全民瘋AI系列 [經典機器學習]

第12屆iT邦幫忙鐵人賽 影片教學組
第13屆iT邦幫忙鐵人賽 AI & Data 組

Tsai Yi Lin
10程式中

Table of contents

第一部分 AI基礎概念

[Day 1] 目標介紹

[Day 2] 快來探索AI的世界

[Day 3] 你真了解資料嗎?試試看視覺化分析吧!

[Day 4] 咱們一起做資料清理和前處理

[Day 5] 機器學習大補帖

第二部分 機器學習入門

[Day 6] 非監督式學習 K-means 分群

[Day 7] 非監督式學習-降維

[Day 16] 每個模型我全都要 - 堆疊法 (Stacking)

[Day 17] 輕量化的梯度提升機 - LightGBM

[Day 18] 機器學習 boosting 神器 - CatBoost

第三部分 進階概念與應用

[Day 19] 自動化機器學習 - AutoML

[Day 20] 機器學習金手指 - Auto-sklearn

[Day 25] 交叉驗證 Cross-Validation 簡介

[Day 26] 交叉驗證 K-Fold Cross-Validation

[Day 27] 機器學習常犯錯的十件事

第一部分 AI基礎概念

[Day 1] 目標介紹

第13屆iT邦幫忙鐵人賽

前言

哈囉大家好我是10程式中的10！我是上一屆 *(https://ithelp.ithome.com.tw/users/20107247/ironman/3719)*鐵人賽影片教學組全民瘋AI系列的作者，當時講解了人工智慧的基礎以及常見的機器學習演算法與手把手教學。由於大家反應很熱烈，讓我看到了大家對於AI的學習熱忱。也因為上一屆獲得了影片教學組優選，收到了許多書商的出版邀請，由於我沒有時間與動力將這些大量知識寫成文章因此都婉拒了。因此我想藉由這一次鐵人賽將上一屆的影片內容整理成電子書版本，提供大家影片教學與文字版的筆記內容(唷呼書商快看過來～)當然內容會以之前影片教學為基底，並加入一些新的元素讓文章內容變得更紮實。在全新的全民瘋AI系列2.0中我會介紹實用的機器學習演算法並含有程式手把手實作，以及近年來熱門的機器學習套件與模型調參技巧。除此之外我還會提到大家最感興趣的 AI 模型落地與整合。希望在這次的鐵人賽能夠將AI的資源整理得更詳細並分享給各位。

此系列教學適合誰?

如果您是之前的舊讀者，歡迎回來為自己充電～新的系列文章保證讓你收穫滿滿！若您是新來的讀者歡迎加入人工智慧的世界，此系列文章正適合初學者閱讀。另外建議可以搭配我上一屆 *(https://ithelp.ithome.com.tw/users/20107247/ironman/3719)*鐵人賽的影片教學進行學習。

系列文章內容規劃

在本次鐵人賽預計新增了許多新內容，特別是近年來比較新的演算法套件，以及在模型訓練中必須注意的大小事。本系列要在短短30天內講完所有 AI 領域相關應用是不太可能的事情，因此我的規劃是從認識人工智慧開始切入主題。先讓大家知道何謂人工智慧以及相關應用有哪些。接著帶各位了解成為資料科學家的第一步，就是資料分析與視覺化，再來會有一系列經典的機器學習演算法介紹。最後也是大家可能會有興趣的整合部分，會以實際的帶大家手把手部署我們的AI模型以及前後端串接的概念。

前置作業資源

本系列教學將有大量的程式實作，並採用 Google Colab 做為程式雲端運行的編輯執行環境。各位可以直接利用 Colab 開啟本系列文章的範例程式。在使用此平台之前每個人都必須要有自己的 Google 帳號，才能順利的開啟並執行程式碼。Colab 可讓你輕鬆地在瀏覽器上撰寫並執行 Python 程式語言，它可以說是機器學習新手的入門工具。此外 Colab 具備了以下幾個優點：

- 不必進行任何設定與安裝
- 免費額度使用 GPU、TPU 資源
- 輕鬆共用與分享檔案

因此讀者必須先熟悉 Colab 的操作模式，想了解該如何操作的朋友們可以先來看這一步影片 *(https://youtu.be/C9mvGMtrPXo?t=266)*教學。

回報錯誤與建議

本系列文章若有問題或是內容建議都可以來 GitHub 中的 issue *(https://github.com/andy6804tw/2021-13th-ironman/issues)* 提出。歡迎大家一同貢獻為這系列文章有更好的閱讀品質。

關於作者

曾任職於台灣人工智慧學校，擔任AI工程師，擁有豐富的教學經驗，熱衷於網頁前後端整合與AI演算法的開發。希望藉由鐵人賽，將所學貢獻出來，為AI領域提供更多資源。

@andy6804tw *(https://github.com/andy6804tw)*

歡迎大家訂閱我的 YouTube *(https://www.youtube.com/channel/UCSNPCGvMYEV-yIXAVt3FA5A)* 頻道。

本系列教學簡報 PDF & Code 都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 2] 快來探索AI的世界

今日學習目標

- 人工智慧的演進
- 人工智慧的分級
- 機器是如何學習的

人工智慧的演進

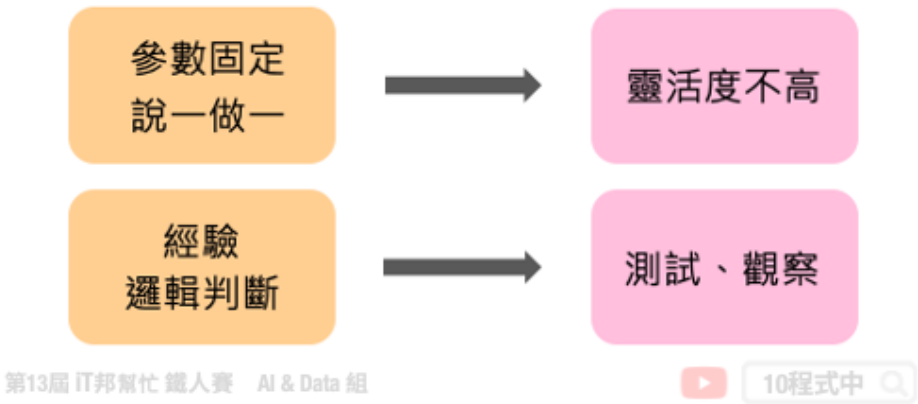
AI 與機器學習技術正在蓬勃發展中，你能想像人工智慧曾被認為是一個毫無出路的領域嗎？ 從人工智慧的時間軸來看可以分為三個熱潮。第一次熱潮（1950~1960年），由於早期的電腦硬體資源的不足導致複雜的問題無法輕易的解決。第二次熱潮（1980~1990年）將帶有知是本體的代理人放入機器人中使具有智慧，也就是所謂的專家系統。但人類資源有限不可能把所有的知識都逐一地輸入到電腦。因此大家開始思考機器是否能夠讓他自己去學習？而不是人類一昧的餵入這些知識。第三次熱潮（2000年~現在）由於 CPU、GPU 以及雲端運算資源普及，早期複雜難解的演算法陸續可以透過超級電腦來解決。當手邊有了大量的數據就能拿來機器學習，因此大家踏入了大數據以及深度學習的時代。時間不斷的往前走，你能想像未來的 AI 在世界上是扮演什麼樣的角色嗎？

人工智慧的分級

現今人工智慧與我們生活無所不在，例如我們只要對著手機喊一聲「Hey Siri !」蘋果手機的語音助理就能幫你打理好大小事。或者正在超市購物的你正在為購買哪一項商品煩惱時，推薦系統機器人能夠即時地為你做商品推薦。看似著簡單的動作，但人工智慧的情景在你我日常生活中息息相關。人工智慧依照機器能夠處理與判斷的能力區分為四個分級，分別為自動控制、探索推論、機器學習、深度學習：

第一級人工智慧：自動控制

機器含有自動控制的功能，並且經由感測器偵測環境的資訊。例如透過溫度感測器來偵測產線的馬達是否過熱，並達到停止運轉效果。或是冷氣低於20度時就進入待機模式……等。因此程式設計師必須先把所有可能的情況都考慮進去才能寫出控制程式。這就衍伸出一些問題，像是靈活度不高，且需要有經驗的專家介入才能完成。



第二級人工智慧：探索推論

第二級逐漸開始強調邏輯推理，可以說是補足第一級的問題。透過將知識組織成知識本體並讓機器從現有的資訊中去推理。典型的例子就是專家系統，它是透過特定領域的專家訂定出一套知識庫與規則庫，並產生大量輸入與輸出資料的排列組合來解決日常生活中的問題。當然所謂的專家系統就必須邀請領域的專家為系統量身打造一套獨一無二的規則。然而每個人的觀點可能都不同，因此不同專家間所制定的規則可能都不太一樣。

第三級人工智慧：機器學習

機器可以根據資料學習如何將輸入與輸出資料產生關聯。機器學習是一種學習的演算法，並從資料中去學習並找出問題的解決方法。其應用包括搜尋引擎、大數據分析等。我們依據資料與學習方式可大致分為監督式學習、非監督式學習、增強式學習，此外自監督學習這個名詞最近也熱烈的討論中。

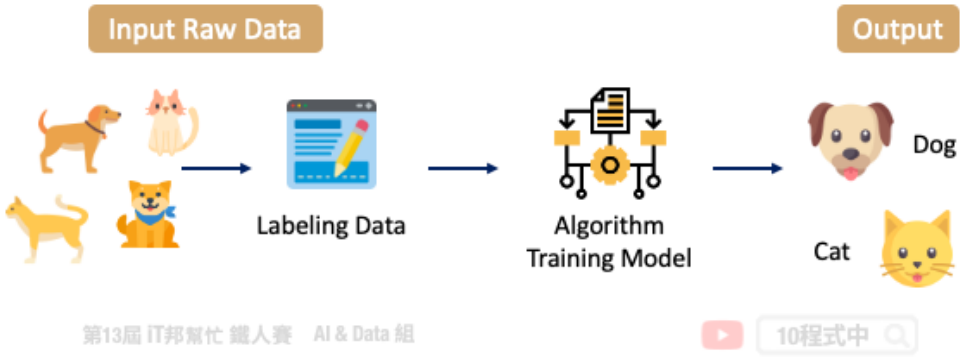
第四級人工智慧：深度學習

深度學習是一種機器學習的方法。它藉由模仿人類大腦神經元的結構，定義解決問題的函式。所謂深度學習是一種具有深度多層的神經網路。機器可以自行學習並且理解機器學習時用以表示資料的「特徵」，因此又稱為「特徵表達學習」，其應用包括：影像分類、機器翻譯...等。

機器如何學習？

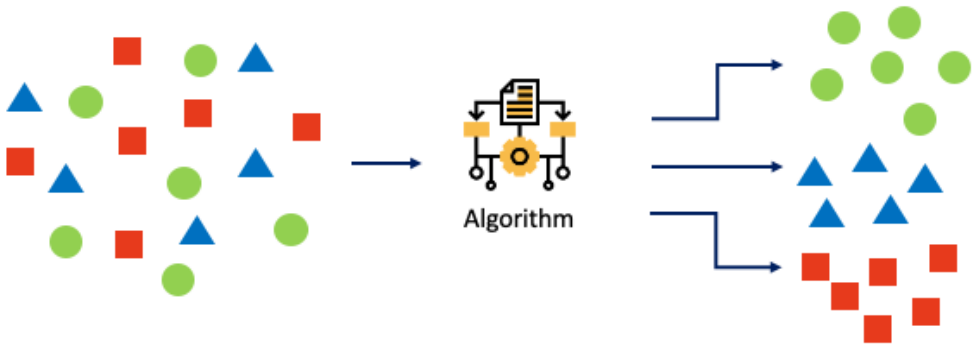
監督式學習 (Supervised Learning)

所謂的監督式學習是給許多資料並給與答案，透過損失函數計算來找出一個最佳解。舉一個簡單的例子，比如給機器各看了 1000 張貓和狗的照片後再詢問機器新的一張照片中是貓還是狗。一直不斷的迭代訓練並從錯誤中去學習，最終機器能成功的分類了。



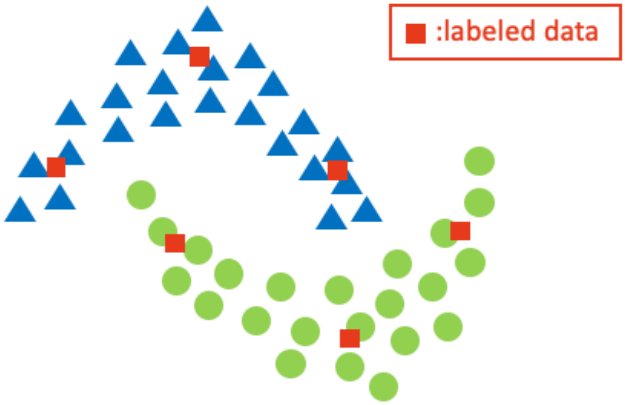
非監督式學習 (Unsupervised Learning)

非監督式學習只給定特徵，機器會想辦法會從中找出規律。非監督式學習最常見的方法就是集群分析(Cluster Analysis)，目標是根據特徵將資料樣本分為幾群。簡單來說非監督式學習就是給許多資料但不給予答案，模型會從資料中自己去找出關係。透過分群演算法來計算資料與資料間的相似程度與距離。



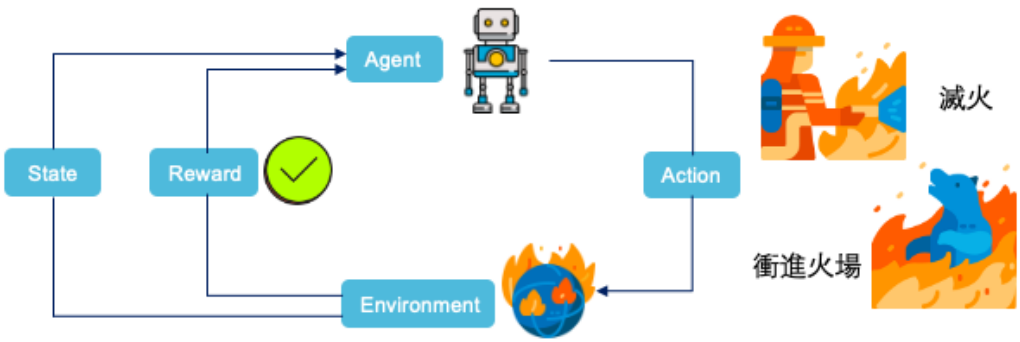
半監督式學習 (Semi-Supervised Learning)

介於監督式學習與非監督式學習之間。在現實生活中，未標記樣本多、有標記樣本少是一個比價普遍現象，如何利用好未標記樣本來提升模型泛化能力，就是半監式督學習研究的重點。半監式督學習的應用主要在於收集資料很簡單，但標記的資料太少了，我們希望可以自動標記資料。



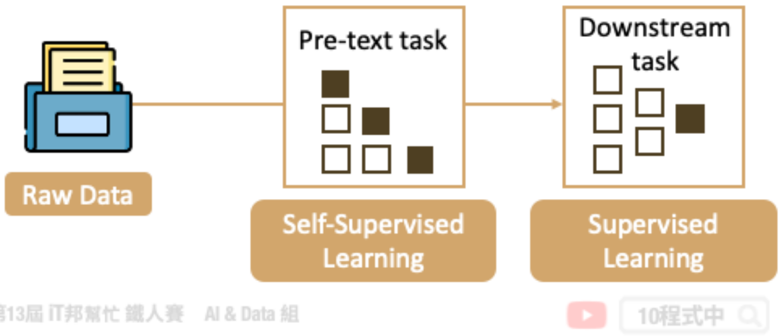
強化式學習 (Reinforcement Learning)

在強化式學習中機器會進行一系列的動作，而每做一個動作、環境都會跟著發生變化。若環境的變化是離目標更接近，我們就會給予一個正向反饋。若離目標更遠，則給予負向反饋。機器透過不斷的從錯誤中去學習，最終學到了如何去解決一件事情。



自監督學習 (Self-Supervised Learning)

自監督學習是由卷積神經之父 Yann LeCun 於 2019 年所提出來的一種學習機制。此學習機制模仿模仿人類的學習行為，透過當前任務觀察所得到的特徵，並訓練一個目標任務的模型。而且學習過程中並不仰賴人類給定的標籤。簡單來說訓練過程是拿一個訓練好的模型透過非監督式技巧 pre-text task 訓練好模型，訓練完成後再接到下游任務做最後的模型微調 (fine tune)。



學 AI 該用哪種程式語言？

Python 是近年來高速成長並且逐漸普及的程式語言，也可以說是最容易上手的程式語言之一。主要在於它的語法是簡化而不複雜的，同時強調程式碼的可讀性因此更能貼近程式設計者的思維。當然也些人使用 R 語言進行統計分析、繪圖以及資料探勘甚至建模。如果你正在猶豫要入坑哪一類程式語言，筆者這裡推薦 Python 程式語言。因為 Python 不僅可以進行資料分析、機器學習也能進行前/後端開發。另外 Python 有豐富的討論社群以及許多開源套件支援，大幅的降低學習門檻。

說了這麼多！大家準備好了嗎？快準備好電腦與筆記本，好好的為自己進行三十天的充電吧～ Let's Go!

本系列教學內容都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 3] 你真了解資料嗎?試試看視覺化分析吧!

今日學習目標

- 探索式分析 (EDA)
  - 聊聊何謂 EDA，為何要做數據分析?
- 撰寫第一支 EDA 程式
  - 透過鳶尾花 (iris) 資料集，來查看資料的分佈狀態

範例程式： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/3.你真了解資料嗎試試看視覺化分析吧/3.你真了解資料嗎試試看視覺化分析吧.ipynb)

探索式分析 (EDA)

探索式資料分析（Exploratory Data Analysis，EDA)，主要概念是利用數據統計的方式視覺化資料。透過資料的探索式分析可以查看資料集當中每個特徵彼此的重要程度以及其資料分布狀況，有良好的數據分析習慣能夠幫助你更了解資料集的特性。另外做 EDA 的好處是可以從各種面向先了解資料的狀況，以利後續的模型分析。

EDA 必要的套件

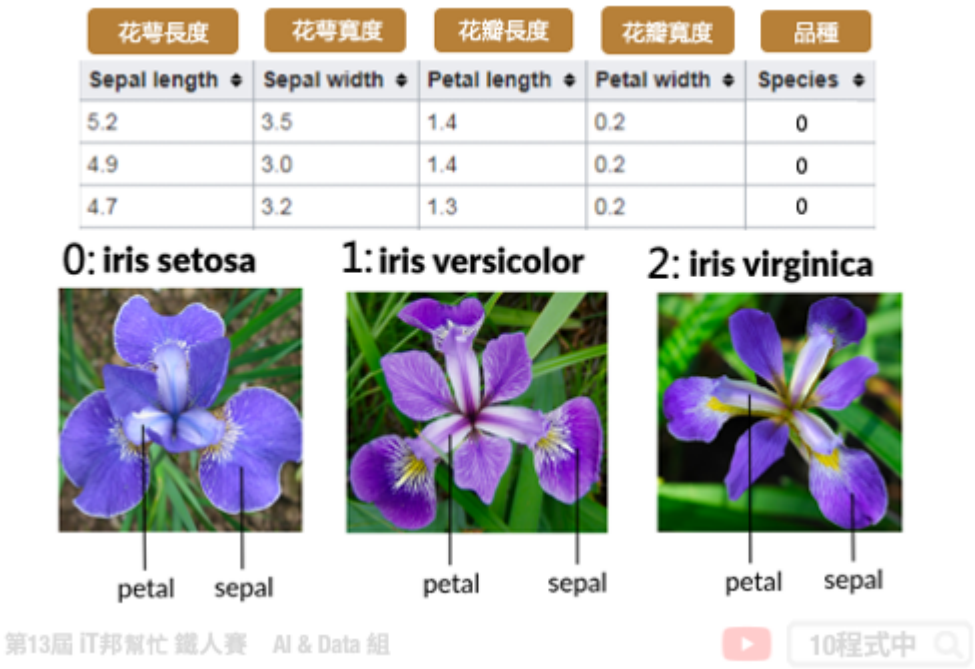
- 資料處理 – Pandas, Numpy
  - Pandas *(https://pandas.pydata.org/)*：Python 表格資料處理的重要工具
  - Numpy *(https://numpy.org/)*：針對多維陣列的平行運算進行優化的強大函式庫
- 繪圖相關 – Matplotlib, Seaborn
  - Matplotlib *(https://matplotlib.org/)*：Python 最常被使用到的繪圖套件
  - Seaborn *(https://seaborn.pydata.org/)*：以 matplotlib 為底層的高階繪圖套件

試試看第一支EDA程式

Example : 鳶尾花朵

鳶尾花朵資料集一覽

此資料集總共有4個輸入特徵。分別為花萼長度、花萼寬度、花瓣長度與花瓣寬度。輸出特徵為花朵的品種， 共有三種類別分別為 0: iris setosa、 1: iris versicolor、 2: iris virginica。



載入必要套件

首先我們載入資料探索式分析所需的套件。分別有進行數據處理的函式庫的 `pandas`、高階大量的維度陣列與矩陣運算的 `numpy`、處理資料視覺化的繪圖庫 `matplotlib` 與 `seaborn`。最後一個是資料集來源，此系列範例我們採用 Sklearn 所提供的鳶尾花分類的資料集。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
```

Sklearn Toy datasets

Sklearn 套件中提供了七個快速入門的 Toy datasets *(https://scikit-learn.org/stable/datasets/toy_dataset.html)* 很推薦初學者可以載入來玩玩看，並且練習做資料探索與建模。每一個資料集呼叫的方法非常簡單。以鳶尾花朵資料集為例，我們可以透過 API 取得輸入與輸出。

```
from sklearn.datasets import load_iris

iris = load_iris()
# 輸入特徵
X = iris.data
# 輸出特徵
y = iris.target
```

Sklearn 提供了許多 API 方法可以呼叫：

- data: 取得輸入特徵
- target: 取得輸出特徵
- feature_names: 取得輸入特徵的名稱
- target_names: 取得輸出的類別標籤(分類資料集)
- DESCR: 資料集詳細描述

如果想試試其他的資料集可以參考：

- 迴歸問題
  - load_boston *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston)* 波士頓房價預測
  - load_diabetes *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)* 糖尿病預測
  - load_linnerud *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_linnerud.html#sklearn.datasets.load_linnerud)* 體能評估預測
- 分類問題
  - load_iris *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris)* 鳶尾花種類預測
  - load_digits *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits)* 手寫數字辨識
  - load_wine *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine)* 葡萄酒種類預測
  - load_breast_cancer *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer)* 乳癌預測

參考 (https://zhuanlan.zhihu.com/p/95412564)

載入資料集

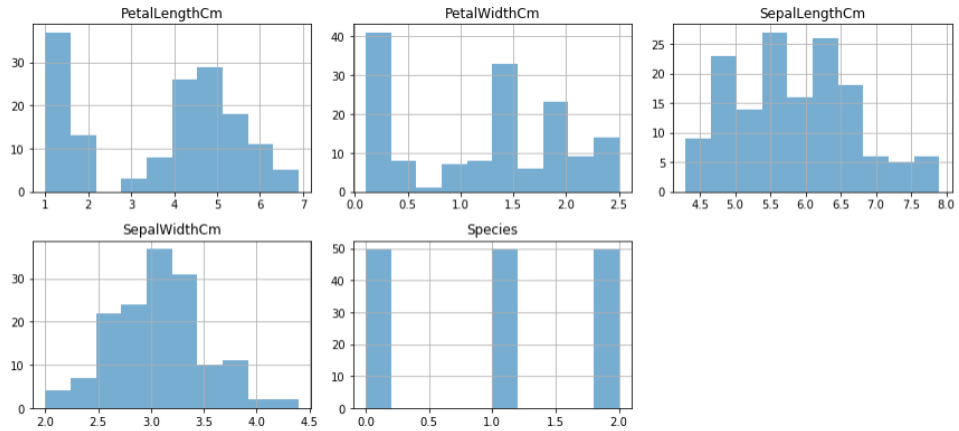
首先我們載入鳶尾花朵資料集。為了方便分析我們將 numpy 格式的資料轉換成 DataFrame 的格式進行資料探索。因為透過 Pandas 的 DataFrame 格式我們更能用表格的形式觀察資料。

```
iris = load_iris()
df_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= ['SepalLengthCm','SepalWidthCm','PetalL
df_data
```

直方圖

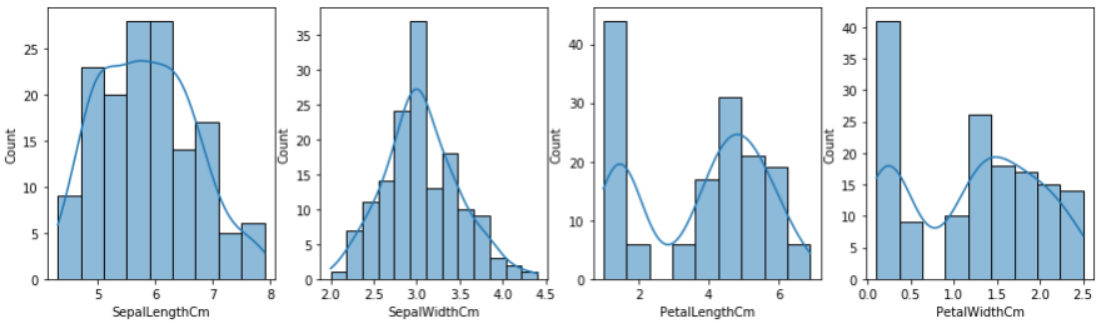
直方圖是一種對數據分布情況的圖形表示，是一種二維統計圖表。我們可以直接呼叫 Pandas 內建函式 `hist()` 進行直方圖分析。其中我們可以設定 bins(箱數)，預設值為 10。如果設定的輸量越大，其代表需要分割的精度越細。通常取一個適當的箱數即可觀察該特徵在資料集中的分佈情況。藉由直方圖我們可以知道每個值域的分佈大小與數量。我們也能發現輸出項的類別共有三個，並且這三個類別的數量都剛好各有 50 筆資料。我們也能得知這一份資料集的輸出類別是一個非常均勻的資料。

```
#直方圖 histograms
df_data.hist(alpha=0.6,layout=(3,3), figsize=(12, 8), bins=10)
plt.tight_layout()
plt.show()
```



我們也可以透過 Seaborn 的 `histplot` 做出更詳細的直方圖分析。並利用和密度估計 `kde=True` 來查看每個特徵的分佈狀況。

```
fig, axes = plt.subplots(nrows=1,ncols=4)
fig.set_size_inches(15, 4)
sns.histplot(df_data["SepalLengthCm"][:],ax=axes[0], kde=True)
sns.histplot(df_data["SepalWidthCm"][:],ax=axes[1], kde=True)
sns.histplot(df_data["PetalLengthCm"][:],ax=axes[2], kde=True)
sns.histplot(df_data["PetalWidthCm"][:],ax=axes[3], kde=True)
```

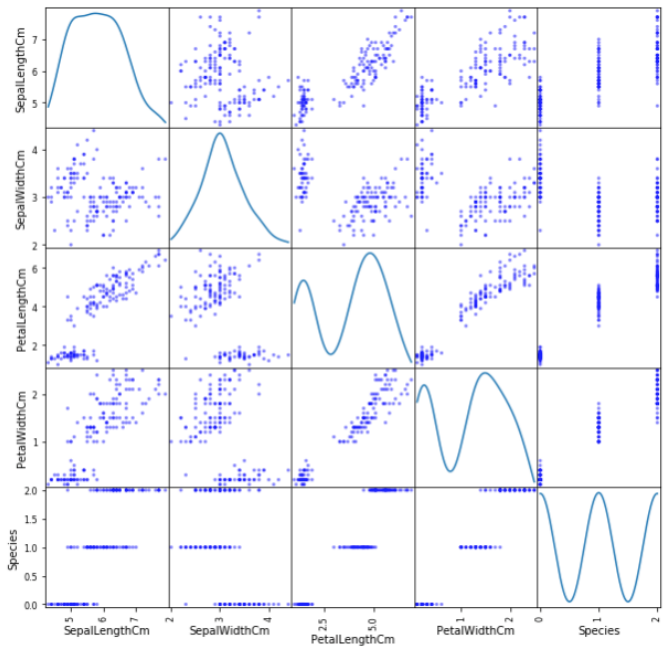


核密度估計

核密度估計分爲兩部分，分別有對角線部分和非對角線部分。在對角線部分是以核密度估計圖（Kernel Density Estimation）的方式呈現，也就是用來看某一個特徵的分佈情況，x軸對應著該特徵的數值，y軸對應著該特徵的密度也就是特徵出現的頻率。在非對角線的部分為兩個特徵之間分佈的關聯散點圖。將任意兩個特徵進行配對，以其中一個爲橫座標，另一個爲縱座標，將所有的數據點繪製在圖上，用來衡量兩個變量的關聯程度。

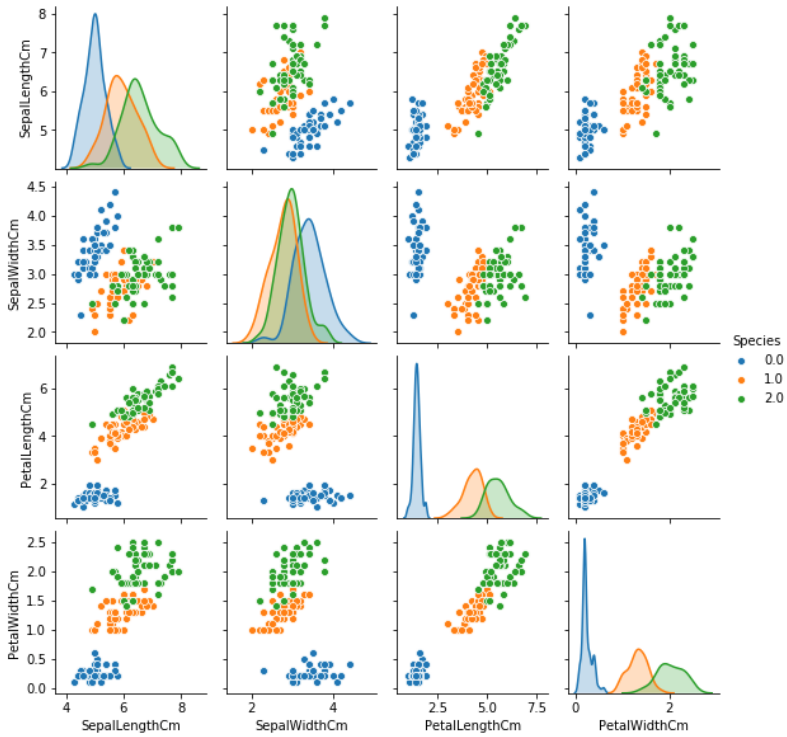
使用 Pandas 繪製：

```
from pandas.plotting import scatter_matrix
scatter_matrix( df_data,figsize=(10, 10),color='b',diagonal='kde')
```



使用 Seaborn 繪製：

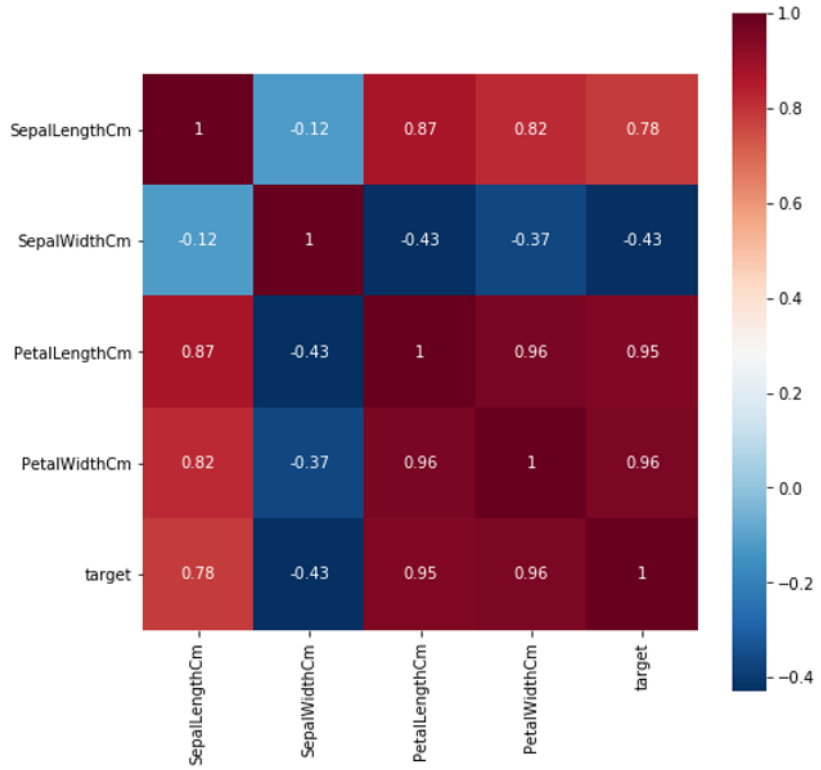
```
sns.pairplot(df_data, hue="Species", height=2, diag_kind="kde")
```



關聯分析

透過 pandas 的 `corr()` 函式可以快速的計算每個特徵間的彼此關聯程度。其區間值為-1~1之間，數字越大代表關聯程度正相關越高。相反的當負的程度很高我們可以解釋這兩個特徵之間是有很高的負 關聯性。

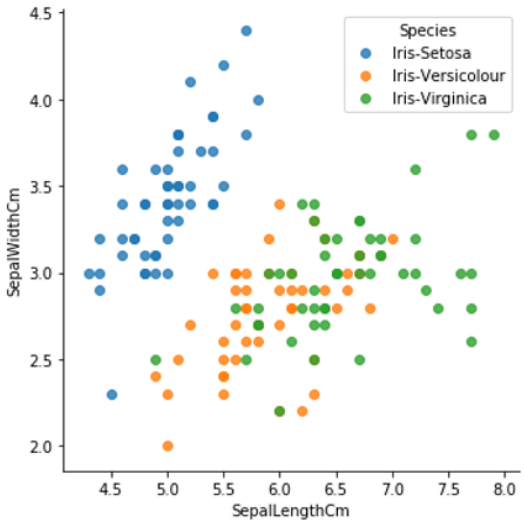
```
# correlation 計算
corr = df_data[['SepalLengthCm','SepalWidthCm','PetalLengthCm','Petal
plt.figure(figsize=(8,8))
sns.heatmap(corr, square=True, annot=True, cmap="RdBu_r")
```



散佈圖

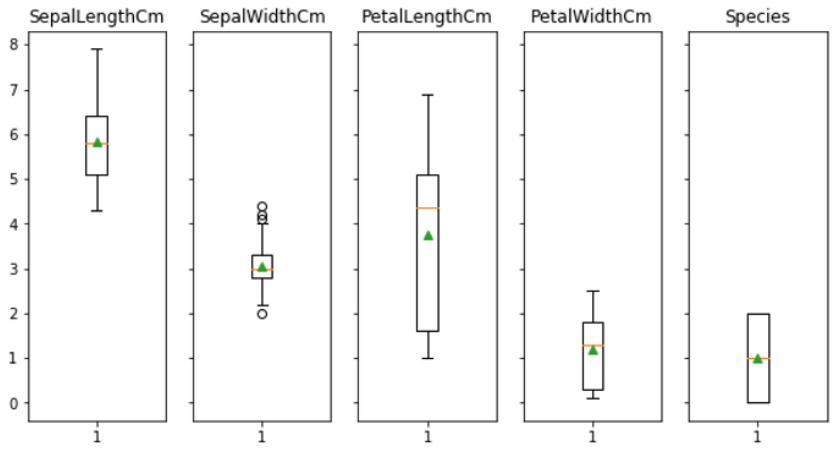
透過散佈圖我們可以從二維的平面上觀察兩兩特徵間彼此的分佈狀況。如果該特徵重要程度越高，群聚的效果會更加顯著。

```
sns.lmplot("SepalLengthCm", "SepalWidthCm", hue='Species', data=df_da
plt.legend(title='Species', loc='upper right', labels=['Iris-Setosa',
```



箱形圖

透過箱形圖可以分析每個特徵的分布狀況以及是否有離群值。我們利用箱形圖來表示四分位數來觀察數據分散情況。箱形的兩端為第一個四分位數涵蓋25%之資料(Q1)與第三個四分位數涵蓋75%之資料(Q3)，而箱形圖的中間線為中位數顯示涵蓋前50%資料之位置。箱形上虛線的端點為極大值，箱型下虛線的點為極小值。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 4] 咱們一起做資料清理和前處理

今日學習目標

- 資料如何清理
    - 什麼是資料清理？
- 資料前處理的方式
    - 為什麼資料要前處理呢？前處裡有何好處？
- 學習 Sklearn 中四種不同資料前處理方式
    - StandardScaler (平均值和標準差)
    - MinMaxScaler(最小最大值標準化)
    - MaxAbsScaler（絕對值最大標準化）
    - RobustScaler

範例程式： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/4.咱們一起做資料清理和前處理/4.咱們一起做資料清理和前處理.ipynb)

前言

很多演算法對數據範圍非常的敏感。因此為了要讓模型訓練的更強大，通常的做法是對特徵進行調節，使得數據更適合這些演算法。一般來說，我們在做機器學習時往往會做特徵的正規化。

載入相關套件

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

np.set_printoptions(suppress=True)
```

1) 載入資料集

今天的範例我們延續昨天的例子，鳶尾花朵資料集進行資料正規化的示範。

```
iris = load_iris()
df_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= ['SepalLengthCm','SepalWidthCm','PetalL
df_data
```

2) 檢查缺失值

使用 numpy 所提供的函式來檢查是否有 NA 缺失值，假設有缺失值使用 `dropna()` 來移除。使用的時機在於當只有少量的缺失值適用，若遇到有大量缺失值的情況，或是本身的資料量就很少的情況下建議可以透過機器學習的方法補值來預測缺失值。

```
X = df_data.drop(labels=['Species'],axis=1).values # 移除Species並取得剩
y = df_data['Species']
# checked missing data
print("checked missing data(NAN mount):",len(np.where(np.isnan(X))[0]
```

輸出結果：

```
checked missing data(NAN mount): 0
```

由於 Sklearn 所提供的資料集非常乾淨，若你收集到的資料有許多的缺失值或是本身資料量就不多的強況下，建議好好的去處理這些缺漏的值。通常補值的方法可分為手動填值與插值法。首先手動填值可以以該欄位所有資料的算術平均數或中位數做填補的依據。再者使用以出現頻率最高的值做填補也是常見的補值方式。另一種差值法是透過時間或空間上的技巧處理這些缺值，例如當資料是有時間序列的因素存在時，可以利用該筆缺失欄位附近的時間點的資料加總並平均。

3) 切割訓練集與測試集

我們透過 Sklearn 所提供的 `train_test_split()` 方法來為我們的資料進行訓練集與測試集的切割。在此方法中我們可以設定一些參數來讓我們切割的資料更多樣性。其中 `test_size` 參數就是設定測試集的比例，範例中我們設定 0.3 即代表訓練集與測試集的比例為 7:3。另外預設資料切割的方式是隨機切割 `shuffle=True` 對原始數據進行隨機抽樣，以保證隨機性。若想要每次程式執行時切割結果都是一樣的可以設定亂數隨機種子 `random_state` 並給予一個隨機數值。最後一個是 `stratify` 分層隨機抽樣，特別是在原始數據中樣本標籤分佈不均衡時非常有用。使用時機是確保分類問題 `y` 的類別數量分佈要與原資料集一致。以免資料集切割不平均導致模型訓練時有很大的偏差。

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

print('train shape:', X_train.shape)
print('test shape:', X_test.shape)
```

輸出結果：

```
train shape: (105, 4)
test shape: (45, 4)
```

Standardization平均&變異數標準化

將所有特徵標準化，也就是高斯分佈。使得數據的平均值為 0，方差為 1。適合的使用時機於當有些特徵的方差過大時，使用標準化能夠有效地讓模型快速收斂。

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
# scaled之後的資料零均值，單位方差
print('資料集 X 的平均值 : ', X_train.mean(axis=0))
print('資料集 X 的標準差 : ', X_train.std(axis=0))

print('\nStandardScaler 縮放過後訓練集的平均值 : ', X_train_scaled.mean(ax
print('StandardScaler 縮放過後訓練集的標準差 : ', X_train_scaled.std(axis=
```

輸出結果：

```
資料集 X 的平均值 :  [5.87333333 3.0552381  3.7847619  1.20571429]
資料集 X 的標準差 :  [0.85882164 0.45502087 1.77553646 0.77383751]

StandardScaler 縮放過後訓練集的平均值 :  [ 0. -0. -0. -0.]
StandardScaler 縮放過後訓練集 X 的標準差 :  [1. 1. 1. 1.]
```

訓練集的 Scaler 擬合完成後，我們就能做相同的轉換在測試集上。

```
X_test_scaled = scaler.transform(X_test)

print('\nStandardScaler 縮放過後測試集的平均值 : ', X_test_scaled.mean(axi
print('StandardScaler 縮放過後測試集的標準差 : ', X_test_scaled.std(axis=0
```

輸出結果：

```
StandardScaler 縮放過後測試集的平均值 :  [0.40925926 0.44259259 0.44750958
StandardScaler 縮放過後測試集的標準差 :  [0.20457725 0.15915694 0.29647499
```

如果想將轉換後的資料還原可以使用 inverse_transform() 將數值還原成原本的輸入。

```
# 將縮放的資料還原
X_test_inverse = scaler.inverse_transform(X_test_scaled)
```

```
[36]: X_test_inverse[:3]

[36]: array([[7.3, 2.9, 6.3, 1.8],
             [6.1, 2.9, 4.7, 1.4],
             [6.3, 2.8, 5.1, 1.5]])

[37]: X_test[:3]

[37]: array([[7.3, 2.9, 6.3, 1.8],
             [6.1, 2.9, 4.7, 1.4],
             [6.3, 2.8, 5.1, 1.5]])
```

MinMaxScaler最小最大值標準化

在 MinMaxScaler 中是給定了一個明確的最大值與最小值。每個特徵中的最小值變成了0, 最大值變成了1。數據會縮放到到[0,1]之間。

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
# scaled 之後的資料最小值、最大值
print('資料集 X 的最小值 : ', X_train.min(axis=0))
print('資料集 X 的最大值 : ', X_train.max(axis=0))

print('\nStandardScaler 縮放過後訓練集的最小值 : ', X_train_scaled.min(axi
print('StandardScaler 縮放過後訓練集的最大值 : ', X_train_scaled.max(axis=
```

輸出結果：

```
資料集 X 的最小值 :  [4.3 2.  1.1 0.1]
資料集 X 的最大值 :  [7.9 4.4 6.9 2.5]

StandardScaler 縮放過後訓練集的最小值 :  [0. 0. 0. 0.]
StandardScaler 縮放過後訓練集的最大值 :  [1. 1. 1. 1.]
```

```
X_test_scaled = scaler.transform(X_test)

print('\nStandardScaler 縮放過後測試集的最小值 : ', X_test_scaled.min(axis
print('StandardScaler 縮放過後測試集的最大值 : ', X_test_scaled.max(axis=0
```

```
StandardScaler 縮放過後測試集的最小值 :  [ 0.02777778  0.125      -0.01724
StandardScaler 縮放過後測試集的最大值 :  [0.83333333 0.83333333 0.89655172
```

MaxAbsScaler絕對值最大標準化

MaxAbsScaler 與 MinMaxScaler 類似，所有數據都會除以該列絕對值後的最大值。 數據會縮放到到[-1,1]之間。

```
from sklearn.preprocessing import MaxAbsScaler

scaler = MaxAbsScaler().fit(X)
X_scaled = scaler.transform(X)
```

```
X_test_scaled = scaler.transform(X_test)
```

RobustScaler

可以有效的縮放帶有 outlier 的數據，透過 Robust 如果數據中含有異常值在縮放中會捨去。

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler().fit(X)
X_scaled = scaler.transform(X)
```

```
X_test_scaled = scaler.transform(X_test)
```

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 5] 機器學習大補帖

今日學習目標

- 了解機器學習是什麼
    - 何謂機器學習?
    - 人工智慧的範疇
    - 什麼是人工智慧?
- 資料科學三劍客
- 機器學習的種類有哪些？
    - 從人類學習到機器學習
    - 認識什麼是資料
- 機器學習的流程

何謂機器學習?

機器學習是一種學習的演算法，是一種從一大群資料中去學習找出解決問題的方法。簡單來說你只要將大量的資料餵給電腦，機器學習的演算法會為你量身打造學習出一個特定的模型給你，而不是再透過人類手動的給予規則。透過一堆資料有標籤給答案，並從資料集學習與標記間的關聯，最後再從非特定資料去辨認答案。

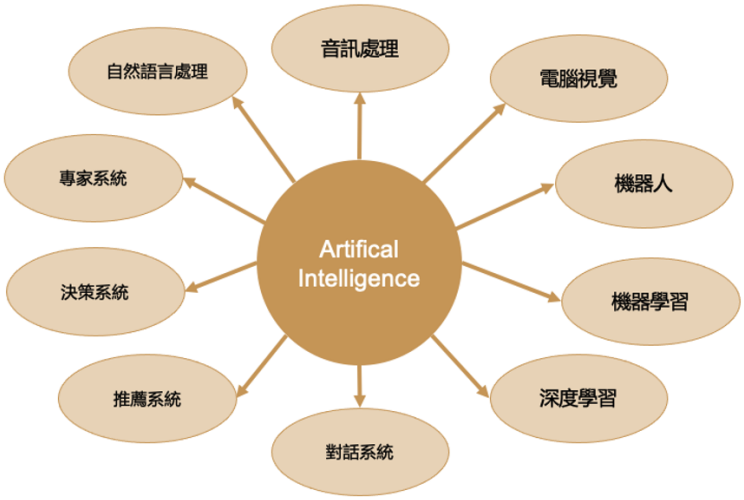
『機器學習是一種學習的演算法，是一種從一大群資料中去學習找出解決問題的方法。』

現今機器學習無所不在

人工智慧的範疇

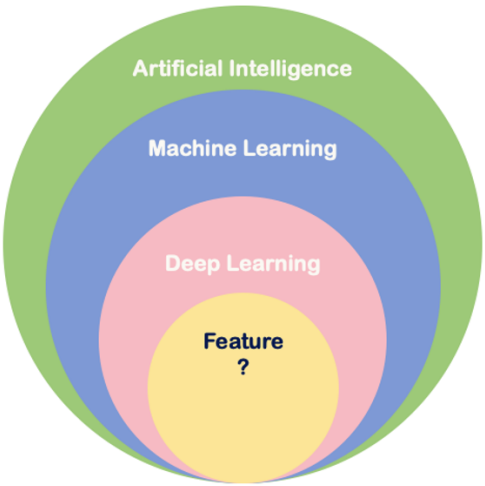
其實人工智慧的應用在現實生活中隨處可見，從製造、醫療、金融、交通、安防、　　零售、物流、農業.....等都可以看到與　AI　的相關應用。當然人工智慧的出現並不是曇花一現，Artificial Intelligence 這一詞其實早在 20 世紀中就被提出，起初當然不被看好甚至大家都覺得要一個機器人學會人類的智慧是天方夜譚的事情，中間也經歷好幾次 AI 寒冬，現在回過頭來看 AI 的研究

領域起伏伏。不過隨著軟硬體的進步，逐漸使得需要大量計算的人工智慧技術慢慢的被挖掘出來。近年來 AI 新創如與春筍般冒出，智慧機器人、感知識別、自然語言處理、對話客服、自動駕駛、瑕疵檢測、預防性維修、自動流程控制、原料組合最佳化……等。



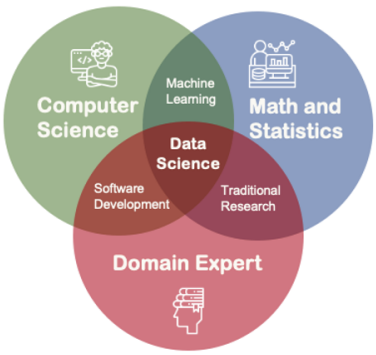
什麼是人工智慧?

