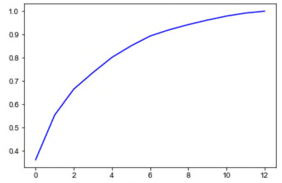
1. 「特徵縮放」
（**PCA** 靠最大方差法，一定得做！）
2. **n_components=None**（需要幾個特徵）
強制列出每一特徵所含資訊量
3. **info_covered**（**NDArray**）
列出每一特徵所含訊息量

```
array([0.36198848, 0.1920749 , 0.11123631, 0.0706903 , 0.06563294,
       0.04935823, 0.04238679, 0.02680749, 0.02222153, 0.01930019,
       0.01736836, 0.01298233, 0.00795215])
```

4. **cumulated_sum**（**NDArray**）
累進總和

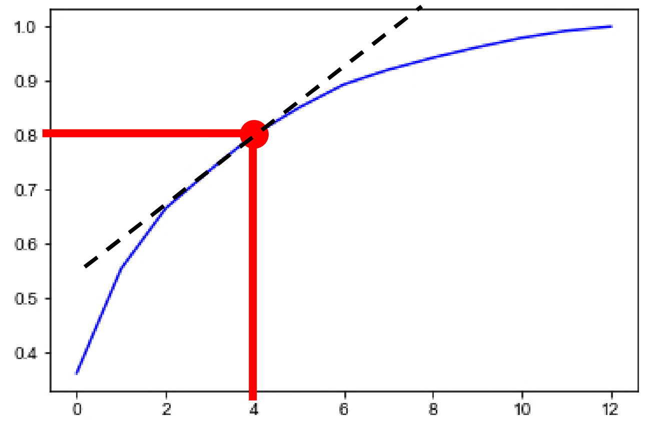
```
array([0.36198848, 0.55406338, 0.66529969, 0.73598999, 0.80162293,
       0.85098116, 0.89336795, 0.92017544, 0.94239698, 0.96169717,
       0.97906553, 0.99204785, 1.        ])
```

5. 累進總和作圖



使用「標準函式庫」實作

- 如何選取適合的 **n_components** = ?



包含 1 2 3 4 5 個元素的「資訊量」

```
array([0.36198848, 0.55406338, 0.66529969, 0.73598999, 0.80162293,
       0.85098116, 0.89336795, 0.92017544, 0.94239698, 0.96169717,
       0.97906553, 0.99204785, 1.        ])
```

- 「**凸邊形優化法**」（Convex Optimization）
  → 現況：斜率最接近 1 者，有最佳解。
- 「**資訊量選擇法**」
  → 至少涵蓋 80% 資訊量的「特徵數」
- 「**直接指定法**」
  → 我想要「2 個特徵」

```
import pandas as pd

pca = PCA(n_components=2)
X_columns = ["PCA_{}".format(i+1) for i in range(2)]
X = pd.DataFrame(pca.fit_transform(X),
                 index=X.index, columns=X_columns)
```