其實人工智慧這項領域又分成很多門派，從最早的符號邏輯、專家系統開始說起。早期的 AI 是將人類的專家知識透過知識庫與規則庫放到機器人的大腦中，並賦予機器人智慧使得有能力判斷事物。當然人類專家的知識始終有限，隨著網路與個人電腦普及並進入了大數據時代。各個科學家於是開始思考如何將這些搜集來的大量數據進行應用與分析？機器學習一詞就出現了，目標是透過現實生活中所收集的資料，搭配各種不同機器學習演算法訓練出來一個模型，使得機器人有判斷與預測的能力。當然近幾年熱門的深度學習其實僅是個機器學習裡面的其中一種學習的方法，他是模仿人類的神經系統，透過大量的神經元與多層的神經網路建構出來的複雜數學模型。然而在本系列教學中我們會從最基礎的機器學習演算法開始提起，並一步一步的帶領讀者成為一位真正的資料科學家。



資料科學 × 三劍客

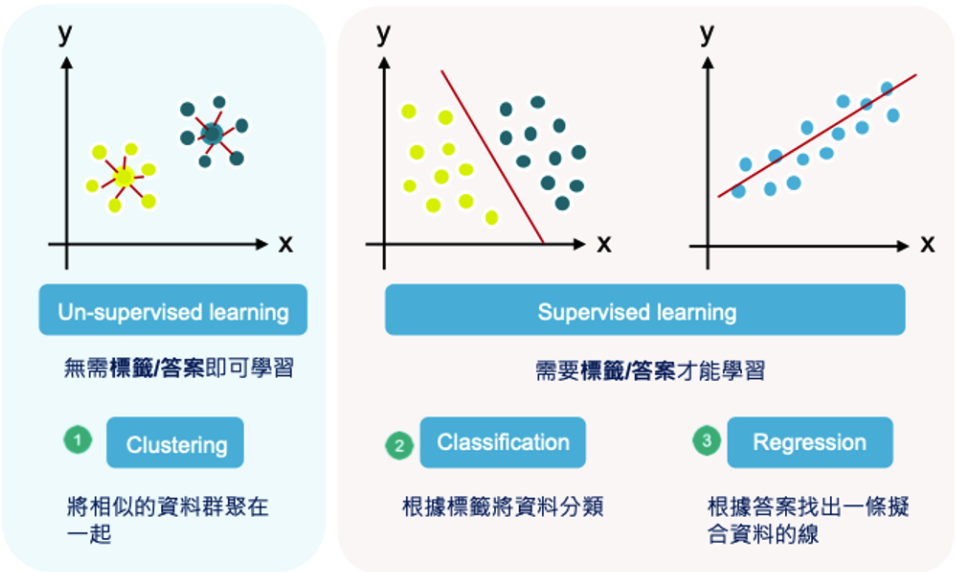
資料科學主要透過機器學習的技術，讓機器可以預測或者推論。其中這幾年很夯的資料科學家這一名詞其實是由三種人所組合起來的。第一個是數學與統計背景的人，他們能夠透過對資料的敏感度從一大群原始資料中探索有意義的資訊。並設計一套適合的模型為這一群資料進行數據擬合。第二種人是電腦科學背景的工程師，他們擅長程式語言能夠將複雜的數學模型夠過程式實作並且協助落地整合。當然現今有非常多機器學習的套件例如 Sklearn、TensorFlow..... 等，降低了大家學習的門檻，不一定是要理工背景的人都可以透過這些機器學習套件一窺人工智慧的奧秘。除此之外 MLOps 是近年來延伸出來的新名詞，其實概念與 DevOps 類似並將這一套機制複製在機器學習專案上，我們平時所執行的 AI 專案必須透過持續性整合與維運的觀念不斷的週期性更新從最新收集到得數據重新學習模型越來越貼近使用者。最後一個關鍵的人物就是各行各業的領域專家，因為 AI 再也不是資訊背景人的專利。我們可以夠過 AI 解決日常生活中的問題，因此我們必須與領域專家進行合作協助資料清與與建立機器學習模型。總之要成為一個好的資科學家上述三種人的特性缺一不可。



機器學習種類

機器學習是一種學習的演算法， 是一種從資料中去學習並找出解決方法。其依照機器學習的種類大致可以分成以下幾類：

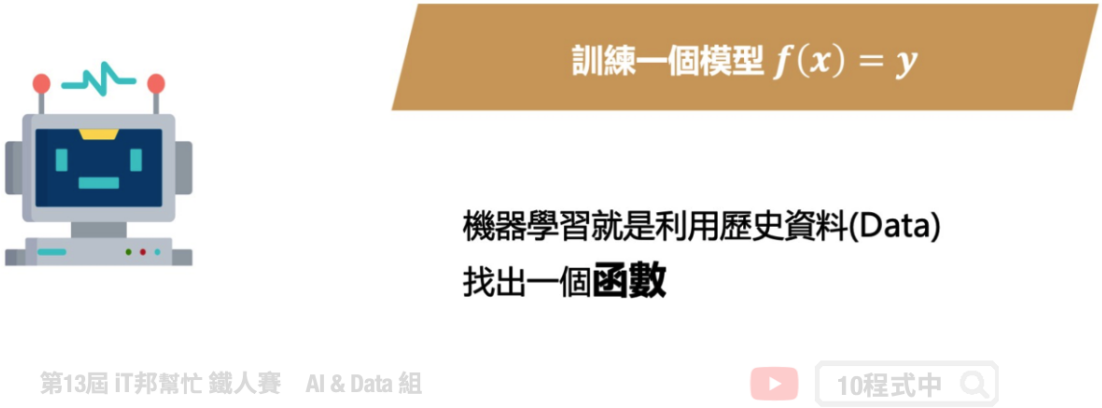
- 非監督式學習
  - 無需標籤/答案即可學習
  - Ex: 集群 (Clustering)
- 監督式學習
  - 需要標籤/答案才能學習
  - Ex: 分類 (Classification) 、 回歸 (Regression)
- 半監督式學習
- 自監督學習
- 強化學習



如何擷取好的特徵是在機器學習中很重要的一件事

從人類學習到機器學習

簡單來說機器學習就是要從一大群資料當中找出一個數學模型。這個數學模型可以稱作是一個 `f(x)=y` 其中 `x` 為輸入的資料， `y` 為該筆資料所相對應的輸出。其中 `f` 即為函數， 也就是任一種機器學習模型。至於典型的機器學習模型有哪些呢？例如線性迴歸、邏輯回歸、KNN、SVM、決策樹、隨機森林、XGBoost.....等。之後的系列文章都會依序向各位解釋。



什麼是資料?

一般來說資料可以分成兩個部分。以一個分類的問題來說，分別有輸入的特徵以及該筆資料相對應的答案稱作標記。AI 這個領域就是讓機器有學習解決問題的能力，而不是我們告訴他應該怎麼解決問題。我們舉一個簡單的例子，假設我們需要預測明天是否會下雨。我們的輸入特徵就可以有各個觀測站的雲量與溫濕度作為模型訓練的資料。而每一筆的天氣資訊都對應著是否會下雨的標準答案。

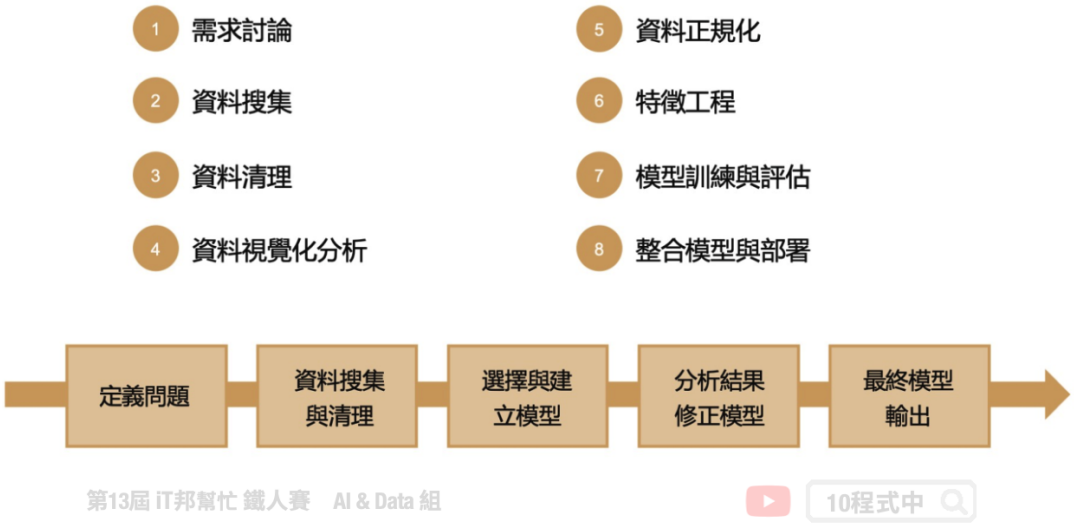
- 特徵 (Feature): 用來描述每一筆資料，通常會用 X 來表示
- 標記 (Label): 用來表示每一筆資料所對應的輸出，這個輸出樣式可以有不同的狀態(可能是類別或者實數值等)，通常會用 Y 來表示。

X	Y
雲量、溫度、濕度	是否下雨

機器學習流程

完整的機器學習流程大致分成八個步驟。首先定義問題，經過需求討論與評估後有個明確的目標並開始執行專案。接著開始搜集資料，由於各場域所收集到的原始數據可能尚未整理以及格式尚未統一。因此第三步的資料清理極為重要，有個乾淨的資料可以對模型表現有大幅的提升。資料一切就緒後建議在建模之前先對資料進行視覺化分析，並為數據做前處理以及專業知

識的特徵工程。對資料有初步的認識後，接著挑選合適的機器學習演算法訓練與評估模型。在模型正式上線之前，先透過測試集或是交叉驗證等機制確認模型泛化能力。模型確認沒有問題後即可將模型打包輸出，並且與實際場域應用進行整合。最終就是部署模型以及維運，持續將場域蒐集到的新資料進行再訓練，形成一個開發循環。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

第二部分 機器學習入門

[Day 6] 非監督式學習 K-means 分群

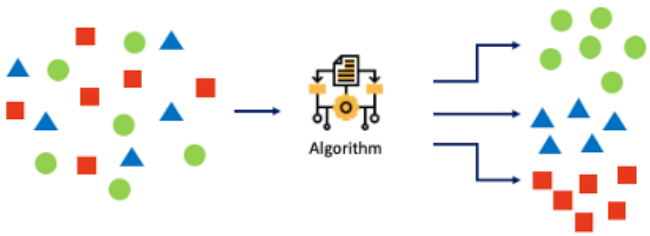
今日學習目標

- 非監督式學習
  - 何謂非監督式學習? 集群分析?
- 分群演算法介紹
  - K-means 分群分類演算法

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/6.非監督式學習k-means分群/6.非監督式學習k-means分群.ipynb)*

非監督式學習(Un-supervised learning)

在訓練過程中沒有所謂的標準答案，故機器會自己從資料群中找出一套分群的法則。非監督式學習的優點是不需要事先以人力標籤，只給定特徵讓機器想辦法會從中找出規律。常見的非監督式的分群演算法有 K-means，它根據物以類聚的原理目標是根據特徵把資料樣本分為 K 群。其中在訓練模型時僅須對機器提供輸入的特徵，並利用分群演算法自動從這些特徵中找出鄰近的集群中心作為該類別。

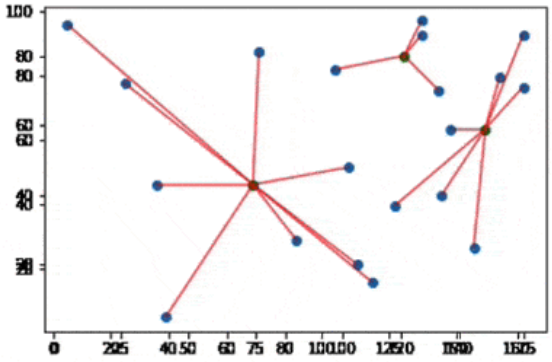


K-means 演算法

透過分群分類演算法我們能夠將多種維度的資料進行分類。K-means 演算法的概念很簡單也非常容易實作，僅一般加減乘除就好不需複雜的計算公式。

1. 初始化: 指定 K 個分群，並隨機挑選 K 個資料點的值當作群組中心值
2. 分配資料點: 將每個資料點設為距離最近的中心
3. 計算平均值: 重新計算每個分群的中心點

重複步驟2、3，直到資料點不再變換群組為止



[程式實作]

載入相關套件

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
```

1) 載入資料集

我們今天要實作分群分類的問題，因此鳶尾花朵資料集非常適合當作範例。其資料集載入方式在第四天有提過，是一樣的內容！

```
iris = load_iris()
df_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= ['SepalLengthCm','SepalWidthCm','PetalL
df_data
```

K-Means

K-means 演算法在 Sklearn 套件中已經幫我們封裝好了，使用者只要呼叫 API 即可將分群分類演算法快速實作。

Parameters: - n_cluster: K的大小，也就是分群的類別數量。 - random_state: 亂數種子，設定常數能夠保證每次分群結果都一樣。 - n_init: 預設為10次隨機初始化，選擇效果最好的一種來作為模型。 - max_iter: 迭代次數，預設為300代。

Attributes: - inertia_: inertia_：float，每個點到其他叢集的質心的距離之和。 - cluster_centers_：特徵的中心點 `[n_clusters, n_features]`。

Methods: - fit: K個集群分類模型訓練。 - predict: 預測並回傳類別。 - fit_predict: 先呼叫fit()做集群分類，之後在呼叫predict()預測最終類別並回傳輸出。 - transform: 回傳的陣列每一行是每一個樣本到kmeans中各個中心點的L2(歐幾里得)距離。 - fit_transform: 先呼叫fit()再執行transform()。

```
from sklearn.cluster import KMeans

kmeansModel = KMeans(n_clusters=3, random_state=46)
clusters_pred = kmeansModel.fit_predict(X)
```

評估分群結果

使用者設定 K 個分群後，該演算法快速的找到 K 個中心點並完成分群分類。擬合好模型後我們可以計算各個樣本到該群的中心點的距離之平方和，用來評估集群的成效，其 `inertia` 越大代表越差。

```
kmeansModel.inertia_
```

輸出結果：

```
78.94084142614602
```

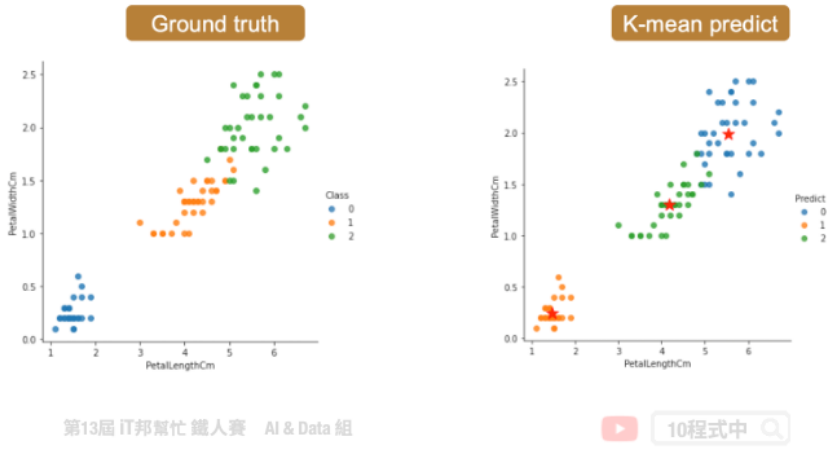
若要查看各群集的中心點，可以參考以下程式碼。

```
kmeansModel.cluster_centers_
```

輸出結果：

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

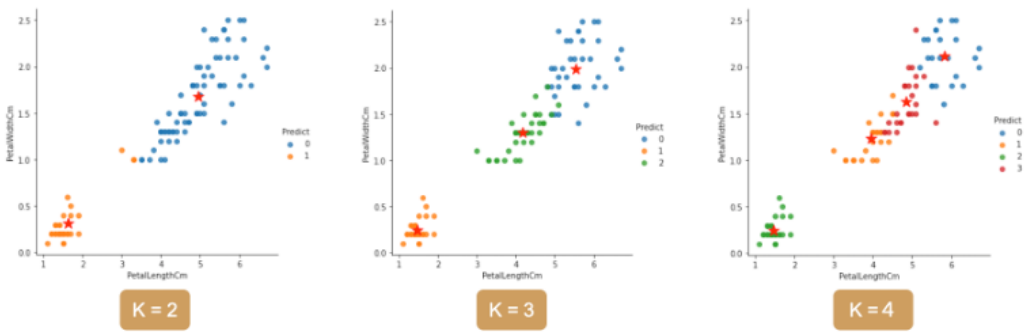
分類結果



如何決定K?

當你手邊有一群資料，且無法一眼看出有多少個中心的狀況。可用使用下面兩種方法做 k-means 模型評估。

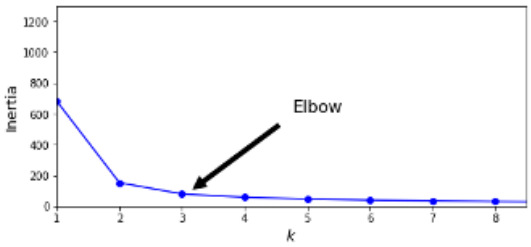
- Inertia 計算所有點到每群集中心距離的平方和。
- silhouette scores 側影函數驗證數據集群內一致性的方法。



使用 inertia 做模型評估

當K值越來越大，inertia 會隨之越來越小。正常情況下不會取K最大的，一般是取 elbow point 附近作為 K，即 inertia 迅速下降轉為平緩的那個點。

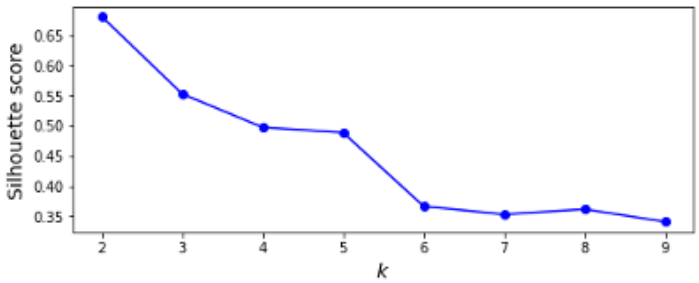
```
# k = 1~9 做9次kmeans, 並將每次結果的inertia收集在一個list裡
kmeans_list = [KMeans(n_clusters=k, random_state=46).fit(X)
               for k in range(1, 10)]
inertias = [model.inertia_ for model in kmeans_list]
```



使用 silhouette scores 做模型評估

另外一個方法是用 silhouette scores 去評估，其分數越大代表分群效果越好。

```
from sklearn.metrics import silhouette_score
silhouette_scores = [silhouette_score(X, model.labels_)
                     for model in kmeans_list[1:]]
```



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 7] 非監督式學習-降維

今日學習目標

- 降維觀念
  - 何謂降維? 降維有什麼優點?
- 常見兩種降維方法
  - PCA & t-SNE

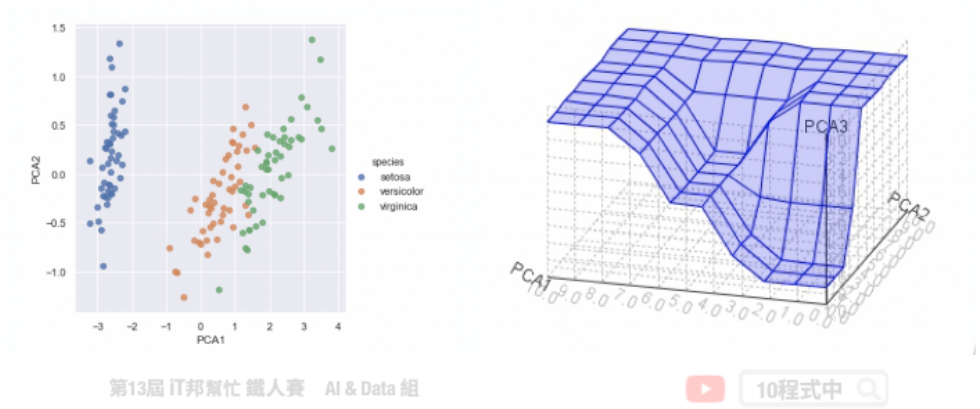
範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/7.非監督式學習-降維/7.非監督式學習-降維.ipynb)*

降維 (Dimension Reduction)

一般資料常見的表示方法有一維(數線)、二維(XY平面)和三維(XYZ立體)。當大於三維的資料就難以視覺化呈現，那麼我們該如何表示高維度的資料同時又不能壓縮原本資料間彼此的關連性呢？這時降維就能幫助你了！降維顧名思義，就是原本的資料處於在一個比較高的維度作標上，我們希望找到一個低維度的作標來描述它，但又不能失去資料本身的特質。

為什麼要降維?

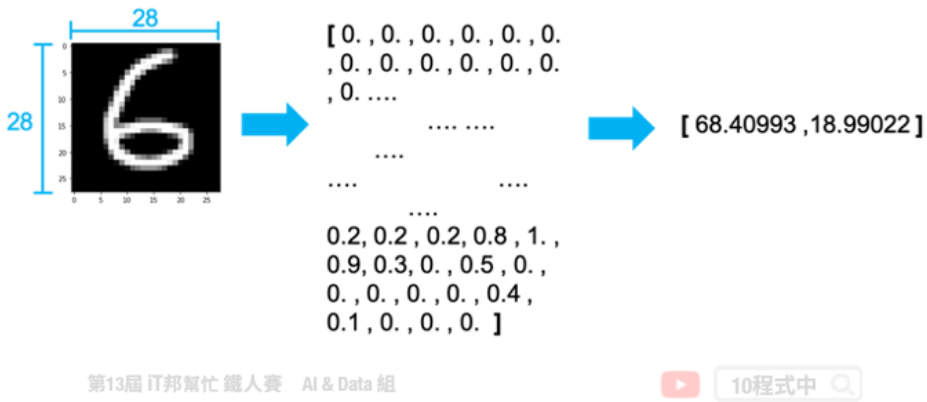
想想看如果我們能夠把一些資料做壓縮，同時又能夠保持資料原來的特性。因此我們可以用比較少的空間，或是計算時用比較少的資源就可以得到跟沒有做資料壓縮之前得到相似的結果。此外資料降維可以幫助我們進行資料視覺化，二維可以用平面圖表示、三維可以用立體圖作表示，而大於三維的空間難以視覺化做呈現。



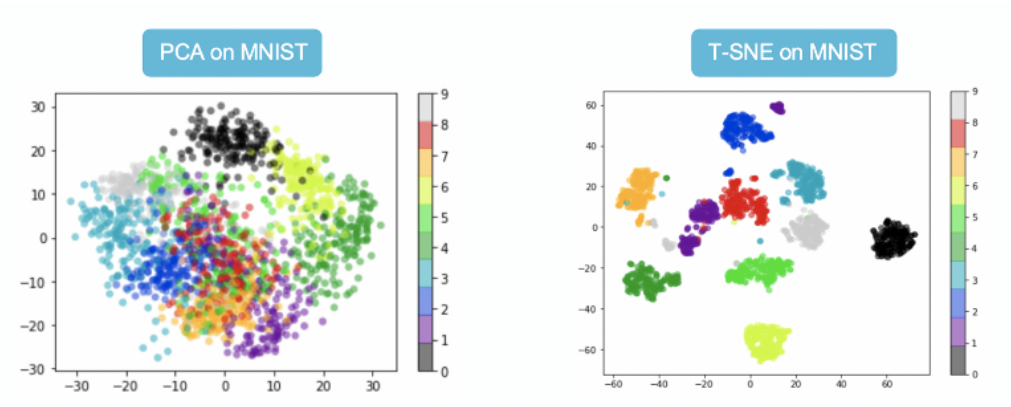
降維演算法

常見的降維方法有兩種分別有線性方法的主成分分析(PCA)以及非線性的 t-隨機鄰近嵌入法(t-SNE)。下圖例子是將 28*28 大小的手寫數字照片，分別透過上述兩種降維方法將一張 784 個像素的影像降成 2 維並投射在平面座標上。我們可以發現 PCA 降為後可以大致將 0~9 的手寫數字照片在平面上分成十群，不過彼此間的界線還是很模糊。而我們透過 t-SNE 方法降為後可以看到平面上很清楚的將這十個數字分成十群。因此我們可以得知手寫數字的影像在非線性的降維轉換效果是比較好的。

- Principal component analysis (PCA)
- T-Distributed Stochastic Neighbor Embedding (t-SNE)



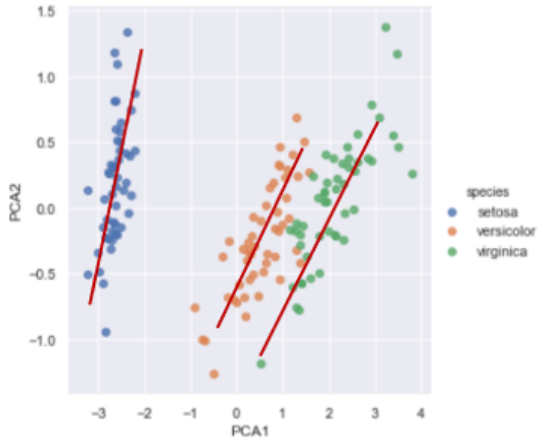
因為 t-SNE 允許非線性的轉換，此外 t-SNE 使用了更複雜的公式來表達高維與低維之間的關係。因此在這種 0~9 有十個分類的情況下可以確保彼此間的距離會被區隔該而不會重疊。



Principal component analysis (PCA)

主成份分析(Principal component analysis, PCA)。其主要目的是把高維的點頭影到低維的空間上，並且低維度的空間保有高維空間中大部分的性質。透過將一個具有 n 個特徵空間的樣本，轉換為具有 k 個特徵空間的樣本，其中 k 必定要小於 n。此外 PCA 只允許線性的轉換。如下圖

所示，我們將捐尾花朵資料集進行 PCA 降維。將原有四個特徵分別有花瓣與花萼的長與寬，透過線性轉換成兩維並投射在平面上。我們可以發現三種花的類別在平面上各自都有線性的趨勢，也就是圖中紅色的線條。



PCA的主要步驟

首先一開始先求出所有資料點中心 μ，也就是將每一個資料點的平均。接著將每一個資料點減去 μ，也就是做資料點的平移，平移後原點是所有點的中心。第三步計算特徵協方差矩陣，其中矩陣對角線上分別是每個特徵的方差，而非對角線上的數值是不同特徵間彼此的協方差。協方差是衡量兩個變數同時變化的變化程度，協方差絕對值越大兩者對彼此的影響越大。第四步驟對矩陣進行特徵值分解，計算協方差矩陣的特徵向量和特徵值並選取特徵向量。第五步驟將特徵值由小到大排序，並選取其中最大的 k 個特徵。然後將這些 k 個特徵向量作為特徵向量矩陣。最後對資料集中的每一個特徵轉換為新的特徵。

1. 先求出所有資料點中心 μ
2. 將每一個資料點減去 μ
3. 計算特徵的協方差矩陣
4. 對矩陣進行特徵值分解
5. 取出最大的 k 個特徵值對應的特徵向量
6. 將資料點投影到選取的特徵向量上

T-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE 目標跟 PCA 是一樣的，它們都希望把高維的資料投影到低維中，並且保留高維中的點與點之間的關係與特性。兩者不同的點在於 t-SNE 允許非線性的轉換。因為 t-SNE 使用了更複雜的公式來表達高維與低維之間的關係。主要是將高維的數據用高斯分佈的機率密度函數近似，而低維數據的部分使用 t 分佈的方式來近似。

高維度

$$p_{ij} = \frac{\exp\frac{\left(-\|x_i - x_j\|^2\right)}{2\sigma^2}}{\sum\frac{\exp(-\|x_k - x_l\|^2)}{2\sigma^2}}$$

低維度

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum\left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

PCA & t-SNE 整理

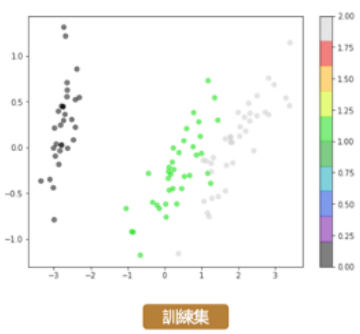
PCA和t-SNE是兩個不同降維的方法，PCA的優點在於簡單若新的點要映射時直接代入公式即可得出降維後的點。若t-SNE有新的點近來時我們沒有去計算新的點和舊的點之間的關係因此 我們無法將新的點投影下去。t-SNE的優點是可以保留原本高維距離較遠的點降維後依然保持 遠的距離，因此這些群降維後依然保持群的特性。

- PCA允許線性的轉換
- t-SNE允許非線性的轉換

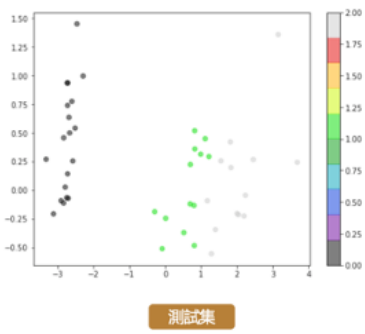
[程式實作]

PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, iterated_power=1)
train_reduced = pca.fit_transform(X_train)

print('PCA方差比: ',pca.explained_variance_ratio_)
print('PCA方差值:',pca.explained_variance_)
```



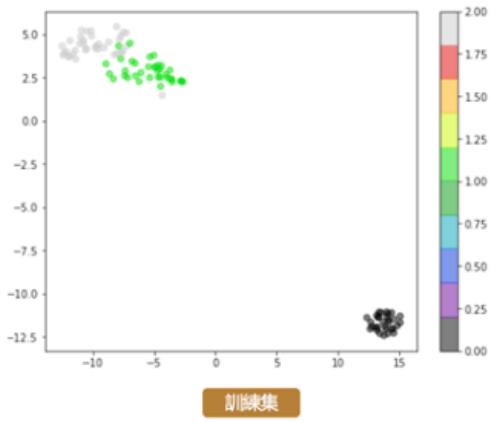
訓練集



測試集

t-SNE

```
from sklearn.manifold import TSNE

tsneModel = TSNE(n_components=2, random_state=42,n_iter=1000)
train_reduced = tsneModel.fit_transform(X_train)
```



t-SNE 不適用於新資料。PCA 降維可以適用新資料，可呼叫transform() 函式即可。而 t-SNE 則不行。因為演算法的關係在 scikit-learn 套件中的 t-SNE 演算法並沒有transform() 函式可以呼叫。

Reference

- 深入學習主成分分析（PCA）演算法原理及其Python實現 *(https://www.itread01.com/content/1547122639.html)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 8] 線性迴歸 (Linear Regression)

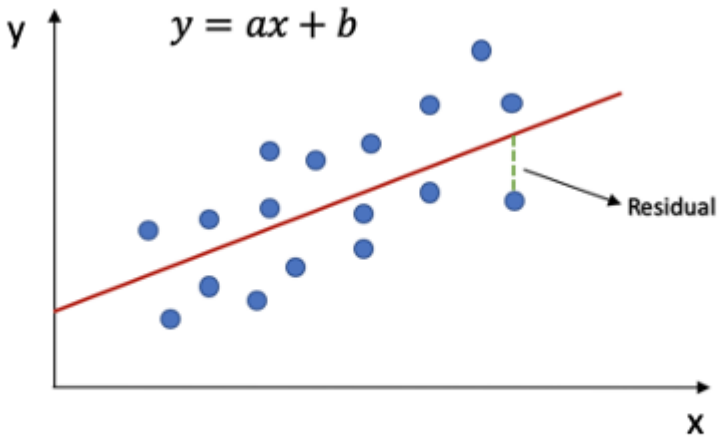
今日學習目標 - 認識線性迴歸 - 透過機器學習來找出一條函式，來最佳化模型 - 兩種求解方法 - 線性迴歸程式手把手 - 簡單線性迴歸、多元迴歸、非線性迴歸

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/8.線性迴歸/8.線性迴歸.ipynb)*

認識線性迴歸

線性迴歸是統計上在找多個自變數和依變數之間的關係所建出來的模型。只有一個自變數(x)和一個依變數(y)的情形稱為簡單線性迴歸大於一個自變數(x_1,x_2,...)的情形稱為多元迴歸。

一個簡單線性迴歸: y=ax+b，其中 b：截距(Intercept)，a：斜率(Slope) 為 x 變動一個單位 y 變動的量，如下圖:



迴歸分析的目標函數或稱損失函數(loss function)就是希望找到的模型最終的殘差越小越好，來找參數 a 和 b。

兩種求解方法

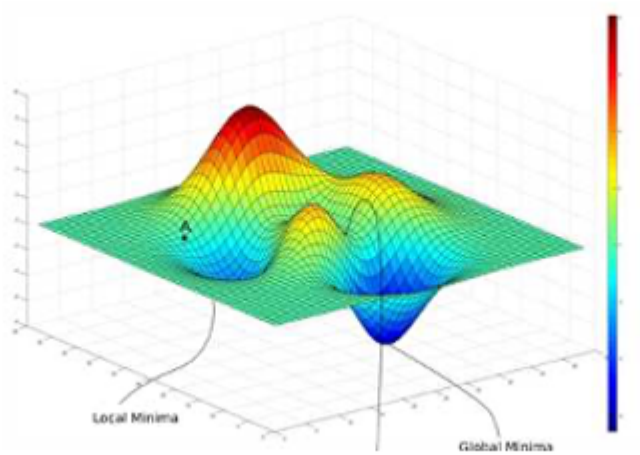
線性模型最常見的解法有兩種，分別為 Closed-form (閉式解) 與梯度下降 (Gradient descent)。當特徵少時使用 Closed-form 較為適合，使用下面公式來求出 θ 值。我們又可以說線性模型的

最小平方法的解即為 Closed-form。若當是複雜的問題時 Gradient descen 較能解決，其原因是大部分的問題其實是沒有公式解的。我們只能求出一個函數 f(x) 使其誤差最小越好。

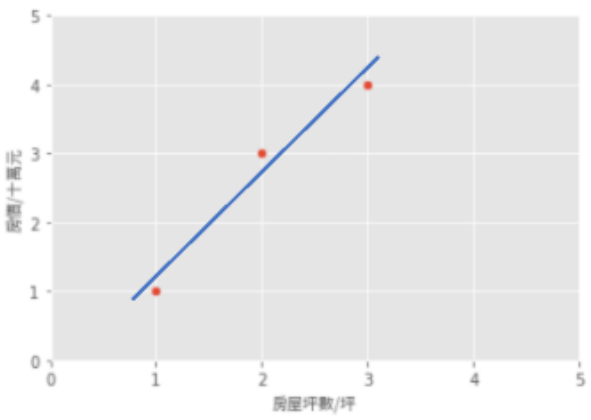
- Closed-form

$$\theta = (X^T X)^{-1} X^T y$$

- Gradient descent



Least Square Method (最小平方法) 假設一個地區的房價與坪數是呈線性關係，並以下圖中的三個點表示。如果我們想透過房子的坪數來預測房價，那麼我們的目標就是找到一條直線，並與座標平面上這三個點的差距越小越好。那這條直線該怎麼找呢？首先我們隨機找一條直線，並計算這三點的 loss。損失函數可以自己定義，假設我們使用 MSE 均方誤差來計算。透過一系列計算我們得到一個 loss 即為 MSE 值。接著我們將這個直線稍稍的轉一個角度後又可以計算一個新的 MSE，此刻我們可以發現 MSE 值又比剛剛更小了。也就是說這一條新的直線能夠更法應出訓練集中 A、B、C 的數據點所反映的房屋坪數與房價之間的線性關係。



簡單來說我們在一個二維空間中，我們可以找到無數條直線。現在我們能做的事情就是從這無數條直線中選出一條最佳的當作我們的預測模型，同時它面對這三點的誤差是要最小的。因此我們的目標就是要最小化 MSE 也就是所謂的損失函數 (loss function)。所以整個線性迴歸的目標就是最小化我們的損失函數，其中一個解法就是最小平方法。因為 MSE 等於 1/n 倍的残差平方和 (RSS)，其中分母 n 為常數，不影響極小化故拿掉。因此最終的求解是滿足最小化平方和，使其最小化。經過數學推導後，簡化的公式如下：

$$\theta = (X^T X)^{-1} X^T y$$

小試身手

基於上面的公式我們想找出一組參數權重 θ。也就是下圖問題中的 a (θ_0)、b (θ_1) 兩參數，使得平面上這三點平方和有極小值。這個函式對 θ_0, θ_1 偏做微分設他們為0，接著我們對方程式求解。 此函式只有極小值，因此我們得到的 θ_0, θ_1 最小極值的解。

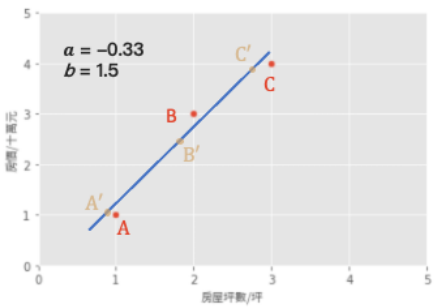
- 找 $y = a + bx$ 盡量滿足 $(1, 1), (2, 3), (3, 4)$?
- $g(a, b) = (a + b - 1)^2 + (a + 2*b - 3)^2 + (a + 3*b - 4)^2$
- $|kA - Y|$ 最小值，取 $k = (ATA)^{-1}A^TY$

$$\begin{bmatrix} a+b-1 \\ a+2b-3 \\ a+3b-4 \end{bmatrix} \text{最小} \Rightarrow \begin{matrix} a+b=1 \\ a+2b=3 \\ a+3b=4 \end{matrix} \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 8 \\ 19 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} 8 \\ 19 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} -2 \\ 9 \end{bmatrix} = \begin{bmatrix} -0.33 \\ 1.5 \end{bmatrix}$$



範例程式 (房價預測)

手刻線性迴歸

我們透過 Sklearn 所提供的房價預測資料集進行線性迴歸模型建模，並採用最小平法。首先為了要驗證我們上面的公式，因此我們先利用 Numpy 套件自己手刻做一系列的矩陣運算求出每一項的係數與截距。

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_boston

# 載入 Sklearn 房價預測資料集 13個輸入特徵 1個輸出特徵
boston_dataset = load_boston()
# 輸入特徵共13個
X = boston_dataset.data
# 設定截距項 b 權重值為 1
b=np.ones((X.shape[0], 1))
# 添加常數項特徵, 最終有 13+1 個輸入特徵
X=np.hstack((X, b))
# 輸出(房價)
y = boston_dataset.target

# 計算 Beta (@ 為 numpy 中 2-D arrays 的矩陣乘法)
Beta = np.linalg.inv(X.T @ X) @ X.T @ y
y_pred = X @ Beta

# MSE: 21.8948311817292
print('MSE:', mean_squared_error(y_pred, y))
```

計算出來 Beta 後我們再把所有的 X 帶入並做計算，算出來的結果 MSE 為 21.89。最後我們可以試著把 Beta 變數列印出來。總共會有 14 個參數，由 13 個輸入特徵係數與最後一項截距所組成的。

輸出結果：

```
array([-1.08011358e-01,  4.64204584e-02,  2.05586264e-02,  2.68673382
       -1.77666112e+01,  3.80986521e+00,  6.92224640e-04, -1.47556685
        3.06049479e-01, -1.23345939e-02, -9.52747232e-01,  9.31168327
       -5.24758378e-01,  3.64594884e+01])
```

使用 Sklearn LinearRegression

線性迴歸簡單來說，就是將複雜的資料數據，擬和至一條直線上，就能方便預測未來的資料。接下來我們一樣使用房價預測資料集，並使用 Sklearn 提供的 LinearRegression 來求解。

Parameters: - fit_intercept: 是否有截距，如果沒有則直線過原點。

Attributes: - coef_: 取得係數。 - intercept_: 取得截距。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: R2 score 模型評估。

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_boston

# 載入 Sklearn 房價預測資料集 13個輸入特徵 1個輸出特徵
boston_dataset = load_boston()
# 輸入特徵共13個
X = boston_dataset.data
# 輸出(房價)
y = boston_dataset.target

# 訓練模型
linearModel = LinearRegression()
linearModel.fit(X, y)

y_pred = linearModel.predict(X)
# 21.894831181729202
print('MSE:', mean_squared_error(y_pred, y))
```

Sklearn 的 LinearRegression 模型也是採用小平方法求解。我們可以發現其 MSE 與稍早手刻的方法相當很接近。另外 Sklearn 模型同時也提供了 `coef_` 和 `intercept_` 兩個屬性可以取得模型的特徵係數與截距。

```
# 取得13個特徵係數
linearMmodel.coef_
```

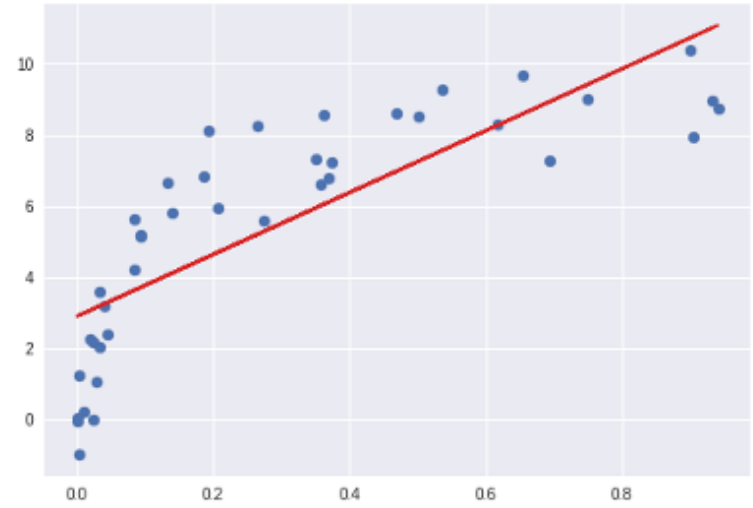
```
array([-1.08011358e-01,  4.64204584e-02,  2.05586264e-02,  2.68673382e+00,
       -1.77666112e+01,  3.80986521e+00,  6.92224640e-04, -1.47556685e+00,
        3.06049479e-01, -1.23345939e-02, -9.52747232e-01,  9.31168327e-03,
       -5.24758378e-01])
```

```
# 取得截距
linearMmodel.intercept_
```

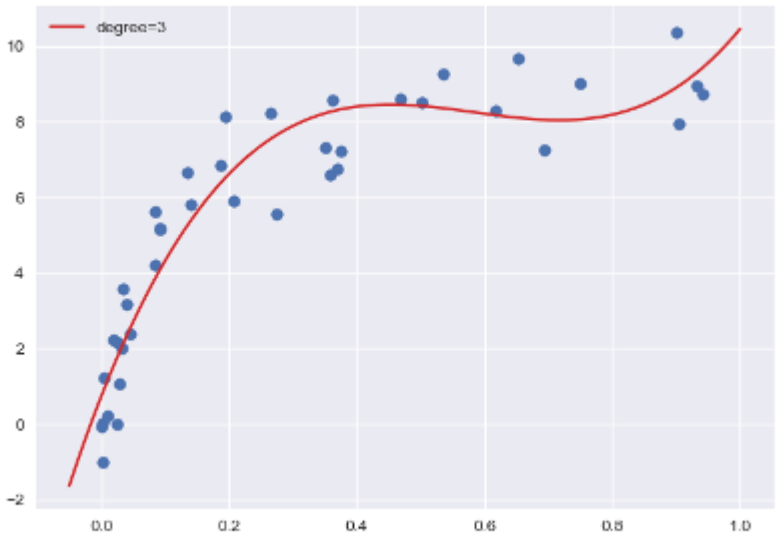
```
36.459488385089855
```


多項式的迴歸模型

對於線性迴歸來說，資料都是很均勻地分布在一條直線上，但現實的資料往往是非線性的分佈。如果我們一樣使用上述方法取得線性模型，在實際場域上預測效果可能並不大。



多項式迴歸中，數據不太具有線性關係，因此應尋找一些非線性曲線去擬合。對於以上的數據，原本是只有一個 x 特徵，但是我們可以建構許多新的特徵。如下圖，用一條三次曲線去擬合數據效果更好。我們將三次函數看成 ax^3+bx^2+cx+d。這樣就又變成解多元，其我們就是要找出 a、b、c、d 使其損失函數最小。



線性模型的擴展

從上述問題中我們可以發現線性迴歸在實務上所面臨的問題。首先我們來迴顧一下稍早所提到的線性方程式，這組線性方程式說明了每個特徵 x 一次方與目標值是有一個線性的關係。

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

接著我們再來看一下另一個例子，比如說特徵 x_1 與目標值存在著以下的關係。我們發現這組方程式已經不是一個線性關係了，因為他有了 x_1 的二次方。

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$$

那麼該怎麼做我們才能又把它轉換成線性關係呢？這時候我們就可以用一個新的特徵 x_2。我們讓 x_2 等於 x_1 的平方，這樣我們再把 x_2 帶回原方程式中。此時這兩個特徵 x_1 與 x_2 與目標值又迴到了線性關係。

$$\text{Let } x_2 = x_1^2$$

$$\Rightarrow y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

同樣的我們再來看另一個例子。我們如果引入了 x_1 的三次方的話，他的方程式如下：

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3$$

同理我們這時一樣可以引入新的特徵 x_2 等於 x_1 的二次方，以及 x_3 等於 x_1 的三次方。這樣經過一個轉換以後我們的 y 值與所有的特徵間依然存在著線性關係。

$$\text{Let } x_2 = x_1^2 \text{ and } x_3 = x_1^3$$

$$\Rightarrow y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

這裡做一個小結。我們可以透過引入轉變過後的 x 作為一個新的特徵來滿足線性假設。此時的迴歸方程式就是一個多項式迴歸(polynomial regression)。

Sklearn 實作多項式迴歸

由於 Sklearn 沒有封裝好的多項式迴歸模型可以直接呼叫。不過我們可以透過 `make_pipeline` 將 `PolynomialFeatures` 與 `LinearRegression` 封裝成一個多項式迴歸模型，並且使用者可以隨意設定 degree(次方)值。

我們可以對原本的特徵進行 PolynomialFeatures 構造新樣本特徵採。並將轉換後的特徵送到線性迴歸模型進行擬合。因此我們可以自定義一個 `PolynomialRegression()` 的函式，使用者可以輸入 degree 大小控制模型的強度。在這個函式中我們使用 Sklearn 的 pipeline 方法將 `PolynomialFeatures` 特徵轉換與 `LinearRegression` 線性迴歸模型封裝起來。另外以下範例是透過自訂義的 `make_data()` 函式產生一組隨機的 x 和 y。該函式中可以設定隨機資料的比數，下面程式中我們先隨機建立 100 筆數據。

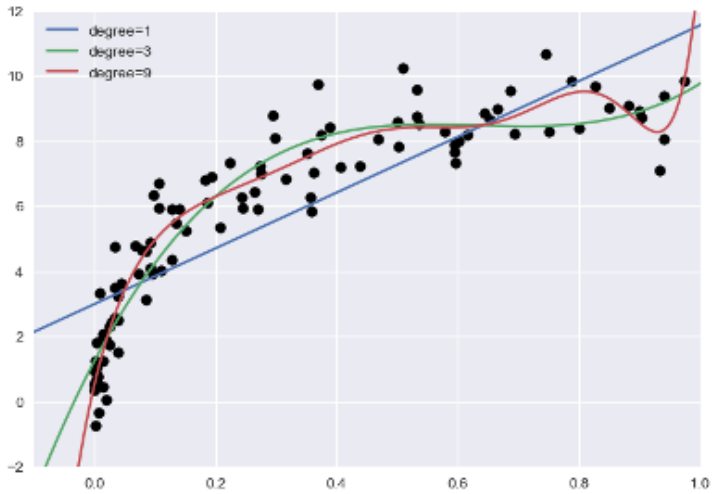
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

```
plt.style.use('seaborn')

# make_pipeline是指可以將多個Sklearn的function一起執行
def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))

# 隨機定義新的x,y值
def make_data(N,err=1,rseed=42):
    rng=np.random.RandomState(rseed)
    x = rng.rand(N,1)**2
    y = 10-1/(x.ravel()+0.1)
    if err>0:
        y+=err*rng.randn(N)
    return x,y

X, y = make_data(100)
```

訓練資料與測試資料都建立完成後。我們就可以將訓練資料丟入建立好的 `PolynomialRegression()` 並進行數據擬合。下面範例程式中我們演示 degree 等於 1、3、9, 並來查看隨著次方數的增長對於模型的擬合程度的影響。

```
# 測試資料集
x_test = np.linspace(-0.1,1.1,500)[:,None]
# 繪製真實答案的分佈
plt.scatter(X.ravel(),y,color='black')

# 測試 1,3,7 的degree
for degree in [1,3,9]:
    y_test=PolynomialRegression(degree).fit(X,y).predict(x_test)
    plt.plot(x_test.ravel(),y_test,label='degree={}'.format(degree))
plt.xlim(-0.1,1.0)
plt.ylim(-2,12)
plt.legend(loc='best')
```