為了與 KBestSelector 比較，先用 2 個特徵

使用「標準函式庫」實作

- 程式**從頭執行一遍**的結果

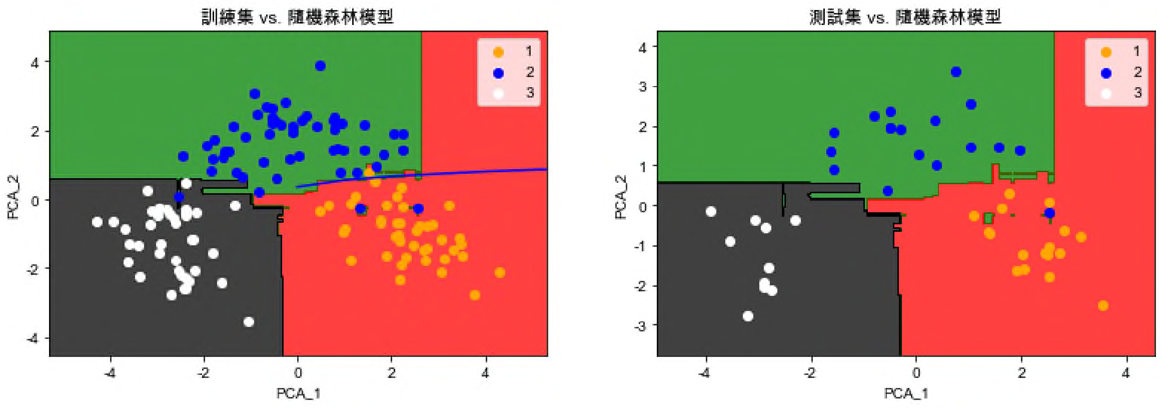
<u>PCA Components = 2</u>

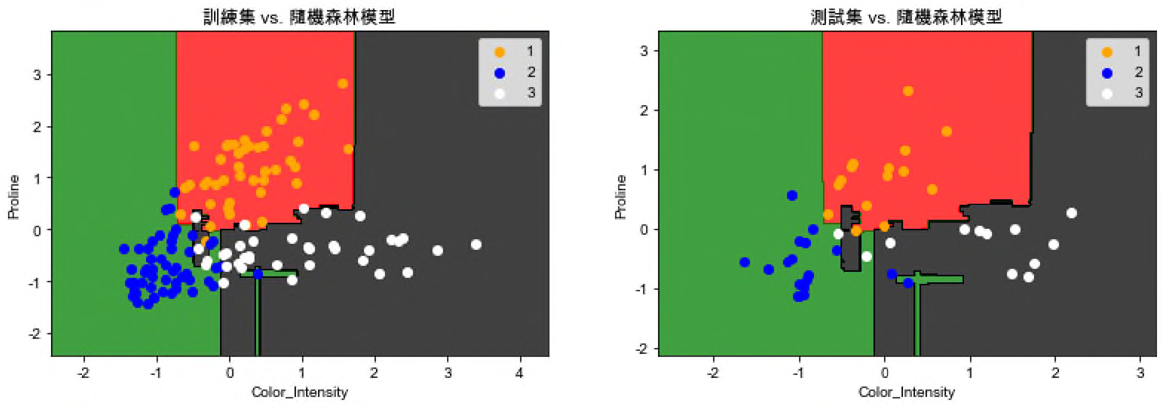
```
----- Random Forest Classification -----
10 Folds Mean Accuracy: 0.9329377364981081
10 Folds Mean Recall: 0.9330952380952381
10 Folds Mean Precision: 0.9451190476190476
10 Folds Mean F1-Score: 0.9334974284974284
```

VS.

<u>KBestSelector K=2</u>

```
----- Random Forest Classification -----
10 Folds Mean Accuracy: 0.8783281733746129
10 Folds Mean Recall: 0.8668253968253969
10 Folds Mean Precision: 0.89048941798941 8
10 Folds Mean F1-Score: 0.8635534889946654
```





隨堂練習：使用「標準函式庫」實作

- 請撰寫下列程式碼，用 Python 實作 PCA 降維：

特徵縮放

引入必要套件

計算特徵值含金量

累進總和與製圖

決定降剩兩維

實際降維 &
自製欄位名稱

```
# Feature Scaling
X = pp.feature_scaling(fit_ary=X, transform_arys=X)

# PCA (with Python Libararies)
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

pca = PCA(n_components=None)
pca.fit(X)
info_covered = pca.explained_variance_ratio_
cumulated_sum = np.cumsum(info_covered)
plt.plot(cumulated_sum, color="blue")

import pandas as pd
pca = PCA(n_components=2)
X_columns = ["PCA_{}".format(i+1) for i in range(2)]
X = pd.DataFrame(pca.fit_transform(X),
                 index=X.index, columns=X_columns)
```

使用「快樂版函式庫」實作

- 原始碼講解（1）：

/HappyML/preprocessor.py

引入必要套件

類別的成員變數

建構函數

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

class PCASelector:
    __selector = None
    __best_k = None
    __info_covered = None
    __strategy = None

    def __init__(self, best_k="auto"):
        if type(best_k) is int:
            self.__strategy = "fixed"
            self.__best_k = best_k
        elif type(best_k) is float:
            self.__strategy = "percentage"
            self.__info_covered = best_k
            self.__best_k = None
        else:
            self.__strategy = "auto"
            self.__best_k = None

        self.__selector = PCA(n_components=self.__best_k)
```

- best_k = “auto”
  由程式幫您選擇最佳的 K 值。
- best_k = 0.80（浮點數）
  選擇至少涵蓋 80% 的 K 值。
- best_k = 2（整數）
  直接選擇 2 個特徵。

使用「快樂版函式庫」實作

- 原始碼講解（2）：

/HappyML/preprocessor.py

selector 的 getter

best_k 的 getter

.fit()

是否逐步顯示

是否繪圖

先計算每個特徵所包含的資訊量

如果是「自動」

把 Y 軸份數，調整成與 X 軸相同

當上升比例不足1，停止！

找到最佳的 K 值 →