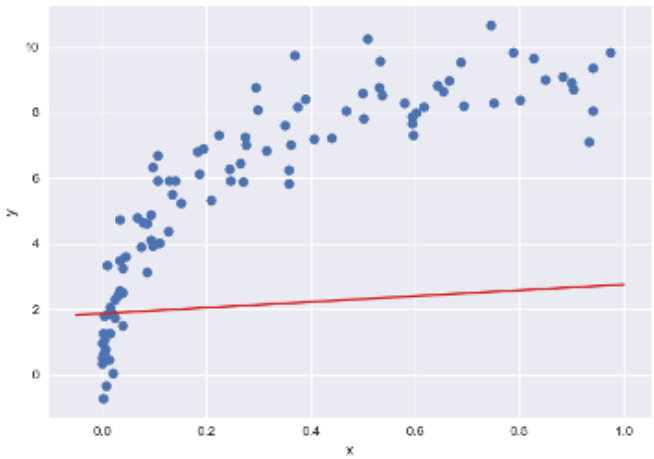
從訓練結果可以發現隨著次方數 degree 的增長模型會變得越複雜。同時對於訓練數據的擬合結果越好。但是這裡必須注意並非越大的 degree 就是越好的, 因為隨著模型複雜會有過度擬合的跡象。因此我們必須找出一個適當的 degree 數值並與測試集驗證與評估。目標是訓練集與測試集的 MSE 差距要越小越好。如果我們一昧的追求訓練集的損失最小化, 可能會影響到測試集的表現能力導致預測結果變差。



Gradient descent (梯度下降法)

接下來我們來討論優化問題的第二種方法，就是梯度下降法。梯度下降不僅限於線性迴歸，在非線性和神經網絡同樣適用。下圖中每一個點是訓練集的樣本 x 軸為輸入值 y 軸為輸出值。也就是平面上每個點 x 都會有一個相對應 y 的輸出，因此我們需要做的事情是為這些點訓練一個模型，使得這條直線能夠盡可能反映出 x 與 y 之間的關係。此外我們都知道在一個二維空間中我們能找到無數條直線，那我們該如何找到這條最佳的直線呢？簡單來說我們的目標是要使得這些訓練資料中的每個樣本點到這一條直線的距離平方和要最小。因此這裡我們將討論該如何使用梯度下降法來最佳化我們的模型。首先我們假設一個直線的方程式是 $y = \beta_0 + \beta_1 x$。那首先我們可以先隨機的給予 β_0 和 β_1 一個初始值。並得到下圖中的結果，我們可以發現這一條直線並不能反映出 x 和 y 的關聯性。

線性方程式：$y = \beta_0 + \beta_1 x$



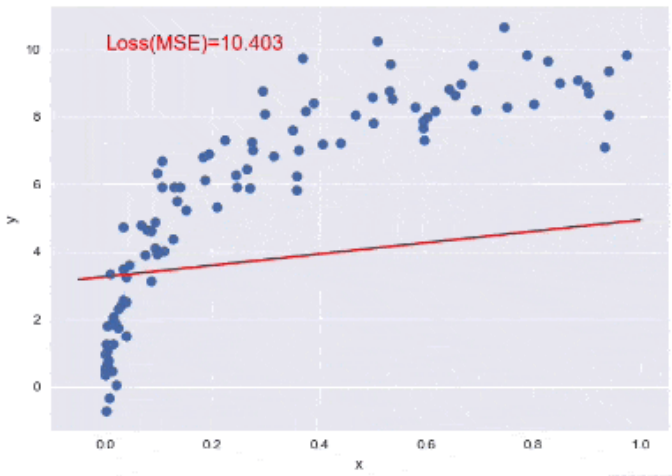
如果我們不斷的迭代，每一次的迭代都讓這一條直線朝著更符合數據點的方向移動一點，那麼經過許多次的更新我們就可以得到最佳的結果。簡單來說就是在每次的迭代要更新所有的參數，例如：β_0 和 β_1，直到得到最小的 MSE 或是預定的迭代次數。以下的公式就是梯度下降法的表達式。它反映的是每次迭代，我們的 β_0 和 β_1 這些參數是如何調整的。我們可以從這個公式得知，他是對損失函數求了某一個特定參數的偏導。這就是所謂的梯度，我們朝著梯度的反方向在更新。然而每一次要更新多大可以依靠 η（(eta) 來控制，因此我們算出來的梯度還會乘上一個學習速率來防止更新步伐太大而導致找不到解。所以 η 的大小要適中以免影響到模型最終的收斂。

$$minimize_{\beta 0,\,\beta 1} J(\beta_0, \beta_1)$$

$$\beta_i := \beta_i - \alpha \frac{\partial J}{\partial \beta_i}$$

α 為學習速率$(learning\ rate)$

此外這個模型如果透過梯度下降法還有一個缺點，那就是當我們的損失函數不是一個凸函數 (convex function) 的時候它就會存在許多個最低點，進而導致在我們選擇不同的 β_0 和 β_1 作為初始值的時候很可能會收斂於不同的局部最佳解(local optimum)。也就是說我們求得的最佳的模型很有機會是局部最佳解而不是全局最佳解(global optimum)。

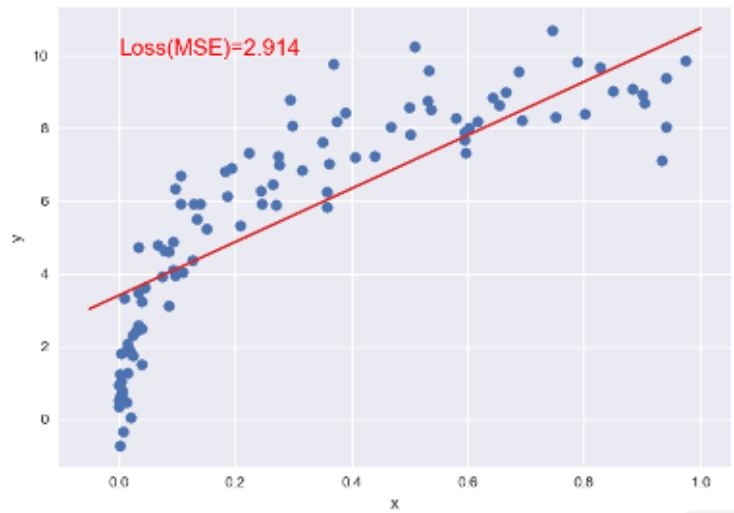


使用 Sklearn SGDRegressor

Sklearn 提供了 SGDRegressor 並實現了隨機梯度下降學習。你可能會問梯度下降與隨機梯度下降兩者差別在哪？簡單來說一般的梯度下降法是一次用全部訓練集的數據計算損失函數的梯

度，然後做一次參數的更新修正。而隨機梯度下降法就是一次跑一個樣本或是小批次樣本，然後算出一次梯度並更新。而所謂的隨機就是在訓練過程中隨機地抽取樣本，所以才會稱為隨機梯度下降法。

```
import numpy as np
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error

# 隨機產生一個特徵的X與輸出y
X, y = make_data(100)

# 建立 SGDRegressor 並設置超參數
regModel = SGDRegressor(max_iter=100)
# 訓練模型
regModel.fit(X, y)
# 建立測試資料
x_test = np.linspace(-0.05,1,500)[:,None]
# 預測測試集
y_test=regModel.predict(x_test)
# 預測訓練集
y_pred=regModel.predict(X)
# 視覺化預測結果
plt.scatter(X,y)
plt.plot(x_test.ravel(),y_test, color="#d62728")
plt.xlabel('x')
plt.ylabel('y')
plt.text(0, 10, 'Loss(MSE)=%.3f' % mean_squared_error(y_pred, y), fon
plt.show()
```



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 9] 邏輯迴歸 (Logistic Regression)

今日學習目標

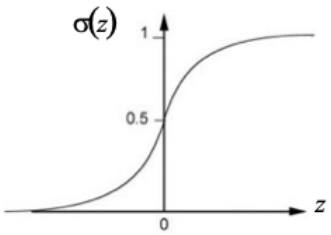
- 認識邏輯迴歸
  - 線性分類器
  - 邏輯迴歸學習機制
  - 比較線性迴歸與邏輯迴歸
  - 多元分類邏輯迴歸
- 邏輯迴歸程式手把手
  - 使用邏輯迴歸建立鳶尾花朵分類器

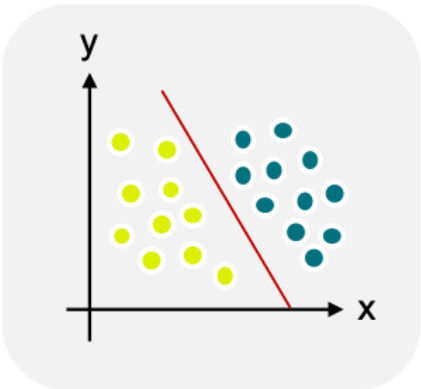
範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/9.邏輯迴歸/9.邏輯迴歸.ipynb)*

認識邏輯迴歸

邏輯迴歸 (Logistic Regression) 是由線性迴歸變化而來的，它是一種分類的模型。其目標是要找出一條直線能夠將所有數據清楚地分開並做分類，我們又可以稱迴歸的線性分類器。邏輯迴歸其實是在說明一個機率的意義，透過一個 function 去訓練得到的一組參數，不同的 w,b 就會得到不同的 function。於是我們可以說 $f_{w,b}(x)$ 即為 posteriror probability。

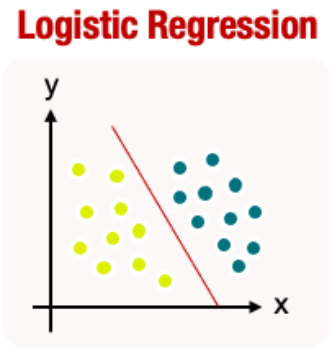
$$\sigma(z) = \frac{1}{1 + exp(-z)}$$

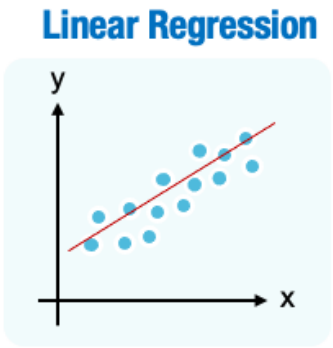




線性迴歸與邏輯迴歸

邏輯迴歸是用來處理分類問題，目標是找到一條直線可以將資料做分類。主要是利用 sigmoid function 將輸出轉換成 0~1 的值，表示可能為這個類別的機率值。而線性迴歸是用來預測一個連續的值，目標是想找一條直線可以逼近真實的資料。





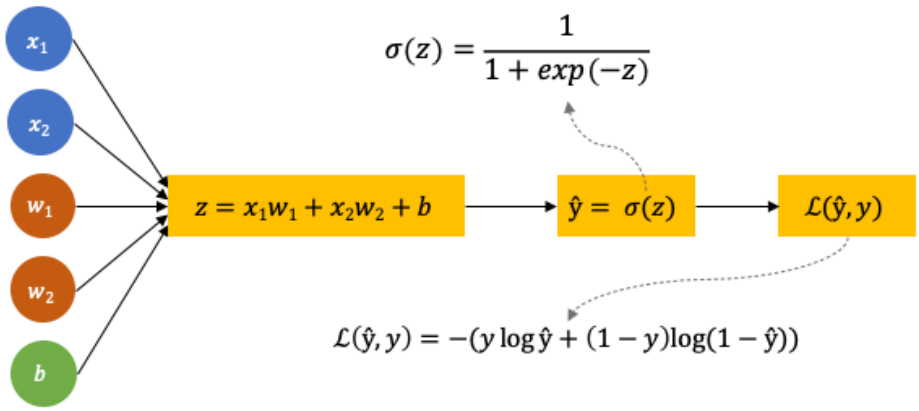
邏輯迴歸學習機制

邏輯迴歸是一個最基本的二元線性分類器。我們要找一個機率 (posterior probability) 當機率 P(C1|x) 大於 0.5 時則輸出預測 Class 1，反之機率小於 0.5 則輸出 Class 2。如果我們假設資料是 Gaussian 機率分佈，我們可以說這個 posterior probability 就是 $\sigma(z)$。其中 `z=w*x+b`，x 為輸入特徵，而 w 與 b 分別為權重(weight)與偏權值(bias) 他們是透過訓練得到的一組參數。

Function set:

$$f_{w,b}(x) = P_{w,b}(C_1|x)$$

以下就是一個邏輯迴歸的運作機制，如果以圖像化表示會長這樣。我們的 function 會有兩組參數，一組是 w 我們稱為 weight，另一個常數 b 稱為 bias。假設我們有兩個輸入特徵，並將這兩個輸入分別乘上 w 再加上 b 就可以得到 z，然後通過一個 sigmoid function 得到的輸出就是 posterior probability。



在邏輯迴歸中我們定義的損失函數是要去最小化的對象是所有訓練資料 cross entropy 的總和。我們希望模型的輸出要跟目標答案要越接近越好。因此我們可以將最小化的目標寫成一個函數：

Cross entropy:

$$C(f(x^n), \hat{y}^n) = -[\hat{y}^n ln f(x^n) + (1 - \hat{y}^n) ln(1 - f(x^n))]$$

最後是尋找一組最好的參數，使得 loss 能夠最低。因此這裡採用梯度下降 (Gradient Descent) 來最小化交叉熵 (Cross Entropy)。我們將損失函數對權重求偏導後，可以得到下面的權重更新的式子：

$$w_{i+1} = w_i - \eta \sum_n -(\hat{y}^n - f_{w,b}(x^n))x_i^n$$

多元分類邏輯迴歸 (Multinomial Logistic Regression)

在 Sklearn 中也能使用邏輯迴歸分類器應用在多類別的分類問題上，對於多元邏輯迴歸有 one-vs-rest(OvR) 和 many-vs-many(MvM) 兩種方法。兩者的做法都是將所有類別的資料依序作二元分類訓練。MvM 相較於 OvR 比較精準，但 `liblinear` 只支援 OvR。

- one-vs-rest(OvR): 訓練時把某個類別的資料歸為一類，其他剩餘的資料歸為另一類做邏輯迴歸，因此若有 k 個類別的資料會有 k 個二元分類器。假如有三個類別 A、B、C，首先抽取 A 類別的資料做為正集，B、C 類別資料做為負集; B 類別的資料作為正集，A、C 類別類別資料做為負集; C 類別的資料作為正集，A、B 類別類別資料做為負集。透過這三組訓練集分別進行訓練，然後的得到三個分類器 f1(x)、f2(x)、f3(x)。預測的時候就是把資料丟進三個分類器，查看哪個分類器預測的分數最高就決定該類別。
- many-vs-many(MvM): 與 OvR 差別在於訓練時每次只會挑兩個類別訓練一個分類器，因此 k 個類別的資料就需要 k(k-1)/2 個二元分類器。假如有三個類別 A、B、C，因此我們會有

三組二元分類器分別有 (A、B)、(A、C) 與 (B、C)。訓練完成後當有新資料要預測時，把資料分別對三個二元分類器進行預測，最終多數決的方式得到預測結果。

[程式實作]

邏輯迴歸 (分類器)

邏輯迴歸雖然有迴歸兩字但他其實是被用來做分類的，目的是要找出一條直線能夠將兩個類別分開。本範例採用鳶尾花朵資料集做分類器實驗，希望能夠透過線性分類器將三個類別彼此區隔開。

Parameters: - penalty: 正規化l1/l2，防止模型過度擬合。 - C: 數值越大對 weight 的控制力越弱，預設為1。 - n_init: 預設為10次隨機初始化，選擇效果最好的一種來作為模型。 - solver: 優化器的選擇。newton-cg,lbfgs,liblinear,sag,saga。預設為liblinear。 - multi_class: 選擇分類方式，ovr就是one-vs-rest(OvR)，而multinomial就是many-vs-many(MvM)。預設為 auto，故模型訓練中會取一個最好的結果。 - max_iter: 迭代次數，預設為100代。 - class_weight: 若遇資料不平衡問題可以設定balance，預設=None。 - random_state: 亂數種子僅在solver=sag/liblinear時有用。

Attributes: - coef_: 取得斜率。 - intercept_: 取得截距。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - predict_proba: 預測每個類別的機率值。 - score: 預測成功的比例。

```
from sklearn.linear_model import LogisticRegression

# 建立Logistic模型
logisticModel = LogisticRegression(random_state=0)
# 使用訓練資料訓練模型
logisticModel.fit(X_train, y_train)
# 使用訓練資料預測分類
predicted = logisticModel.predict(X_train)
```

使用Score評估模型

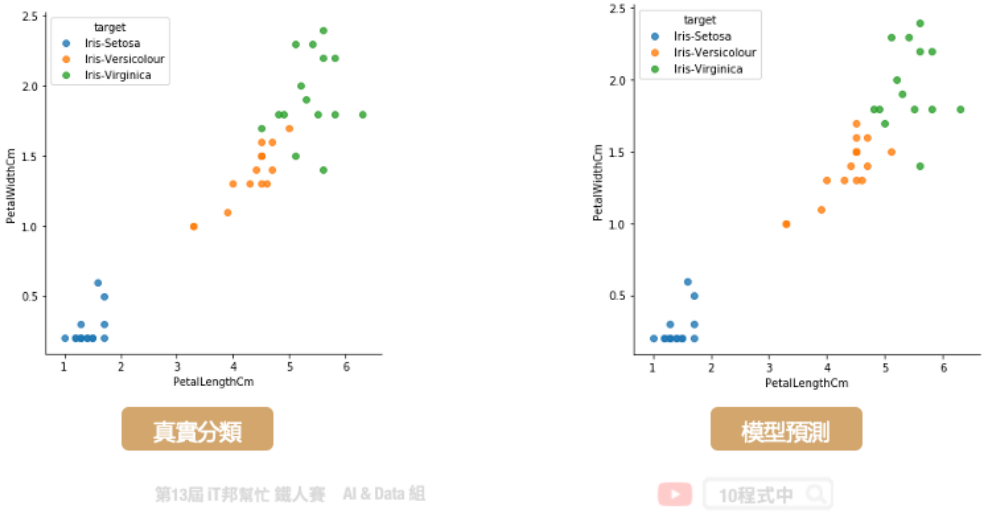
我們可以直接呼叫 `score()` 直接計算模型預測的準確率。

```
# 預測成功的比例
print('訓練集: ',logisticModel.score(X_train,y_train))
print('測試集: ',logisticModel.score(X_test,y_test))
```

輸出結果：

```
訓練集:  0.9714285714285714
測試集:  0.9333333333333333
```

透過 Sklearn 的 `LogisticRegression` 可以實作一個典型的二元分類器。不過當有多個類別的時候，我們可以透過參數 `multi_class` 來設定多元分類器的學習機制。我們可以觀察一下訓練好的模型在測試集上的預測能力，為了方便觀察訓練結果，因此我們只挑選其中兩個特徵並繪製平面的點散圖。下圖中左邊的是測試集的真實分類，右邊的是模型預測的分類結果。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 10] 近朱者赤，近墨者黑 - KNN

今日學習目標

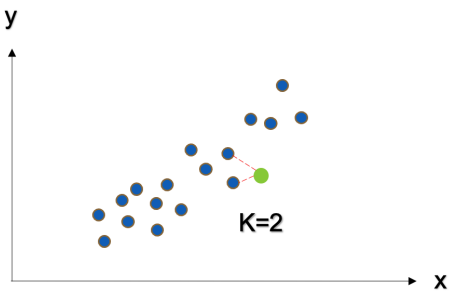
- K-近鄰演算法介紹
  - KNN 演算法解析
  - KNN 於分類器和迴歸器的做法
  - 比較 KNN 與 k-means 差異
- 實作 KNN 分類器與迴歸器
  - 實作 KNN 分類器，觀察不同 k 值會對分類結果造成什麼影響
  - 實作 KNN 迴歸迴器

範例程式　KNN(Classification)：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/10.KNN/10.1.KNN(Classification-iris).ipynb)*

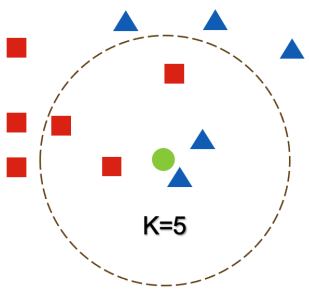
範例程式　KNN(Regression)：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/10.KNN/10.2.KNN(Regression).ipynb)*

K-近鄰演算法 (KNN)

KNN 的全名 K Nearest Neighbor 是屬於機器學習中的 Supervised learning 其中一種算法，顧名思義就是 k 個最接近你的鄰居。分類的標準是由鄰居「多數表決」決定的。在 Sklearn 中 KNN 可以用作分類或迴歸的模型。



KNN 迴歸器



KNN 分類器

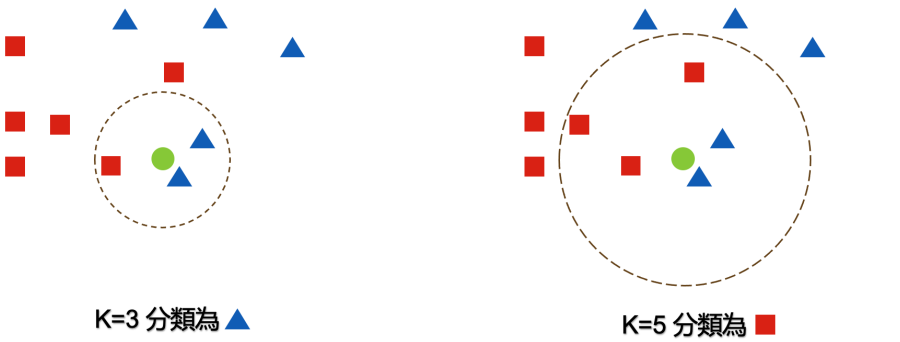
KNN 分類器

在分類問題中 KNN 演算法採多數決標準，利用 k 個最近的鄰居來判定新的資料是在哪一群。其演算法流程非常簡單，首先使用者先決定 k 的大小。接著計算目前該筆新的資料與鄰近的資料間的距離。第三步找出跟自己最近的 k 個鄰居，查看哪一組鄰居數量最多，就加入哪一組。

1. 決定 k 值
2. 求每個鄰居跟自己之間的距離
3. 找出跟自己最近的 k 個鄰居，查看哪一組鄰居數量最多，就加入哪一組

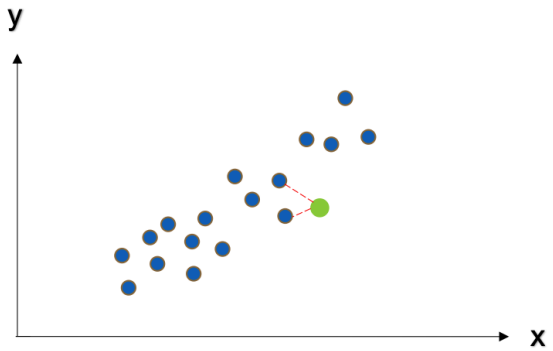
如果還是沒辦法決定在哪一組，回到第一步調整 k 值，再繼續

k 的大小會影響模型最終的分類結果。以下圖為例，假設綠色點是新的資料。當 k 等於 3 時會搜尋離綠色點最近的鄰居，我們可以發現藍色三角形為預測的結果。當 k 設為 5 的時候結果又不一樣了，我們發現距離最近的三個鄰居為紅色正方形。



KNN 迴歸器

KNN 同時也能運用在迴歸問題上面。迴歸模型輸出的結果是一個連續性數值，其預測該值是 k 個最近鄰居輸出的平均值。以下圖為例當 k=2 時，假設我們有一個輸入特徵 x 要預測的輸出為 y。當有一筆新的 x 進來的時候， KNN 迴歸器會尋找鄰近 2 個 x 的輸出做平均當作是該筆資料的預測結果。



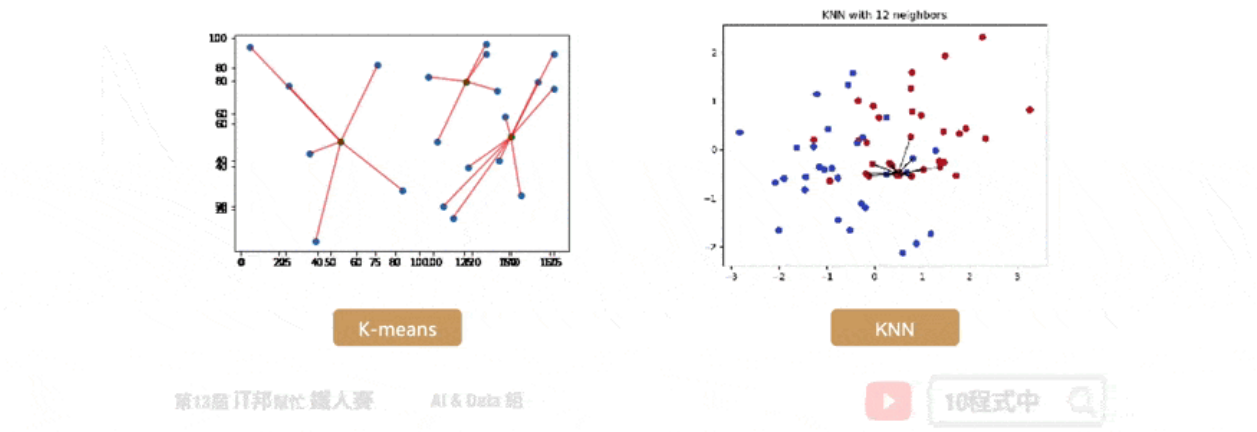
KNN 度量距離的方法

要判斷那些是鄰居的話，首先要量化相似度，而歐幾里得距離 (Euclidean distance) 是比較常用的方法來量度相似度。除此之外還有明可夫斯基距離(Sklearn 預設)、曼哈頓距離、柴比雪夫距離、夾角餘弦、漢明距離、傑卡德相似係數 都可以評估距離的遠近。

identifier	class name	args	distance function
“euclidean”	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
“manhattan”	ManhattanDistance	•	`sum(|x - y|)`
“chebyshev”	ChebyshevDistance	•	`max(|x - y|)`
“minkowski”	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
“wminkowski”	WMinkowskiDistance	p, w	`sum(|w * (x - y)|^p)^(1/p)`
“seuclidean”	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
“mahalanobis”	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

KNN 與 k-means 勿混淆

KNN 的缺點是對資料的局部結構非常敏感，因此調整適當的 k 值極為重要。另外大家很常將 KNN 與 K-means 混淆，雖然兩者都有 k 值要設定但其實兩者無任何關聯。KNN 的 k 是設定鄰居的數量採多數決作為輸出的依據。而 K-means 的 k 是設定集群的類別中心點數量。



[程式實作]

KNN 分類器

採用鳶尾花朵資料集做為分類範例，使用 Sklearn 建立 k-nearest neighbors(KNN) 模型。以下是 KNN 常見的模型操作參數：

Parameters: - n_neighbors: 設定鄰居的數量(k)，選取最近的k個點，預設為5。 - algorithm: 搜尋數演算法{'auto'，'ball_tree'，'kd_tree'，'brute'}，可選。 - metric: 計算距離的方式，預設為歐幾里得距離。

Attributes: - classes_: 取得類別陣列。 - effective_metric_: 取得計算距離的公式。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。

```
from sklearn.neighbors import KNeighborsClassifier

# 建立 KNN 模型
knnModel = KNeighborsClassifier(n_neighbors=3)
# 使用訓練資料訓練模型
knnModel.fit(X_train,y_train)
# 使用訓練資料預測分類
predicted = knnModel.predict(X_train)
```

使用Score評估模型

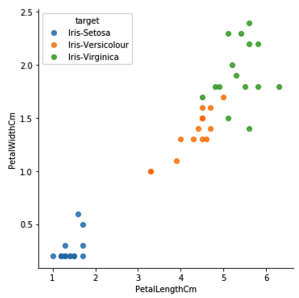
我們可以直接呼叫 `score()` 直接計算模型預測的準確率。

```
# 預測成功的比例
print('訓練集: ',knnModel.score(X_train,y_train))
print('測試集: ',knnModel.score(X_test,y_test))
```

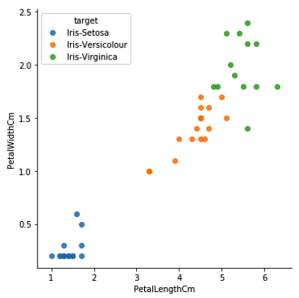
執行結果：

```
訓練集:  0.9619047619047619
測試集:  0.9555555555555556
```

我們可以查看訓練好的模型在測試集上的預測能力，下圖中左邊的是測試集的真實分類，右邊的是模型預測的分類結果。從圖中可以發現藍色的 Setosa 完整的被分類出來，而橘色與綠色的分佈是緊密相連在交界處分類的結果比較不穩定。但最終預測結果結果在訓練集與測試集都有百分之95以上的準確率。



真實分類



模型預測

KNN 迴歸器

KNN 不僅能夠作為分類器，也可以做迴歸連續性的數值預測。其預測值為k個最近鄰居的值的平均值。

Parameters: - n_neighbors: 設定鄰居的數量(k)，選取最近的k個點，預設為5。 - algorithm: 搜尋數演算法{'auto', 'ball_tree', 'kd_tree', 'brute'}，可選。 - metric: 計算距離的方式，預設為歐幾里得距離。

Attributes: - classes_: 取得類別陣列。 - effective_metric_: 取得計算距離的公式。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。

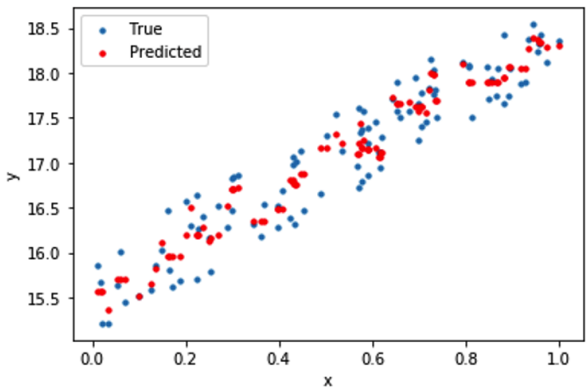
```
from sklearn.neighbors import KNeighborsRegressor

# 建立 KNN 模型
```

```
knnModel = KNeighborsRegressor(n_neighbors=3)
# 使用訓練資料訓練模型
knnModel.fit(x,y)
# 使用訓練資料預測
predicted= knnModel.predict(x)
```

模型評估

Sklearn 中 KNN 迴歸模型的 score 函式是 R2 score，可作為模型評估依據，其數值越接近於1代表模型越佳。除了 R2 score 還有其他許多迴歸模型的評估方法，例如： MSE、MAE、RMSE。

```
from sklearn import metrics
print('R2 score: ', knnModel.score(x, y))
mse = metrics.mean_squared_error(y, predicted)
print('MSE score: ', mse)
```



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 11] 核模型 - 支持向量機 (SVM)

今日學習目標

- SVM 分類器
  - 何謂支持向量機? 非線性與線性?
  - 多元分類支持向量機。
- SVR 迴歸器
  - 學習 SVR 方法如何處理連續性輸出。
- SVM 分類器與 SVR 迴歸器手把手實作
  - 藉由圖形化的邊界，來了解使用不同的 Kernel 及不同參數的意義。
  - 查看 SVR 方法在簡單線性迴歸和非線性迴歸表現。

範例程式　SVM(Classification)： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/11.SVM/11.1.SVM(Classification-iris).ipynb)

範例程式　SVR(Regression)： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/11.SVM/11.2.SVR(Regression).ipynb)

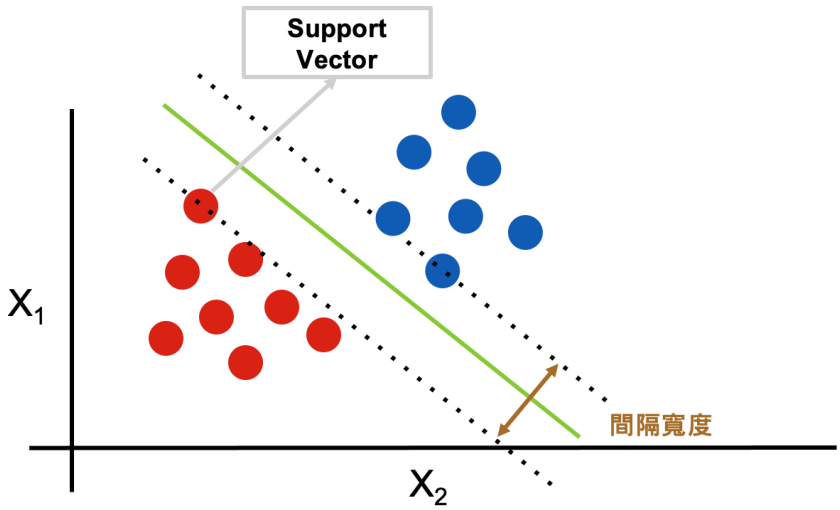
SVM 分類器

支持向量機 (support vector machine, SVM) 是一個基於統計學習的監督式演算法，透過找出一個超平面，使之將兩個不同的集合分開。一般的分類問題我們就是要，找出在不同的資料類別中的分隔線。但在一般狀況下這個分隔線非常複雜且有很多種可能。然而 SVM 就是要在這很多種的可能當中找出最佳的解。SVM　演算法的精神就是找出一條分隔線使所有在邊界上的點離得越遠越好，使模型抵抗雜訊的能力更佳。

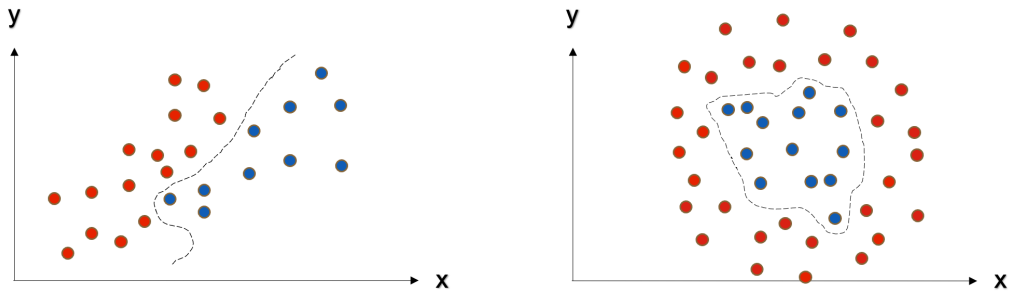
SVM 可分為以下兩種： - 線性可分支持向量機 - 非線性可分支持向量機

線性可分支持向量機

線性可分支持向量機就是在下圖範例的二維圖形中找出一條線，目標讓這條直線與兩個類別之間的間隔寬度距離最大化。其中離兩條虛線(間隔超平面)距離最近的點，就稱為支持向量(support vector)。



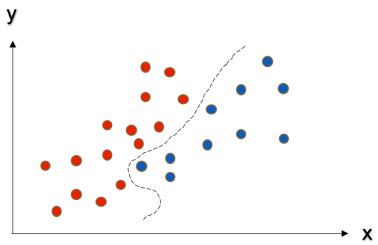
當然現實生活中的資料往往稍微複雜，那如果不是線性可分集合怎麼辦呢？我們可以運用核函數(kernel function) 幫我們造出不可分的分割平面。

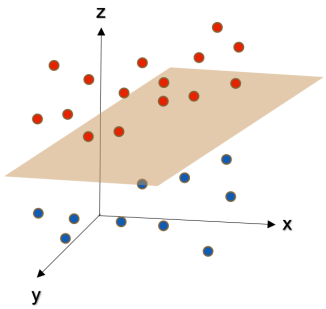


非線性可分支持向量機

除了進行線性分類之外 SVM 還可以使用核技巧有效地進行非線性分類，將其輸入的資料投到更高維度的空間，並在高維度的空間進行高維度的分類或降維。簡單來說透過多維度的投影技巧，將原本在二維空間中不可分的點到了三維空間就可分了。但是隨著資料量增加其運算也會變多，相對的執行速度就會變慢。

兩個非線性的 Kernel： - Polynomial 高次方轉換 - Radial Basis Function 高斯轉換





多元分類支持向量機

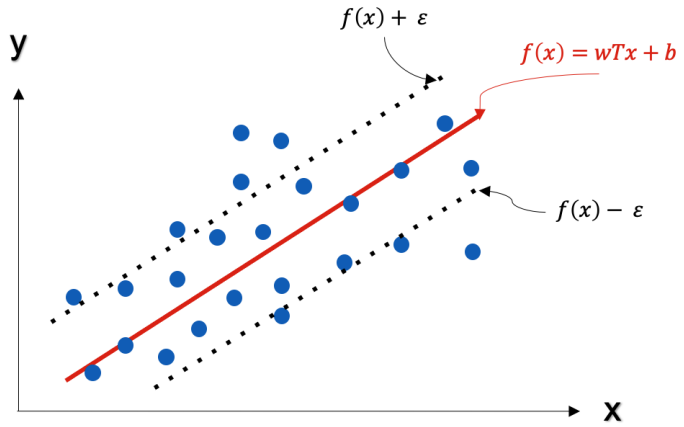
SVM 演算法最初是為二元分類問題所設計的，但是現實生活中的例子一定不只有兩類的問題要解決。他的解決方式與 [Day 9 邏輯迴歸] 所提到的多元分類邏輯迴歸是一樣的。主要是將一個多元分類問題轉換為多個二元分類問題。常見方法包括 one-vs-rest(OvR) 和 many-vs-many(MvM) 兩種。

- one-vs-rest(OvR)
  - 將某個類別的樣本歸為一類，其他剩餘的樣本歸為另一類
- many-vs-many(MvM)
  - 在任意兩類樣本之間設計一個 SVM

詳細介紹可以參考 [Day 9 邏輯迴歸] *(https://ithelp.ithome.com.tw/articles/10269006)*

SVR 迴歸器

支持向量機（SVM）是專門處理分類的問題，還有另一個名詞稱為支持向量迴歸（Support Vector Regression, SVR）專門處理迴歸問題。SVR 是 SVM 的延伸，而支持向量迴歸只要 f(x) 與 y 偏離程度不要太大，既可以認為預測正確。如下圖中的迴歸範例，在線性的 SVR 模型中會在左右加上 ε 作為模型容忍的區間。因此在訓練過程中只有在虛線以外的誤差才會被計算。此外 SVR 也提供了線性與非線性的核技巧，其中在非線性的模型中可以使用高次方轉換或是高斯轉換。



[程式實作]

支持向量機 (Support Vector Machine, SVM) 模型

SVM 能夠透過超參數 C 來達到 weight regularization 來限制模型的複雜度。除了這點我們還能透過 SVM 的 Kernel trick 的方式將資料做非線性轉換, 常見的 kernel 除了 linear 線性以外還有兩了非線性的 Polynomial 高次方轉換以及 Radial Basis Function 高斯轉換。

四種不同SVC分類器: 1. LinearSVC (線性) 2. kernel='linear' (線性) 3. kernel='poly' (非線性) 4. kernel='rbf' (非線性)

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。

LinearSVC

Parameters: - C: 限制模型的複雜度, 防止過度擬合。 - max_iter: 最大迭代次數, 預設1000。

```
from sklearn import svm

# 建立 linearSvc 模型
linearSvcModel=svm.LinearSVC(C=1, max_iter=10000)
# 使用訓練資料訓練模型
linearSvcModel.fit(train_reduced, y_train)
# 使用訓練資料預測分類
predicted=linearSvcModel.predict(train_reduced)
# 計算準確率
accuracy = linearSvcModel.score(train_reduced, y_train)
```

訓練集 Accuracy: 0.96

kernel='linear'

Parameters: - C: 限制模型的複雜度，防止過度擬合。 - kernel: 此範例採用線性。

```
from sklearn import svm

# 建立 kernel='linear' 模型
svcModel=svm.SVC(kernel='linear', C=1)
# 使用訓練資料訓練模型
svcModel.fit(train_reduced, y_train)
# 使用訓練資料預測分類
predicted=svcModel.predict(train_reduced)
# 計算準確率
accuracy = svcModel.score(train_reduced, y_train)
```

訓練集 Accuracy: 0.97

kernel='poly'

Parameters: - C: 限制模型的複雜度，防止過度擬合。 - kernel: 此範例採用 Polynomial 高次方轉換。 - degree: 增加模型複雜度，3 代表轉換到三次空間進行分類。 - gamma: 數值越大越能做複雜的分類邊界。

```
from sklearn import svm

# 建立 kernel='poly' 模型
polyModel=svm.SVC(kernel='poly', degree=3, gamma='auto', C=1)
# 使用訓練資料訓練模型
polyModel.fit(train_reduced, y_train)
# 使用訓練資料預測分類
predicted=polyModel.predict(train_reduced)
# 計算準確率
accuracy = polyModel.score(train_reduced, y_train)
```

訓練集 Accuracy: 0.97

kernel='rbf'

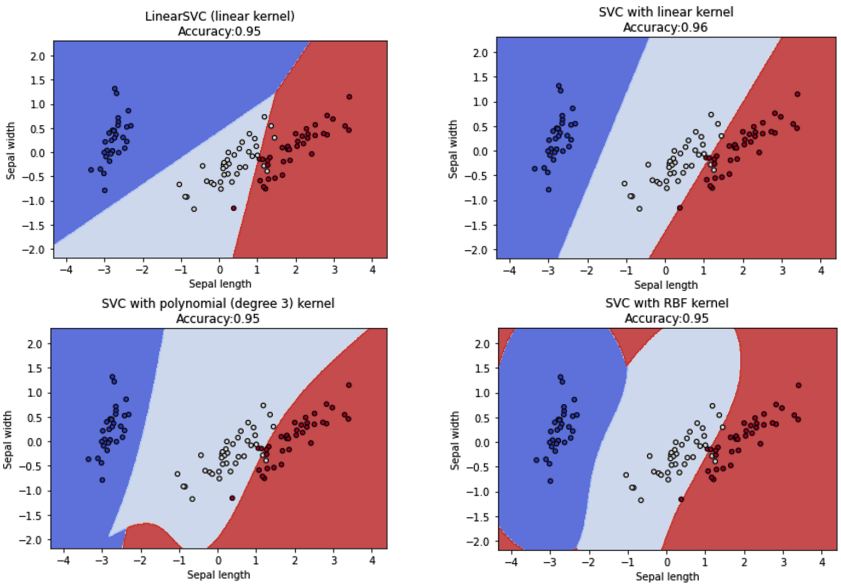
Parameters: - C: 限制模型的複雜度，防止過度擬合。 - kernel: 此範例採用 Radial Basis Function 高斯轉換。 - gamma: 數值越大越能做複雜的分類邊界

```
from sklearn import svm
```

```
# 建立 kernel='rbf' 模型
rbfModel=svm.SVC(kernel='rbf', gamma=0.7, C=1)
# 使用訓練資料訓練模型
rbfModel.fit(train_reduced, y_train)
# 使用訓練資料預測分類
predicted=rbfModel.predict(train_reduced)
# 計算準確率
accuracy = rbfModel.score(train_reduced, y_train)
```

訓練集 Accuracy: 0.97

我們藉由圖形化的邊界，來了解使用不同的 Kernel 及不同參數的意義。以下範例將原先 鳶尾花朵資料集四個特徵透過 PCA 降成二維，以利我們做視覺化觀察。透過四種不同的 SVC 實驗我們可以發現不同的核技巧所預測出來的決策邊線都不盡相同。然而越複雜的模型相對的邊界就會變得越扭曲，因為非線性的模型能夠有比較好的擬合使得錯誤率降低。



支持向量迴歸（Support Vector Regression, SVR）模型

在 Sklearn 中 SVM 提供迴歸的模型稱作 SVR。此外 SVR 迴歸器也提供了三種不同的核函數，分別有一個線性以及兩個非線性的模型可以呼叫。在 SVR 迴歸的實驗，我們拿一組非線性的資料作為例子。並查看在不同的核技巧下模型所擬合的成效為何？

三種不同SVR迴歸器: 1. kernel='linear' (線性) 2. kernel='poly' (非線性) 3. kernel='rbf' (非線性)

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。

kernel='linear'

Parameters: - C: 限制模型的複雜度，防止過度擬合。 - kernel: 此範例採用線性。

```
from sklearn import svm

# 建立 kernel='linear' 模型
linearModel=svm.SVR(C=1, kernel='linear')
# 使用訓練資料訓練模型
linearModel.fit(x, y)
# 使用訓練資料預測分類
predicted=linearModel.predict(x_test)
```

訓練集 MSE: 5.903802524650818

kernel='poly'

Parameters: - C: 限制模型的複雜度，防止過度擬合。 - kernel: 此範例採用 Polynomial 高次方轉換。 - degree: 增加模型複雜度，3 代表轉換到三次空間進行分類。 - gamma: 數值越大越能做複雜的預測。

```
from sklearn import svm

# 建立 kernel='poly' 模型
polyModel=svm.SVR(C=6, kernel='poly', degree=3, gamma='auto')
# 使用訓練資料訓練模型
polyModel.fit(x, y)
# 使用訓練資料預測分類
predicted=polyModel.predict(x_test)
```

訓練集 MSE: 8.296270605383441

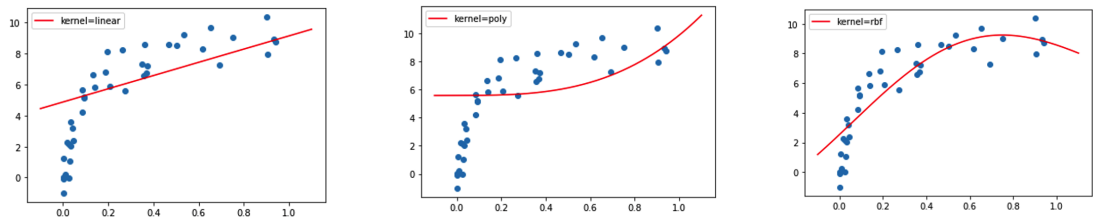
kernel='rbf'

Parameters: - C: 限制模型的複雜度，防止過度擬合。 - kernel: 此範例採用 Radial Basis Function 高斯轉換。 - gamma: 數值越大越能做複雜的分類邊界。

```
from sklearn import svm

# 建立 kernel='rbf' 模型
rbfModel=svm.SVR(C=6, kernel='rbf', gamma='auto')
# 使用訓練資料訓練模型
rbfModel.fit(x, y)
# 使用訓練資料預測分類
predicted=rbfModel.predict(x_test)
```

訓練集 MSE: 2.2551572190243157

這裡的迴歸模型採用非線性的資料進行數據擬合的實驗。我們可以發現線性的核函數無法有效的預測所有數據點的趨勢。而非線性的模型中 RBF 的模型對於此資料有比較好的預測結果。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 12] 決策樹 (Decision tree)

今日學習目標

- 決策樹演算法介紹
  - 決策樹如何生成？
  - 如何處理分類問題？
  - 如何處理迴歸問題？
- 實作決策樹分類器
  - 觀察決策樹是如何生成的。
- 實作決策樹迴歸器
  - 查看決策樹方法在簡單線性迴歸和非線性迴歸表現。

範例程式 決策樹(Classification)： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/12.決策樹/12.1.決策樹(Classification-iris).ipynb)

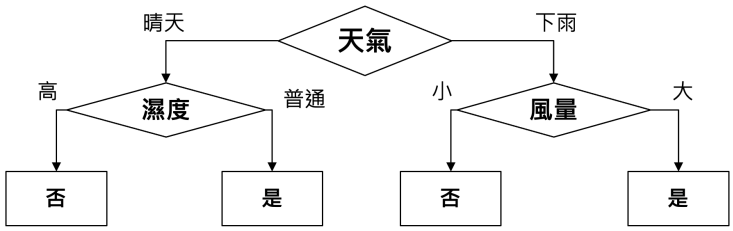
範例程式 決策樹(Regression)： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/12.決策樹/12.2.決策樹(Regression).ipynb)

決策樹

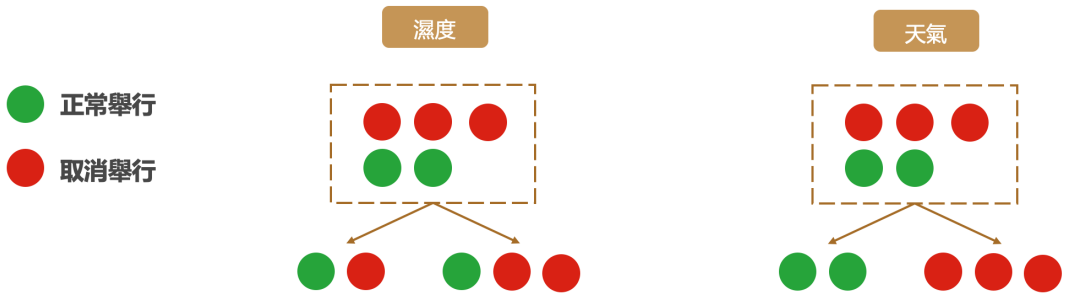
決策樹會根據訓練資料產生一棵樹，依據訓練出來的規則來對新樣本進行預測。決策樹演算法可以使用不同的方式來評估分枝的好壞(亂度)，例如像是 Information gain、Gain ratio、Gini index。依據訓練資料找出合適的規則，最終生成一個規則樹來決策所有事情，其目的使每一個決策能夠使訊息增益最大化。就好比我們評估今天比賽是否舉行，天氣因子可能站比較大的因素，而 Co2 的濃度高低可能佔的因子程度較低。因此在第一層的決策中以天氣的特徵先進行第一次的決策判斷。接著第二層再從所有特徵中尋找最適合的決策因子，直到設定的最大樹的深度即停止樹的生長。