```python
    @property
    def selector(self):
        return self.__selector

    @property
    def best_k(self):
        return self.__best_k

    def fit(self, x_ary, verbose=False, plot=False):
        pca = PCA(n_components=None)
        pca.fit(x_ary)
        info_covered = pca.explained_variance_ratio_
        cumulated_sum = np.cumsum(info_covered)
        info_covered_dict = dict(zip([i+1 for i in range(info_covered.shape[0])], cumulated_sum))

        if self.__strategy == "auto":
            scaler = MinMaxScaler(feature_range=(0, info_covered.shape[0]-1))
            scaled_info_covered = scaler.fit_transform(info_covered.reshape(-1, 1)).ravel()
            for i in range(1, scaled_info_covered.shape[0]):
                if (scaled_info_covered[i-1]-scaled_info_covered[i]) < 1:
                    break
            self.__best_k = i
```

使用「快樂版函式庫」實作

- 原始碼講解（3）：

若是「涵蓋資訊」

```python
        elif self.__strategy =="percentage":
            current_best = 1
            cummulated_info = 0.0
            for i in info_covered:
                cummulated_info += i
                if cummulated_info < self.__info_covered:
                    current_best += 1
                else:
                    break
            self.__best_k = current_best

        self.__selector = PCA(n_components=self.best_k)
        self.selector.fit(x_ary)

        if verbose:
            print("Select {} components, covered information {:.2%}"
                  .format(self.best_k, info_covered_dict[self.best_k]))

        if plot:
            np.insert(cumulated_sum, 0, 0.0)
            plt.plot(cumulated_sum, color="blue")
            plt.scatter(x=self.best_k-1, y=cumulated_sum[self.best_k-1], color="red")
            plt.title("Components vs. Information")
            plt.xlabel("# of Components (0-based)")
            plt.ylabel("Covered Information (%)")
            plt.show()

        return self

    def transform(self, x_ary):
        X_columns = ["PCA_{}".format(i) for i in range(1, self.best_k+1)]
        return pd.DataFrame(self.selector.transform(x_ary), index=x_ary.index, columns=X_columns)
```

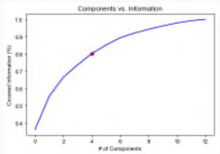
一直累加涵蓋資訊量，直到超過指定量為止

執行 PCA 降維

顯示降維結果

```
Select 4 components, covered information 73.60%
```

繪製圖形



合成指定個數的特徵

使用「快樂版函式庫」實作

- 呼叫範例：

1

```
# Feature Scaling
X = pp.feature_scaling(fit_ary=X, transform_arys=X)

# PCA with HappyML's Class
from HappyML.preprocessor import PCASelector

selector = PCASelector(best_k="auto")
X = selector.fit(x_ary=X, verbose=True, plot=True).transform(X)
```

不要漏掉「特徵縮放」

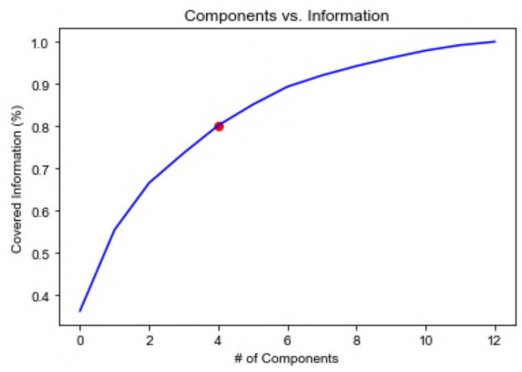
2

3

開 “auto” 讓它自行選擇

- 執行結果：

```
Select 4 components, covered information 73.60%
```




```
----- Random Forest Classification -----
10 Folds Mean Accuracy: 0.9499656002751978
10 Folds Mean Recall: 0.9528571428571428
10 Folds Mean Precision: 0.9585978835978837
10 Folds Mean F1-Score: 0.952495130142189
```

隨堂練習：使用「快樂版函式庫」實作

- 請先註解掉「用 Python 製作 PCA」與「繪製決定邊界」程式碼：

```python
# # PCA without HappyML's Class
# from sklearn.decomposition import PCA
# import numpy as np
# import matplotlib.pyplot as plt
# import pandas as pd

# pca = PCA(n_components=None)
# pca.fit(X)
# info_covered = pca.explained_variance_ratio_
# cumulated_sum = np.cumsum(info_covered)
# plt.plot(cumulated_sum, color="blue")