天氣	濕度	風量	是否舉行
晴天	高	大	否
陰天	低	小	是



決策樹如何生成？

決策樹是以一個貪婪法則來決定每一層要問什麼問題，目標是分類過後每一群能夠很明顯的知道是屬於哪一種類別。延續上面的例子，以分類問題來說假設要評估明天比賽是否舉行。在樹的第一層節點中我們要從已知的兩個特徵分別是溫度與特徵選一個作為該層的決策因子。假設目前訓練集有五筆資料，其中正常舉行的有兩筆資料，取消舉行的有三筆資料。在樹的結構中左子樹為決策正常取行，而右子樹是決策取消舉行。我們可以發現當特徵為天氣的時候可以一很清楚的將這兩類別完整分開，因此我們會將天氣作為這一層判斷的因子。這就是決策樹在生成中的貪婪機制。然而要如何去判斷每次決策的好壞，就必須依靠亂度的評估指標。



決策樹的混亂評估指標

我們需要客觀的標準來決定決策樹的每個分支，因此我們需要有一個評斷的指標來協助我們決策。決策樹演算法可以使用不同的指標來評估分枝的好壞，常見的決策亂度評估指標有 Information gain、Gain ratio、Gini index。我們目標是從訓練資料中找出一套決策規則，讓每一

個決策能夠使訊息增益最大化。以上的指標都是在衡量一個序列中的混亂程度，其數值越高代表越混亂。然而在 Sklearn 套件中預設使用 Gini。

- Information gain (資訊獲利)
- Gain ratio (吉尼獲利)
- Gini index (吉尼係數) = Gini Impurity (吉尼不純度)

評估分割資訊量

Information Gain 透過從訓練資料找出規則，讓每一個決策能夠使訊息增益最大化。其算法主要是計算熵，因此經由決策樹分割後的資訊量要越小越好。而 Gini 的數值越大代表序列中的資料亂，數值皆為 0~1 之間，其中 0 代表該特徵在序列中是完美的分類。常見的資訊量評估方法有兩種：資訊獲利 (Information Gain) 以及 Gini 不純度 (Gini Impurity)。

$$Entropy = -\sum_j p_j \log_2 p_j \qquad Gini = 1 - \sum_j p_j^2$$

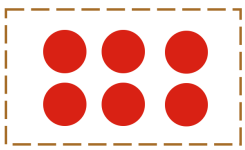
熵 (Entropy)

熵 (Entropy) 是計算 Information Gain 的一種方法。在了解 Information Gain 之前要先了解熵是如何被計算出來的。其中在下圖公式中 p 代表是的機率、q 代表否的機率。我們可以從圖中範例很清楚地知道當所有的資料都被分類一致的時候 Entropy 即為 0，當資料各有一半不同時 Entropy 即為 1。

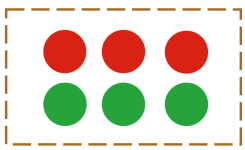
$$\textbf{Entropy} = -\sum p_j \log_2 p_j$$

$$\textbf{Information Gain} = -p * \log_2 p - q * \log_2 q$$

p：是的機率　q：否的機率



$$\textbf{Info(6, 0)} = -\frac{6}{6}\log_2\left(\frac{6}{6}\right) - \frac{0}{6}\log_2\left(\frac{0}{6}\right) = 0$$



$$\textbf{Info(3, 3)} = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) = 1$$

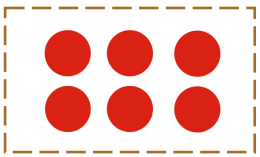
Gini 不純度 (Gini Impurity)

Gini 不純度是另一種亂度的衡量方式，它的數字越大代表序列中的資料越混亂。公式如下所示，其中 p 代表是的機率、q 為代表否的機率。我們可以從圖中範例很清楚地知道當所有的資料都被分類一致的時候混亂程度即為 0，當資料各有一半不同時混亂程度即為 0.5。

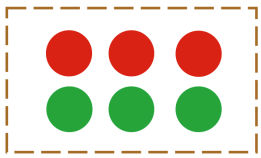
$$\text{Gini} = 1 - \sum p_j^2$$

$$\text{Gini Impurity} = 1 - (p^2 + q^2)$$

p：是的機率　q：否的機率



$\text{Info}(6, 0) = 1 - (1^2 + 0^2) = 0$



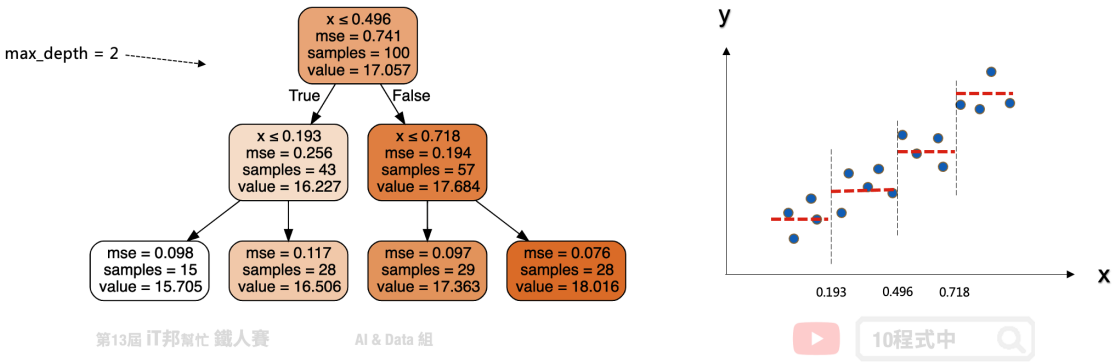
$\text{Info}(3, 3) = = 1 - (0.5^2 + 0.5^2) = 0.5$

迴歸樹

決策樹迴歸方法與分類有點類似差別僅在於評估分枝好壞的方式不同，我們又可以稱作迴歸樹。當數據集的輸出爲連續性數值時，該樹算法就是一個迴歸樹。透過樹的展開，並用葉節點的均值作爲預測值。從根節點開始，對樣本的某一特徵進行測試。經過評估後，將樣本分配到其子結點。此時每一個子節點對應著該特徵的一個值。依照這樣方式進行，直至到達葉結點。此時誤差值要最小化，並且越接近零越好。

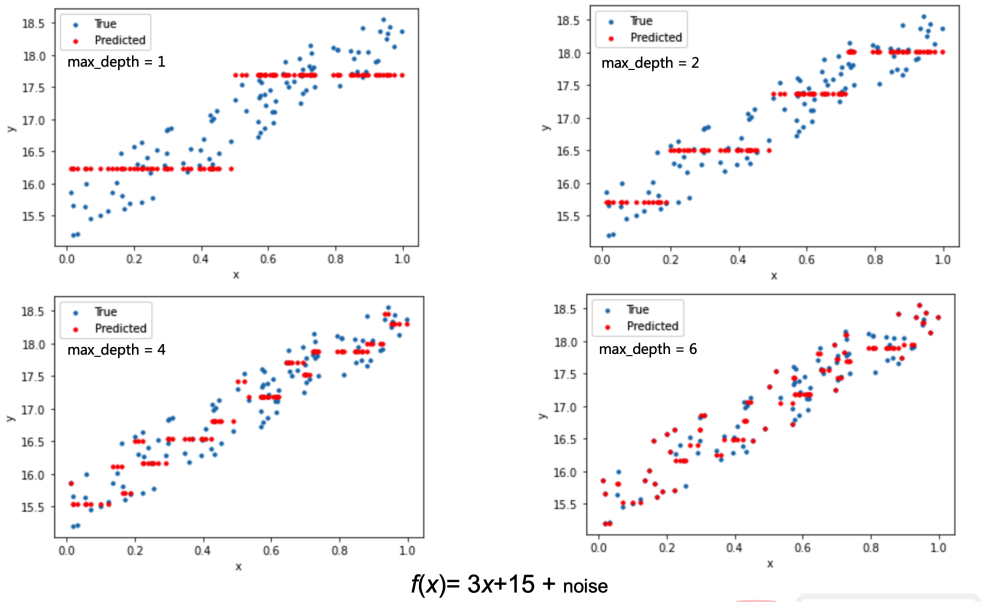
迴歸樹的生長過程很推薦看這篇 *(https://zhuanlan.zhihu.com/p/82054400)*文章

以下舉一個例子假設 x 是輸入 y 是輸出，我們可以在一個平面上繪製出資料與正確答案間的分佈。假設迴歸樹的最大深度設定兩層。首先在第一層中會將所有的資料從中間切一刀此斷點為 `x=0.496` 當大於設定的值的數據點會繼續往右子樹下去延伸，反之小於 0.496 的資料點會往左子樹走。此時將會切出一個分支出來並往下擴展並形成第二層的決策分支。一直不斷持續拓展直到設定的最大深度終止，此時的節點即為葉節點也就是最終的模型輸出值。



樹越深模型越複雜

假設我們生成一個 `f(x) = 3x+15 + noise` 的資料，其中 noise 為一個 0~1 之間的隨機數。從以下的測試可以看出隨著決策樹深度的增加，決策樹的擬合能力不斷上升。決策樹已經不僅僅擬合了我們的線性函式 `3x+15`，同時也擬合了我們添加的噪音(noise)。



$f(x)= 3x+15 + \text{noise}$

迴歸樹該如何選擇切割點?

在分類模型中決策樹是以亂度作為決策樹生成時候的評估指標。但是迴歸樹透過是 `MSE` 或 `MAE` 來評估模型，並找出誤差最小的值作為樹的特徵選擇與切割點。其中前者是均方差，後者是和均值之差的絕對值之和。

- Mean Square Error

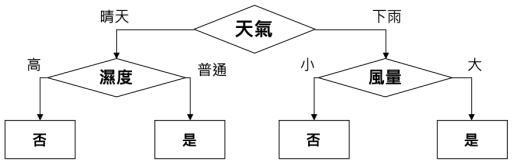
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$$

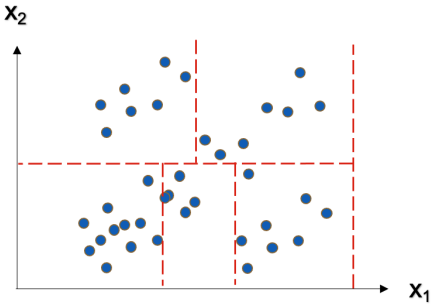
- Mean Absolute Error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|$$

CART 決策樹

在 Sklearn 套件中決策樹演算法是採用 CART (Classification and Regression Tree) 演算法，並且可以被拿來做分類和迴歸的預測。在決策樹的每一個節點上都是採用二分法，也就是每一個決策節點只分枝出兩個子節點。並且不斷地往下拓展，直到設定的最大深度為止，此時最大深度的節點稱為葉節點即為模型的預測輸出。





決策樹模型的優缺點

建立決策樹的過程就是不斷的尋找特徵進行決策，透過這些決策盡量的使這些資料被分為同一個類別，且試著讓混亂程度越小越好。切記樹的深度越深不一定越好，他可能會造成過度擬合的問題。訓練好的模型我們能夠視覺化決策樹的結構，相對的可解釋性就變高。此外與其它的ML模型比較起來，決策樹執行速度是它的一大優勢。因為是樹狀結構，因此在進行機器學習的時候每個決策階段都相當的明確清楚，不是 0 就是 1。

優點

- 簡單且高度可解釋性
- 低計算時間複雜度
- 每個決策階段都相當的明確清楚
- 幾乎沒有要調整的超參數

缺點

- 模型容易過度擬合
- 當標籤類別種類多時樹會很複雜

決策樹總結

決策樹透過所有特徵與對應的值將資料切分，來找出最適合的分枝並繼續往下拓展。若決策樹深度越深則決策的規則將越複雜，模型預測也會越接近真實答案。但若訓練集中含有過多的雜訊，太深的樹就有可能產生過擬合的情形。因此單一的決策樹肯定是不夠用的，我們可以利用集成學習中的 Boosting 架構，對迴歸樹進行改良升級。

樹

森林

[程式實作]

分類決策樹

一個決策樹會根據訓練資料自動產生一棵樹。決策樹會根據資料產生很多樹狀的規則，最終訓練出來的規則會對新樣本進行預測。

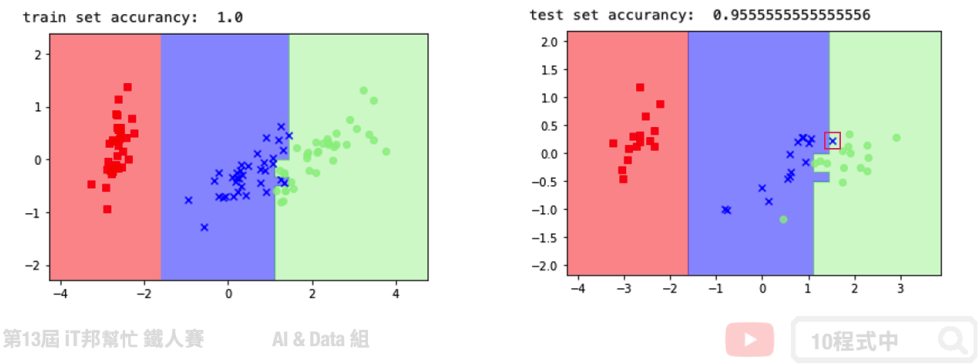
Parameters: - criterion: 亂度的評估標準，gini/entropy。預設為gini。 - max_depth: 樹的最大深度。 - splitter: 特徵劃分點選擇標準，best/random。預設為best。 - random_state: 亂數種子，確保每次訓練結果都一樣，splitter=random 才有用。 - min_samples_split: 至少有多少資料才能再分 - min_samples_leaf: 分完至少有多少資料才能分

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。 - get_depth: 取得樹的深度。

```
from sklearn.tree import DecisionTreeClassifier

# 建立 DecisionTreeClassifier 模型
decisionTreeModel = DecisionTreeClassifier(criterion = 'entropy', max
# 使用訓練資料訓練模型
decisionTreeModel.fit(train_reduced, y_train)
# 使用訓練資料預測分類
predicted = decisionTreeModel.predict(train_reduced)
# 計算準確率
accuracy = decisionTreeModel.score(train_reduced, y_train)
```

我們透過鳶尾花朵資料集進行 PCA 降維並訓練一個決策樹模型。透過繪製訓練決策邊界可以看到，在下圖右手邊的訓練集完整地將三個類別切割開來。而在右邊的測試集中僅有一筆紅色框起來的資料預測錯誤。



迴歸決策樹

Parameters: - criterion: 評估切割點指標，mse/friedman_mse/mae。 - max_depth: 樹的最大深度。 - splitter: 特徵劃分點選擇標準，best/random。預設為best。 - random_state: 亂數種子，確保每次訓練結果都一樣，splitter=random 才有用。 - min_samples_split: 至少有多少資料才能再分 - min_samples_leaf: 分完至少有多少資料才能分

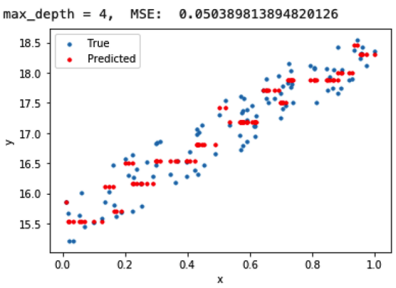
Attributes: - feature_importances_: 查詢模型特徵的重要程度。

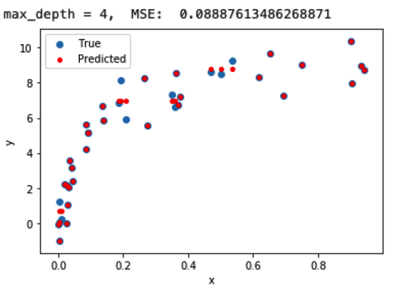
Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - get_depth: 取得樹的深度。

```
from sklearn.tree import DecisionTreeRegressor

# 建立 DecisionTreeRegressor 模型
decisionTreeModel = DecisionTreeRegressor(criterion = 'mse', max_dept
```

```
# 使用訓練資料訓練模型
decisionTreeModel.fit(x, y)
# 使用訓練資料預測
predicted=decisionTreeModel.predict(x)
```

在迴歸決策樹中我們使用了簡單線性迴歸與非線性迴歸兩種資料集進行數據擬合實驗。在簡單線性迴歸中我們將數據點添加一些噪音讓資料分布在斜直線上。左圖是迴歸樹在最大深度為 4 的訓練結果，可以隱約地看到模型決策的方式呈現階梯狀態。如果我們嘗試的將數的深度增加，模型相對複雜因此可以擬合得更好。而右邊是透過隨機產生的非線性資料進行模型訓練。從訓練結果可以發現在最大深度為 4 的時候，訓練結果就還不錯了。大家可以試看看調整模型的樹最大深度以及其他的超參數對模型訓練結果的影響。





本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 13] 整體學習 (Ensemble Learning)

今日學習目標

- 了解整體學習
    - 何謂整體學習？
- 三種不同的整體學習
    - Bagging、Boosting、Stacking

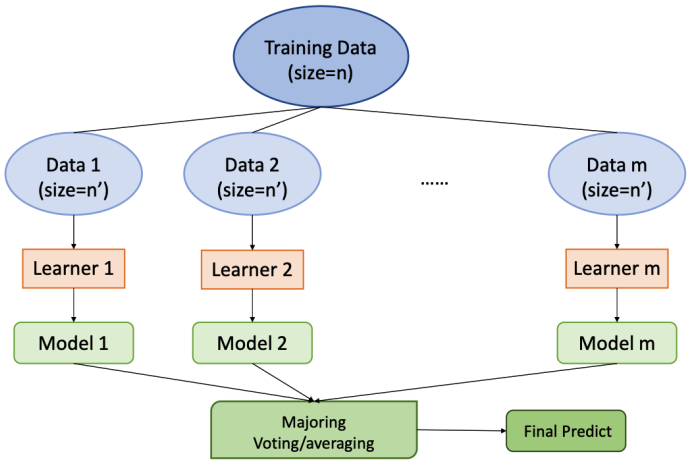
何謂整體學習？

整體學習 (Ensemble Learning) 又稱集成學習、整合學習，指的是以一個系統化的方式將好幾個監督式學習的模型結合在一起，目的是希望結合眾多的模型產生一個更強大的模型。在許多科學競賽中 Ensemble Learning 在實務上是非常有效的提升預測準確率。依照 Ensemble 的處理方式的不同，我們可以將它分為三類。第一類為 Bagging，第二類為 Boosting，第三類為 Stacking。

- Bagging:
    - Random forest
- Boosting:
    - AdaBoost
    - Gradient Boosting
    - XGBoost
- Stacking

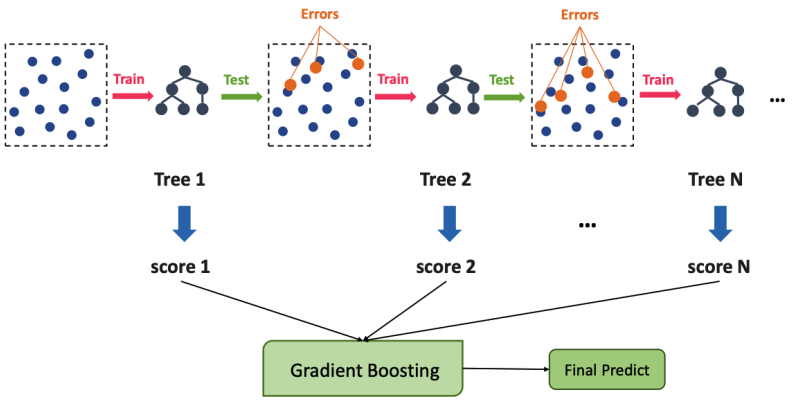
Bagging 自助重抽總合法

Bagging 指的是我們把訓練資料重新採樣產生不同組的訓練資料，如下圖為整體學習 Bagging 之架構。根據不同組的訓練資料即使我們用同一種演算法我們也會得到不一樣的模型，他的樹是各自獨立因此可以平行化處理。代表的方法是隨機森林，隨機森林除了 Bagging 之外，還有另一個隨機的因素是每一棵樹都只能看到一部分的特徵，這些特徵是由隨機決定的。



Boosting 推升法

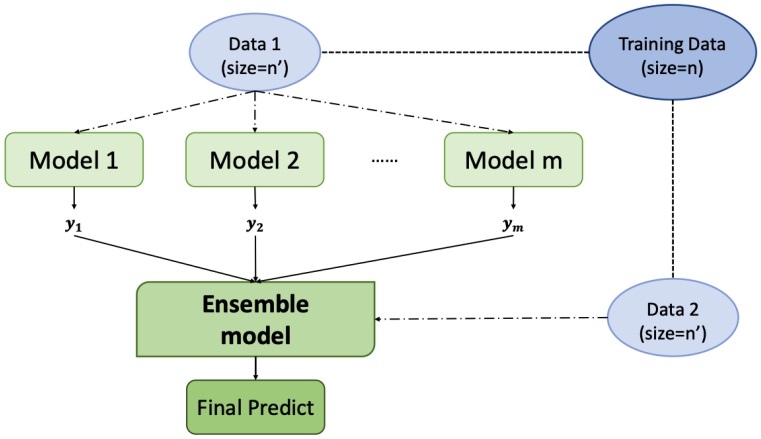
Boosting 則會根據每一筆訓練資料的難或簡單給予不同的權重。如下圖所示，首先我們會訓練一個 base learner 然後根據 base learner 預測的結果對或錯來分辨該筆資料是一個簡單還是困難的資料。對於難的資料我們加強他的權重再訓練一個新的分類器或迴歸器。我們目標是希望訓練後，新的模型在這些難的資料能夠表現得更好。我們不斷重複這些步驟，不斷地加入新的 base learner，且新的 base learner 把過去表現不好的地方改善，這就是 Boosting 精神。因此 Boosting 的每一棵樹是互相有關聯性的做完第一棵樹可能進行下一棵樹的生成。代表的方法有 AdaBoost、Gradient Boosting，兩種都是產生非常多棵的樹，但是每一棵都是很簡單的決策樹。Boosting 目標是希望新的樹可以針對舊的樹預測不太好的部分做一些補強。所以最終我們要把這麼多簡單的樹合再一起才能當最後的預測。



Stacking 堆疊法

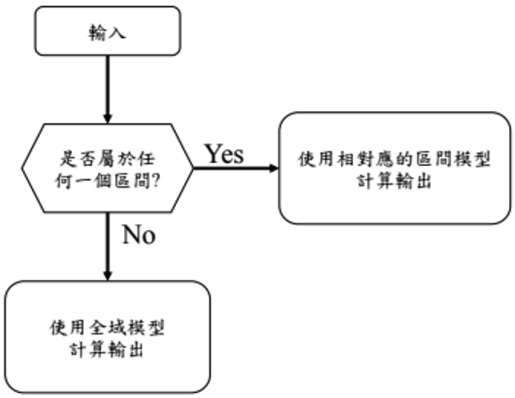
下圖為整體學習 Stacking 架構。Stacking 首先產生出 m 個模型，彼此間並互相無關連，例如第一個模型為 logistic regression 第二個為決策樹。訓練完 m 個模型後，我們要把這 m 個模型合

併在一起。合併的方式是我們再另外訓練一個模型，這個模型把m個模型的輸出當成新的模型的輸入因此我們會根據這m個特徵利用整體學習其中的演算法來學習一個模型並預測最終結果。

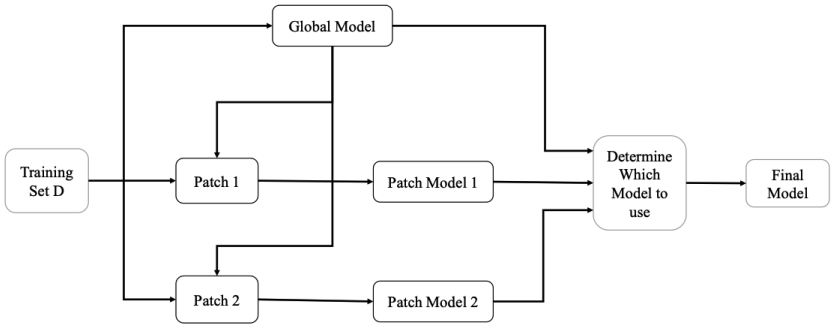


區域學習 (Patch Learning)

區域學習 (Patch Learning *(https://arxiv.org/abs/1906.00158)* , PL) 於 2019 年由美國南加州大學 Jerry M. Mendel 與 Dongrui Wu 所提出的一種機器學習方法。所謂的區域學習是能夠有效的掌握表現不好的區間，經過訓練一個全域的模型後並任一機器學習模型找出這些誤差大的 Patch，透過多個斷點的學習我們會得到 Global Model、Patch1 Model、Patch2 Model…Patch(n) Model。然而在機器學習中我們有很多種方法可以改善我們的模型，例如加深和加寬神經網路或是添加一些非線性的激勵函數來最佳化我們的模型。或是使用整體學習的方法集合許多策略，最終形成一個強學習器並改善某些區域的弱點。下圖為一個簡單的區域學習預測流程圖。在使用區域學習模型前我們要找出該段輸入所對應的 Patch，若該區間剛好落於所劃定的範圍內，這些輸入就將會對應到相對應的區域學習，否則就會使用全域模型進行預測。



區域學習機制包括三個部分，簡要說明如下: (1)使用所有訓練資料訓練一個全域模型; (2)挑出影響錯誤率較高的資料，再放入個別的 Patch 模型進行區域訓練; (3)自訓練資料中去掉已經被局部模型用過的資料，再使用剩下的所有資料更新全域模型。當輸入資料進來時，首先判斷這個輸入是不是在剛剛記下的 Patch 模型中，如果是的話，就執行 Patch 模型。如果不是的話，執行更新後的全域模型。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 14] 多棵決策樹更厲害：隨機森林 (Random forest)

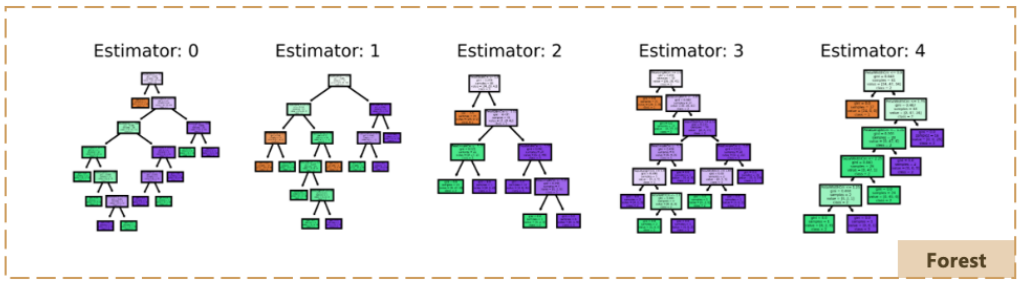
今日學習目標

- 隨機森林介紹
  - 隨機森林的樹是如何生成？隨機森林的優點？
  - 隨機森林如何處理分類問題？
  - 隨機森林如何處理迴歸問題？
- 實作隨機森林分類器
  - 比較隨機森林與決策樹兩者差別。

範例程式　隨機森林(Classification)： Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/14.隨機森林/14.1.隨機森林(Classification-iris).ipynb)* 範例程式　隨機森林(Regression)： Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/14.隨機森林/14.2.隨機森林(Regression).ipynb)*

隨機森林

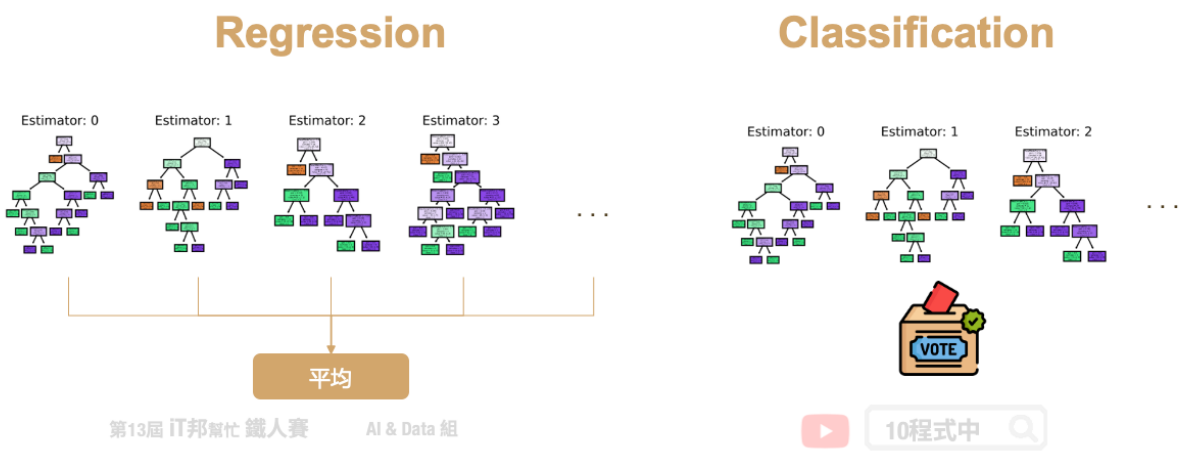
隨機森林其實就是進階版的決策樹，所謂的森林就是由很多棵決策樹所組成。隨機森林是使用 Bagging　加上隨機特徵採樣的方法所產生出來的整體學習演算法。還記得在前幾天的決策樹演算法中，當模型的樹最大深度設定太大的話容易讓模型過擬合。因此隨機森林藉由多棵不同樹的概念所組成，讓結果比較不容易過度擬合，並使得預測能力更提升。



隨機森林的生成方法

首先從訓練集中抽取 n' 筆資料出來，然而這 n' 筆資料是可以被重複抽取的。假設我們有一千筆資料我們要從中抽取 100 筆資料出來，這 100 筆資料裡面可能會有重複的數據。接著第二步從這些抽取出來的資料中挑選 k 個特徵當作決策因子的後選，因此每一棵樹只能看見部分的特徵。第三步重複以上步驟 m 次並產生 m 棵決策樹。透過 Bootstrap 步驟重複 m 次，做完之後我們會有 m 組的訓練資料，每一組訓練資料內都有 n' 筆資料。最後再透過每棵樹的決策並採多數決投票的方式，決定最終預測的類別。因為隨機森林每一棵樹的特徵數量可能都不同，所以最後決策出來的結果都會不一樣。最後再根據任務的不同來做迴歸或是分類的問題，如果是迴歸問題我們就將這些決策數的輸出做平均得到最後答案，若是分類問題我們則用投標採多數決的方式來整合所有樹預測的結果。

1. 從訓練集中抽取 n' 筆資料出來
2. n' 筆資料隨機挑選 k 個特徵做樣本
3. 重複 m 次，產生 m 棵決策樹
4. 分類: 多數投票機制進行預測、迴歸: 平均機制進行預測



隨機森林中的隨機？

隨機森林中的隨機有兩種方面可以解釋。首先第一個是隨機取樣，在模型訓練的過程中每棵樹的生成都會先從訓練集中隨機抽取 n' 筆資料出來，而這 n' 筆資料是可以被重複抽取的。此抽取資料的方式又稱為 Bootstrap，它是一種在統計學上常用的資料估計方法。第二個解釋隨機的理由是在隨機森林中每一棵樹都是隨機的特徵選取。每一棵樹都是從 n' 筆資料中隨機挑選 k 個特徵做樣本。

在 sklearn 中，最多隨機選取 $log_2 N$ 個特徵

隨機森林的優點

- 每棵樹會用到哪些訓練資料及特徵都是由隨機決定
- 採用多個決策樹的投票機制來改善決策樹
- 與決策樹相比，不容易過度擬合
- 隨機森林每一棵樹都是獨立的
- 訓練或是預測的階段每一棵樹都能平行化的運行

[程式實作]

隨機森林(分類器)

Parameters: - n_estimators: 森林中樹木的數量，預設=100。 - max_features: 劃分時考慮的最大特徵數，預設auto。 - criterion: 亂度的評估標準，gini/entropy。預設為gini。 - max_depth: 樹的最大深度。 - splitter: 特徵劃分點選擇標準，best/random。預設為best。 - random_state: 亂數種子，確保每次訓練結果都一樣，splitter=random 才有用。 - min_samples_split: 至少有多少資料才能再分 - min_samples_leaf: 分完至少有多少資料才能分

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。 - get_depth: 取得樹的深度。

```
from sklearn.ensemble import RandomForestClassifier

# 建立 Random Forest Classifier 模型
randomForestModel = RandomForestClassifier(n_estimators=100, criterio
# 使用訓練資料訓練模型
randomForestModel.fit(X_train, y_train)
# 使用訓練資料預測分類
predicted = randomForestModel.predict(X_train)
```

使用Score評估模型

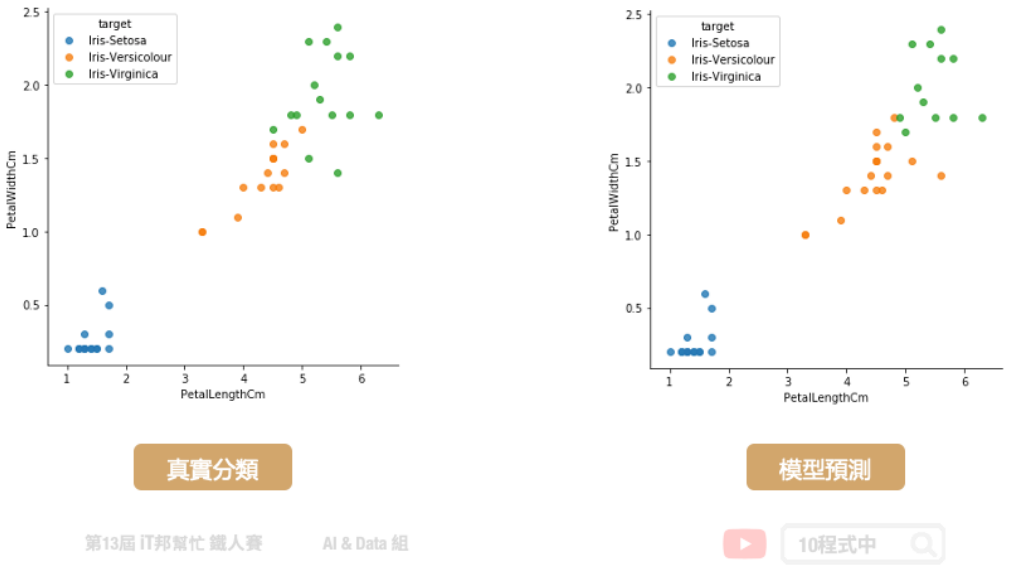
我們可以直接呼叫 `score()` 直接計算模型預測的準確率。

```
# 預測成功的比例
print('訓練集: ',randomForestModel.score(X_train,y_train))
print('測試集: ',randomForestModel.score(X_test,y_test))
```

輸出結果：

```
訓練集:  1.0
測試集:  0.8888888888888888
```

我們可以查看訓練好的模型在測試集上的預測能力，下圖中左邊的是測試集的真實分類，右邊的是模型預測的分類結果。由於訓練資料筆數不多，因此模型訓練容易過度擬合訓練集的分布。最終在測試及預測的表現上僅有 0.88 的準確率。



特徵重要程度

只要是決策樹系列演算法，不管是分類器或是迴歸器都能透過 `feature_importances_` 來檢視模型預測對於特徵的重要程度。

```
print('特徵重要程度: ',randomForestModel.feature_importances_)
```

輸出結果：

```
特徵重要程度:  [0.09864249 0.01363871 0.44211602 0.44560278]
```

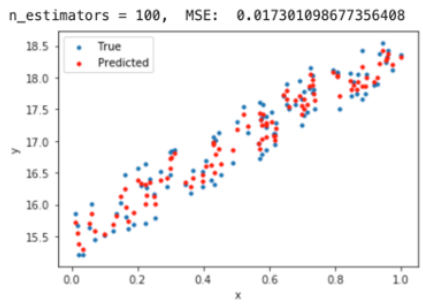
隨機森林(迴歸器)

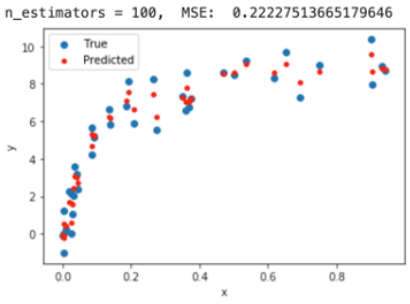
Parameters: - n_estimators: 森林中樹木的數量，預設=100。 - max_features: 劃分時考慮的最大特徵數，預設auto。 - criterion: 評估切割點指標，mse/mae。 - max_depth: 樹的最大深度。 - splitter: 特徵劃分點選擇標準，best/random。預設為best。 - random_state: 亂數種子，確保每次訓練結果都一樣，splitter=random 才有用。 - min_samples_split: 至少有多少資料才能再分 - min_samples_leaf: 分完至少有多少資料才能分

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測。 - score: 預測成功的比例。 - get_depth: 取得樹的深度。

```
from sklearn.ensemble import RandomForestRegressor

# 建立RandomForestRegressor模型
randomForestModel = RandomForestRegressor(n_estimators=100, criterion
# 使用訓練資料訓練模型
randomForestModel.fit(x, y)
# 使用訓練資料預測
predicted=randomForestModel.predict(x)
```





本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 15] 機器學習常勝軍 - XGBoost

今日學習目標

- XGBoost 介紹
  - XGBoost 是什麼？為什麼它那麼強大？
  - XGBoost 優點
- 比較兩種整體學習架構差異？
  - Bagging vs. Boosting
  - Boosting vs. Decision Tree
- Boosting 方法有哪些
- 實作 XGBoost 分類器與迴歸器
  - 比較 Bagging 與 Boosting 兩者差別

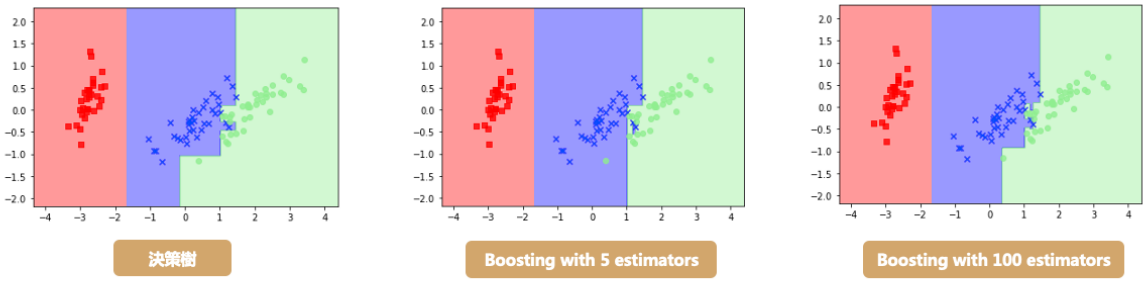
範例程式 XGBoost(Classification)： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/15.XGBoost/15.1.XGBoost(Classification-iris).ipynb) 範例程式 XGBoost(Regression)： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/15.XGBoost/15.2.XGBoost(Regression).ipynb)

人人驚奇的 XGBoost

XGboost 全名為 eXtreme Gradient Boosting，是目前 Kaggle 競賽中最常見到的算法，同時也是多數得獎者所使用的模型。此機器學習模型是由華盛頓大學博士生陳天奇所提出來的，它是以 Gradient Boosting 為基礎下去實作，並添加一些新的技巧。它可以說是結合 Bagging 和 Boosting 的優點。XGboost 保有 Gradient Boosting 的做法，每一棵樹是互相關聯的，目標是希望後面生成的樹能夠修正前面一棵樹犯錯的地方。此外 XGboost 是採用特徵隨機採樣的技巧，和隨機森林一樣在生成每一棵樹的時候隨機抽取特徵，因此在每棵樹的生成中並不會每一次都拿全部的特徵參與決策。此外為了讓模型過於複雜，XGboost 在目標函數添加了標準化。因為模型在訓練時為了擬合訓練資料，會產生很多高次項的函數，但反而容易被雜訊干擾導致過度擬合。因此 L1/L2 Regularization 目的是讓損失函數更佳平滑，且抗雜訊干擾能力更大。最後 XGboost 還用到了一階導數和二階導數來生成下一棵樹。其中 Gradient 就是所謂的一階導數，而 Hessian 即為二階導數。



XGBoost 優點

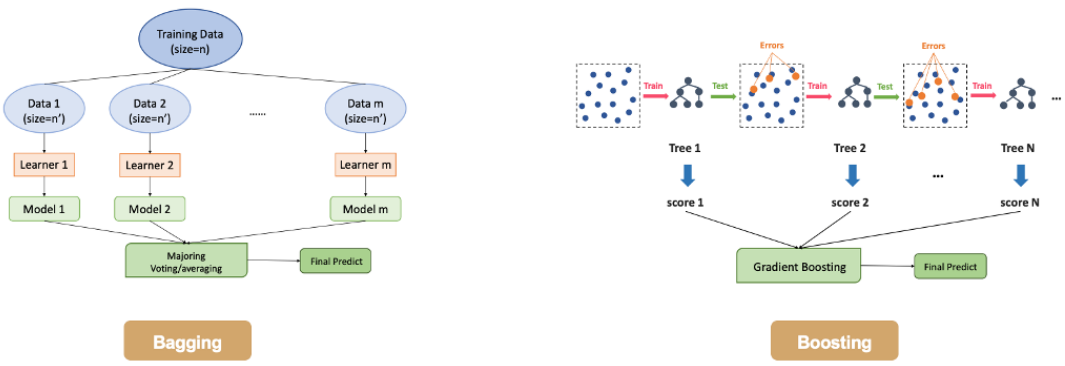
XGBoost 除了可以做分類也能進行迴歸連續性數值的預測，而且效果通常都不差。並透過 Boosting 技巧將許多弱決策樹集成在一起形成一個強的預測模型。

- 利用了二階梯度來對節點進行劃分
- 利用局部近似算法對分裂節點進行優化
- 在損失函數中加入了 L1/L2 項，控制模型的複雜度
- 提供 GPU 平行化運算

Bagging vs. Boosting

在這裡幫大家回顧一下整體學習中的 Bagging 與 Boosting 兩者間的差異。首先 Bagging 透過隨機抽樣的方式生成每一棵樹，最重要的是每棵樹彼此獨立並無關聯。先前所提到的隨機森林就是 Bagging 的實例。另外 Boosting 則是透過序列的方式生成樹，後面所生成的樹會與前一棵樹相關。本章所提及的 XGBoost 就是 Boosting 方法的其中一種實例。正是每棵樹的生成都改善了上一棵樹學習不好的地方，因此 Boosting 的模型通常會比 Bagging 還來的精準。

- Bagging 透過抽樣的方式生成樹，每棵樹彼此獨立
- Boosting 透過序列的方式生成樹，後面生成的樹會與前一棵樹相關



Boosting vs. Decision Tree

我們再與最一開始所提的決策樹做比較。決策樹通常為一棵複雜的樹，而在 Boosting 是產生非常多棵的樹，但是每一棵的樹都很簡單的決策樹。Boosting 希望新的樹可以針對舊的樹預測不太好的部分做一些補強。最終我們要把所有簡單的樹合再一起才能當最後的預測輸出。

Boosting 方法有哪些

AdaBoost 是由 Yoav Freund 和 Robert Schapire 於 1995 年提出。所謂的自適應是表示根據弱學習的學習誤差率表現來更新訓練樣本的權重，然後基於調整權重後的訓練集來訓練第二個弱學習器，藉由此方法不斷的迭代下去。

- AdaBoost（Adaptive Boosting）
  - AdaBoostClassifier *(https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)*
  - AdaBoostRegressor *(https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor)*

Gradient Boosting 由 Friedman 於 1999 年提出。其中 GBDT (Gradient Boosting Decision Tree) 的弱學習器僅限於只能使用 CART 決策樹模型，並採用加法模型的前向分步算法來解決分類和迴歸問題。

- Gradient Boosting
  - GradientBoostingClassifier *(https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)*
  - GradientBoostingRegressor *(https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor*

接下來介紹三個近年三個強大的開源機器學習專案。首先 XGBoost *(https://xgboost.readthedocs.io/en/latest/)* 最初是由陳天奇於 2014 年 3 月發起的一個研究項目，並在短時間內成為競賽中的熱門的模型。接著於 2017 年 1 月微軟發布了第一個穩定的 LightGBM *(https://lightgbm.readthedocs.io/en/latest/)* 版本。它是一個基於 Gradient Boosting 的輕量級的演算法，優點在於使用少量資源、更快的訓練效率得到更好的準確度。另外在同年的 4 月，俄羅斯的一家科技公司 Yandex 發布了 CatBoost *(https://catboost.ai/)*，其核心依然使用了 Gradient Boosting 技巧，並為類別型的特徵做特別的轉換並產生新的數值型特徵。

未來幾天將會介紹 LightGBM 與 CatBoost 哦！

[程式實作]

XGBoost 分類器

Parameters: - n_estimators: 總共迭代的次數，即決策樹的個數。預設值為100。 - max_depth: 樹的最大深度，默認值為6。 - booster: gbtree 樹模型(預設) / gbliner 線性模型 - learning_rate: 學習速率，預設0.3。 - gamma: 懲罰項係數，指定節點分裂所需的最小損失函數下降值。

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。

```
from xgboost import XGBClassifier

# 建立 XGBClassifier 模型
xgboostModel = XGBClassifier(n_estimators=100, learning_rate= 0.3)
# 使用訓練資料訓練模型
xgboostModel.fit(X_train, y_train)
# 使用訓練資料預測分類
predicted = xgboostModel.predict(X_train)
```

使用Score評估模型

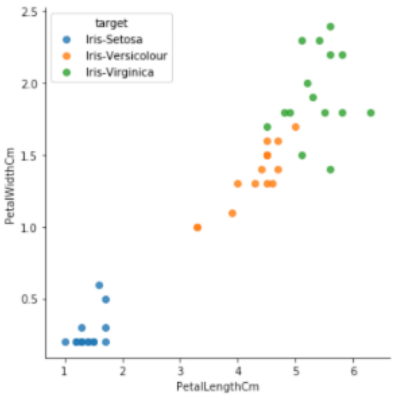
我們可以直接呼叫 `score()` 直接計算模型預測的準確率。

```
# 預測成功的比例
print('訓練集: ',xgboostModel.score(X_train,y_train))
print('測試集: ',xgboostModel.score(X_test,y_test))
```

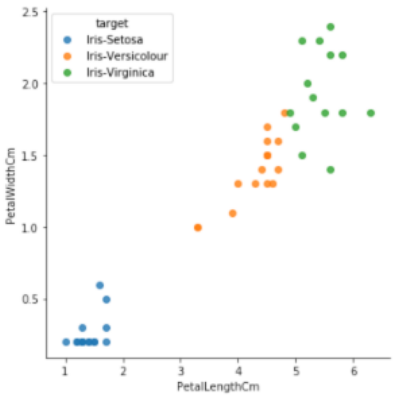
輸出結果：

```
訓練集:  1.0
測試集:  0.9333333333333333
```

大家可以試著與前幾天的決策樹和隨機森林兩個模型相比較。是不是 XGBoost 有著更好的預測結果呢？因為有了 Gradient Boosting 學習機制，大幅提升了預測能力。在學習過程中將預測不好的地方，尤其是橘色 (Versicolour) 與綠色 (Virginica) 交界處有更好的評估能力。



真實分類



模型預測

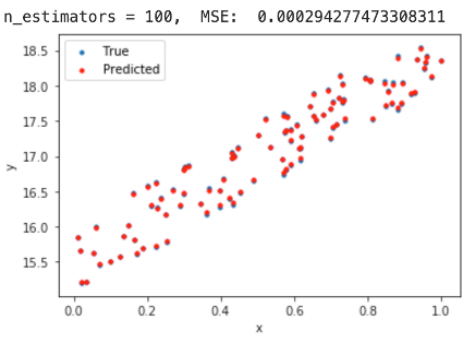
XGBoost (迴歸器)

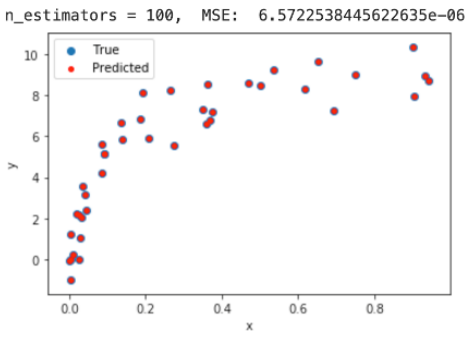
Parameters: - n_estimators: 總共迭代的次數, 即決策樹的個數。預設值為100。 - max_depth: 樹的最大深度, 默認值為6。 - booster: gbtree 樹模型(預設) / gbliner 線性模型 - learning_rate: 學習速率, 預設0.3。 - gamma: 懲罰項係數, 指定節點分裂所需的最小損失函數下降值。

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。

```
import xgboost as xgb

# 建立 XGBRegressor 模型
xgbrModel=xgb.XGBRegressor()
# 使用訓練資料訓練模型
xgbrModel.fit(x,y)
# 使用訓練資料預測
predicted=xgbrModel.predict(x)
```





Reference

- XGboost入門經驗分享-超參數解析 *(https://medium.com/@pahome.chen/xgboost%E5%85%A5%E9%96%80%E7%B6%93%E9%A9%97%E5%88%86%E4%BA%AB-e06931b835f5)*
- 關於 XGBoost 20 個 FAQ *(https://towardsdatascience.com/20-burning-xgboost-faqs-answered-to-use-the-library-like-a-pro-f8013b8df3e4)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 16] 每個模型我全都要 - 堆疊法 (Stacking)

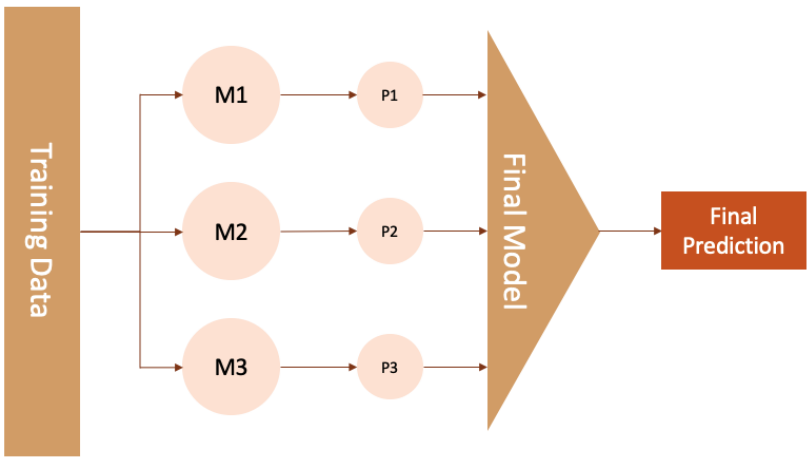
今日學習目標

- 了解 Stacking 方法
  - 堆疊法的學習機制為何？
- 利用 Stacking 實作迴歸器
  - 透過 Stacking Regressor 建立房價預測模型

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/16.Stacking/16.house-price-prediction-stacking.ipynb)*

前言

堆疊法 (Stacking) 是整體學習中其中一種實例。它是結合許多獨立的模型所預測出來的結果，並將每個獨立模型的輸出視為最終模型預測的輸入特徵，最後再訓練一個最終模型。以下圖為例，假設我們事先訓練三個基底的模型 (base learner)，這三個模型彼此互相無關連。由於每一個模型所訓練出來的預測能力都不同，也許模型一在某個區段的資料有不太好的預測能力，而模型二能補足模型一預測不好的地方。藉由上述這個觀點我們將三個訓練好的模型輸出集合起來(P1、P2、P3)，如果是分類問題可以透過投票方式，而迴歸問題可以採用平均法或是加權平均法將所有的預測做最後評估。又或者是可以將這三個輸出值當作是新模型的特徵再丟入一個機器學型模型做最後的預測得到最終輸出。



[程式實作]

在此範例中我們透過 Sklearn 所提供的波士頓房價預測資料集進行 Stacking 方法建模。並觀察同一組資料在單一模型下預測，與加入 Stacking 機制後的結果有無改善。

1) 載入資料集

首先我們夠過 Sklearn 套件讀入波士頓房價資料集，並將輸入特徵與房價合併成一個 DataFrame。在此資料集中總共有 13 個輸入特徵，以及一個輸出 MEDV 即為房價。

```
# load boston_dataset
boston_dataset = load_boston()
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.fea
boston['MEDV'] = boston_dataset.target
boston
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

2) 切割訓練集與測試集

在此範例中我們著重於比較模型的差異，因此沒有按照正常的機器學習流程走。資料視覺化以及前處理...等是非常重要的哦！在此步驟我們快速物的將乾淨的資料切出訓練集與測試集，其中訓練集 X_train 與 y_train 是實際參與行訓練的資料。而 X_test 與 y_test 是未參與訓練的資料，它是被拿來測試評估最終訓練好的模型。

```
from sklearn.model_selection import train_test_split
X  = boston.drop(['MEDV'],axis=1).values
y = boston[['MEDV']].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

```
print('Training data shape:', X_train.shape)
print('Testing data shape:', X_test.shape)
```

由於 Sklearn 資料集提供的資料樣本數比較少，因此測試集僅切出 0.1 的資料。

執行結果：

```
Training data shape: (455, 13)
Testing data shape: (51, 13)
```

XGBoost 模型

因為要與 Stacking 做一個比較。因此這裡使用 XGBoost 先訓練一個模型，並將結果與 Stacking 做比較。

```
from xgboost import XGBRegressor

# 建立 XGBRegressor 模型
xgboostModel = XGBRegressor()
# 使用訓練資料訓練模型
xgboostModel.fit(X_train, y_train)
# 使用訓練資料預測
predicted = xgboostModel.predict(X_train)

print("訓練集 Score: ", xgboostModel.score(X_train,y_train))
print("測試集 Score: ", xgboostModel.score(X_test,y_test))
```

從預測結果我們先來查看 R2 score，一切看似還 ok。不過這裡要呼籲各位讀者絕不要看 R2 分數高就高興得太早！

執行結果：

```
訓練集 Score:  0.9999920949016282
測試集 Score:  0.9292786904177338
```

我們來看一下 MSE 實際算一下訓練集與測試集的誤差。可以發現很明顯的過度擬合了，簡單來說在訓練集的資料算出來的 MSE 很小，但是在測試集中 MSE 預測能力不足造成誤差變大。

```
from sklearn import metrics

# 訓練集 MSE
train_pred = xgboostModel.predict(X_train)
mse = metrics.mean_squared_error(y_train, train_pred)
print('訓練集 MSE: ', mse)
```

```
# 測試集 MSE
test_pred = xgboostModel.predict(X_test)
mse = metrics.mean_squared_error(y_test, test_pred)
print('測試集 MSE: ', mse)
```

執行結果：

```
訓練集 MSE:  0.0006847746512112584
測試集 MSE:  4.415429632025227
```

Stacking 模型

Stacking 結合許多弱學習器，將所有的弱學習器的輸出當作新的模型的輸入接著預測最終結果。在此範例中我們建立了四種迴歸器，分別有隨機森林、支持向量機、KNN 與決策樹。最終的模型我們採用兩層隱藏層的神經網路作為最後的房價預測評估模型。

Parameters: - estimators: m 個弱學習器。 - final_estimator: 集合所有弱學習器的輸出，訓練一個最終預測模型。預設為LogisticRegression。

Attributes: - estimators_: 查看弱學習器組合。 - final_estimator: 查看最終整合訓練模型。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.neural_network import MLPRegressor


estimators = [
    ('rf', RandomForestRegressor(random_state = 42)),
    ('svr', svm.SVR()),
    ('knn', KNeighborsRegressor()),
    ('dt', DecisionTreeRegressor(random_state = 42))
]
clf = StackingRegressor(
    estimators=estimators, final_estimator= MLPRegressor(activation =
                           learning_rate = "constant", max_iter = 20
)

clf.fit(X_train, y_train)
```

```
print("訓練集 Score: ", clf.score(X_train,y_train))
print("測試集 Score: ", clf.score(X_test,y_test))
```

我們先觀察訓練後的 R2 score 在訓練集與測試集上的分數。從數值看觀察可以發現透過堆疊法兩者間的分數差距變小了。

輸出結果：

```
訓練集 Score:  0.9608703782891547
測試集 Score:  0.9371735287625855
```

```
from sklearn import metrics

# 訓練集 MSE
train_pred = clf.predict(X_train)
mse = metrics.mean_squared_error(y_train, train_pred)
print('訓練集 MSE: ', mse)
# 測試集 MSE
test_pred = clf.predict(X_test)
mse = metrics.mean_squared_error(y_test, test_pred)
print('測試集 MSE: ', mse)
```

接著我們一樣計算 MSE 實際觀察模型在訓練集與測試集上的誤差。從計算解果可以看到兩者的誤差都是差不多的。從這裡我們就可以很清楚的知道透過 Stacking 可以避免模型過擬合，並且透過多個基底的模型讓最終預測結果有比較平滑的輸出。

輸出結果：

```
訓練集 MSE:  3.389581229598408
測試集 MSE:  3.9225215768179433
```

> 本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 17] 輕量化的梯度提升機 - LightGBM

今日學習目標

- LightGBM 與 XGBoost 比較
- 了解 LightGBM 優點
- 實作 LightGBM 處理資料不平衡資料
    - 信用卡盜刷偵測 (二元分類)

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/17.LightGBM/17.creditcard-fraud-detection-lightgbm.ipynb)*

前言

LightGBM 是屬於 GBDT 家族中成員之一，相較於先前介紹的 XGBoost 兩者可以拿來做比較。簡單來說從 LightGBM 名字上觀察，我們可以看出它是輕量化 (Light) 的梯度提升機 (GBM) 的實例。其相對 XGBoost 來說它具有訓練速度快、記憶體佔用低的特點，因此近幾年 LightGBM 在 Kaggle 上也算是熱門模型一。

LightGBM 與 XGBoost 比較

這兩種演算法都使用貪婪的方法來最小化損失函數的梯度來構建所有的弱學習器。其 tree-based 演算法所面臨的挑戰是如何挑選最佳的葉節點的切割方式，然而 LightGBM 和 XGBoost 分別使用不同的優化技術與方法來識別最佳的分割點。

LightGBM 優點

LightGBM 由微軟團隊於 2017 年所發表的論文 LightGBM: A Highly Efficient Gradient Boosting Decision Tree *(https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf)* 被提出。其主要想法是利用決策樹為基底的弱學習器，不斷地迭代訓練並取

得最佳的模型。同時該演算法進行了優化使得訓練速度變快，並且有效降被消耗的資源。LightGBM 也是個開源專案大家可以在 GitHub *(https://github.com/microsoft/LightGBM)* 上可以取得相關資訊。

在官方的文件中也條列了幾個 LightGBM 的優點： - 更快的訓練速度和更高的效率 - 低記憶體使用率 - 更好的準確度 - 支援 GPU 平行運算 - 能夠處理大規模數據

LightGBM 使用 leaf-wise tree 演算法，因此在迭代過程中能更快地收斂。但是 leaf-wise tree 方法較容易過擬合。詳細的內容可以參考文章最後提供的相關資源。

處理 unbalance 資料

在使用 LightGBM 做分類器時該如何處理樣本類別分佈不平衡的問題？一個簡單的方法是設定 `is_unbalance=True`，或是 `scale_pos_weight` 注意這兩個參數只能擇一使用。以下我們就使用一個不平衡的資料集，信用卡盜刷預測來做示範。首先我們可以載入 Google 所提供的信用卡盜刷資料集，詳細資訊可以參考這裡 *(https://www.tensorflow.org/tutorials/structured_data/imbalanced_data)*。

```
import pandas as pd
raw_df = pd.read_csv('https://storage.googleapis.com/download.tensorf
X=raw_df.drop(columns = ['Class'])
y=raw_df['Class']
print('X:', X.shape)
print('Y:', y.shape)
```

載入成功後我們可以看到該資料集共有 284807 筆資料，每一筆資料有 30 個特徵。

```
X: (284807, 30)
Y: (284807,)
```

為了方便檢視實驗結果，我們依照 y 的比例進行訓練集與測試集的切割。這裡值得一提的是，`stratify` 為分層隨機抽樣。特別是在原始數據中樣本標籤分佈不均衡時非常有用，一些分類問題可能會在目標類的分佈中表現出很大的不平衡時例如：負樣本可能比正樣本多幾倍。在這種情況下，建議使用分層抽樣。

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
print('X_train:', X_train.shape)
print('X_test:', X_test.shape)
```

輸出結果：

```
X_train: (199364, 30)
X_test: (85443, 30)
```

訓練集與測試集經由 7:3 的比例下去隨機切割資料。我們可以透過 Pandas 做更近一步的分析，可以發現切割出來的訓練集與測試集在盜刷(1)與非盜刷(0)的資料比例是差不多的。

```
# 查看訓練集的分佈
pd.Series(y_train).value_counts(normalize=True)
```

```
0    0.998275
1    0.001725
Name: Class, dtype: float64
```

```
# 查看測試集的分佈
pd.Series(y_test).value_counts(normalize=True)
```

```
0    0.998268
1    0.001732
Name: Class, dtype: float64
```


接下來重頭戲出場。我們採用 LightGBM 分類器，若還沒安裝的讀者可以參考以下指令進行安裝。

```
pip install lightgbm
```

安裝結束後即可載入 lightgbm 套件並選用 LGBMClassifier 分類器。另外我們可以在建立分類器同時設定模型超參數，這裡我們來示範使用 `is_unbalance=True` 訓練模型。除此之外模型的超參數有很多，可以由官方 *(https://lightgbm.readthedocs.io/en/latest/Parameters.html)* 文件中查閱。以下幫各位整理常用的方法：

Parameters: - num_iterations: 總共迭代的次數，即決策樹的個數。預設值為100。 - learning_rate: 學習速率，預設0.1。 - boosting: 選擇 boosting 種類。共四種 gbdt、rf、dart、goss，預設為 gbdt。 - max_depth: 樹的最大深度，預設值為-1即表示無限制。 - min_data_in_leaf: 一個子葉中最少數據，可用於處理過擬合。預設20筆。 - max_bin: 將特徵值放入桶中的最大bins數。預設為255。

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。

```
import lightgbm as lgb
model = lgb.LGBMClassifier(is_unbalance=True)
model.fit(X_train,y_train)
```

訓練結束後即可使用剛切割好的測試集進行模型評估。我們可以發現準確率高達 94%。

```
from sklearn.metrics import accuracy_score
pred=model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, pred))
```

輸出結果：

```
Accuracy: 0.9401706400758401
```

如果要判斷分類器的好壞，僅使用準確率來評估是一個不好的習慣。我們應該善用混淆矩陣做更近一步的分析，並查看正樣本與負樣本在預測上的表現。首先我們先來寫一個計算混淆矩陣的函式，並用 seaborn 繪製出熱力圖矩陣。

```
import seaborn as sns
import matplotlib.pyplot as plt
def plot_confusion_matrix(actual_val, pred_val, title=None):
    confusion_matrix = pd.crosstab(actual_val, pred_val,
                                   rownames=['Actual'],
                                   colnames=['Predicted'])

    plot = sns.heatmap(confusion_matrix, annot=True, fmt=',.0f')

    if title is None:
        pass
    else:
        plot.set_title(title)

    plt.show()
```

在評估模型之前我們先來查看測試集輸出 y 的分佈各是多少。透過 numpy 的 unique 方法可以計算 y_test 中每個類別的數量。從輸出結果可以得知，85443 筆測試集中共有 85295 筆是標籤 0(未盜刷)、148 筆是標籤 1(盜刷)。知道這些真實數據的數量後，接下來我們就可以透過混淆矩陣來查看模型是否有將這些盜刷的資料被正確預測出來。

```
import numpy as np
unique, counts = np.unique(y_test, return_counts=True)
dict(zip(unique, counts))
```

輸出結果：

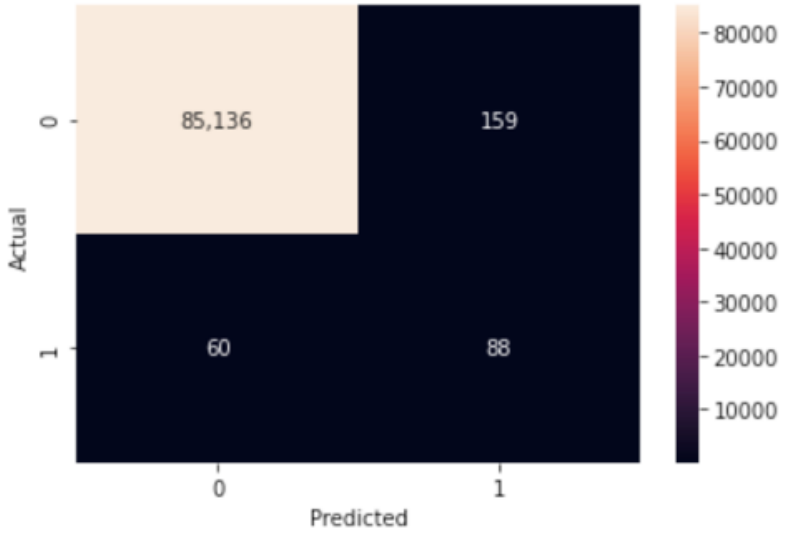
```
{0: 85295, 1: 148}
```

plot_confusion_matrix 函式建立完成後即可呼叫。此函式有三個輸入，分別為 y_test 實際輸出答案、 pred 模型預測結果、title 圖表標題(預設None)。相對應的變數輸入後即可得到計算好的混淆矩陣。

```
plot_confusion_matrix(y_test, pred)
```

下圖為實際 is_unbalance=True 的訓練結果。我們可以發現在測試集中有 148 筆盜刷資料，其中有 124 筆盜刷被成功辨識出來。另外我們可以發現真實答案是沒盜刷的資料居然有 5088 筆被誤判成盜刷。

我們再來試試將 is_unbalance 設為 False 並觀察混淆矩陣。可以發現雖然誤判的數量減少了，但是真實答案中有 148 筆盜刷資料僅有 88 筆被成功辨識出來。我們可以猜想模型在大多數狀況都會預測資料未被盜刷的機率較大。



Reference

- 終於有人把XGBoost 和 LightGBM 講明白了，項目中最主流的集成演算法！ *(https://codingnote.cc/zh-tw/p/22596/)*
- Lightgbm基本原理介紹 *(https://www.twblogs.net/a/5baa44f32b717750855c8ac6)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 18] 機器學習 boosting 神器 - CatBoost

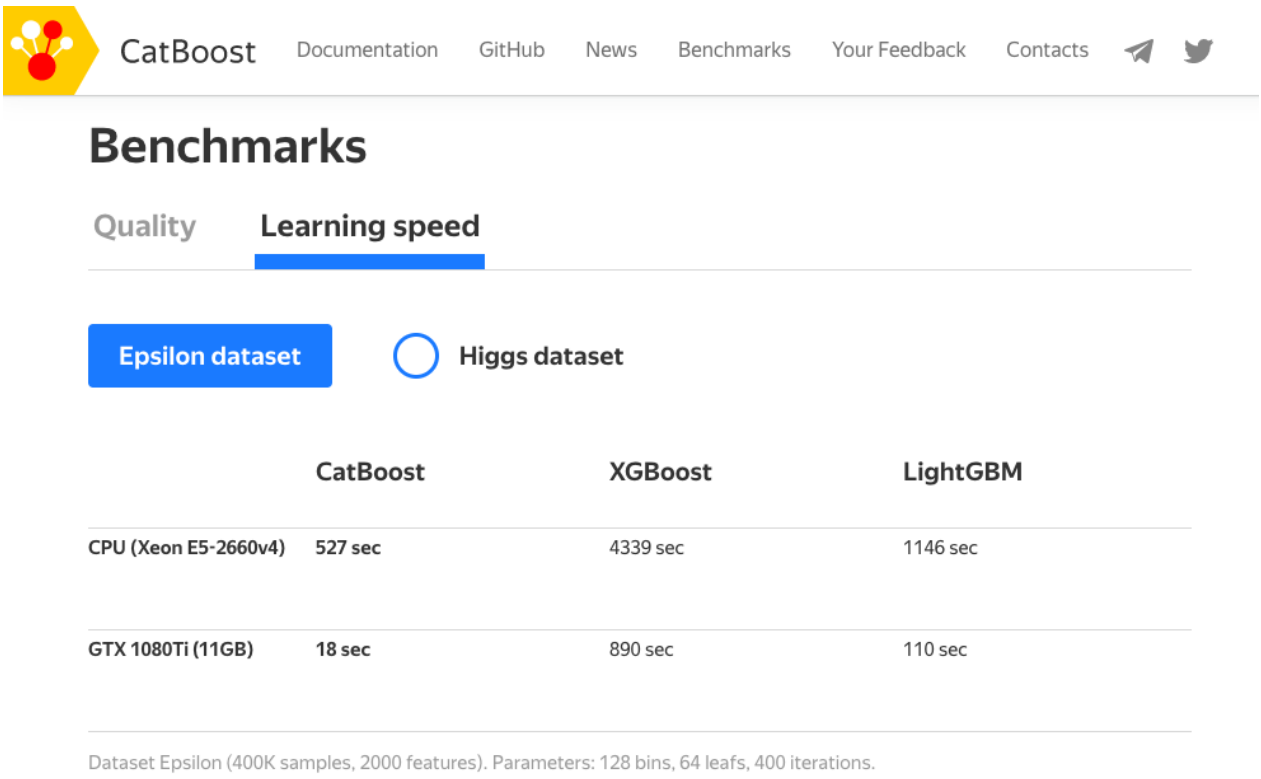
今日學習目標

- 了解 CatBoost 模型
- 實作 CatBoost 迴歸模型-房價預測
    - 模型訓練、特徵篩選
    - 超參數搜索
    - 自動處理類別型的特徵
    - 可解釋化模型

範例程式：  *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/18.CatBoost/18.CatBoost(house_prices).ipynb)*

前言

CatBoost 同樣是基於 Gradient Boosting Tree 的梯度提升樹模型框架，最大的特點對類別特徵的直接支援，甚至允許字串類型的特徵進行模型訓練。近年來在 Kaggle 上的比賽陸續有人使用 CatBoost 方法並取得不錯的成績，於是就來撰寫文章順便來瞧瞧它與其他 Boosting 演算法不同之處。其中最特別的地方是 CatBoost 能夠處理非數值型態的資料，也就是說無需對數據特徵進行任何的預處理就可以將類別轉換爲數字。CatBoost 採用決策樹梯度提升方法並宣稱在效能上比 XGBoost 和 LightGBM 更加優化，同時支援 CPU 和 GPU 運算。與其他 Boosting 方法相比 CatBoost 是一種相對較新的開源機器學習算法。該演算法是由一間俄羅斯的公司 Yandex 於 2017 年所提出，同時在 arxiv 有一篇 CatBoost: unbiased boosting with categorical features *(https://arxiv.org/pdf/1706.09516.pdf)* 的論文，文中作者詳細說明了 CatBoost 的方法與優點。



CatBoost 優點

CatBoost 名稱源於 Category 和 Boost 兩個單詞，承襲 Boosting 的優點之外該演算法在類別型的特徵上做了一些更公平的特徵工程。訓練過程中允許沒有編碼的類別特徵，透過分類和數字特徵組合的各種統計量為類別型的特徵做編碼。不過在訓練前必須確保該特徵中無缺失值。其訓練資料若有缺失值 CatBoost 預設會將數值型的資料補上最小值，詳細內容可以參考 *(https://catboost.ai/docs/concepts/algorithm-missing-values-processing.html#numerical-features)*。另外對於 GPU 的使用者，它也能處理字串(類別)型態的特徵。

- 自動處理類別型的特徵
- 自動處理缺失值
- 可以處理各種數據類型，如音頻、文字、圖像
- 減少人工調參的需要，並降低了過擬合的機會

CatBoost 安裝

CatBoost 演算法可以解決分類 (CatBoostClassifier) 和迴歸 (CatBoostRegressor) 的問題。安裝的方式也非常簡單，使用 `pip` 就能輕鬆安裝。

```
pip install catboost
```

CatBoost Parameters

CatBoost 基本上可以自由的讓演算法去選擇最佳的模型，不過 API 還是提供一些基本的超參數讓使用者手動調整。

Parameters: - iterations: 總共迭代的次數，即決策樹的個數。預設值為 1000。 - use_best_model: 設定 True 時必須給定驗證集，將會留下驗證集分中數最高的模型。 - early_stopping_rounds: 連續訓練N代，若結果未改善則提早停止訓練。 - od_type: IncToDec/Iter，預設 Iter 防止 Overfitting 評估方式，若設定前者需要設定閥值。 - eval_metric: 模型評估方式。 - loss_function: 計算loss方法。 - verbose: True(1)/Flase(0)，預設1顯示訓練過程。 - random_state: 亂數種子，確保每次訓練結果都一樣。 - learning_rate: 預設 automatically。 - depth: 樹的深度，預設 6。 - cat_features: 輸入類別特徵的索引，它會自動幫你處理。

參考 (https://catboost.ai/docs/concepts/python-reference_parameters-list.html)

Attributes: - feature_importances_: 查詢模型特徵的重要程度。

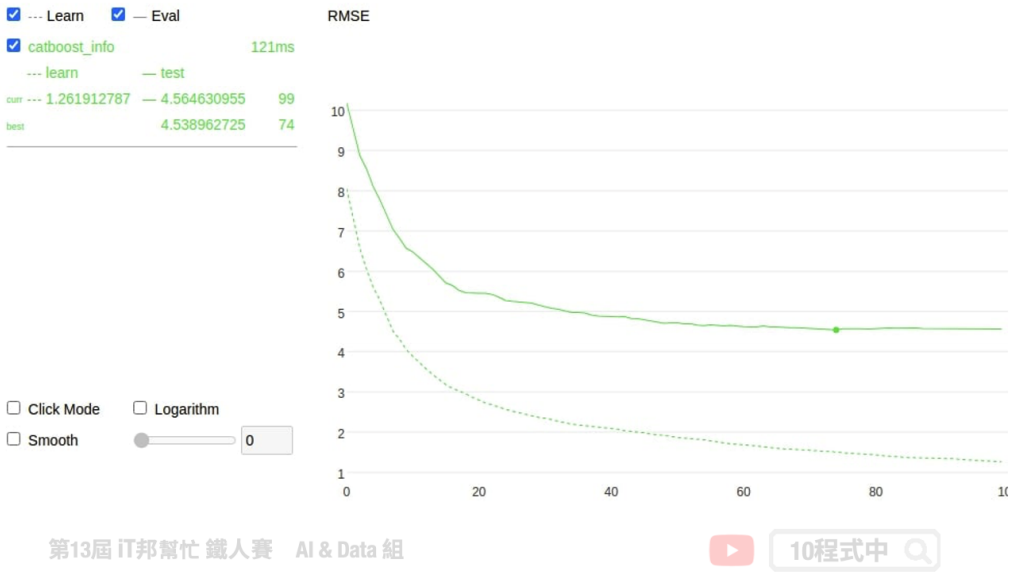
Methods: - fit: 放入X、y進行模型擬合。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。

如果需要手動處理 Overfitting 問題可以參考這份官方文件 *(https://catboost.ai/docs/features/overfitting-detector-desc.html)*

模型訓練

模型訓練方式基本上與 XGBoost 一樣，如果你熟悉 sklearn 的話 CatBoost 的使用方式基本上大同小異。只不過在 CatBoost 中多了一些方便的方法和參數可以使用。像是在訓練過程中可以加上 `plot=True`，並在 eval_set 參數中插入測試集可以即時看到訓練過程的視覺化分析。甚至可以使用交叉驗證，在不同的分割上觀察模型準確度的平均和標準偏差。

```
from catboost import CatBoostRegressor

# 建立模型
model = CatBoostRegressor(random_state=42,
                          loss_function='RMSE',
                          eval_metric='RMSE',
                          use_best_model=True)
# 使用訓練資料訓練模型
model.fit(X_train,y_train, eval_set=(X_test, y_test), verbose=0, plot
```



特徵篩選

訓練過程中會自動從資料中篩選對模型預測有用的特徵，並移除無幫助預測的特徵。

- 參考 *(https://catboost.ai/docs/concepts/python-reference_catboost_select_features.html)*

```
from catboost import CatBoostRegressor, Pool, EShapCalcType, EFeature

# feature_names = ['F{}'.format(i) for i in range(X_train.shape[1])]
train_pool = Pool(X_train, y_train, feature_names=boston_dataset.feat
test_pool = Pool(X_test, y_test, feature_names=boston_dataset.feature

model = CatBoostRegressor(random_state=42,
                          loss_function='RMSE',
                          eval_metric='RMSE',
                          use_best_model=True)
summary = model.select_features(
    train_pool,
    eval_set=test_pool,
    features_for_select='0-12',
    num_features_to_select=3,
    steps=2,
    algorithm=EFeaturesSelectionAlgorithm.RecursiveByShapValues,
    shap_calc_type=EShapCalcType.Regular,
    train_final_model=True,
    logging_level='Silent',
    plot=False
)
summary
```

由於在訓練將 num_features_to_select 設為三，即表示模型訓練時會拿取三個最重要特徵當作做中模型預測方式。我們採用 sklearn 的房價預測資料集，結果可以發現三個最重要特徵為 ['RM', 'PTRATIO', 'LSTAT']。如果你有做 EDA 可以發現這三個特徵與房價的關聯性都很高。

```
{'selected_features': [5, 10, 12],
 'eliminated_features_names': ['DIS',
  'B',
  'ZN',
  'CHAS',
  'RAD',
  'INDUS',
  'CRIM',
  'AGE',
  'TAX',
  'NOX'],
 'eliminated_features': [7, 11, 1, 3, 8, 2, 0, 6, 9, 4],
 'selected_features_names': ['RM', 'PTRATIO', 'LSTAT']}
```

Grid search

除此之外 CatBoost 提供對模型的指定參數值進行簡單的網格搜索，如果有使用過 sklearn 的 Grid Search 其實他就是一樣的使用方式。

- *參考 (https://catboost.ai/docs/concepts/python-reference_catboostregressor_grid_search.html)*

```
from catboost import CatBoostRegressor
grid = {'iterations': [100, 150, 200],
        'learning_rate': [0.03, 0.1],
        'depth': [2, 4, 6, 8],
        'l2_leaf_reg': [0.2, 0.5, 1, 3]}

model = CatBoostRegressor(random_state=42,
                          loss_function='RMSE',
                          eval_metric='RMSE')
model.grid_search(grid, X_train,y_train)
```

自動處理類別型的特徵

CatBoost 無需對數據特徵進行任何的預處理就可以將類別轉換爲數字。下面程式為一個分類問題的範例，其中輸入特徵的第一個為季節。在機器學習上的認知我們必須將所以字串型資料必須透過標籤編碼方式轉換成數值，然而在 CatBoost 完全不需要。僅需在訓練模型時給予參數

`cat_features = [0]` 即代表資料的第一個特徵需要進行類別轉換。另外輸出葉不一定要編碼後的結果，你也可以丟入文字進行訓練只要加上 `loss_function='MultiClass'` 即可。

```
from catboost import Pool, CatBoostClassifier

train_data = [["summer", 1924, 44],
              ["summer", 1932, 37],
              ["winter", 1980, 37],
              ["summer", 2012, 204]]

eval_data = [["winter", 1996, 197],
             ["winter", 1968, 37],
             ["summer", 2002, 77],
             ["summer", 1948, 59]]

train_label = ["France", "USA", "USA", "UK"]
eval_label = ["USA", "France", "USA", "UK"]


# Initialize CatBoostClassifier
model = CatBoostClassifier(iterations=10,
                           learning_rate=1,
                           depth=2,
                           cat_features = [0],
                           loss_function='MultiClass')
# Fit model
model.fit(train_data, train_label)
# Get predicted classes
preds_class = model.predict(eval_data)
# Get predicted probabilities for each class
preds_proba = model.predict_proba(eval_data)
# Get predicted RawFormulaVal
preds_raw = model.predict(eval_data,
                          prediction_type='RawFormulaVal')
```

參考 (https://catboost.ai/docs/concepts/python-usages-examples.html)

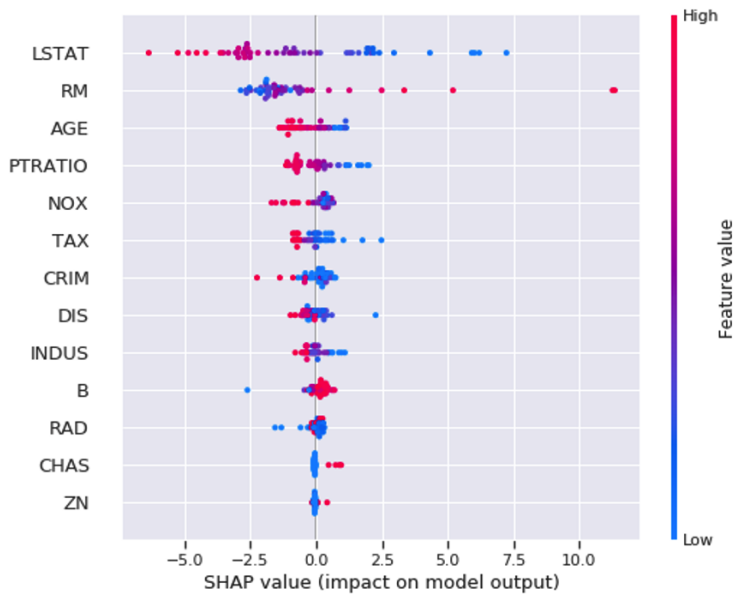
善用 Verbose

訓練過程中可以隨時觀察訓練集與測試集的loss，使用verbose=10即代表每10次迭代會顯示一次資訊，這種方式也解決每次疊代顯示一次的困擾。訓練過程中剩餘時間也會顯示出來。

```
[*]: from catboost import CatBoostRegressor

     # 建立模型
     model = CatBoostRegressor(random_state=42,
                               loss_function='RMSE',
                               eval_metric='RMSE',
                               use_best_model=True)
     # 使用訓練資料訓練模型
     model.fit(X_train,y_train, eval_set=(X_test, y_test), verbose=10, plot=False)

     Custom logger is already specified. Specify more than one logger at same time is not threa
     d safe.
     Learning rate set to 0.226387
     0:      learn: 7.5323882        test: 6.6107054 best: 6.6107054 (0)     total: 147ms    re
     maining: 2m 27s

[ ]:
```


模型的解釋

CatBoost 提供了 `plot` 可以方便在訓練時查看並作即時分析訓練趨勢。除此之外 CatBoost 也支援 SHAP 增加了模型可解釋。詳細的使用方式可以參考官方教學 *(https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/tree_based_models/Catboost%20tutorial.html)*。



小結

CatBoost 的優點和使用方法都介紹完了，是不是覺得十分簡單易用且功能強大。尤其是遇到資料需要大量處理和特徵數值化的任務時最適合使用 CatBoost 了。真的是所謂的懶人套件，名符其實的 Ying Train Yi Fa(硬Train一發)!

Reference

- Tutorial: CatBoost Overview *(https://www.kaggle.com/mitribunskiy/tutorial-catboost-overview)*
- SHAP Catboost tutorial *(https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/tree_based_models/Catboost%20tutorial.html)*
- CatBoost regression in 6 minutes *(https://towardsdatascience.com/catboost-regression-in-6-minutes-3487f3e5b329)*
- Catboost：超越Lightgbm和XGBoost的又一個boost算法神器 *(https://www.xuehua.us/a/5ebf84b0fdbefd9ba19e8d2f?lang=zh-tw)*
- CatBoost、LightGBM、XGBoost，這些算法你都瞭解嗎？ *(https://www.xuehua.us/a/5eb594cf86ec4d63e698d3e7?lang=zh-tw)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

第三部分 進階概念與應用

[Day 19] 自動化機器學習 - AutoML

今日學習目標

- 了解何謂 AutoML
- 超參數調參方法
    - Grid Search
    - Random Search
    - Bayesian Optimization

AutoML 的動機

大家還記得在 [Day 5] 機器學習大補帖 *(https://ithelp.ithome.com.tw/articles/10265942)* 中有提到完整的機器學習流程大致分成八個步驟。然而模型的訓練與超參數調整僅扮演其中的一環，選擇一個好的模型是件重要的事情。想必大家在訓練模型時一定會遇到一個棘手的問題，就是該如何正確選擇模型以及調整超參數？隨著越來越多的演算法不斷地被開發出來，要從茫茫大海中挑選一個合適的模型是件耗時的事。因此自動化機器學習 (Automated Machine Learning ,AutoML) 可以幫助我們在有限的時間內找出一個滿意的模型。在近年來有許多人開始研究這類的問題，筆者彙整了幾個 Python 熱門的 AutoML 開源套件：

- AutoGluon *(https://www.automl.org/automl/#:~:text=AutoML%20packages%20include%3A-,AutoGluon,-is%20a%20multi)*
- Auto-sklearn *(https://automl.github.io/auto-sklearn/master/)*
- FLAML *(https://github.com/microsoft/FLAML)*
- H2O AutoML *(http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html)*
- LightAutoML *(https://github.com/sberbank-ai-lab/LightAutoML)*
- Pycaret *(https://pycaret.org/)*
- MLJAR *(https://mljar.com/)*
- TPOT *(https://github.com/EpistasisLab/tpot)*
- MLBox *(https://github.com/AxeldeRomblay/MLBox)*
- Auto-PyTorch *(https://github.com/automl/Auto-PyTorch)*
- AutoKeras *(https://autokeras.com/)*
- talos *(https://github.com/autonomio/talos)*

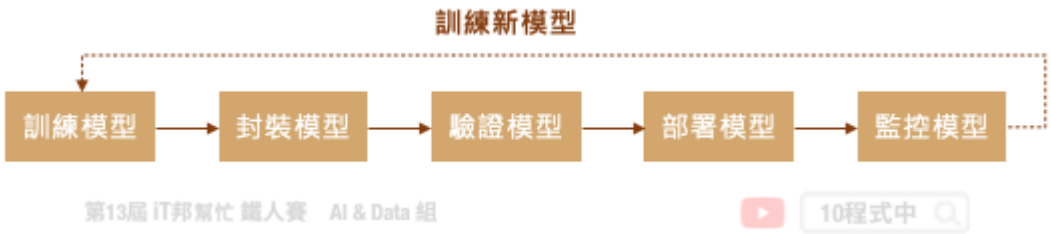
AutoML 扮演的角色

自動化機器學習提供了一系列的方法和自動化的學習流程，以提高機器學習的效率並加速機器學習的研究。透過 AutoML 集結專家的先驗知識，大幅降低了機器學習建模的困難度。雖然領域專家與 AI 工程師必然扮演重要的角色，但是近年來 No Code 無程式碼開發平台形成一股潮流。AI 再也不是需要資訊背景的人才能做的事，目的是讓大家不用透過寫程式也能快速地進行資料探索與建立預測模型。然而近年來許多企業開發了各種需求的 AutoML 平台，如雨後春雨般的出現：

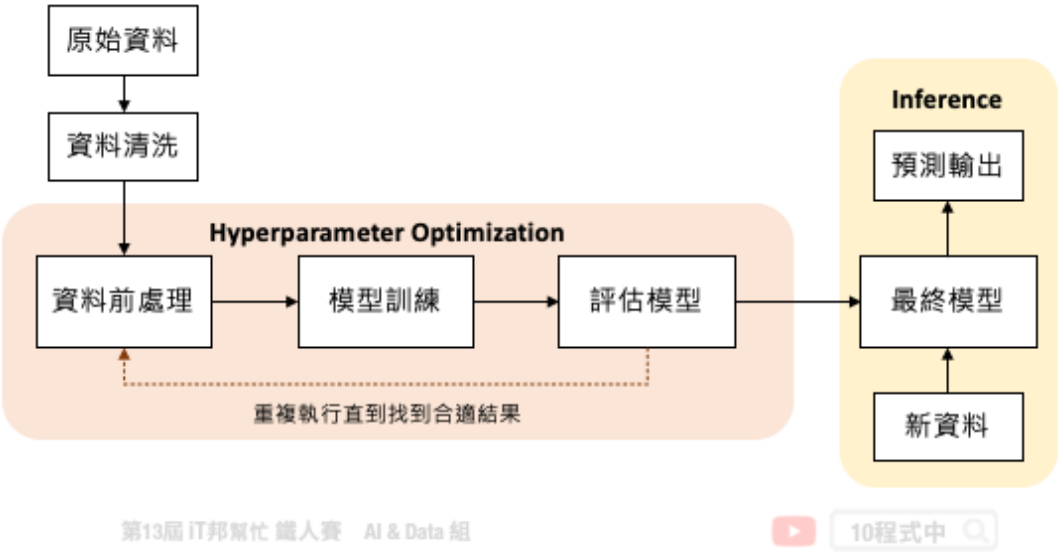
- Google: Cloud AutoML
- Microsoft: Azure Machine Learning
- Amazon: SageMaker Autopilot
- Landing AI: LandingLens
- Chimes AI: tukey

AutoML 能幫助多少事情

典型的機器學習流程是一個迭代的循環週期，從定義問題、資料收集與處理、模型設計到最終模型部署，每個步驟極為重要且缺一不可。此外一個好的機器學習的專案需要執行 MLOps 的流程，才能夠讓模型在實際應用場景越來越好真實地解決問題。MLOps 指的是從 AI 模型訓練到部署上線的一套完整機器學習工作流程，近年來這一名詞非常熱門，它其實就是 ML (機器學習) 與 DevOps (開發與維運) 的結合。如下圖所示從訓練模型到正式部署中間還有許多事情要處理，而模型上線後還是要持續監控並收集新的場域資料。最後將資料收集到一定程度，又回到週期的第一步重新訓練新模型。至於模型該如何重新訓練並保持資料的隱私性就是另一門議題。這時候我們就能採用一個技術叫做 Federated Learning (聯合學習) 想辦法處理這類的事情。



但是我們可以發現訓練一個機器學習模型，在 MLOps 的週期中僅扮演小小的一塊角色。下圖是一個訓練機器學習模型的基本流程，中間橘色的部分就是 AutoML 可以幫助我們的事。從資料前處理、訓練模型到評估模型需要不斷地的進行試驗，並且嘗試各種不同的模型演算法與模型超參數。除此之外還有資料前處理與特徵工程，都可以透過 AutoML 自動化的訓練找到一個滿意的模型。



超參數調參方法

機器學習自動化的困難點在於資料清洗與特徵工程技巧。一個好的特徵表達可以讓模型快速地抓到關鍵因子，並讓模型預測能力提升。慶幸的是模型挑選和超參數調整已經有比較成熟的方法可以協助我們有效的搜尋。

- Grid Search 網格搜索/窮舉搜索
- Random Search 隨機搜索
- Bayesian Optimization 貝葉斯優化

Grid Search

Grid Search (網格搜索) 又稱窮舉搜索。它的搜索方式是在所有可能的參數中，透過排列組合嘗試每一種可能性。並將表現最好的參數最為最終的超參數搜尋結果。他的缺點就是當有許多超參數要尋找時，他的排列組合就會變得非常多，導致搜索的時間變長花費的資源也變大。因此這種暴力式的搜索方法適合在小的資料集上被採用。然而在 Sklearn 套件中有提供 GridSearchCV *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)* 方法，使用者可以自己設定參數列表，並透過所有可能的參數組合一個一個嘗試找到最合適的參數。

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
# 載入鳶尾花朵資料集
iris = datasets.load_iris()
# 設定想要的搜索參數並給予候選值
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
# 建立 SVC 分類器
svc = svm.SVC()
```

```
# 網格搜索所有可能的組合(2*2)共四種
clf = GridSearchCV(svc, parameters)
# 擬合數據並回傳最佳模型
clf.fit(iris.data, iris.target)
```

搜索結束後也能夠過 `cv_results_` 查看所有組合的超參數所對應的訓練結果。

```
clf.cv_results_
```

Random Search

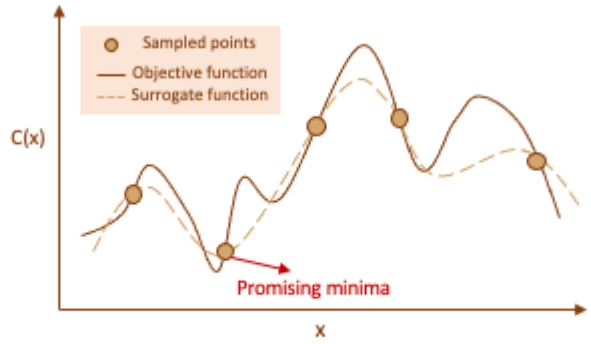
Random Search (隨機搜索) 按照字面上的意思就是在所有可能的候選參數中隨機挑選一個數值並嘗試。如果需要調的參數較多的時候，使用隨機搜索可以降低搜索時間，同時又能確保一定的模型準確性。在 Sklearn 套件中也有提供 RandomizedSearchCV *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)* 方法可以呼叫，與網格搜索的差別在於使用者可以將欲搜尋的超參數設定一個期望的範圍。該方法會在此範圍中隨機抽一個數值並進行模型訓練並驗證模型。並找出所有隨機組合中表現最好的一組超參數。

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
# 載入鳶尾花朵資料集
iris = load_iris()
# 建立邏輯迴歸模型
logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
# 設定欲搜尋的超參數並給予一個期望的範圍
distributions = dict(C=uniform(loc=0, scale=4),
                     penalty=['l2', 'l1'])
# 隨機搜索預設 n_iter=10
clf = RandomizedSearchCV(logistic, distributions, random_state=0, n_i
# 擬合數據並回傳最佳模型
search = clf.fit(iris.data, iris.target)
search.best_params_
```

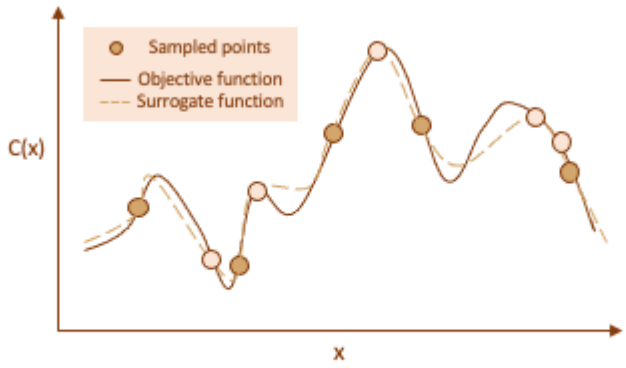
Bayesian Optimization

Bayesian Optimization (貝葉斯優化) 目標是要在最少的試驗下尋找一組最佳的超參數使得錯誤率能夠越低越好。由於我們所收集到的資料無從得知該模型的目標函數是長怎樣，因此機器學習的目的就是要從這些資料中去擬合一個函數，目標是給予一筆輸入 X 該函數的輸出要與真實的答案越接近越好。透過代理優化 (surrogate optimization) 使用一個代理函數來估計目標函數。簡

單來說代理函數是指目標函數的一種近似，此外代理函數可基於取樣得到的資料點被構建出來。



代理函數的目的是在給定一組特定的候選超參數的情況下快速估計實際模型的錯誤率。透過這種方式可以快速決定該組超參數是否可以被拿來實際訓練模型。隨著試驗次數的增加，代理函數隨著先前的試驗結果而更新改進，並開始推薦更好的候選超參數。



Auto-sklearn *(https://automl.github.io/auto-sklearn/master/)* 就是一個透過貝葉斯優化來尋找最佳超參數的一個工具。同時它也能搜索在 Sklearn 中所有可能的算法，並為你推薦一個合適的模型與資料前處理方式。明天我們就來一探究竟該套件背後的神秘原理以及程式實作吧！

Reference

- automl.org *(https://www.automl.org/automl/)*
- Sklearn 官方文件 GridSearchCV *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)*
- Sklearn 官方文件 RandomizedSearchCV *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)*
- 用簡單術語讓你看到貝葉斯優化之美 *(https://www.gushiciku.cn/pl/p7cE/zh-tw)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 20] 機器學習金手指 - Auto-sklearn

今日學習目標

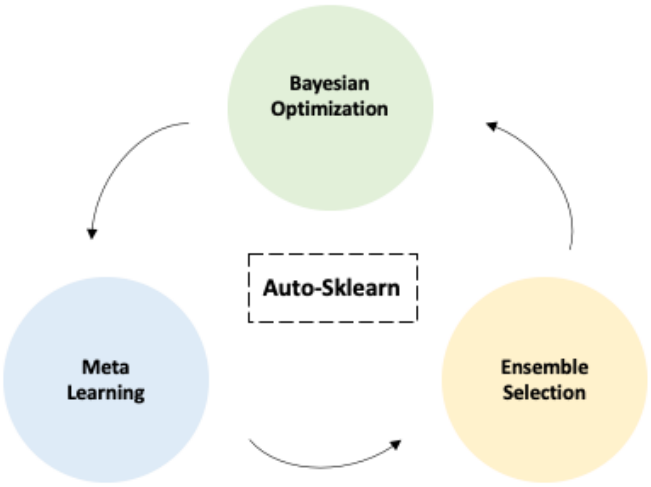
- 了解 Auto-sklearn 運作原理
  - Meta Learning
  - Bayesian Optimization
  - Build Ensemble
- 實作 Auto-sklearn
  - 採用鳶尾花朵資料集訓練，並比較兩種不同版本的 Auto-sklearn。
  - 使用 pipelineprofiler 視覺化 AutoML 模型。

範例程式： 

(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/20.Auto-Sklearn/20.Auto-sklearn(iris-classification).ipynb)