# pca = PCA(n_components=2)
# X_columns = ["PCA_{}".format(i+1) for i in range(2)]
# X = pd.DataFrame(pca.fit_transform(X), index=X.index, columns=X_columns)
```

```python
# import HappyML.model_drawer as md

# md.classify_result(x=X_train, y=Y_train, classifier=classifier.classifier,
#                    fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
#                    title="訓練集 vs. 隨機森林模型", font="DFKai-sb")
# md.classify_result(x=X_test, y=Y_test, classifier=classifier.classifier,
#                    fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
#                    title="測試集 vs. 隨機森林模型", font="DFKai-sb")
```

- 撰寫下列程式碼，並重跑整個程式：

```python
# Feature Scaling
X = pp.feature_scaling(fit_ary=X, transform_arys=X)

# PCA with HappyML's Class
from HappyML.preprocessor import PCASelector

selector = PCASelector(best_k="auto")
X = selector.fit(x_ary=X, verbose=True, plot=True).transform(X)
```

課後作業：動物園分類

- 資料集說明
  - 請下載資料集 Zoo_Data.csv、Zoo_Class_Name.csv。
  - Zoo_Data.csv 包含動物如何被分類的資料
    - **自變數**：hair（是否有毛髮）、feathers（是否有羽毛）、eggs（是否卵生）...
    - **應變數**：class_type（類別）= 1, 2, 3, 4, 5, 6, 7。
  - Zoo_Class_Name.csv 包含「類別」vs.「類別名稱」的對應
    - 1=Mammal、2=Bird...。

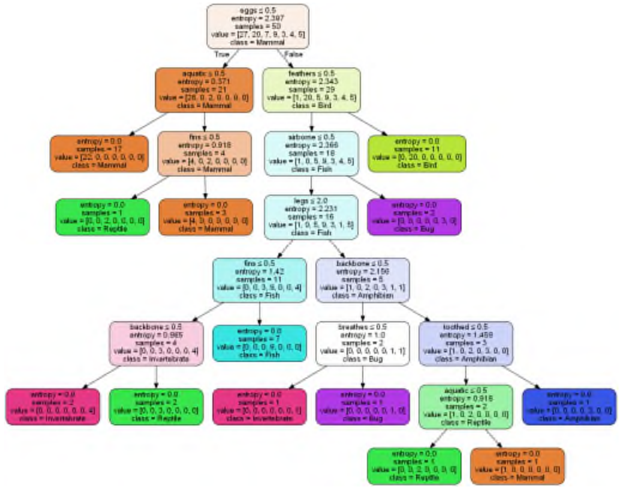
- 要求
  - 請用「**隨機森林**」製作一款分類器，能根據生物特徵分類動物園裡的動物。
  - 請自行選用「**PCA**」或「**KBest**」作為您的特徵選擇演算法。留下表現好的那一個。
  - 請印出該分類器的「**確度、廣度、精度、F 值**」，來說明您的模型有多好。
  - 請**繪製**出隨機森林其中一棵樹（使用 HappyML.model_drawer.**tree_drawer**()）。注意！決策樹「樹葉」要能夠清楚印出「class=Mammal」（哺乳類），而不是僅印出「class=1」。

課後作業：動物園分類

- 提示
  - 要能夠在決策樹的「樹葉」，印出「class=Mammal」，重點在於 tree_drawer() 的 target_names= 是否傳入正確的參數。
  - 如果希望 1=Mammal, 2=Bird, 3=Reptile...，則 target_names=["Mammal", "Bird", "Reptile", ...]。
  - 類別名稱可以用下列手法讀入：
    - dataset_classname = pp.dataset("Zoo_Class_Name.csv")
    - class_names = [row['Class_Type'] for index, row in dataset_classname.iterrows()]

- 輸出

```
----- Random Forest Classification -----
10 Folds Mean Accuracy: 0.9614646464646464
10 Folds Mean Recall: 0.9095238095238095
10 Folds Mean Precision: 0.9
10 Folds Mean F1-Score: 0.9038095238095238
```



本章總結

- **隨機森林的原理**
  - 建立多棵「**決策樹**」，**投票**決定答案。

- **隨機森林的優點**
  - 能處理**大量、無相關性**的自變數
  - 對「**缺失資料**」（Missing Data）耐受度良好
  - 能產生較為「**不偏（Un-bias）」的訓練集**
  - 對於「**離群值**」（Outliers）抗性較佳

- **相關套件**
  - 隨機森林：sklearn.ensemble.**RandomForestClassifier**
  - PCA：sklearn.decomposition.**PCA**