前言

Auto-sklearn 採用元學習 (Meta Learning) 選擇模型和超參數優化的方法作為搜尋最佳模型的重點。此 AutoML 套件主要是搜尋所有 Sklearn 機器學習演算法以模型的超參數，並使用貝葉斯優化 (Bayesian Optimization) 與自動整合 (Ensemble Selection) 的架構在有限時間內搜尋最佳的模型。第一版的 Auto-sklearn 於 2015 年發表在 NIPS(Neural Information Processing Systems) 會議上，論文名稱為 Efficient and Robust Automated Machine Learning *(https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf)*。有別於其他的 AutoML 方法，Auto-sklearn 提出了元學習架構改善了貝葉斯優化在初始冷啟動的缺點，並提供一個好的採樣方向更快速尋找最佳的模型[1]。第二個版本於 2020 年發布，論文名稱為 Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning *(https://arxiv.org/abs/2007.04074)*。在新的版本中修改了元學習架構，並不依賴元特徵來選擇模型選擇與調參策略。而是引入了一個元學習策略選擇器，根據資料集中的樣本數量和特徵，訂定了一個模型選擇的策略[3]。



AutoML 視為 CASH 問題

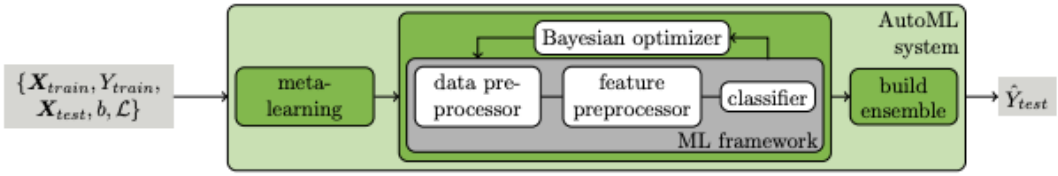
在論文中作者將 AutoML 視為演算法選擇和超參數優化 (Combined Algorithm Selection and Hyperparameter, CASH) 的組合最佳化問題。因為在 AutoML 領域當中將會面臨兩個問題。第一個是沒有任何的演算法模型是可以保證在所有的資料集中表現最好，因此挑選一個好的演算法是自動化機器學習的首要任務。第二許多的機器學習模型往往依賴於超參數，透過不同的超參數設定可以取得更好的學習結果。例如在 SVM 方法中我們可以設定不同的核技巧讓模型具有非線性的能力，或是透過超參數 C 限制模型的複雜度防止過度擬合。然而貝葉斯優化如今成為 AutoML 超參數搜尋的重要核心方法。

Auto-sklearn 架構

Auto-sklearn 可以被拿來處理迴歸和分類的問題。下圖為第一版論文中所繪製的架構圖。我們可以將 Auto-sklearn 切成三個部分，其中第一個是引入元學習機制來模仿專家在處理機器學習的先驗知識。並採用元特徵讓我們更有效率的去決定在新的資料集中該挑選哪一種機器學習模型。接著挑好模型後並透過貝葉斯優化來挑選合適的模型超參數，以及嘗試一些資料前處理與

特徵工程。最後挑選幾個不錯的模型並透過整體學習的技巧進行模型堆疊，將表現不錯的模型輸出結果做一個加權和或是投票。

- Meta Learning
- Bayesian Optimization
- Build Ensemble

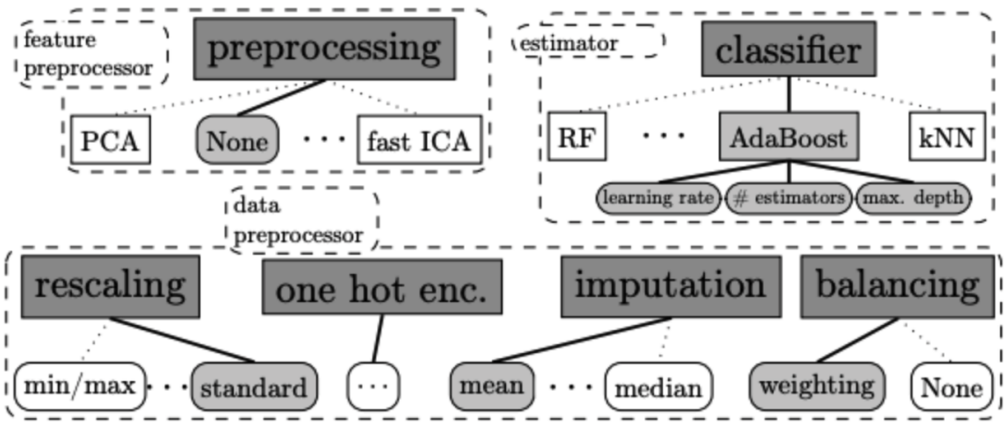


Meta Learning

當我們想對新資料集做分類或迴歸時，Auto-sklearn 會先提取元特徵，具有相似元特徵的資料集在同一組超參數應該會有相似的表現。因此透過元特徵可以有效地評估在新資料集上應該使用哪種算法。元學習在這裡的目的是為了要找一個不錯的超參數做初始化，使其在一開始的表現優於隨機的方法。並提供貝葉斯優化有個明確的方向。Auto-sklearn 參考了 OpenML 140 個資料集，並彙整了 38 個元特徵，例如：偏度、峰度、特徵數量、類別數量.....等。首先為這 140 個資料集使用貝葉斯優化進行模型訓練，並將這些資料集對應的模型與最佳的超參數儲存起來。當有新的資料集進來時會先透過元特徵進行相似度匹配，並將匹配程度最高的前 k 個資料集 (預設k=25) 所對應的模型和超參數作為貝葉斯優化的初始設定。

Bayesian Optimization

在貝葉斯優化當中主要會尋找該資料集中最合適的資料前處理 (data pre-processors)、特徵前處理 (feature pre-processors) 與分類/迴歸模型。以上三大類合計共有 110 個超參數必須透過貝葉斯優化來尋找最適合的參數組合。其貝葉斯優化主要方法是透過建立目標函數的機率模型，並用它來選擇最有希望的超參數來評估真實的目標函數。



以下內容摘錄自 Auto-sklearn v1.0 論文提供的內容 [1][2]

Data Pre-processors

在資料前處理部分 Auto-sklearn 提供了四種方法。包含特徵縮放、填補缺失值、類別特徵進行 one-hot encoding 與處理目標輸出類別數量不平衡問題。

- Data Pre-processors
  - 特徵縮放
  - 填補缺失值
  - one-hot encoding
  - 類別資料不平衡

在新的版本中多了一些資料前處理方法，詳細可以參考 Auto-sklearn data_preprocessing *(https://github.com/automl/auto-sklearn/tree/master/autosklearn/pipeline/components/data_preprocessing)* 的原始程式。

Feature Pre-processors

在特徵前處理部分 Auto-sklearn 提供了 12 種特徵處理的技巧，然而在眾多方法中僅會挑選其中一種。

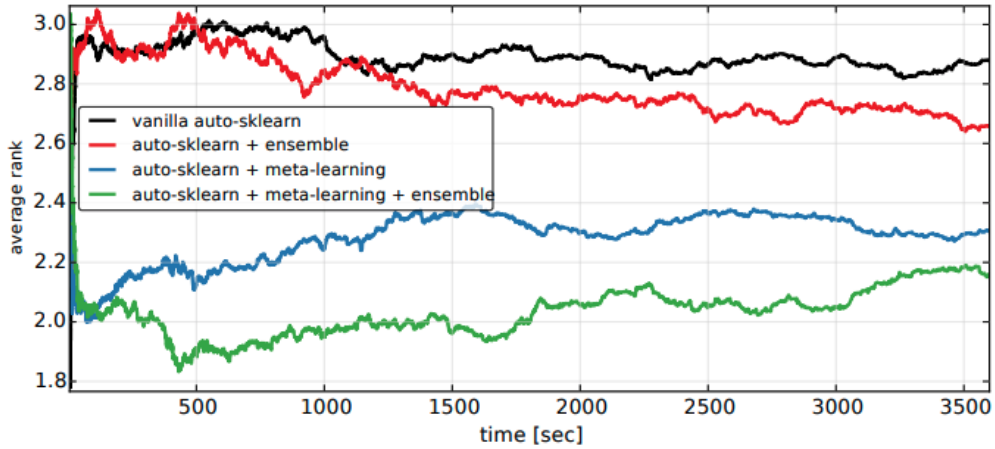
name	#λ	cat (cond)	cont (cond)
extreml. rand. trees prepr.	5	2 (-)	3 (-)
fast ICA	4	3 (-)	1 (1)
feature agglomeration	4	3 ()	1 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
PCA	2	1 (-)	1 (-)
polynomial	3	2 (-)	1 (-)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)

詳細可以參考 Auto-sklearn feature_preprocessing *(https://github.com/automl/auto-sklearn/tree/master/autosklearn/pipeline/components/feature_preprocessing)* 的原始程式。

Build Ensemble

在 Auto-sklearn 訓練階段會產生許多表現優良的模型，最終透過貪婪法的 Bagging Ensemble Selection *(http://www.cs.cornell.edu/~alexn/papers/shotgun.icml04.revised.rev2.pdf)* 方法來合併

多個模型組合成一個更強更大的模型，並提高預測的準確性。下圖為第一版論文中進行的實驗，其中橫軸為程式執行時間，縱軸為在時間內搜尋到的最佳模型的排名。我們可以發現綠色線條再加入了整體學習機制表現效果比尚未加入的藍色線條實驗來得好。並且在短時間內就可以得到不錯的結果。

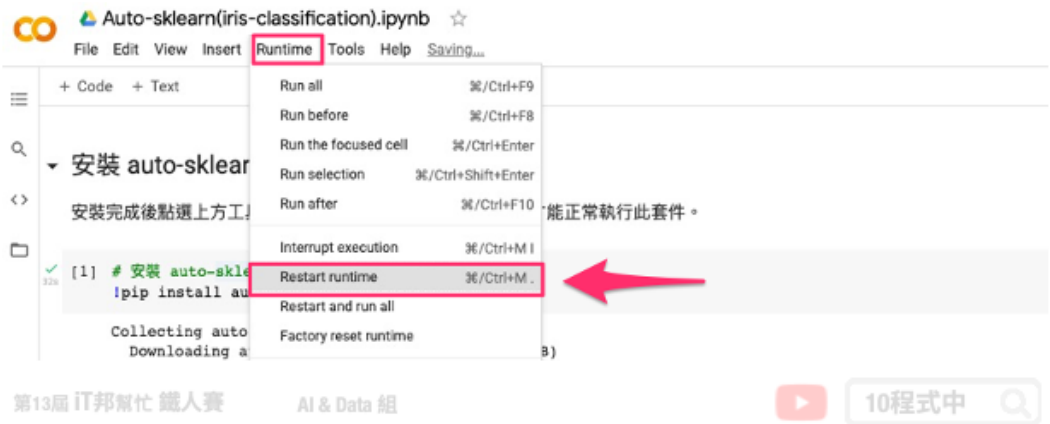


安裝 Auto-sklearn

目前 Auto-sklearn 僅支援 Lunux 系統。若沒有此系統的讀者可以透過 Colab 體驗。另外若安裝過程中出現錯誤，必須先確認 swig *(https://automl.github.io/auto-sklearn/master/installation.html)* 是否已完成安裝。

```
pip install auto-sklearn
```

> 若使用 Colab 執行，安裝完成後點選上方工具列 Runtime -> Restart runtime 重啟才能正常執行此套件。



載入資料集

本次範例沿用鳶尾花朵資料集，並使用 Auto-sklearn 來搜尋最佳的分類器模型。此外大家可以試著觀察 Auto-sklearn 找到的最佳模型在訓練集與測試集上的表現，並與前幾天所介紹的那些機器學習演算法來做比較。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

iris = load_iris()
df_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= ['SepalLengthCm','SepalWidthCm','PetalL
df_data
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0
149	5.9	3.0	5.1	1.8	2.0

150 rows × 5 columns

切割訓練集與測試集

我們按照花朵種類的數量對資料集以 7:3 的比例切割出訓練集與測試集。其中參數 `stratify=y` 設定是確保訓練集與測試集對於三種花朵類別的比例在這兩個切出來的資料集中比例要一樣，以免訓練出來的模型有很大的偏差。

```
from sklearn.model_selection import train_test_split
X = df_data.drop(labels=['Species'],axis=1).values # 移除Species並取得剩
y = df_data['Species'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

```
print('train shape:', X_train.shape)
print('test shape:', X_test.shape)
```

輸出結果：

```
train shape: (105, 4)
test shape: (45, 4)
```

Auto-sklearn

以下是模型常用的超參數以及方法，詳細內容可以參考官方 API 文件 *(https://automl.github.io/auto-sklearn/master/api.html)*。

Parameters: - time_left_for_this_task: 搜尋時間(秒)，預設3600秒(6分鐘)。 - per_run_time_limit: 每個模型訓練的上限時間，預設為time_left_for_this_task的1/10。 - ensemble_size: 模型輸出數量，預設50。 - resampling_strategy: 資料採樣方式。為了避免過擬合，可以採用交叉驗證機制。預設方法為最基本的 holdout。

Attributes: - cv_results_: 查詢模型搜尋結果以及每個最佳模型的超參數。

Methods: - fit: 放入X、y進行模型擬合。 - refit: 使用 fit 尋找好的參數後，再使用所有的資料進行最後微調。 - predict: 預測並回傳預測類別。 - score: 預測成功的比例。 - predict_proba: 預測每個類別的機率值。 - leaderboard: 顯示 k 個 ensemble 模型並排名。

首先我們來測試第一版的 Auto-sklearn，建立一個分類器類型的自動化機器學習模型並設定相關的執行參數。在本次實驗中我們設定模型搜尋總時間為 180 秒，每個模型訓練時間限制 40 秒內。此外設定 `resampling_strategy='cv'` 即 K-Fold 交叉驗證。此外必須另外設定 `resampling_strategy_arguments` 並給予 k=5，訓練集切割為五等份。這意味著相同的模型要訓練五次，每一次的訓練都會從這五等份挑選其中四等份作為訓練資料，剩下一等份未參與訓練並作為驗證集。

```
import autosklearn.classification
automlclassifierV1 = autosklearn.classification.AutoSklearnClassifier
    time_left_for_this_task=180,
    per_run_time_limit=40,
    resampling_strategy='cv',
    resampling_strategy_arguments={'folds': 5}
)
automlclassifierV1.fit(X_train, y_train)
```

訓練結束後我們可以來查看模型在訓練集與測試集表現。大家可以試著調整模型訓練時間以及一些控制參數，查看是否有沒有幫助模型準確度提升。

```
# 預測成功的比例
print('automlclassifierV1 訓練集: ',automlclassifierV1.score(X_train,y
print('automlclassifierV1 測試集: ',automlclassifierV1.score(X_test,y_
```

輸出結果：

```
automlclassifierV1 訓練集:  0.9904761904761905
automlclassifierV1 測試集:  0.9111111111111111
```

使用 Auto-sklearn 2.0

在第二版的 Auto-sklearn 對模型搜尋進行了一些優化，並且可以自動搜尋好的資料採樣方式。因此我們不特地去指定 `resampling_strategy`，查看表現是否能夠提升。

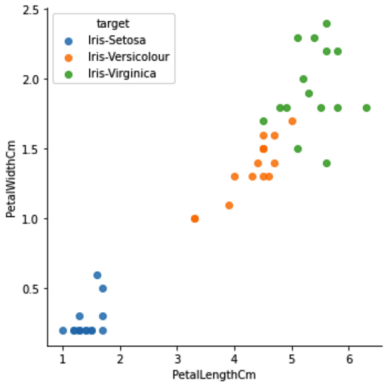
```
from autosklearn.experimental.askl2 import AutoSklearn2Classifier

automlclassifierV2 = AutoSklearn2Classifier(time_left_for_this_task=1
automlclassifierV2.fit(X_train, y_train)
```

```
# 預測成功的比例
print('automlclassifierV2 訓練集: ',automlclassifierV2.score(X_train,y
print('automlclassifierV2 測試集: ',automlclassifierV2.score(X_test,y_
```

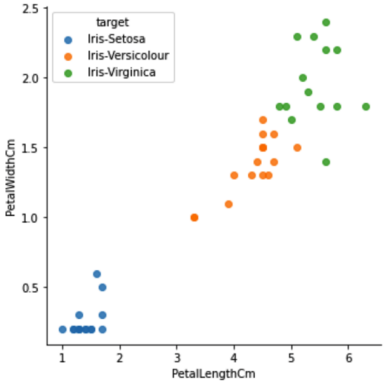
執行結果：

```
automlclassifierV2 訓練集:  0.9904761904761905
automlclassifierV2 測試集:  0.9333333333333333
```

使用一樣的搜尋時間與訓練限制，最終訓練出來的模型在訓練集與測試集都表現不錯。兩者的準確率更接近了。這樣的結果的確比系列教學所介紹的任一個單一模型還來得好。



真實分類



模型預測

查看每個模型的權重

我們可以使用模型提供的方法查看最終訓練結果，並查看 k 個 Ensemble 模型的訓練結果以及每個模型的權重。

```
automlclassifierV2.leaderboard(detailed = True, ensemble_only=True)
```

model_id	rank	ensemble_weight	type	cost	duration	config_id	train_loss	seed	start_time	end_time	budget
5	1	0.02	extra_trees	0.028571	8.158257	4	0.019048	0	1.633139e+09	1.633139e+09	100.00
8	2	0.04	gradient_boosting	0.028571	4.271513	7	0.000000	0	1.633139e+09	1.633139e+09	6.25
11	3	0.04	extra_trees	0.028571	3.994621	10	0.014286	0	1.633139e+09	1.633139e+09	6.25
24	4	0.10	gradient_boosting	0.028571	3.725940	23	0.000000	0	1.633139e+09	1.633139e+09	6.25
28	5	0.02	extra_trees	0.038095	4.176670	27	0.019048	0	1.633139e+09	1.633139e+09	6.25
32	6	0.02	mlp	0.057143	10.952644	31	0.004762	0	1.633139e+09	1.633139e+09	100.00
17	7	0.02	mlp	0.076190	3.610623	16	0.085714	0	1.633139e+09	1.633139e+09	6.25
4	8	0.06	sgd	0.095238	3.963269	3	0.047619	0	1.633139e+09	1.633139e+09	6.25
31	9	0.04	mlp	0.123810	3.625073	30	0.100000	0	1.633139e+09	1.633139e+09	6.25
20	10	0.12	mlp	0.123810	3.486436	19	0.095238	0	1.633139e+09	1.633139e+09	6.25
18	11	0.02	mlp	0.123810	3.604788	17	0.095238	0	1.633139e+09	1.633139e+09	6.25
16	12	0.04	sgd	0.142857	4.134084	15	0.147619	0	1.633139e+09	1.633139e+09	6.25
14	13	0.04	passive_aggressive	0.161905	4.313525	13	0.171429	0	1.633139e+09	1.633139e+09	6.25
13	14	0.02	sgd	0.171429	4.100530	12	0.138095	0	1.633139e+09	1.633139e+09	6.25
33	15	0.02	passive_aggressive	0.171429	3.743019	32	0.171429	0	1.633139e+09	1.633139e+09	6.25
10	16	0.06	sgd	0.228571	4.434203	9	0.190476	0	1.633139e+09	1.633139e+09	6.25
27	17	0.02	passive_aggressive	0.314286	3.785241	26	0.309524	0	1.633139e+09	1.633139e+09	6.25
30	18	0.02	mlp	0.333333	3.599097	29	0.333333	0	1.633139e+09	1.633139e+09	6.25
26	19	0.06	sgd	0.571429	3.941010	25	0.561905	0	1.633139e+09	1.633139e+09	6.25
25	20	0.04	gradient_boosting	0.657143	3.689387	24	0.671429	0	1.633139e+09	1.633139e+09	6.25

輸出模型

如果想將 AutoML 的模型儲存起來，可以透過 `joblib` 將模型打包匯出。

```
from joblib import dump, load

# 匯出模型
dump(automlclassifierV2, 'model.joblib')
# 匯入模型
clf = load('model.joblib')
# 模型預測測試
clf.predict(X_test)
```

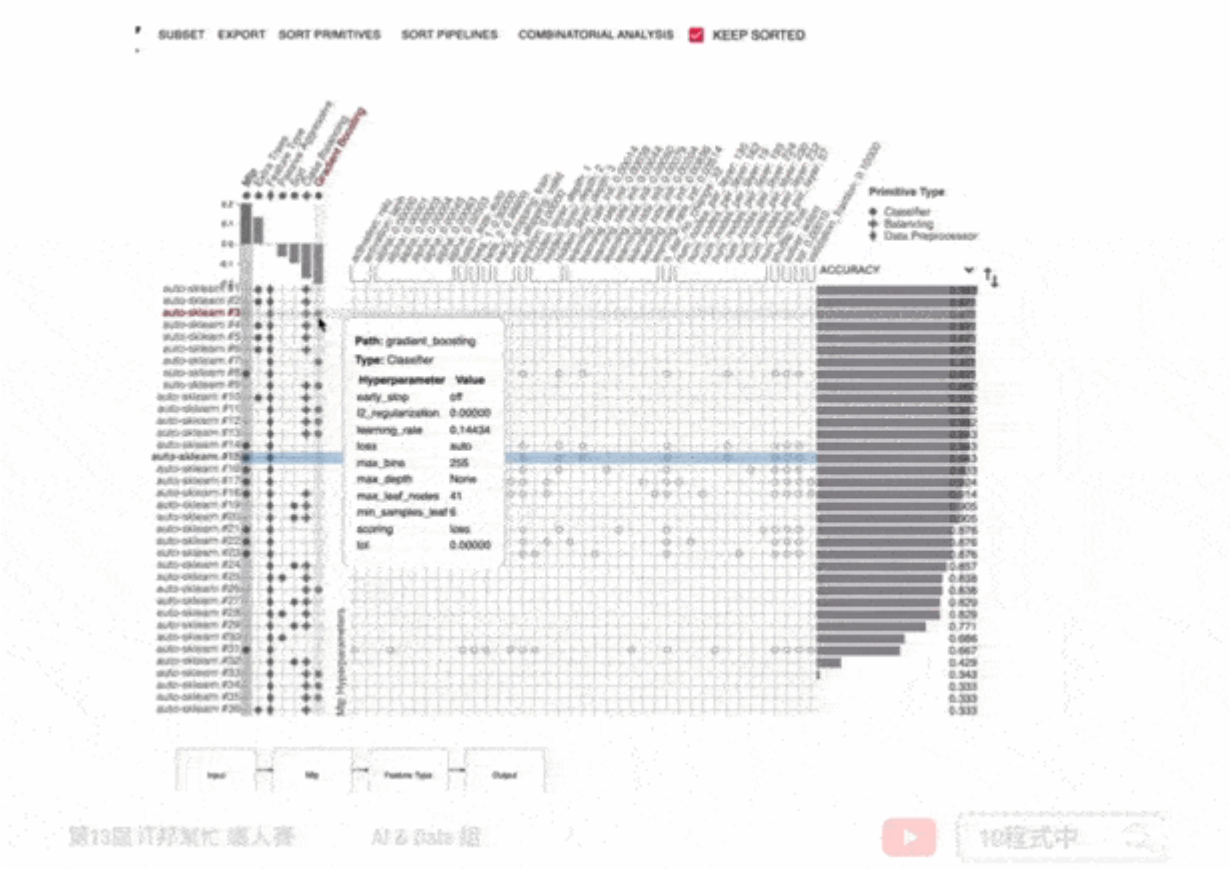
視覺化 AutoML 模型

首先安裝 pipelineprofiler。

```
pip install pipelineprofiler
```

透過 PipelineProfiler 套件可以很快速地檢視模型訓練結果，以及每一個 Ensemble 模型的超參數以及資料前處理方式和特徵處理方法。

```
import PipelineProfiler

profiler_data= PipelineProfiler.import_autosklearn(automlclassifierV2
PipelineProfiler.plot_pipeline_matrix(profiler_data)
```



Reference

- [1] Feurer, Matthias et al. Efficient and Robust Automated Machine Learning *(https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf)*, Advances in neural information processing systems 2015.
- [2] Feurer, Matthias et al. Supplementary Material for Efficient and Robust Automated Machine Learning *(https://ml.informatik.uni-freiburg.de/wp-content/uploads/papers/15-NIPS-auto-sklearn-supplementary.pdf)*, Advances in neural information processing systems 2015.
- [3] Feurer, Matthias et al. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning *(https://arxiv.org/abs/2007.04074)*, arXiv, 2020.
- [4] Ono, Jorge et al. PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines *(https://arxiv.org/abs/2005.00160)*, arXiv, 2020.
- Auto Machine Learning筆記- Bayesian Optimization *(http://codewithzhangyi.com/2018/07/31/Auto%20Hyperparameter%20Tuning%20-%20Bayesian%20Optimization/)*
- A Quickstart Guide to Auto-Sklearn (AutoML) for Machine Learning Practitioners *(https://neptune.ai/blog/a-quickstart-guide-to-auto-sklearn-automl-for-machine-learning-practitioners)*
- Auto-Sklearn: Scikit-Learn on Steroids *(https://towardsdatascience.com/auto-sklearn-scikit-learn-on-steroids-42abd4680e94)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 21] 調整模型超參數利器 - Optuna

今日學習目標

- Optuna 如何採樣參數？
- 實作 Optuna 搜尋最佳超參數
    - 以 XGBoost 迴歸模型於房價預測為例
    - Optuna 視覺化分析搜尋結果

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/21.Optuna/21.optuna-tutorial.ipynb)*

前言

你是否曾經覺得模型有太多的超參數而感到厭煩嗎？要從某一個演算法得到好的解必須要調整超參數，所謂的超參數就是控制訓練模型的一組神秘數字，例如學習速率就是一種超參數。你永遠都不知道 0~1 之間哪一個數字是最適合的，唯一的方法就是試錯 (trial and error)。那萬一模型有多個超參數可以控制，豈不是就有成千上萬種組合要慢慢嘗試嗎？如果你有也這個問題，看這篇就對了！雖然你可能聽過 Sklearn 的 GridSearchCV 同樣也是暴力的找出最佳參數，或是使用 RandomizedSearchCV 指定超參數的範圍並隨機的抽取參數進行訓練，其它們的共同缺點是非常耗時與佔用機器資源。這裡我們要來介紹 Optuna 這個自動找超參數的方便工具，並且可以和多個常用的機器學習演算法整合。Optuna 透過調整適當的超參數來提高模型預測能力，此專案最初於 2019 發表於 arxiv 的一篇論文 Optuna: A Next-generation Hyperparameter Optimization Framework *(https://arxiv.org/abs/1907.10902)* 同時開源在 GitHub *(https://github.com/optuna/optuna)* 上免費提供大家使用。同時 Optuna 也是 2021 年 Kaggle 資料科學競賽中最常見的模型調參工具。那是什麼原因讓 Optuna 受到廣大的機器學習社群如此的歡迎呢？就讓我們來看看他是如此地強大吧！

關於 Optuna

Optuna 是一個專為機器學習設計的自動超參數優化的框架。其最突出的特點是：

- 人性化的定義搜索空間。
- 支援大多數 ML 與 DL 的學習套件。例如: Sklearn、PyTorch、TensorFlow, XGBoost、LightGBM、 CatBoost...等。
- 對對搜索結果提供可解釋性(XAI)。
- 儲存歷史最佳的參數實現平行優化工作。
- 決定並終止不滿足預定義條件的試驗。

Optuna 簡單範例

這裡我們設定一個簡單的目標函式 $(x1+2)^2 + (x2-4)^2$。我們都知道當這個式子 x1=-2, x2=4 時將會有極小值 0。因此我們就用這個簡單的例子透過 Optuna 找出這個函式中極小值所對應的 x1 與 x2 吧。

```
import optuna

def objective(trial):
    x1 = trial.suggest_float("x1", -5, 5)
    x2 = trial.suggest_float("x2", -5, 5)
    return (x1 + 2) ** 2 + (x2 - 4) ** 2
```

首先載入 optuna 套件，如果尚未安裝此套件的的讀者可以參考以下指令進行安裝：

```
pip install optuna
```

接著我們來定義一個找出極小值的目標函式 objective()。在這個函式中我們將要設定 optuna 可以去尋找的一參數，也就是 x1 與 x2。我們可以透過 optuna 所提供的 trial 物件來為我們的超參數設定一組範圍。其中它有一個 suggest_float 方法，該方法採用超參數的名稱和範圍來尋找其最佳值。我們以 x1 來舉例：

```
x1 = trial.suggest_float("x1", -5, 5)
```

上面這一段程式在 GridSearch 中可以表示成 {"x1": np.arange(-5, 5, .1)}。即表示搜尋過程中我們會從 x1 隨機設定 -5~5 之間的任一浮點數。設定完函式後就可以開始優化了，我們從 optuna 建立一個 study 物件，並將 objective 函數傳遞給 study 的 optimize 方法。由於我們的目標是要找出函式中的極小值，因此 direction 設為 minimize。另外在 optimize 方法中我們也可以設定試驗的次數(n_trials)或時間(timeout)。一切就緒後即可開始執行！以下範例是迭代50次並從中找到一組最佳的 x1 與 x2 使其目標函式可以最小化。跑完 50

次後我們可以經由 study 變數中得到一組最佳的解。試驗結束後我們可以發現 x1 趨近於 -2 和 x2 趨近於 4。

```
%%time
# Creating Optuna object and defining its parameters
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials = 50)

# Showing optimization results
print('Number of finished trials:', len(study.trials))
print('Best trial parameters:', study.best_trial.params)
print('Best score:', study.best_value)
```

輸出結果：

```
Number of finished trials: 50
Best trial parameters: {'x1': -1.8154924755761588, 'x2': 3.9141985823
Best score: 0.04140490983908035
CPU times: user 432 ms, sys: 46.3 ms, total: 478 ms
Wall time: 431 ms
```

由上述的簡單例子我們可以知道建立一個 optuna 最佳化流程僅需要三步驟： 1. 建立 objective 函式與設定 trial, 並回傳 loss。 2. 建立 `create_study()` 物件。 3. 使用 `optimize()` 執行搜尋。

End-to-end example with XGBoost

我們以 Sklearn 所提供的房價預測資料來來做範例。此資料集共有 506 筆資料, 其中輸入特徵有 13 個其輸出為預測該筆資料的房價。由於想要快速示範如何使用 optuna, 因此這裡就不做任何資料 EDA 與前處理。

```
from sklearn.datasets import load_boston
X, y = load_boston(return_X_y=True)
print('X:',X.shape)
print('y:',y.shape)
```

輸出結果：

```
X: (506, 13)
y: (506,)
```

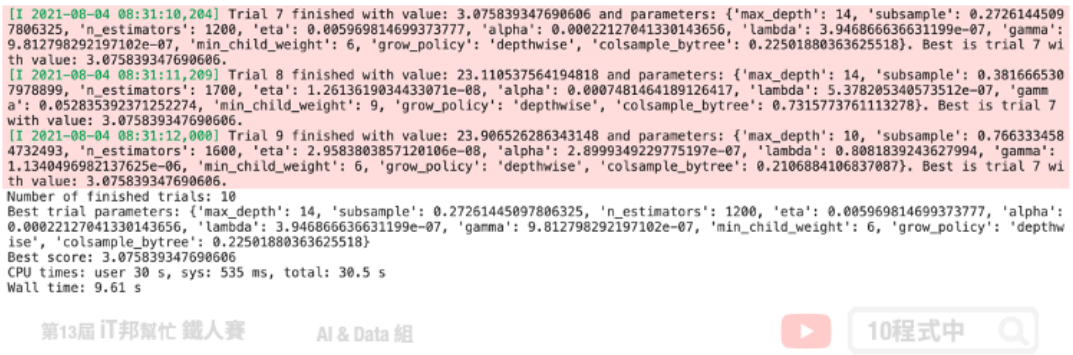
資料集成功被載入後我們就可以建立一個 objective 函式。在這個目標函式中, 我們建立了一個小範圍的的 XGBoost 超參數搜索空間。其每一個超參數都會有一個搜索的範圍, 可以使用 `suggest_*` 方法設定區間。此方法必須輸入超參數的名稱, 以及給予該參數的一組隨機範圍其

型態有很多例如：suggest_int、suggest_discrete_uniform、suggest_float...等。更多詳細的內容可以從官方文件 *(https://optuna.readthedocs.io/en/stable/reference/generated/optuna.trial.Trial.html)*取得。或是也可以參考官方在 GitHub *(https://github.com/optuna/optuna-examples/tree/main/xgboost)* 上對於 XGBoost 的使用範例。

```
import optuna
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def objective(trial, X=X, y=y):
    """
    A function to train a model using different hyperparamerters comb
    """
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_

    params = {
        'max_depth': trial.suggest_int('max_depth', 6, 15),
        "subsample": trial.suggest_float("subsample", 0.2, 1.0),
        'n_estimators': trial.suggest_int('n_estimators', 500, 2000,
        'eta': trial.suggest_float("eta", 1e-8, 1.0, log=True),
        'alpha': trial.suggest_float('alpha', 1e-8, 1.0, log=True),
        'lambda': trial.suggest_float('lambda', 1e-8, 1.0, log=True),
        'gamma': trial.suggest_float("gamma", 1e-8, 1.0, log=True),
        'min_child_weight': trial.suggest_int('min_child_weight', 2,
        'grow_policy': trial.suggest_categorical("grow_policy", ["dep
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0
    }

    reg = xgb.XGBRegressor(**params)
    reg.fit(X_train, y_train,
            eval_set=[(X_valid, y_valid)], eval_metric='rmse',
            verbose=False)
    return mean_squared_error(y_valid, reg.predict(X_valid), squared=
```

設定好調參的區間後，即可開始囉。

```
%%time
# Creating Optuna object and defining its parameters
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials = 10)

# Showing optimization results
print('Number of finished trials:', len(study.trials))
print('Best trial parameters:', study.best_trial.params)
print('Best score:', study.best_value)
```



Optuna 預設的超參數搜尋方法能有效地在短時間內往最佳的方向去尋找一組適合的參數。與 GridSearch 相比原本可能需要數小時的搜索空間在短短的幾分鐘內就可以獲得不錯的經果。並且有效的降低 loss。除了迴歸問題 Optuna 也能對分類問題進行超參數搜尋，官方的 GitHub *(https://github.com/optuna/optuna-examples)* 也有提供各種不同機器學習框架的寫法。

Optuna 如何採樣參數？

TPESampler 為預設的超參數採樣器。它試圖透過提高最後一次試驗的分數來對超參數候選者進行採樣。除此之外 Optuna 提供了以下這幾個參數採樣的方式: - `GridSampler`: 與 Sklearn 的 `GridSearch` 採樣方式相同。使用此方法時建議不要設定太大的範圍。 - `RandomSampler`: 與 Sklearn 的 `RandomizedGridSearch` 採樣方式相同。 - `TPESampler`: 全名 Tree-structured Parzen Estimator sampler。預設採樣方式。 - `CmaEsSampler`: 基於 CMA ES 演算算法的採樣器 (不支援類別型的超參數).

如果需要替換採樣參數的方式可以參考以下程式。

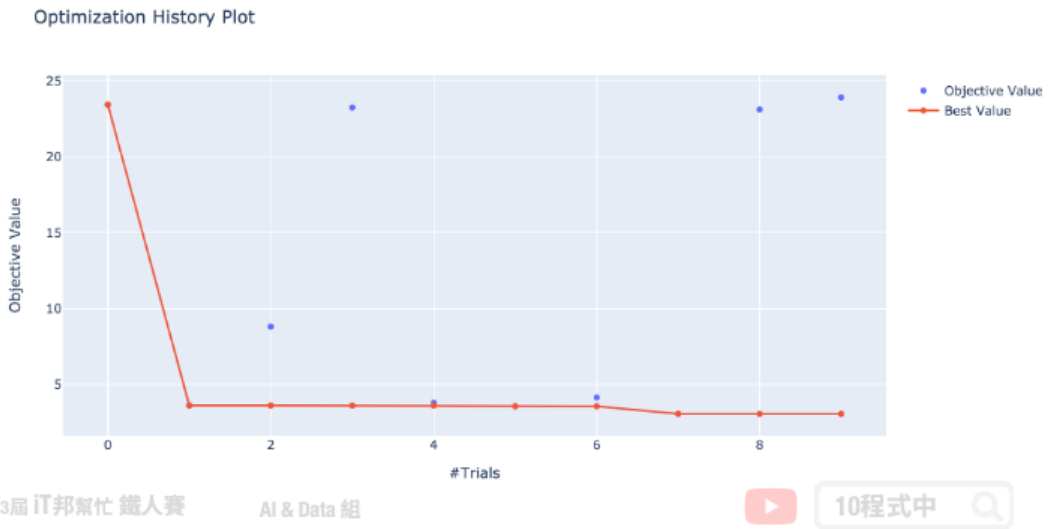
```
from optuna.samplers import CmaEsSampler, RandomSampler

# Study with a random sampler
study_1 = optuna.create_study(sampler=RandomSampler(seed=42))

# Study with a CMA ES sampler
study_2 = optuna.create_study(sampler=CmaEsSampler(seed=42))
```

Optuna 視覺化分析

Optuna 在同時也提供了視覺化的套件: - plot_optimization_history (視覺化優化的過程) - plot_intermediate_values (視覺化學習的曲線) - plot_parallel_coordinate (視覺化高維度中參數間的彼此關係) - plot_contour (視覺化參數間的彼此關係) - plot_slice (視覺化個別參數) - plot_param_importances (參數對模型的重要程度) - plot_edf (視覺化驗分佈函數)

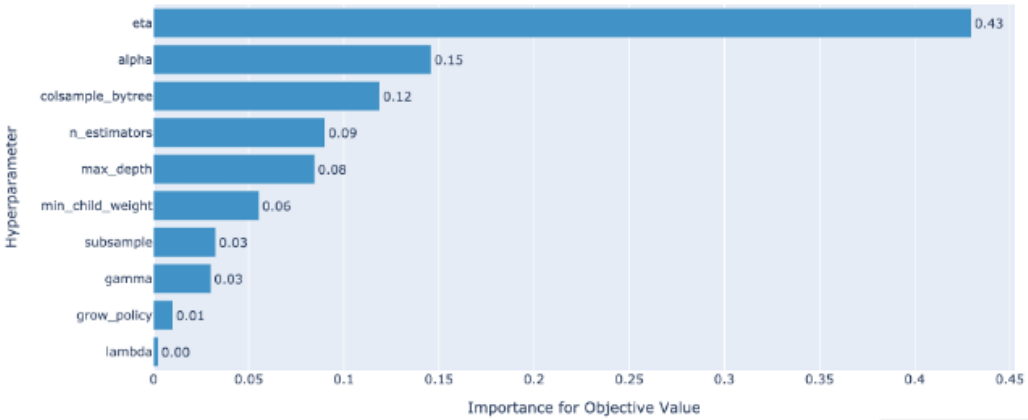
延續上面的範例我們來視覺化展示 Optuna 搜尋的過程與結果。首先我們來繪製 `study` 的優化歷史過程。這張圖告訴我們, Optuna 只經過幾次試驗就使分數收斂到最小值。

```
from optuna.visualization import plot_optimization_history

plotly_config = {"staticPlot": True}

fig = plot_optimization_history(study)
fig.show(config=plotly_config)
```



接下來, 讓我們繪製超參數重要性。從這張圖我們可以發現 eta(learning_rate) 學習速率是最為重要的。此外 grow_policy 與 lambda 對減少 loss 上無太大幫助。因此在下一次執行試驗的時候可以考慮將無用的參數移除, 並將重要的超參數範圍加大取得更好的搜索結果。其他的使用方法可以 *參考 (https://optuna.readthedocs.io/en/stable/reference/visualization/index.html)* 官方的說明文件。

```
from optuna.visualization import plot_param_importances

fig = plot_param_importances(study)
fig.show(config=plotly_config)
```



小結

今天我們介紹了這一個超參數最佳化的工具，裡面有太多功能尚未提到。例如：試驗的剪枝，簡單來說就是設定試驗的例外條件當不滿足預定條件即不執行此次試驗。或是儲存歷史最佳的參數實現平行優化工作。除此之外此套件還支援像是 SQLite 等資料庫可以儲存歷史搜尋結果快速的達到最佳搜尋能力。重點此套件還支援神經網路的參數搜尋以及網路的寬度深度選擇。常見的深度學習框架都能支援例如 TensorFlow、PyTorch， MXNet...等。

Reference

- OPTUNA: A Flexible, Efficient and Scalable Hyperparameter Optimization Framework *(https://towardsdatascience.com/optuna-a-flexible-efficient-and-scalable-hyperparameter-optimization-framework-d26bc7a23fff)*
- optuna.org *(https://optuna.org/)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 22] Python 視覺化解釋數據 - Plotly Express

今日學習目標

- 安裝 plotly
- 手把手實作視覺化鳶尾花朵資料集
  - 直方圖
  - 特徵關聯度分析
  - 散佈圖
  - 箱形圖
  - 複合型視覺化技巧
  - 匯出圖片

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/22.Plotly-Express/22.Plotly-Express.ipynb)*

前言

Plotly Express 是一個高精緻的資料視覺化套件。初學機器學習的你一定碰過像是 matplotlib 和 seaborn 這類型的圖表化套件，不過使用過 Plotly Express 會讓你對於資料視覺化有更不一樣的體驗。它的功能使用起來非常直觀，並且可以很好地與 Pandas DataFrame 配合使用。Plotly Express *(https://plotly.com/python/plotly-express/)* 於 2019 年由加拿大 Plotly 這間公司釋出了第一版高階的 Python 資料視覺化套件。

安裝 plotly

若尚未安裝此套件的讀者，可以開啟終端機輸入以下指令進行安裝：

```
pip install plotly
```

1) 載入資料集

在今天的範例中我們一樣採用鳶尾花朵資料集來做示範，讓大家瞧瞧 Plotly Express 是如優雅的處理資料視覺化。

```
import plotly.express as px
import plotly.graph_objects as go
from IPython.display import HTML

df_data = px.data.iris()
df_data
```

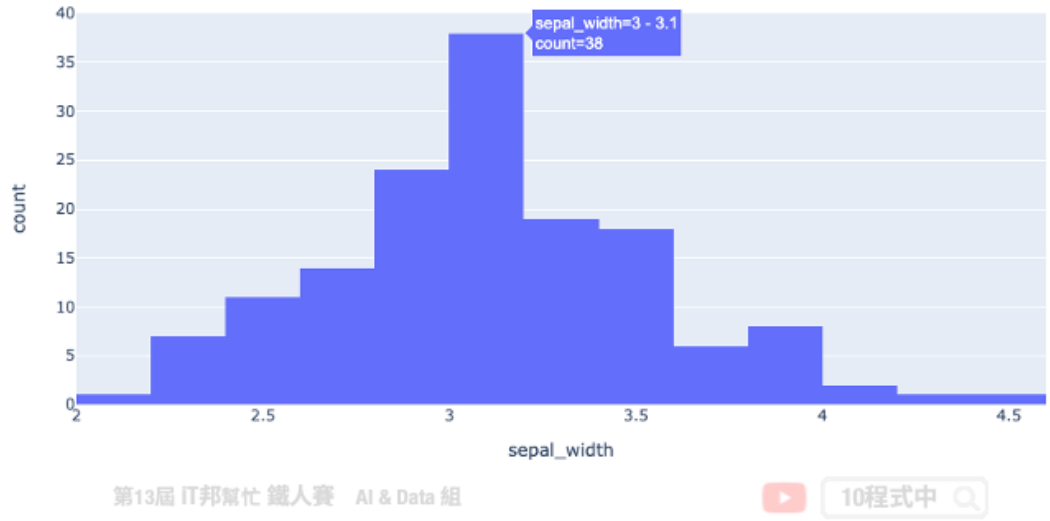
	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	3
146	6.3	2.5	5.0	1.9	virginica	3
147	6.5	3.0	5.2	2.0	virginica	3
148	6.2	3.4	5.4	2.3	virginica	3
149	5.9	3.0	5.1	1.8	virginica	3

150 rows × 6 columns

直方圖

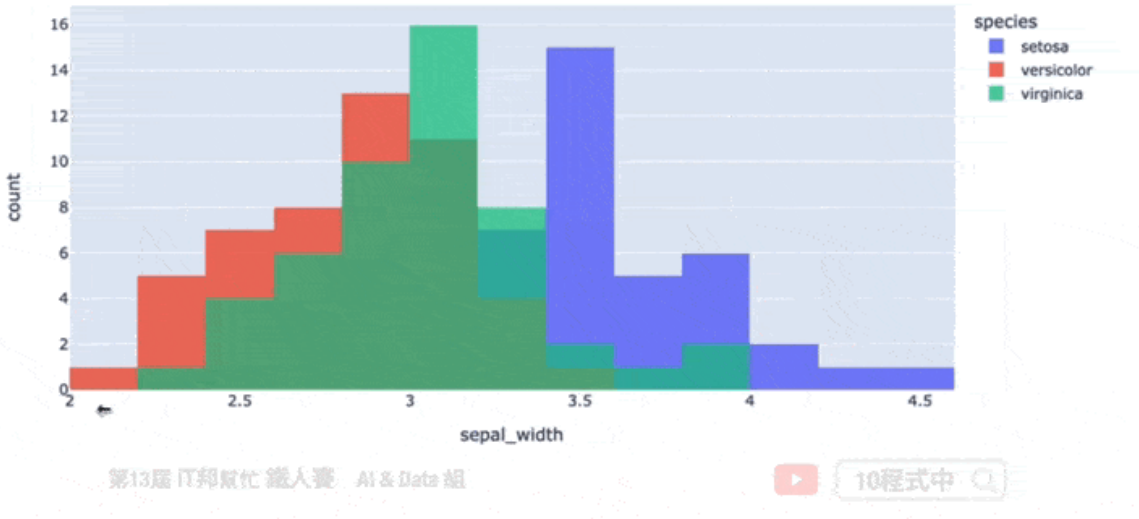
為了更清楚了解特徵的分布狀況，我們可以採用直方圖 histogram 做更近一步的分析。從直方圖我們可以更清楚知道特徵的每個值的頻率分佈。由於目前版本在 Notebook 無法直接使用 `fig.show()` 顯示互動圖，必須安裝一些小插件模組與設定。因此範例中採用最簡單方法，先轉換成 HTML code 並透過 `IPython.display` 中的 HTML 方法顯示出來。

```
fig = px.histogram(df_data, x="sepal_width")
HTML(fig.to_html())
```



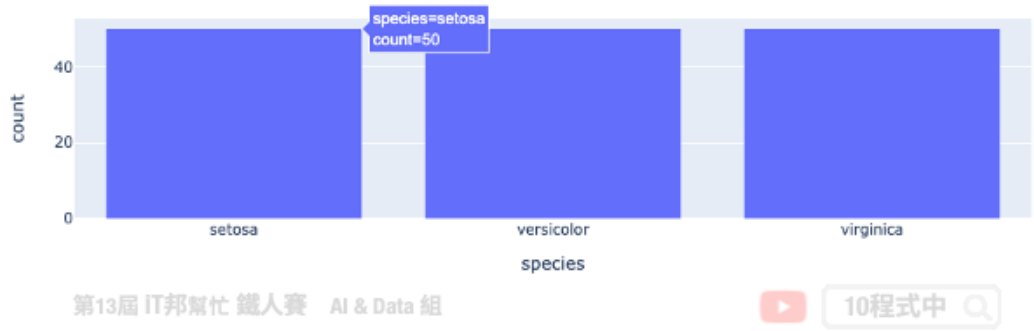
除此之外我們也能觀察每一個獨立特徵對於花的品種的每個分布狀況。每個不同的顏色代表不同的花朵品種，我們可以藉由參數設定每個直方圖是否重疊，以及重疊的透明程度。

```
fig = px.histogram(df_data, x="sepal_width", color="species")
fig.update_layout(barmode='overlay')
fig.update_traces(opacity=0.75)
HTML(fig.to_html())
```



接下來一樣透過直方圖方式來觀察每個花朵品種的數量。從視覺化可以很清楚得知該資料集是否是一個平穩的資料集。

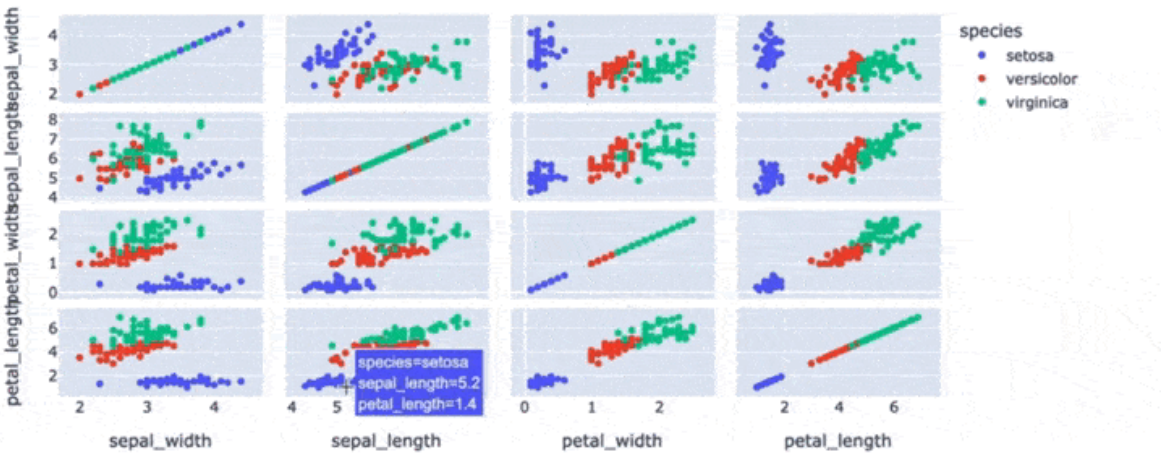
```
fig = px.histogram(df_data, x='species', y='sepal_width', histfunc='c
                   title='Histogram Chart')
HTML(fig.to_html())
```



特徵關聯度分析

我們可以採用 scatter_matrix 為每一個特徵彼此間做一個關聯度分析。透過這種視覺化方式我們可以很清楚的知道兩個特徵間是否正相關與負相關。

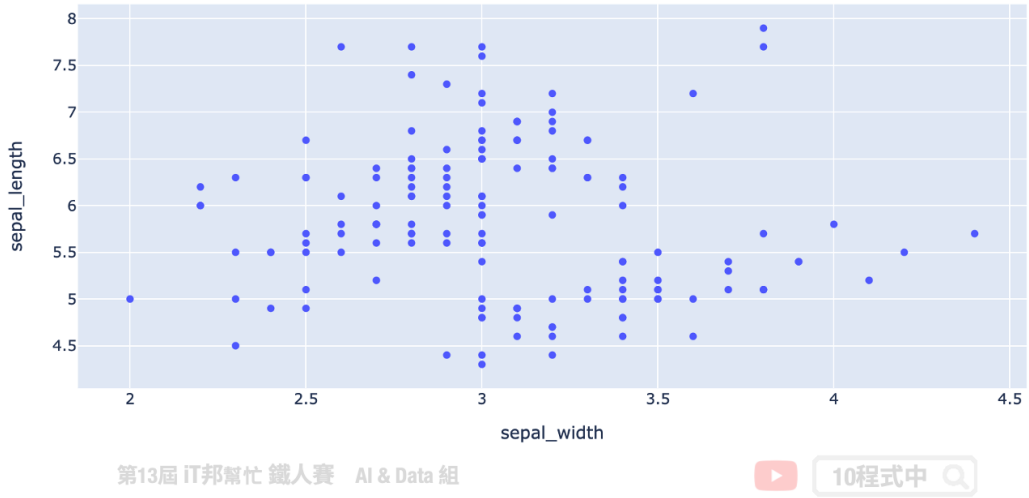
```
fig = px.scatter_matrix(df_data, dimensions=["sepal_width", "sepal_le
HTML(fig.to_html())
```



散佈圖

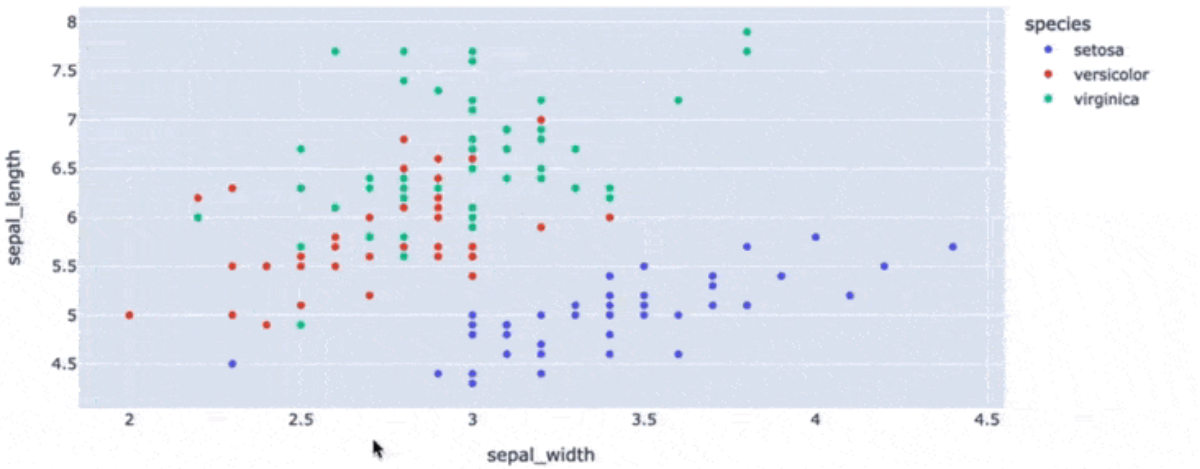
我們使用散佈圖將花萼的長度與寬度顯示在二維坐標平面上。使用 Plotly Express 套件中的 scatter 方法，我們可以輕鬆構建圖形，並放入 DataFrame 格式的資料並指定必要參數 x 軸中的變數和 y 軸中的變數。

```
fig = px.scatter(df_data, x="sepal_width", y="sepal_length")
HTML(fig.to_html())
```



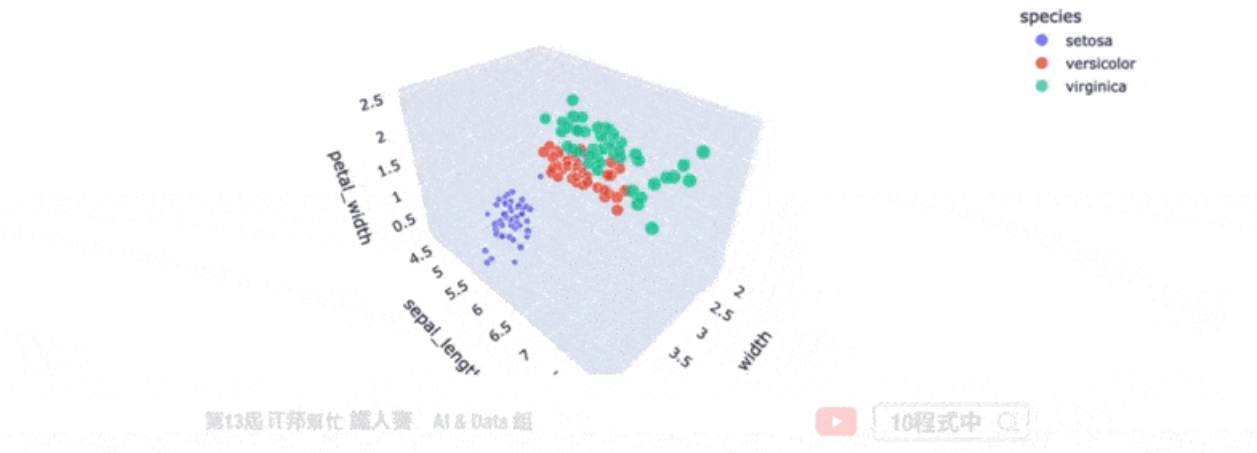
如果想要更清楚表達每個資料點所對應的類別，可以再加上 `color` 並指定種類的欄位即會將所有資料自動分成三類。此外我們也能夠設定滑鼠移到資料點上所顯示的資訊，透過 `hover_data` 並給予指定欄位即可看到輸出。

```
fig = px.scatter(df_data, x="sepal_width", y="sepal_length", color="s
                       , hover_data=['petal_length','petal_width'])
HTML(fig.to_html())
```



Plotly Express 也提供三維的視覺化，此外使用者也能夠過控制變版自由的放大與縮小甚至旋轉。下圖範例中我們將 x 軸設定花萼寬度，y 軸設定花萼長度，z 軸設定花瓣寬度。此外 `size` 可以控制每一個資料點的大小，這裡採用花瓣的長度做為每個資料點大小的依據。因此從這個立體空間可以發現從花瓣長度對於花的種類有很強的關聯性。

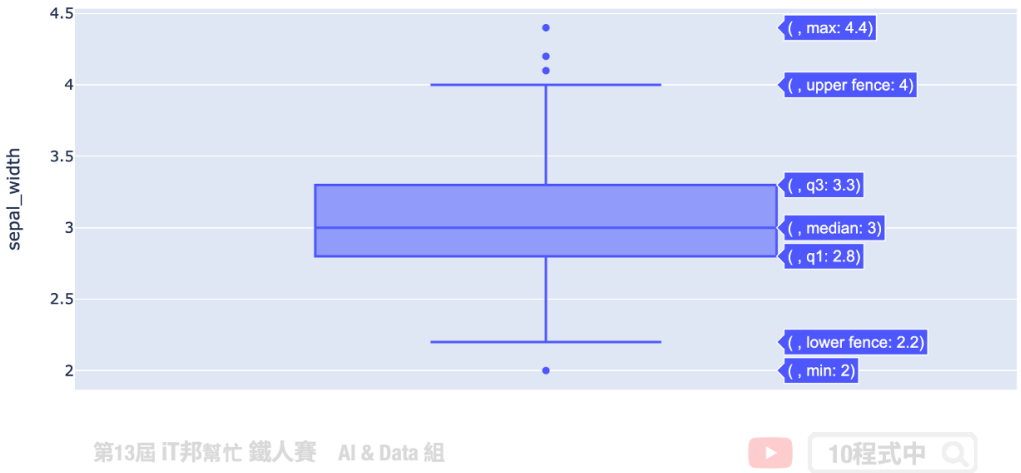
```
fig = px.scatter_3d(df_data, x="sepal_width", y="sepal_length", z="pe
                    color="species",size='petal_length')
HTML(fig.to_html())
```



箱形圖

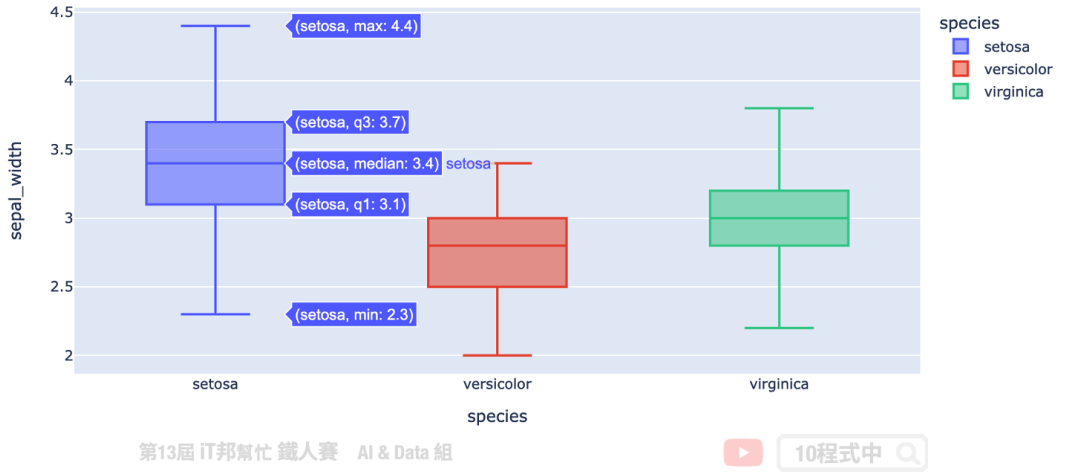
我們可以透過箱形圖進行特徵的分布狀況分析。從視覺化圖中可以清楚地知道花萼的寬度範圍介於 2~4.5 之間，以及四分位數和離群值的訊息。

```
fig = px.box(df_data, y="sepal_width")
HTML(fig.to_html())
```



除此之外我們能夠更近一步的分析花萼寬度對於每個品種的分布狀況。

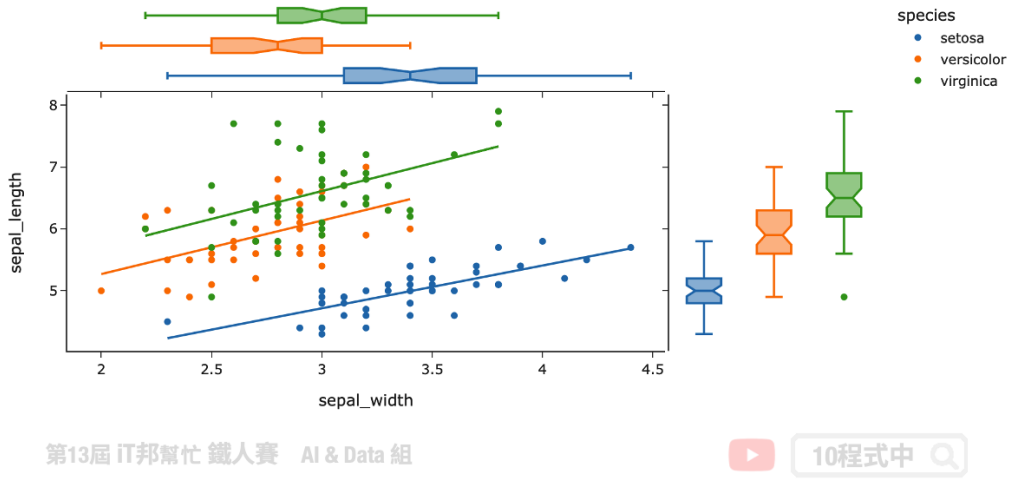
```
fig = px.box(df_data, x="species", y="sepal_width",color="species")
HTML(fig.to_html())
```



複合型視覺化技巧

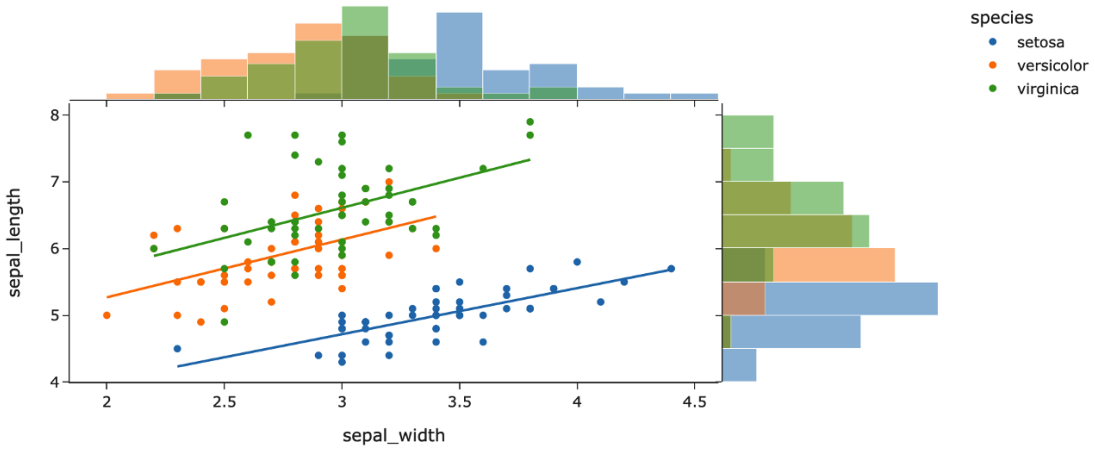
複合型視覺化方式可以同時預覽兩個變數間的散佈圖與箱型圖關係。主要是透過 `marginal_y` 與 `marginal_x` 設置橫軸與縱軸的視覺化方式，因此在一個圖表中可以結合兩種視覺化。此外 `trendline` 可以為散佈圖繪製趨勢線，設置 `ols` 會採用最小平方法位數據建立一個線性迴歸。

```
fig = px.scatter(df_data, x="sepal_width", y="sepal_length", color="s
           marginal_x="box", trendline="ols", template="simple_white"
HTML(fig.to_html())
```



也可以試著將橫軸與縱軸改成直方圖。

```
fig = px.scatter(df_data, x="sepal_width", y="sepal_length", color="s
           marginal_x="histogram", trendline="ols", template="simple_
HTML(fig.to_html())
```



匯出圖片

方法一

直接點選控制面板的相機圖示 (Download plot as a png) 可以立即下載圖片。

方法二

首先要安裝 kaleido 才能匯出 Plotly Express 的靜態圖片。

```
!pip install kaleido
```

- 匯出靜態圖片

```
fig.write_image("./demo.png")
```

- 匯出網頁格式，保留互動形式

```
fig.write_html("./demo.html")
```

Reference

- Plotly Express API Doc *(https://plotly.com/python/plotly-express/)*
- Plotly Express GitHub *(https://github.com/plotly/plotly_express)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 23] 資料分布與離群值處理

今日學習目標

- 資料特徵觀察與離群值分析
- 檢視資料的分布狀態
    - 偏度 (Skewness)
    - 峰度 (Kurtosis)
- 修正特徵偏度的方法

範例程式：  *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/23.資料分布與離群值處理/23.資料分布與離群值處理.ipynb)*

前言

資料前處理 (Data Preprocessing)，是機器學習中最重要的一部分。今日的內容可分為兩部份，前半部份算是一些對資料的觀察與分析，後半部主要是針對特徵 x 進行統計方法的資料分布觀察以及如何修正資料單峰偏左和偏右的常見方法。

載入資料

在今日的範例中我們採用波士頓房價預測的資料集。此資料集共有 506 筆資料。其中我們挑選兩個特徵來進行示範，分別有 LSTAT: 區域中被認為是低收入階層的比例、AGE: 1940年之前建成的自用房屋比例。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston

# 載入資料集
boston_dataset = load_boston()
# 將資料轉換成pd.DataFrame格式。目標輸出是MEDIV，剩下的就是特徵即為輸入特徵。
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.fea
boston['MEDV'] = boston_dataset.target
boston
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

我們可以透過 Pandas 的 `describe()` 方法先來查看每個特徵的平均數、標準差、四分位數以及最大值與最小值。

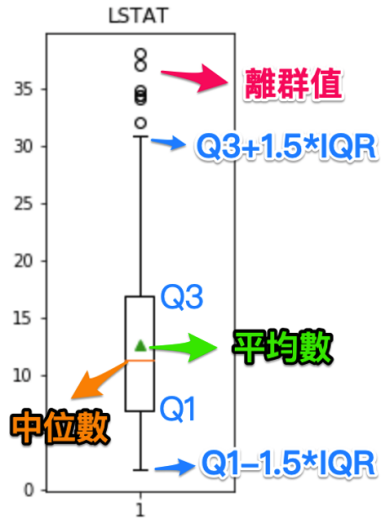
```
# 查看資料分布狀況
boston.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

離群值分析

以 LSTAT 特徵舉例。我們可以透過 boxplot 來查看該特徵在 506 筆資料中的分布狀況，我們可以看出平均值約 12，最大值接近 38，最小值接近 2。我們可以發現大於 32 以外有多個零散的數據點，這些資料我們可以來分析是否為異常點。因為這些異常點所造成的離群值可能會造成特徵的分布狀況嚴重的偏移。

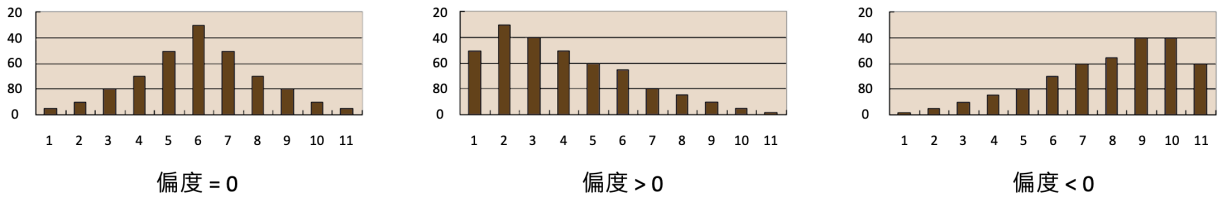
```
plt.figure(figsize=(2,5))
plt.boxplot(boston['LSTAT'],showmeans=True)
plt.title('LSTAT')
plt.show()
```



偏度 & 峰度

偏度 (Skewness)

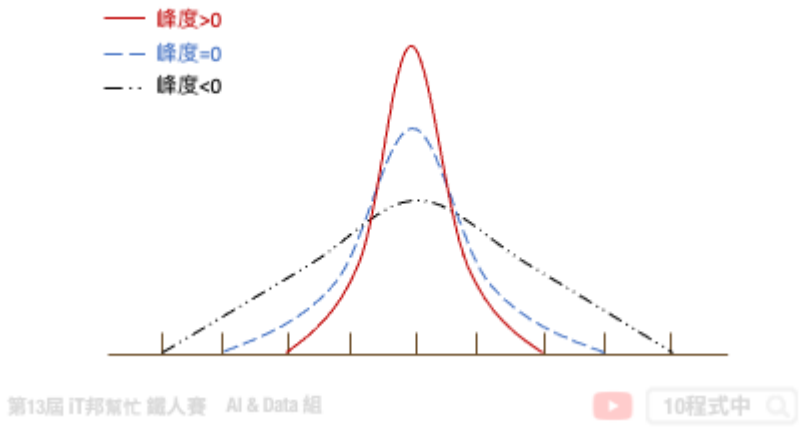
偏度 (Skewness) 是用來衡量資料分布的型態，同時也說明資料分配不對稱的程度。其判別方式如下：



- 右偏(正偏)，表示有少數幾筆資料很大，故平均數>中位數，所以偏度>0。
- 偏度=0 表示資料分布對稱，呈鐘形常態分布。
- 左偏(負偏)，表示有少數幾筆資料很小，故平均數<中位數，所以偏度<0。

峰度 (Kurtosis)

峰度 (Kurtossis) 可以反映資料的分布形狀。例如該資料是否比較高聳或是扁平的形狀。其判別方式如下：



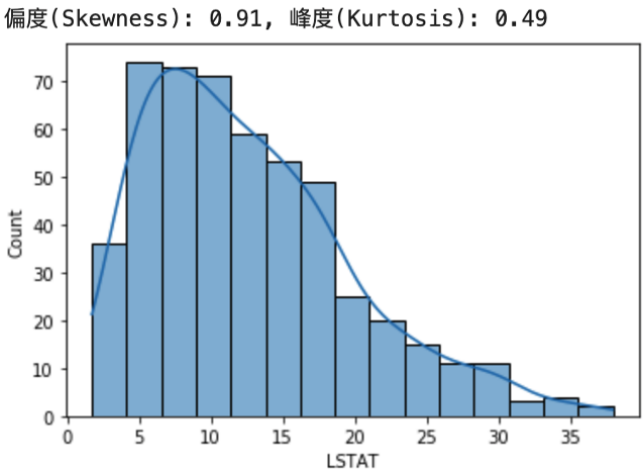
- 峰度>0 表示資料呈現高峽峰。
- 峰度=0 表示資料呈現常態峰。
- 峰度<0 表示資料呈現低潤峰。

分布狀態

LSTAT 特徵觀察

我們可以發現 LSTAT 特徵呈現右偏。透過 Pandas 計算該特徵的偏度與峰度。由結果可以得知偏度 `0.91>0` 呈右偏，而峰度 `0.49>0` 呈現高峽峰形狀。

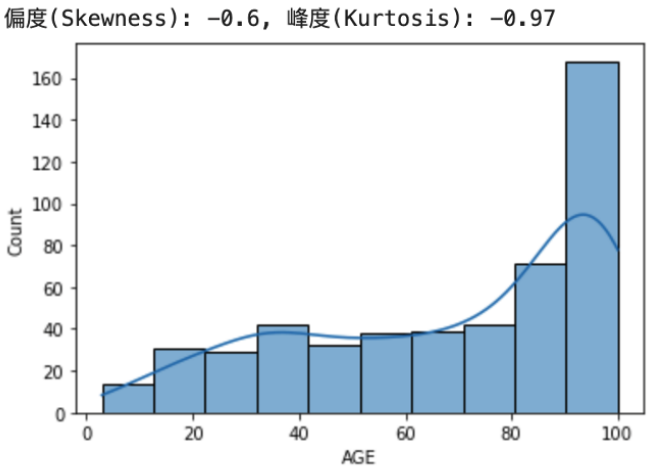
```
# 使用的資料是 LSTAT: 區域中被認為是低收入階層的比例
# skewness 與 kurtosis
skewness = round(boston['LSTAT'].skew(), 2)
kurtosis = round(boston['LSTAT'].kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(boston['LSTAT'], kde=True)
plt.show()
```



AGE 特徵觀察

我們可以發現 AGE 特徵呈現左偏。透過 Pandas 計算該特徵的偏度與峰度。由結果可以得知偏度 `-0.6<0` 呈左偏，而峰度 `-0.97<0` 呈現低潤峰形狀。

```
# 使用的資料是 AGE: 1940年之前建成的自用房屋比例
#skewness 與 kurtosis
skewness = round(boston['AGE'].skew(), 2)
kurtosis = round(boston['AGE'].kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(boston['AGE'], kde=True)
plt.show()
```



修正資料偏態的方法

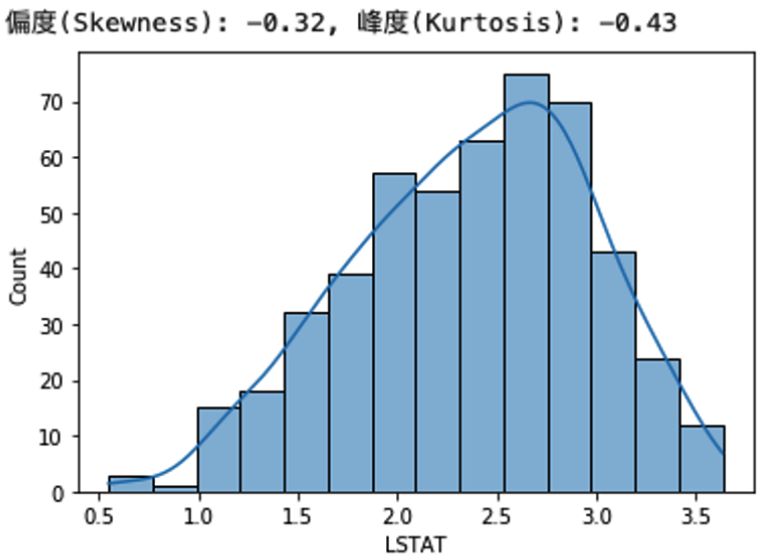
在數學統計或是機器學習中我們都會提出假設，前提是資料樣本是具有常態分佈。我們可以透過剛剛所講的偏度與峰度來評估特徵的分布狀態，或是透過直方圖與核密度估計視覺化查看資料分布。當資料呈現單峰偏斜時，我們會透過一些資料轉換技巧，讓所有資料能夠修正回常態分佈。以下整幾幾個常見的修正特徵偏度的方法：

- 對數轉換 (資料不能有0或負數)
- 平方根轉換 (資料不能是負數)
- 立方根轉換
- 次方轉換 (只能處理左偏)
- Box-Cox 轉換
- 移除離群值

對數轉換

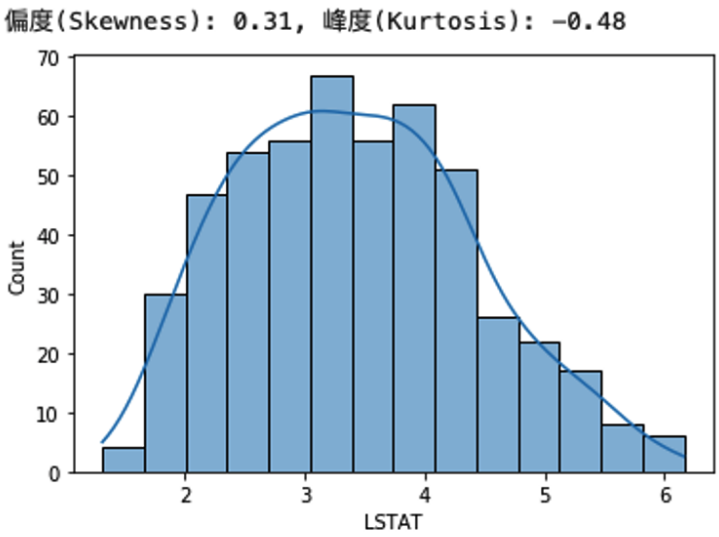
因為資料型態左偏，因此我們可以透過取對數來將資料拉回使為更集中。

```
transform_data = np.log(boston['LSTAT'])
# skewness 與 kurtosis
skewness = round(transform_data.skew(), 2)
kurtosis = round(transform_data.kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(transform_data, kde=True)
plt.show()
```



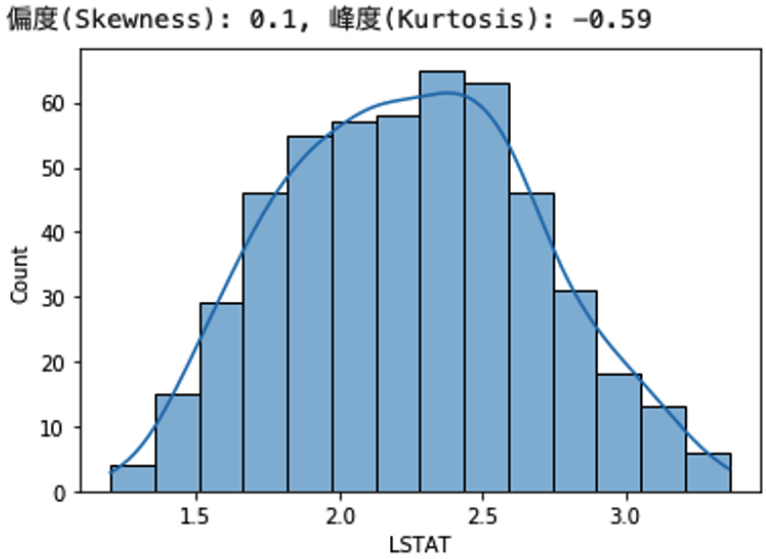
平方根轉換

```
transform_data = boston['LSTAT']**(1/2)
# skewness 與 kurtosis
skewness = round(transform_data.skew(), 2)
kurtosis = round(transform_data.kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(transform_data, kde=True)
plt.show()
```



立方根轉換

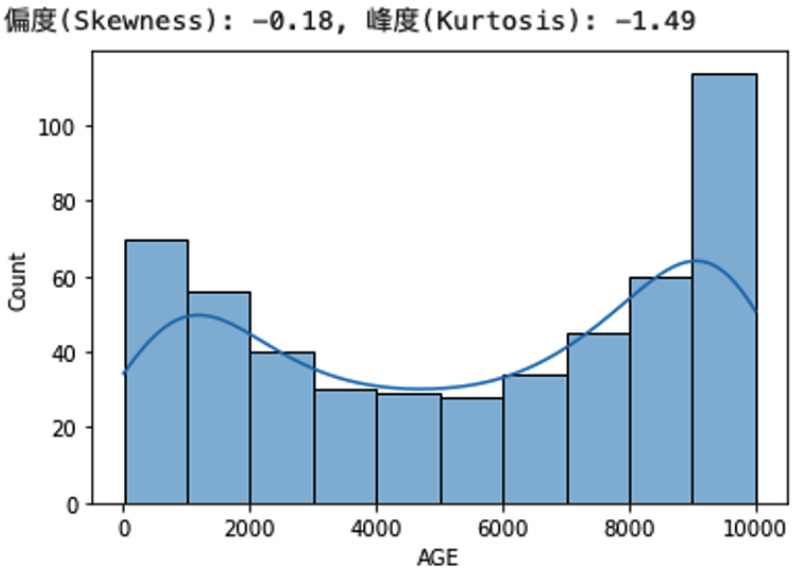
```
transform_data = boston['LSTAT']**(1/3)
# skewness 與 kurtosis
skewness = round(transform_data.skew(), 2)
kurtosis = round(transform_data.kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(transform_data, kde=True)
plt.show()
```



次方轉換

次方轉換僅能使用在偏左的資料上。

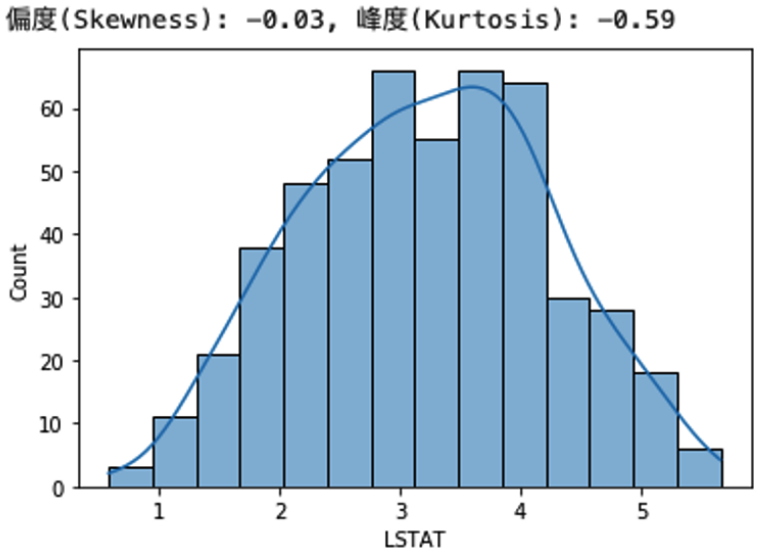
```
transform_data = np.power(boston['AGE'], 2)
# skewness 與 kurtosis
skewness = round(transform_data.skew(), 2)
kurtosis = round(transform_data.kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(transform_data, kde=True)
plt.show()
```



原本的資料分布低潤峰且有點雙峰的趨勢，因此轉換出來會有兩座山的感覺。

Box-Cox 轉換

```
from scipy.stats import boxcox
transform_data, lam = boxcox(boston['LSTAT'])
transform_data = pd.DataFrame(transform_data, columns=['LSTAT'])['LST
# skewness 與 kurtosis
skewness = round(transform_data.skew(), 2)
kurtosis = round(transform_data.kurt(), 2)
print(f"偏度(Skewness): {skewness}, 峰度(Kurtosis): {kurtosis}")

# 繪製分布圖
sns.histplot(transform_data, kde=True)
plt.show()
```



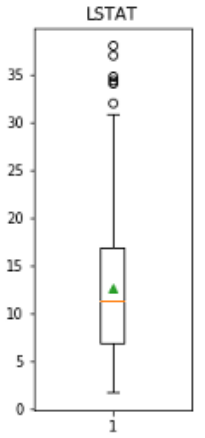
移除離群值

在 Q3＋1.5IQR（四分位距）和 Q1-1.5IQR 處畫兩條與中位線一樣的線段，這兩條線段為異常值截斷點，稱其為內限。在 Q3＋3IQR 和Q1－3IQR 處畫兩條線段稱其為外限。處於內限以外位置的點表示的數據都是異常值，其中在內限與外限之間的異常值為溫和的異常值（mild outliers），在外限以外的為極端的異常值 (extreme outliers)。

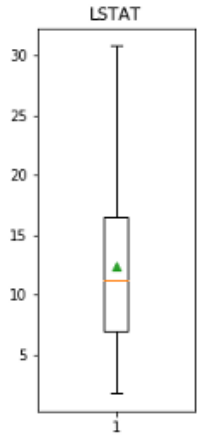
```
# 將所有特徵超出1.5倍IQR的概念將這些Outlier先去掉，避免對Model造成影響。
print ("Shape Of The Before Ouliers: ",boston['LSTAT'].shape)
n=1.5
#IQR = Q3-Q1
IQR = np.percentile(boston['LSTAT'],75) - np.percentile(boston['LSTAT
# outlier = Q3 + n*IQR
transform_data=boston[boston['LSTAT'] < np.percentile(boston['LSTAT']
# outlier = Q1 - n*IQR
transform_data=transform_data[transform_data['LSTAT'] > np.percentile
print ("Shape Of The After Ouliers: ",transform_data.shape)
```

我們必須將超出 1.5 倍的極端異常值清掉。共有 7 筆資料被移除掉。 輸出結果：

```
Shape Of The Before Ouliers:  (506,)
Shape Of The After Ouliers:  (499,)
```



Before



After

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 24] 機器學習 - 不能忽視的過擬合與欠擬合

今日學習目標

- 如何選擇最佳的模型？
- 深入理解度擬合與欠擬合
    - Bias-Variance Tradeoff
- 如何避免過擬合與欠擬合？

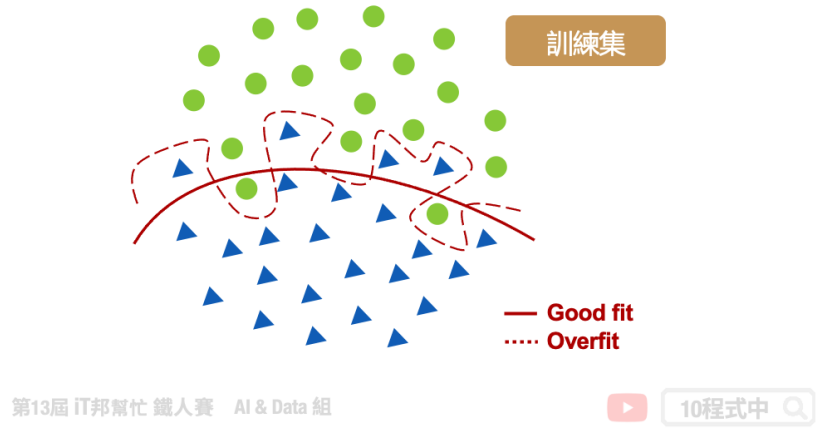
範例程式： 

前言

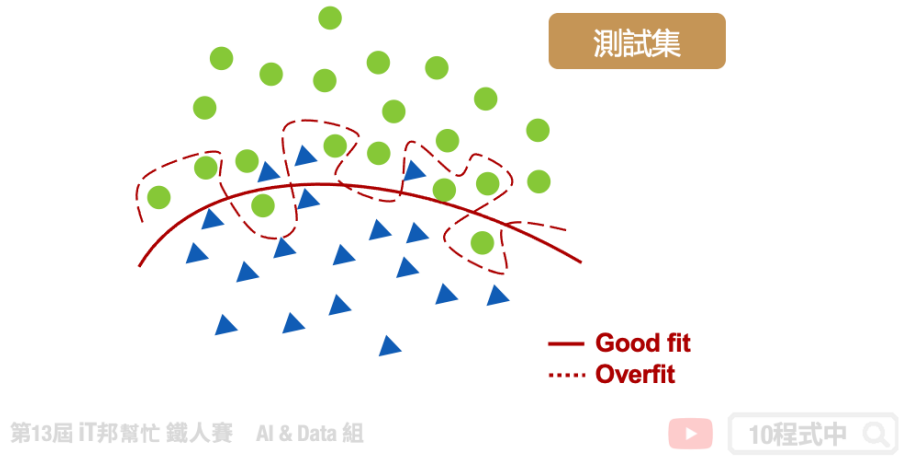
在機器學習中大家可能最常見的問題是，當訓練好了模型並在測試資料也獲得不錯的成績。於是很開心的落地並部署到真實場域中，殊不知預測出來的結果遠遠不如預期。我想這個痛點大家應該都經歷過，尤其是機器學習的新手。這種情況就是所謂的過度擬合，它是一個在機器學習領域中非常棘手的的問題。當你的模型過度的擬合訓練集，這意味著你的模型過於複雜的去記住所有現有的數據點，進而導致模型的泛化能力不佳，這不是我們期望的。所謂的模型泛化能力是指，當我利用訓練集訓練一個模型後再拿另一組模型沒看過的資料進行預測，最終的預測結果如果在沒看過的資料中依然保持不錯的表現我們就可以說此模型泛化能力強。今天我們將來詳細探討何謂過度擬合，以及該如何去解決它使得模型處於一個適當的狀態。

如何選擇最佳的模型？

通常我們希望預測出來的結果要與實際的數值越接近越好，也就是在模型訓練的過程中我們要想辦法最小化誤差使得模型的誤差越小越好。那麼我們該如何評估訓練出來的模型好壞呢？以下圖為例，假設我們要訓練一個二元分類器。最簡單的方法是找出一條線夠將這兩個類別完整地分開，然而這一條切割的線要長得怎樣才是好的模型呢？從下圖我們可以發現紅色虛線的模型完整的擬合於訓練資料，而紅色實線的模型相對的比較沒有那麼嚴厲，在兩個類別間適當的找出一條平滑的曲線來區隔兩類的資料。



接著我們拿測試資料進行模型預測，可以發現由於紅色虛線的模型已經完整記住了訓練集的趨勢，因此在新的沒看過的資料表現就沒有那麼好了。尤其是在兩類別分隔線附近的資料最能看出端倪。於是我們可以很確定紅色虛線的模型已經過度擬合訓練資料了。另外紅色實現的模型雖然在訓練集中有幾筆會預測錯誤，但是它再測試集資料中一樣保持穩定的預測能力。



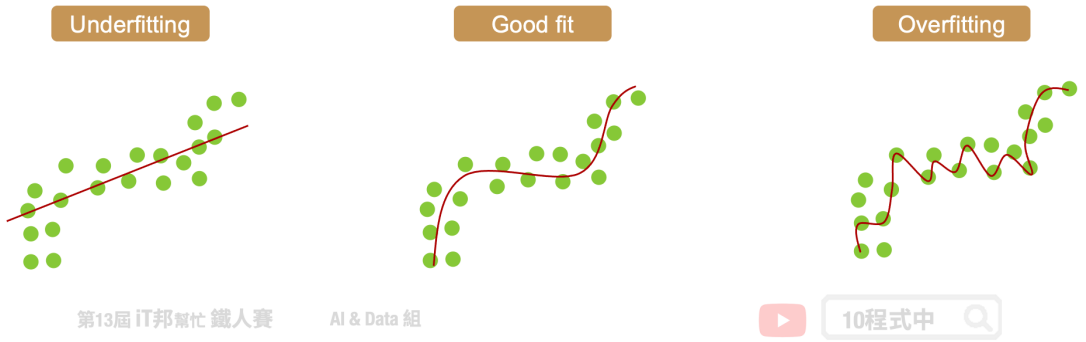
從上述的例子我們可以得知，在訓練模型時並非訓練集的誤差越小越好。我們必須同時拿測試集驗證模型的預測能力，目標是訓練集與測試集的平均誤差要越近越好。

一個適當的機器學習工作流程包括： - 切割訓練集與測試集 - 資料視覺化與前處理 - 尋找適合的模型 - 調整模型超參數 - 使用適當的指標評估模型 - 交叉驗證模型

Overfitting vs. Underfitting

過度擬合的反義就是欠擬合，從字面上可以得知模型預測能力是不好的。當模型太簡單時會發生欠擬合，或是加入太多的 L1/L2 正則化限制模型預測能力，使模型在從數據集中學習時變得不靈活。一個過於簡單的模型在預測中往往具有較小的方差(variance)而導致偏差(bias)就會變大。相反的過於複雜的模型會有較的變異進而導致方差大，同時偏差會變小。偏差和方差都是

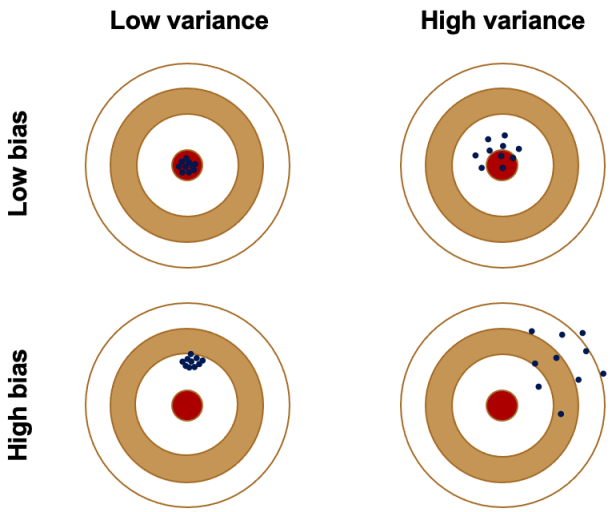
機器學習中的預測誤差的方式。在一般情況下我們可以減少偏差所引起的誤差，但可能會導致增加方差引起的誤差，反之亦然。



這裡我們就要來思考機器學習模型中的 error 從何而來？模型中的 error 是判斷一個模型的好壞依據，但其實我們可以將 error 拆分成兩大部分。分別有 Bias 與 Variance 兩個部分。以實際例子來說，假設輸出 y 是輸入 x 真正的答案，而 ŷ 則是透過模型 f(x) 訓練出來的預測值，我們希望預測的結果要與真實答案越接近越好，當 ŷ≠y 時就會產生 error (誤差)。

Bias-Variance Tradeoff

方差與偏差之間存在著一些關係，我們必須從中找到一個適當的平衡點。因此我們希望透過權衡 bias error 跟 variance error 來使得總誤差達到最小。我們常會以打靶例子解釋方差與偏差之間的關聯性。假設我們發射十次，我們說一個人的打靶技術很精準。其中的精就表示這十個把面上的點彼此間距離都相當近，也就是我的方差非常低(low variance)。另外所謂的準就表示這十個點都離準心很近，也就是我們的偏差非常低(low bias)。

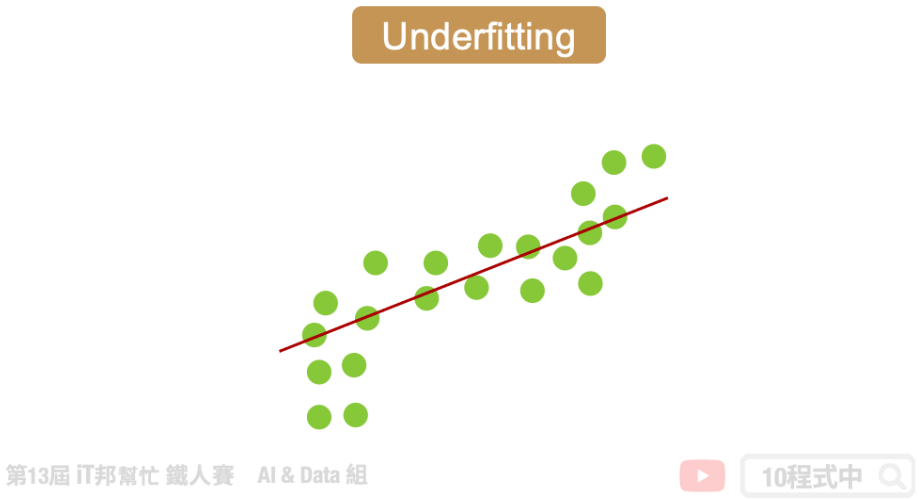


- Underfitting: 過於簡單的模型使得預測結果彈性不高，訓練集與測試集表現都不好。low variance (high bias)。

- Overfitting: 過於複雜的模型使得訓練集完整的被擬合，因此訓練集表現極好，但測試集表現不佳。high variance (low bias)。

Error from Bias

偏差(bias)就是模型的預測與真實值之間的差異。一般我們訓練模型是期望預測的值要與實際的答案要越接近越好。然而當一個簡單的線性模型可能無法完整地擬合到一個複雜非線性的資料集。因此如下圖所示，當一個模型訓練結果偏差過大我們可以得知該模型過於簡單。無論搜集再多的資料，線性的模型永遠無法擬合非線性的曲線。因為比較簡單的模型，他受到不同的資料的影響是比較小的。



- 簡單的模型有大的 bias，小的 variance。
- Error 來自於 bias 很大，稱為欠擬合。

Error from Variance

方差(variance)是指你的模型對於資料集的敏感程度。一個過於複雜的模型會導致輸出的變異性非常大。模型死背所有訓練集中的數據點會導致一個問題發生。當你的訓練資料有需多的隨機誤差或是離群值時，我們又把這些異常值全部擬合進模型裡面，導致學出來的模型過於複雜同時降低泛化能力，對於未知的資料預測的能力就會很差，同時造就了很高的 variance error。因此這樣的結果我們稱為過度擬合。

- 較複雜的模型有小的 bias，大的 variance。
- Error 來自於 variance 很大，稱為過度擬合。

如何避免欠擬合？

通常 bias 大而導致模型過於簡單，而無法擬合訓練資料。我們可以試著增加輸入的特徵，並做一些特徵工程讓模型觀察多點線索。或是調整模型的演算法，使模型更複雜。例如使用項次更高的多項式模型，或是 tree-based 模型中適當的增加樹的深度.....等。這裏更值得一提的是，當模型欠擬合時搜集再多的訓練資料是沒有用的。因為簡單的模型比較不會受資料的影響，所以 variance 相對的會比較低而 bias 大，也就是輸出的變化性不大。從這裡我們可以得知簡單的模型受到不同的輸入資料受到的影響是比較小的。因為模型選得不好，再怎麼訓練他的 bias 還是一樣大。

- 增加輸入特徵或特徵工程
- 提高模型複雜度

如何避免過度擬合？

當模型過於複雜過度擬合發生的機率相對提高，我們可以從訓練集與測試集觀察，很容易地檢測模型是否過度擬合。但是我們應該如何避免模型太過於複雜，而導致過度擬合發生呢？通常我們會診斷這些錯誤的來源，這些錯誤來自於兩種，分別為有 bias 與 variance。如果我們能夠診斷出這些錯誤的來源，我們就能挑出適當的方法來改善模型。以下幾點或許能夠幫助你進行建模：

- 搜集更多訓練資料
    - 增加訊練集的資料量是有效控制 variance 的方法，並且不會增加 bias。
- 模型添加 Regularization
    - 在損失函數中增加一些限制式，降低模型複雜。

- 交叉驗證
    - 從訓練集中切出驗證集，並挑出好的模型。而不是從測試集中求最小 error。
- Early Stopping
    - 設定當模型連續幾帶都無法改善 error，就立即終止訓練。
- Ensembling
    - 透過訓練多個模型，並取得每個模型預測並平均作為最終輸出。

Reference

- Overfitting in Machine Learning: What It Is and How to Prevent It *(https://elitedatascience.com/overfitting-in-machine-learning)*
- WTF is the Bias-Variance Tradeoff? (Infographic) *(https://elitedatascience.com/bias-variance-tradeoff)*
- 【機器學習】偏差與方差之權衡 Bias-Variance Tradeoff *(https://jason-chen-1992.weebly.com/home/-bias-variance-tradeoff)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 25] 交叉驗證 Cross-Validation 簡介

今日學習目標

- 常見的交叉驗證方法
  - K-fold
  - Leave one out cross validation
  - Random Subsampling
  - Bootstrap

前言

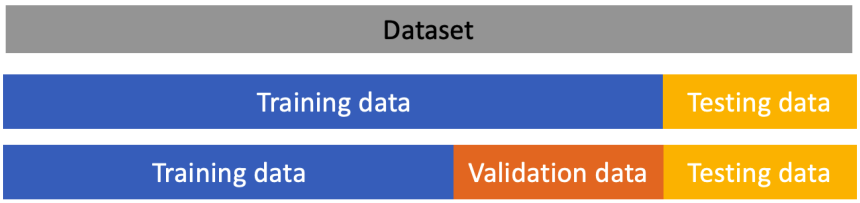
為了避免模型訓練發生過度擬合，通常我們還會從訓練集切一小部分資料出來進行驗證。驗證集的用處則是用來檢視模型在訓練過程中每次的迭代結果訓練的好不好。但該如何切出這個驗證集比較有公信力呢？如果我們僅切一小份的資料他是能有有效的評估訓練時模型的好壞嗎？在某些情況底下單純直接從資料集裡面切一塊出來當驗證集，是沒有辦法很有效的去評估一個模型訓練的好壞。說不定訓練出來的模型在這一份驗證集恰好表現得不錯，如果又隨機抽另一份資料來當驗證集說不定結果會變得很糟糕。這就表示模型泛化能力不足。為了避免這種情況發生並且有效的切割驗證集來評估模型，我們可以採用交叉驗證 Cross-Validation 的技巧來獲得最佳驗證。

什麼是交叉驗證？

在解釋交叉驗證之前我們先來討論將資料集切分為訓練集、測試集和驗證集的問題。在一般狀況下我們會將資料先切割成兩等份，分別為訓練集和測試集。其中在訓練階段模型只會對訓練集進行擬合，另外測試集的資料並未參與訓練，因此可以拿來當作最終評估模型的好壞。但是我們訓練的模型希望找到一個不錯的超參數，使得模型在訓練集和測試集都有不錯的成績，也就是說 loss 要越低越好。因此最常見的作法會將訓練資料再切出一個驗證集來找出一個最佳的模型參數，使得驗證集的表現要最好。但是為了避免模型對於我們所切的驗證集過度擬合，因此可已透過交叉驗證的方法對模型做更好的評估。所謂的交叉驗證簡單來說是將訓練資料進行分組，一部分做為訓練子集來訓練模型，另一部分做為驗證子集來評估模型。用訓練子集的數據先訓練模型，然後用驗證子集去跑一遍，看驗證集的損失函數(loss)或是分類準確率等。等模型訓練好之後，再用測試集去測試模型的性能。主要的交叉驗證法有以下幾個方法:

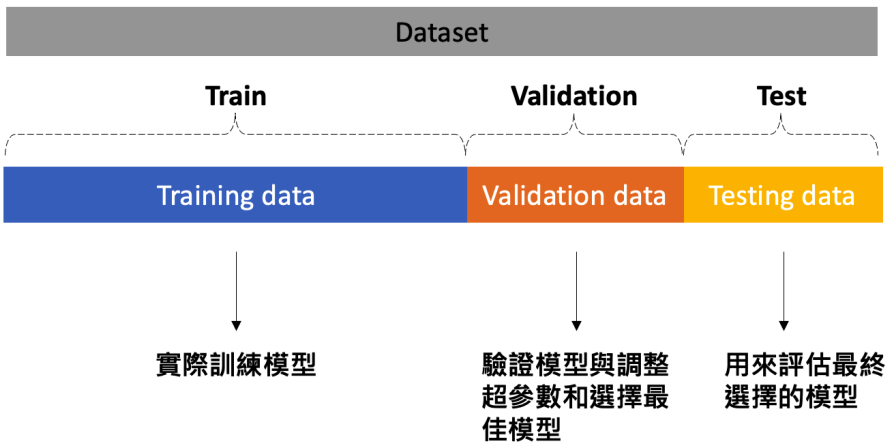
- Holdout
- K-fold
- Leave one out cross validation
- Random Subsampling

- Bootstrap



Holdout Method

此方法是最經典且最簡單實作的交叉驗證法，Holdout 顧名思義就是將資料切出一部分作為模型評估的依據。在這種方法中，我們將資料隨機分為三部分：訓練集、驗證集和測試集。其中只有訓練集資料實際參與訓練，其餘的資料僅拿來評估模型好壞。驗證集使用時機是在訓練過程中可以檢視訓練的趨勢，若有發現過擬合擬合跡象可以提早發現並解決。以及方便我們進行調整超參數以及選擇最佳的模型。當然僅透過驗證集不能代表全部，因此最後確定好模型時。我們會再拿事先切好的測試集進行最終的評估，檢視模型的泛化能力。



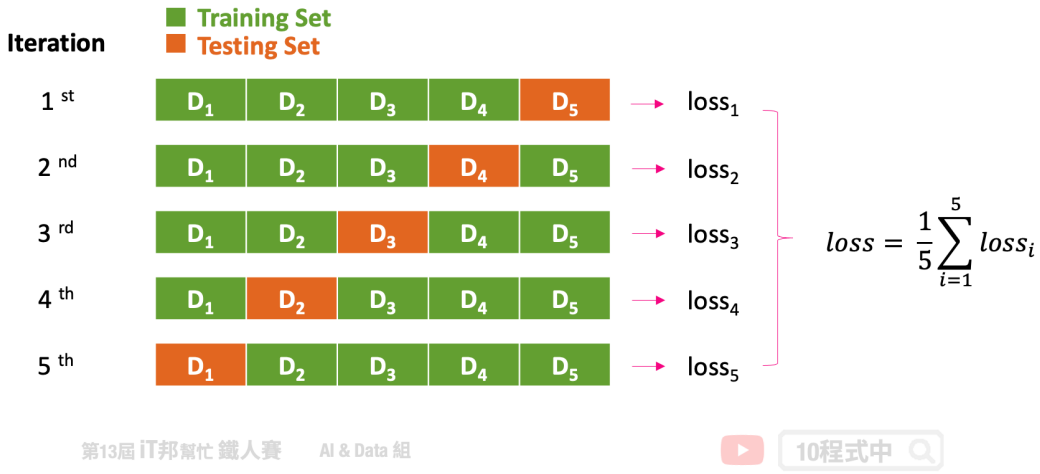
參考 (https://www.datavedas.com/holdout-cross-validation/)

優點: 1. 簡單實作。 2. 驗證集可以被拿來評估模型在訓練過程中的學習成果。 3. 測試集可以評估模型泛化能力。

缺點: 1. 當資料集變異量較大時，驗證集與測試集可能無法足以評估模型。 2. 不適合用在資料不平衡的資料集。

K-fold Cross-Validation

上一個方法雖然簡單，但是在訓練過程中僅切一份驗證集往往不能夠代表全部。因此我們可以透過一些技巧切割驗證集，使得訓練過程中有一個更公正的評估方式。我們可以透過 K-Fold 方法將訓練資料再依序切割訓練集與測試集，K-Fold 裡面的測試集可以當成驗證集。K-Fold 的方法中 K 是由我們自由調控的，在每次的迭代中會選擇一組作為驗證集，其餘 (k-1) 組作為訓練集。透過這種方式學習，不同分組訓練的結果進行平均來減少方差，因此模型的性能對數據的劃分就不會那麼敏感。



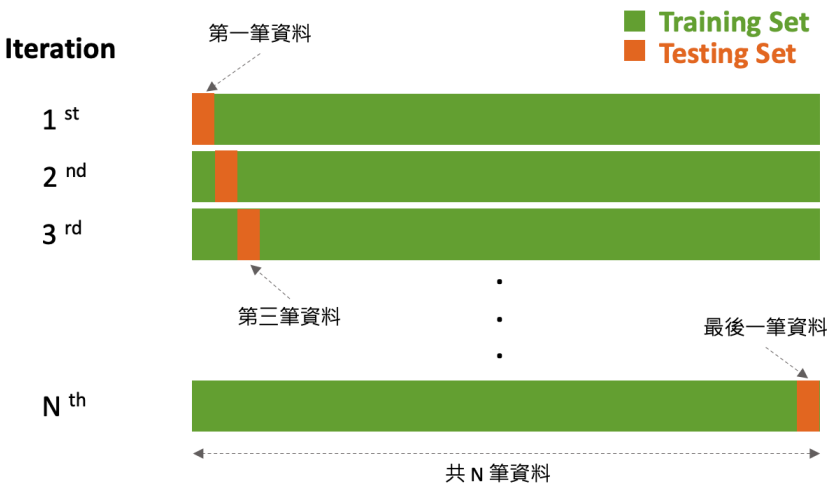
參考 (https://www.datavedas.com/k-fold-cross-validation/)

優點: 1. 降低模型訓練對於資料集的偏差。 2. 訓練集與驗證集完整被充分利用與學習。

缺點: 1. 不適合用於資料不平衡的資料集。 2. 如果要簡單的 K-fold 來尋找超參數會有資料洩漏問題導致訓練結果有偏差，因為在每個 Fold 中都會使用同一組資料進行驗證。 3. 在相同的驗證集計算模型的誤差，當找到了最佳的超參數。這可能會導致重大偏差，有過擬合擬合疑慮。

Leave One Out

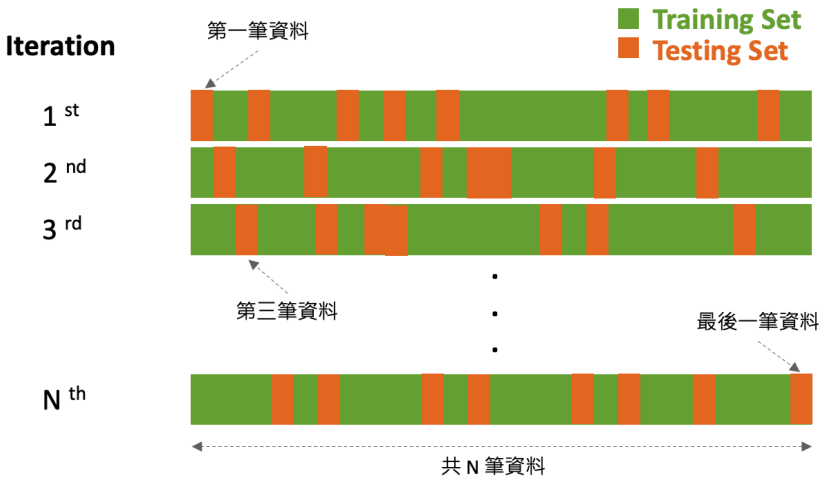
此方法是 K-fold 其中一種特例，當 K 等於資料集的數量時就等於 Leave One Out 方法。也就是在每次訓練時僅會把一筆資料當成測試資料，其餘的 N-1 筆資料作為訓練模型的資料。此作法相當簡單明瞭，但是訓練負擔會非常重且耗時。然而 Leave p-out 是另一種技巧，其中的 p 使用者可以自己設定每次訓練需要留幾筆資料作為測試集。



優點: 1. 簡單且容易理解，好實作。

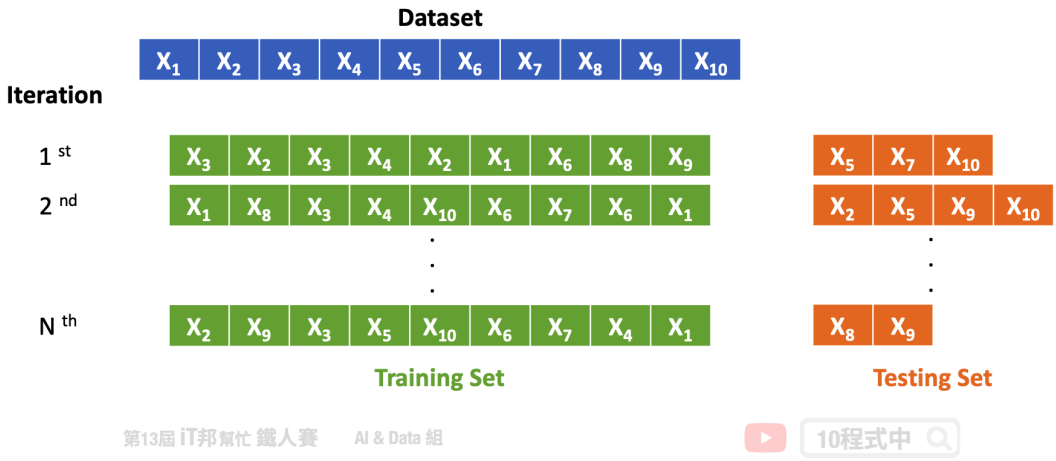
缺點: 1. 需要花費更多的訓練時間。

Random Subsampling



Bootstrapping

還有一種比較特殊的交叉驗證方式，Bootstrapping 自助抽樣法。是一種從給定訓練集中有放回的均勻抽樣，也就是說，每當選中一個樣本，它等可能地被再次選中並被再次添加到訓練集中。假設每次訓練都採樣十個樣本，在這十筆資料中很有可能會再次被隨機抽到。剩下沒有抽到的資料則都變成測試集，用來評估訓練完的模型。



小結

交叉驗證是訓練模型中非常重要的技巧，尤其是當手邊的資料集有限時更應該使用。透過交叉驗證技巧，即使在數據有限的情況下，我們也能夠獲得準確的結果，並且可以避免模型過度擬合。並為我們提供更準確的模型預測性能估計方式，同時也能夠提升模型的泛化能力。以上的方法可以直接使用 scikit-learn 裡面 model_selection 底下的 cross_val_score 方法進行實作。

> 本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 26] 交叉驗證 K-Fold Cross-Validation

今日學習目標

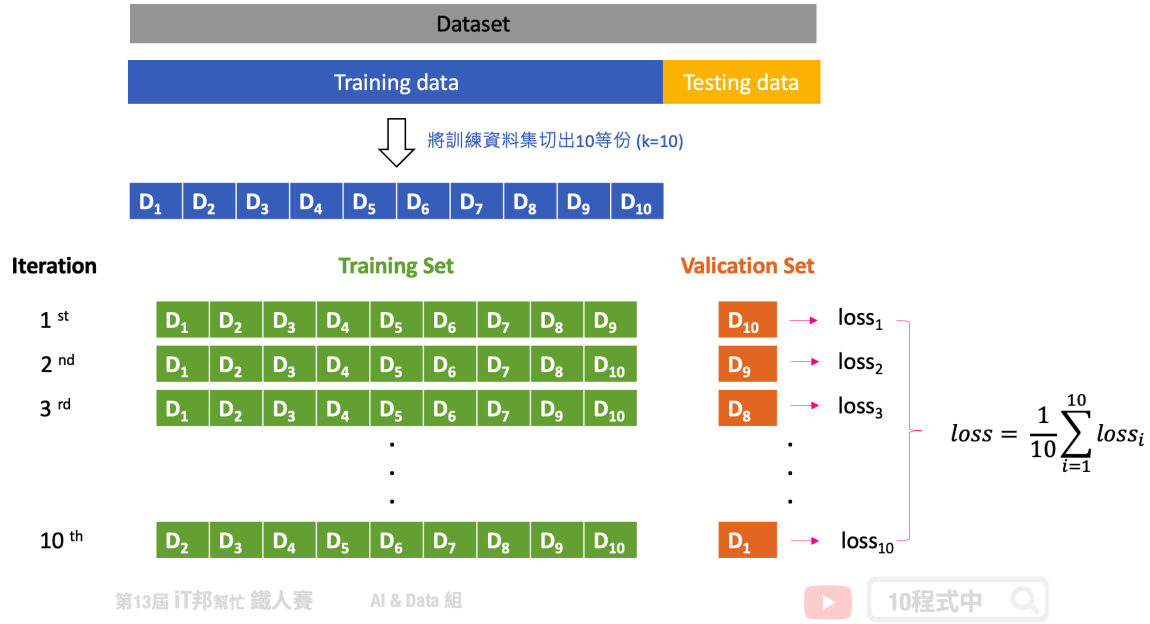
- 了解 K-Fold 各種不同變形
  - K-Fold Cross-Validation
  - Nested K-Fold Cross Validation
  - Repeated K-Fold
  - Stratified K-Fold
  - Group K-Fold

前言

交叉驗證又稱為樣本外測試，是資料科學中重要的一環。透過資料間的重複採樣過程，用於評估機器學習模型並驗證模型對獨立測試數據集的泛化能力。在今天的文章中我們將詳細的來介紹每一種 K-Fold 變型。

K-Fold Cross-Validation

在 K-Fold 的方法中我們會將資料切分為 K 等份，K 是由我們自由調控的，以下圖為例：假設我們設定 K=10，也就是將訓練集切割為十等份。這意味著相同的模型要訓練十次，每一次的訓練都會從這十等份挑選其中九等份作為訓練資料，剩下一等份未參與訓練並作為驗證集。因此訓練十回將會有十個不同驗證集的 Error，這個 Error 通常我們會稱作 loss 也就是模型評估方式。模型評估方式有很多種，以回歸問題來說就有 MSE、MAE、RMSE...等。最終把這十次的 loss 加總起來取平均就可以當成最終結果。透過這種方式，不同分組訓練的結果進行平均來減少方差，因此模型的性能對數據的劃分就不會那麼敏感。

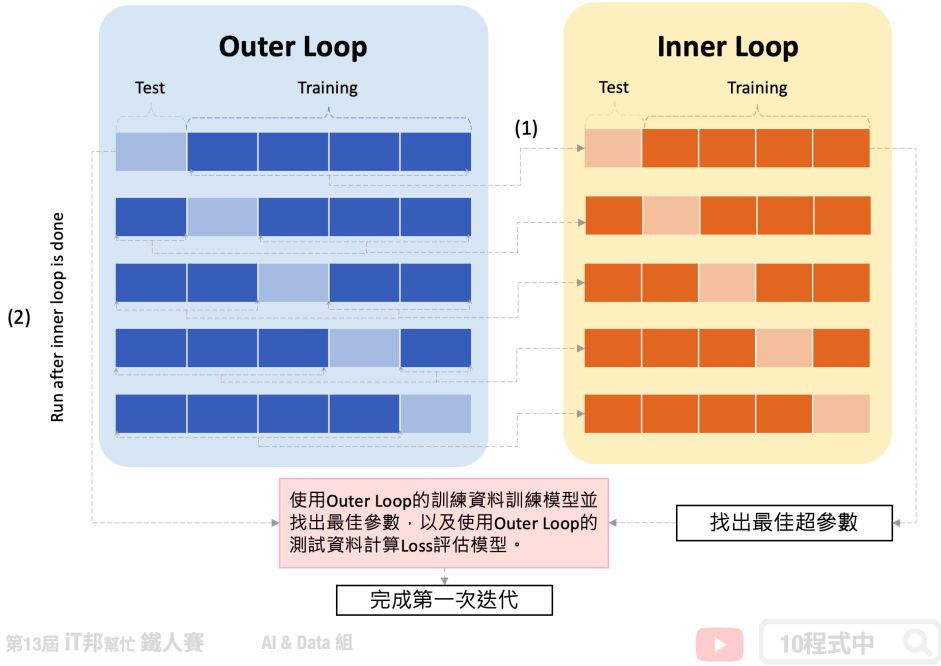


參考 *(https://www.datavedas.com/K-Fold-cross-validation/)*

- [scikit-learn] K-Fold *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html#sklearn.model_selection.KFold)*

Nested K-Fold Cross Validation

此方法為 K-Fold 的變型，Nested 意指雙迴圈(巢狀)的意思。分別有外層迴圈(Outer Loop)為一般正常的 K-Fold。唯一不同的是我們在每一次迭代中會將外層 K-Fold 的訓練集拿出來再進入到內層迴圈(Inner Loop)再做一次 K-Fold。由下圖可以看到，(1)我們可以在第一個外層回圈中將訓練資料又切為五份訓練集和測試集，內層圈透過 Grid Search 等演算法來尋找最佳超參數。等找到最好的模型超參數後，我們再拿(2)外層回圈的測試資料進行模型評估並計算 loss。最終我們會得到五個測試集 loss 的平均作為交叉驗證模型評估結果。



Algorithm 1: K-Fold Nested Cross-Validation with Random Search

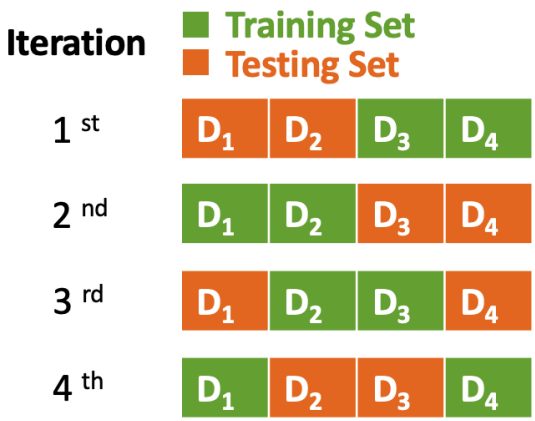
Require: K_1, K_2, where K_1 is number of outer folds and K_2 inner folds
Require: $\mathcal{D}$, dataset containing input features X and output feature y
Require: P_{sets}, set of hyperparameters with different values
Require: $\mathcal{M}$, a single estimator, model.

for $i = 1$ ***to*** K_1 *splits* **do**
  Split $\mathcal{D}$ into $\mathcal{D}_i^{train}, \mathcal{D}_i^{test}$ for the *i'th* split
  for $j = 1$ ***to*** K_2 *splits* **do**
    Split $\mathcal{D}_i^{train}$ into $\mathcal{D}_j^{train}, \mathcal{D}_j^{test}$ for the *j'th* split
    foreach p ***in*** $RandomSample(P_{sets})$ **do**
      Train $\mathcal{M}$ on $\mathcal{D}_j^{train}$ with hyperparameter set p
      Compute test error E_j^{test} for $\mathcal{M}$ with $\mathcal{D}_j^{test}$
  Select optimal hyperparameter set p^* from P_{sets}, where E_j^{test} is best
  Train $\mathcal{M}$ with $\mathcal{D}_i^{train}$, using p^*
  Compute test error E_i^{test} for $\mathcal{M}$ with $\mathcal{D}_i^{test}$

- [scikit-learn] Nested K-Fold *(https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html)*

Repeated K-Fold

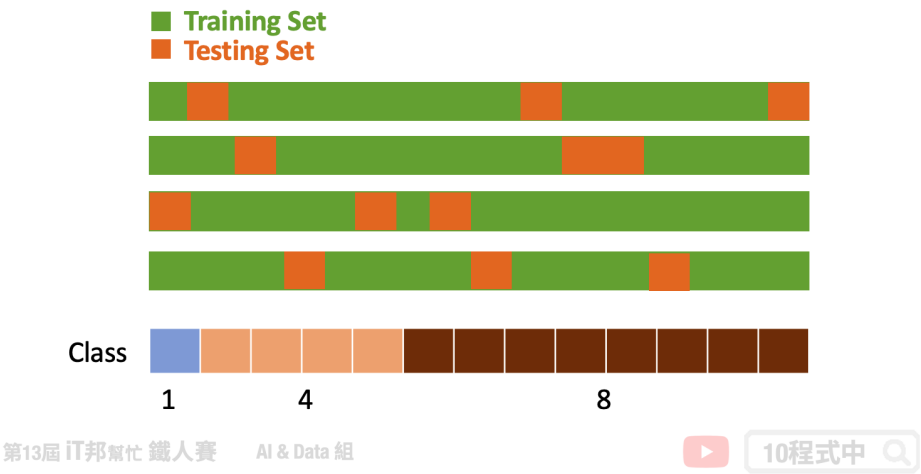
另一個 K-Fold 變型為 Repeated K-Fold 顧名思義就是重複 n 次 K-Fold cross-validation。假設 K=2、n=2 代表 2-fold cross validation，在每一回合又會將資料將會打亂得到新組合。因此最終會得到 4 組的資料，意味著模型將訓練四遍。此種方法會確保每次組合的隨機資料並不會重複。簡單來說執行 K-Fold 交叉驗證，然後重新洗牌數據，然後再次執行 K-Fold。



- [scikit-learn] RepeatedKFold *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedKFold.html#sklearn.model_selection.RepeatedKFold)*

Stratified K-Fold

分層交叉驗證，每個 Fold 都是按照類別的比例抽出來的。假設這個分類任務一共有三個類別A、B、C，它們的比例是1:4:8。那麼每個fold中的A、B、C的比例也必須是1:4:8。其實現方式也非常簡單，首先依序把A、B、C類別的數據隨機分成k組，最後再把它們合併依照比例起來，就得到了k組滿足1:2:10的數據了。



優點: 1. 優於一般的 K-Fold 因為test set能充分代表整體數據。 2. 預測結果的方差也會變小，使得交叉驗證的 error 更可靠。 3. 對於資料不平衡的數據很有用

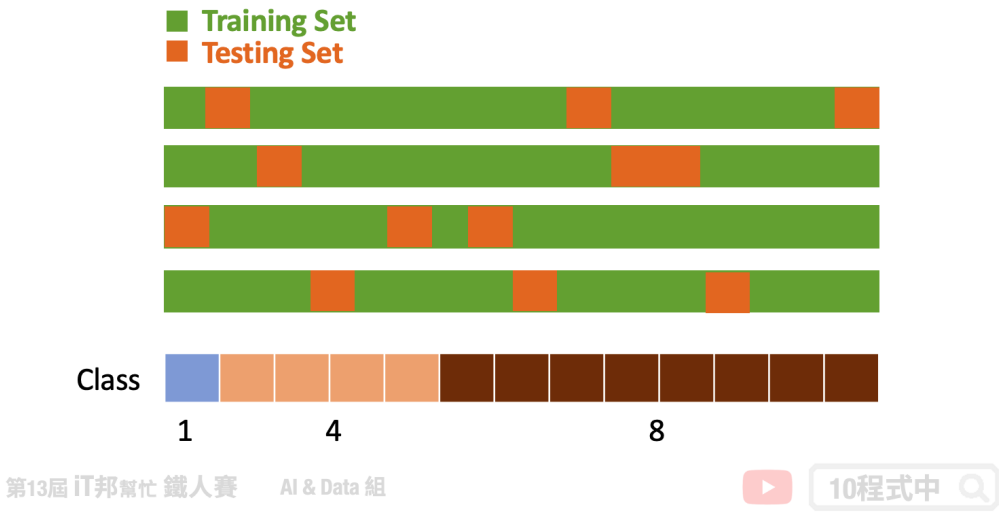
缺點: 1. 大多實例都以分類問題為主

- [scikit-learn] StratifiedKFold *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold)*

- [scikit-learn] StratifiedShuffleSplit *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html#sklearn.model_selection.StratifiedShuffleSplit)*

Group K-Fold

此做法為了避免取連續的資料而造成測試集或驗證集偏向某一特別的狀況而造成過度擬和訓練集，反而在未看過的資料下表現不好。Group K-Fold 為了避免此情況發生，它切割資料時有效的從資料集中每個區塊隨機挑選作為驗證集。同時保證每一個 Fold 的驗證集並不會重複的資料。假設你有三個類別，至少驗證集必須從三個不同的分組中抽樣取出，同時確保每一個 Fold 所抽出來的這三個分組並不會重複。



- [scikit-learn] GroupKFold *(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GroupKFold.html#sklearn.model_selection.GroupKFold)*

  本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 27] 機器學習常犯錯的十件事

今日學習目標

- 探討機器學習常犯的十個錯誤

前言

人工智慧近年來成為任何產業熱門的話題之一，各公司積極地導入機器學習技術協助產業 AI 化。例如：智慧醫療、智慧交通、智慧製造.....等。正是因為 AI 技術的創新與普及，訓練機器學習模型再也不是理工背景的人才能做的事。此外隨著 Python 開發社群茁壯，許多開源的 AI 套件如雨後春筍般的出現大大降低了機器學習建模的門檻。在今天的內容中我想藉由鐵人賽來跟大家分享機器學習常犯錯的十件事，並且從資料面與模型面的角度來探討機器學習應該注意的幾件事。尤其是在初學階段，因缺乏經驗往往會犯一些無可避免的錯誤。所以這篇文章將點出十個機器學習中常犯的隱形錯誤。

- 資料面
- 資料收集與處理不當
- 訓練集與測試集的類別分佈不一致
- 沒有資料視覺化的習慣
- 使用 LabelEncoder 為特徵編碼

- 資料處理不當導致資料洩漏

- 模型面
- 僅使用測試集評估模型好壞
- 在沒有交叉驗證的情況下判斷模型性能
- 分類問題僅使用準確率作為衡量模型的指標
- 迴歸問題僅使用 R2 分數評估模型好壞
- 任何事情別急著想用 AI 解決

1. 資料收集與處理不當

機器學習首要的步驟是定義問題，當確定目標與方向後即可開始搜集資料。相信大家都知道現實生活中的資料得來不易，即使從資料庫取得了這些資料後我們還需要花大量的時間進行資料清洗。所謂的資料清洗是資料庫當中可能會有缺失值，例如：NA、Inf、NaN、NULL。

- NA：表示缺失值，是 Not Available 的縮寫。
- Inf：表示無窮大，是 Infinite 的縮寫。

- NaN：表示非數值，是 Not a Number 的縮寫。
- NULL：表示空值，即沒有內容。

當資料都完成了前處理後，即可開始建立模型與評估模型。但是當訓練出來的模型表現不好有很多的因素。大家最常做的是替換模型演算法，或是嘗試不同的模型超參數取得一個最佳的結果。但是在進行這些做之前，建議大家先把關注的點回到資料處理面。模型訓練不好的其中一個因素是資料的標籤收集不當。Landing.ai 執行長吳恩達也曾經說過當一個小資料集存在著錯誤標籤時，模型很難給出一個正確的輸出。因為資料間夾帶了雜訊往往會使的模型存在著一些偏差，導致訓練結果不穩定。因此筆者建議模型訓練不好的時候，可以回頭觀察資料是否存在一些錯誤。而不是一昧的調整模型演算法與超參數。

2. 訓練集與測試集的類別分佈不一致

在分類的資料中，初學者常見的錯誤是忘記使用分層抽樣（stratify）來對訓練集和測試集進行切割。當測試集的分佈盡可能與訓練相同情況下，模型才更有可能得到更準確的預測。然而在分類的問題中，我們必須更關注每個類別的資料分佈比例。以下舉個例子：假設我們有三個標籤的類別，而這三個類別的分佈比例分別為 4:3:3。同理我們在進行資料切割的時候必須確保訓練集與測試集需要有相同的資料分佈比例。

大家應該都使用過 Sklearn 的 `train_test_split` 進行資料切割。在此方法中 Sklearn 提供了一個 `stratify` 參數達到分層隨機抽樣的目的。特別是在原始數據中樣本標籤分佈不均衡時非常有用，一些分類問題可能會在目標類的分佈中表現出很大的不平衡：例如，負樣本與正樣本比例懸殊(信用卡盜刷預測、離職員工預測)。以下用紅酒分類預測來進行示範，首先我們不使用 `stratify` 隨機切割資料並查看資料切割前後的三種類別比例。

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

X, y = load_wine(return_X_y=True)

# 查看全部資料三種類別比例
pd.Series(y).value_counts(normalize=True)
```

```
# 全部資料三種類別比例
1    0.398876
0    0.331461
2    0.269663
dtype: float64
```

```
# 實驗一：不使用 stratify 進行切割資料
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
# 查看訓練集三種類別比例
pd.Series(y_train).value_counts(normalize=True)
# 查看測試集三種類別比例
pd.Series(y_test).value_counts(normalize=True)
```

```
# 訓練集三種類別比例
1    0.390977
0    0.330827
2    0.278195
dtype: float64

# 測試集三種類別比例
1    0.511111
0    0.266667
2    0.222222
dtype: float64
```

從上面切出來的訓練集與測試集可以發現三個類別的資料分佈比例都不同。因此我們可以使用 `stratify` 參數再切割一次。

```
# 實驗二: 使用 stratify 進行切割資料
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)

# 查看訓練集三種類別比例
pd.Series(y_train).value_counts(normalize=True)
# 查看測試集三種類別比例
pd.Series(y_test).value_counts(normalize=True)
```

```
# 訓練集三種類別比例
1    0.400000
0    0.333333
2    0.266667
dtype: float64

# 測試集三種類別比例
1    0.398496
0    0.330827
2    0.270677
dtype: float64
```

我們可以發現將 `stratify` 設置為目標 (y) 在訓練和測試集中產生相同的分佈。因為改變的類別的比例是一個嚴重的問題，可能會使模型更偏向於特定的類別。因此訓練資料的分佈必須要與實際情況越接近越好。

3. 沒有資料視覺化的習慣

資料視覺化的好處多多，在本系列文章 [Day 3] 你真了解資料嗎？試試看視覺化分析吧！*(https://ithelp.ithome.com.tw/articles/10264416)* 與 [Day 22] Python 視覺化解釋數據 - Plotly Express *(https://ithelp.ithome.com.tw/articles/10277258)* 講解了許多 Python 資料視覺化的技巧。資料視覺化可以幫助我們分析與統計資料的型態，往往有好的資料清洗與前處理對模型預測結果會有大幅的提升。有興趣的讀者可以參考安斯庫姆四重 (Anscombe's quartet) *(https://zh.wikipedia.org/wiki/%E5%AE%89%E6%96%AF%E5%BA%93%E5%A7%86%E5%9B%9B%E9%87%8D%E5%A5%8F)* 。他主要是是透過四個小資料集並透過視覺化與統計來觀察，並說明在分析數據前先繪製圖表的重要性，以及離群值對統計的影響之大。

4. 使用 LabelEncoder 為特徵編碼

通常我們要為類別的特徵進行編碼，直覺會想到 Sklearn 的 LabelEncoder *(https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)*。但是如果一個資料集中有多個特徵是屬於類別型的資料，豈不是很麻煩?必須要一個一個呼叫 LabelEncoder 分別為這些特徵進行轉換。如果你看到這邊有同感的，在這裡要告訴你事實並非如此！我們看看 在官方文件下 LabelEncoder 的描述：

> This transformer should be used to encode target values, i.e. y, and not the input X.

簡單來說 LabelEncoder 只是被用來編碼輸出項 y 而已的！你還在用它來編碼你的每個 x 嗎？（暈

那麼我們該用什麼方法來編碼有順序的類別特徵呢？如果你仔細閱讀有關編碼分類特徵的 Sklearn 用戶指南，你會看到它清楚地說明：

> To convert categorical features to integer codes, we can use the OrdinalEncoder. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1)

看到這邊大家應該知道閱讀官方文件的重要性吧！官方文件中建議 x 項的輸入特徵可以採用 OrdinalEncoder *(https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html)* 一次為所有特徵依序做 Label Encoding。OrdinalEncoder 編碼器的使用方式如下：

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)

print(enc.categories_)
enc.transform([['Female', 3], ['Male', 1]])
```

```
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=obje
array([[0., 2.],
       [1., 0.]])
```

以上的範例是 X 有三筆資料，每筆資料都有兩個特徵。我們可以發現第一個特徵是性別 Male 與 Female，因此 OrdinalEncoder 會依造字母開頭做排序 Female 編碼為 0 而 Male 編碼為 1。另外第二個特徵為數字 1、2、3，同理依序為他們編碼成 0、1、2。只需閱讀官方文檔和用戶指南，你就可以了解很多關於 Sklearn 的知識！是不是很棒～

5. 資料處理不當導致資料洩漏

資料洩漏 (data leakage) 是個隱形殺手，它會在不知不覺中影響模型預測結果。其發生的時機在於你在訓練過程中，不應該將測試的資料的資訊洩漏到訓練過程中。它會造成模型給出一個非常樂觀的結果，即使在交叉驗證中也是如此，但在對實際新數據進行測試時表現會非常地糟糕。

資料洩漏最常發生於資料前處理的階段，尤其是當你的訓練集和測試集尚未切割的時候。Sklearn 提供了許多資料前處理的方法，例如: 缺失值補值(imputers)、正規化 (normalizers)、標準化(standardization)以及對數(log) 轉換...等。這些轉換器都會依賴於你輸入資料的分佈，並依照此分佈做相對應的擬合。

舉例來說，我們在做標準化時(StandardScaler)透過從每筆資料中減去平均值並將其除以標準偏差來獲得縮放後的數據。我們使用 `fit()` 方法在所有資料集 X 上做轉換，並使得轉換器學習每個特徵的整個分佈的平均值和標準差。這些資料轉換後如果再將這些數據拆分為訓練集和測試集，則訓練集會受到污染。因為 StandardScaler 從實際分佈中洩露了測試集重要訊息，一般來說我們不能將測試集的分佈情況與訓練集混在一起。雖然我們希望訓練集的分佈與實際測試集的分佈要越接近越好，因為使得模型表現結果穩定。

雖然我們把測試集與訓練集混在一起並做轉換，這一步驟對我們來說可能沒什麼。但是對於 Sklearn 強大的演算法，可能會透過這個遺漏測試集的分佈的訊息把模型擬合的很好。屆時模型訓練完成後，測試集不夠新穎，無法在實際看不見的數據上測試模型的性能。

最簡單的解決辦法，就是不要使用 `fit()` 一次轉換所有的資料。在做任何資料轉換之前要先確保訓練集與測試集已經完整地被切開。即使切開後也不要再拿測試集呼叫 `fit()` 或 `fit_transform()`，這一樣會導致相同問題發生。因為訓練集和測試集必須進行相同的轉

換，依照官方的範例我們必須先使用 `fit_transform()` 在訓練集上進行擬合與轉換。這確保了轉換器僅從訓練集學習，從中找出參數例如平均值與變異數並同時對其進行變換。接著使用 `transform()` 方法在測試資料上進行轉換，根據從訓練數據中學到的訊息進行轉換。

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

更強大的解決方案是使用 Sklearn 內建的 pipeline，它能夠保護模型免於資料洩漏的問題。此方法能夠確保訓練資料僅參與轉換擬合與模型訓練，而測試資料僅用於計算並驗證模型。

---

6. 僅使用測試集評估模型好壞

如果你的測試資料 R2 score 得到了 0.85 就代表很好了嗎？不盡然！儘管有高的測試分數通常意味著模型表現佳，但在解釋測試結果時仍有一些重要的注意事項。首先最重要的，無論分數值如何測試集的分數一定要與訓練集相比較才能確保模型訓練好與壞。當你的模型訓練集分數高於測試集的分數，並且兩者都足夠高以滿足專案的目標期望時這代表你訓練了一個好模型。然而這並不意味著訓練和測試分數之間的差異越大越好。舉個例子，若訓練集的 R2 score 為 0.85 測試集為 0.8 即代表模型既不過度擬合(overfit)也不欠擬合(underfit)。但是如果訓練集 0.9 測試集 0.8 的時候，你的模型就是過擬合。其原因是該模型沒有在訓練期間進行泛化，而是記住了一些訓練數據，從而導致測試分數低得多。

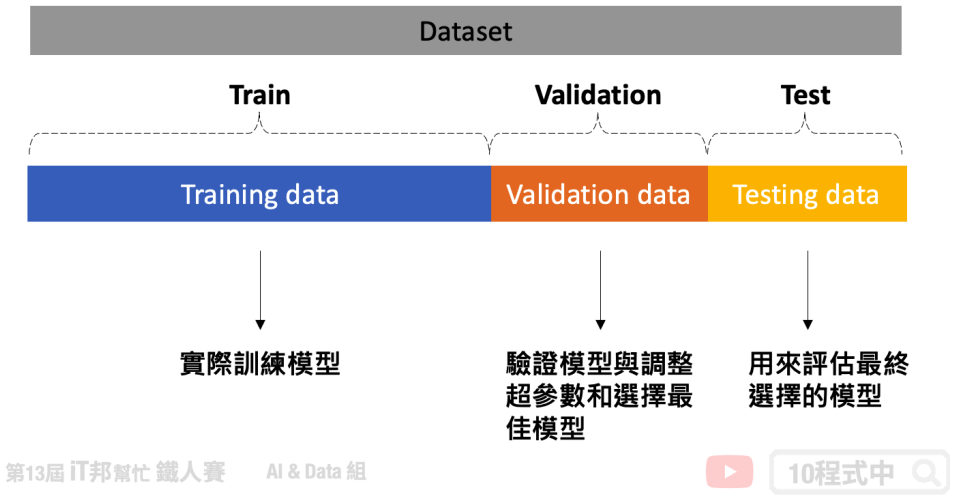
在大多數任務中你將會看到許多人使用 tree-based 模型或是整體學習模型 (ensemble models)。例如在隨機森林演算法當中如果它們的樹深度太深，往往會獲得非常高的訓練分數，從而導致過度擬合。另外也有測試集的分數比訓練集高的情況，若發生此情況時通常都會感覺是不是做錯了什麼。這種情況的主要原因是資料洩漏，也就是上一節我們討論的情況。或是你的測試資料筆數太少，沒辦法足以驗證模型好壞。

另外有時候我們也會得到在訓練集有很好的表現但測試集無敵差的情況。當訓練和測試分數差異很大時，問題往往與測試集有關而不是過度擬合。這時候你可能要檢查資料預處理的方式是否一致 (像是取 log 或 scale)，或是只是忘記對測試集做轉換處理。

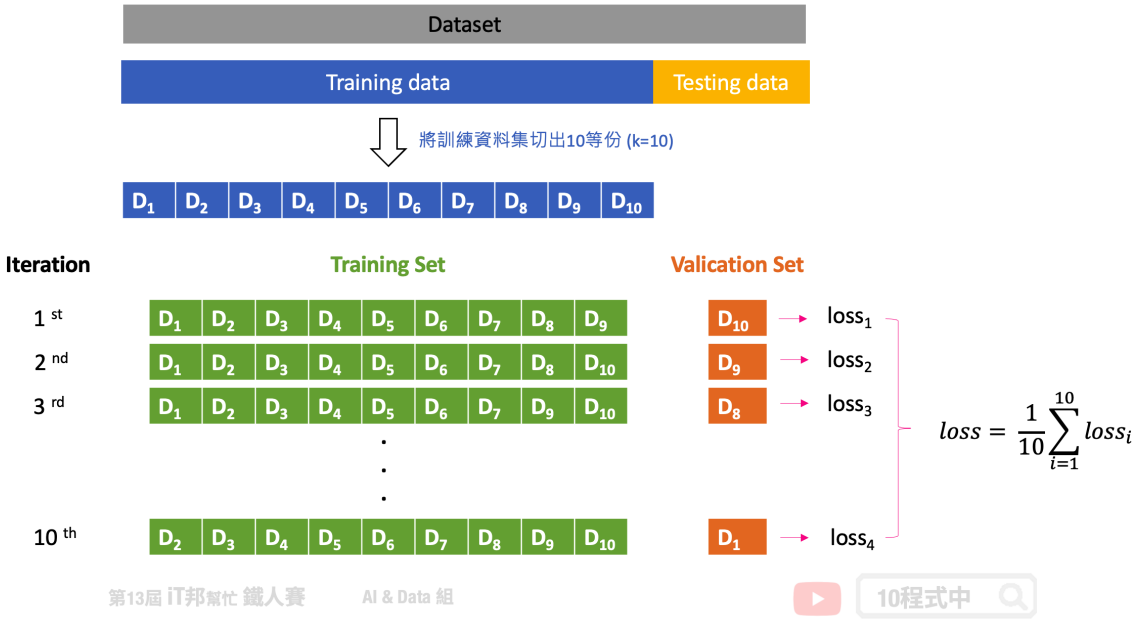
這裡做一個小結，總之在訓練好模型時請仔細檢查訓練和測試分數之間的差距。並且可以透過此評估方式檢視模型是否過擬合，同時也能進行模型條參或是選擇最佳的資料預處理方式。並為最終的模型做最佳的準備。

7. 在沒有交叉驗證的情況下判斷模型性能

我想大家應該都熟練掌握了 overfitting 這個議題。這是機器學習中一個迫切問題，並已經設計了無數個方法來解決它。最基本的方法是將一部分數據作為測試集來模擬和測量模型在看不見的數據上的性能。但是我們可以調整模型的超參數，直到模型在該特定測試集上達到最高分數，這又意味著某種含義的過度擬合。因此我們可以會將完整數據的另一部分作為驗證集再次解決這個問題。模型將在訓練數據上進行訓練，並在驗證集上微調其參數，並在測試集上進行最終評估。



但是將我們寶貴的數據分成三組意味著模型可以學習的數據量更少。此外模型的整體預測性能將取決於那對特定的訓練集和驗證集。因此在進行機器學習時最常使用 K-Fold cross-validation 解決上述問題。詳細內容可以參考我的前兩天文章[Day 25] 交叉驗證 Cross-Validation 簡介 *(https://ithelp.ithome.com.tw/articles/10278851)*以及[Day 26] 交叉驗證 K-Fold Cross-Validation *(https://ithelp.ithome.com.tw/articles/10279240)*。根據我們設定的 K 值，可以完整的將數據被分成 K 組 folds，對於每個 folds 每次模型訓練會把 K-1 組作為訓練集，而剩下的被歸類為驗證集。當模型交叉驗證結束後，訓練集所有資料會被完整的訓練。



8. 分類問題僅使用準確率作為衡量模型的指標

在預設的情況下所有 Sklearn 分類器在呼叫 `score()` 函數時都使用準確度作為評分方法。由於準確率的計算方式簡單與容易理解，因此經常會看到初學者廣泛使用它來判斷其模型的性能。不幸的是這種一般準確率的評估方式只對類別平衡的二元分類問題有用。

然而在其他的狀況下它是一個誤導性的指標，即使是表現最差的模型也可能背後隱藏著高準確度的分數。舉例來說有個偵測垃圾郵件的模型它的準確率 90%，但是實際上它根本無法偵測到垃圾郵件。這是為什麼？由於垃圾郵件並不常見，分類器可以檢測所有非垃圾郵件，即使分類器完全無法達到其目的這也可以提高其準確性。因為這個分類器僅可以分類這些正常郵件，稀少的垃圾郵件根本變認不出來。

對於多元類分類的問題更是應該注意你的模型評估指標。如果達到 80% 的準確率，是否意味著模型在預測類別1、類別2、類別3甚至所有類時一樣準確呢？一般的準確率永遠無法回答此類問題，但幸運的是其他分類指標提供了更多的訊息指標。它就是混淆矩陣 *(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)*(confusion matrix)。

```
from sklearn.metrics import confusion_matrix
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
```

```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

組成混淆矩陣的四個元素分別有 TP、TN、FP、FN。基本上混淆矩陣會拿這四個指標做參考，同時算出來的分數也更能去評估你的模型訓練的結果。此外我們可以利用混淆矩陣來計算 Precision、Recall、Accuracy 等分數。

- TP(True Positive): 正確預測成功的正樣本，例如真實答案(Ground True)是貓，成功的把一張貓的照片預測成貓，即為TP
- TN(True Negative): 正確預測成功的負樣本，成功的把一張狗的照片標示成不是貓，即為TN
- FP(False Positive): 錯誤預測成正樣本，實際上為負樣本，例如：錯誤的把一張狗的照片預測成貓
- FN(False Negative): 錯誤預測成負樣本，實際上為正樣本，例如：錯誤的把一張貓的照片預測成不是貓

9. 迴歸問題僅使用 R2 分數評估模型好壞

在預測連續性數值輸出的迴歸模型中，大家往往會直接呼叫模型提供的評估方法直接計算 `score`。然而這個分數在迴歸模型中是計算 R2 分數，又稱判定係數 (coefficient of determination)。所謂的判定係數是輸入特徵 (x) 去解釋輸出 (y) 的變異程度有多少，其計算公式是：迴歸模型的變異量 (SSR)/總變異量 (TSS) 。用以下變異數分析表（ANOVA table）來說 TSS 就是計算總變異，把每個實際的 y 減去平均數的平方加總起來。而 SSR 就是把所有的模型預測 y 減去平均數的平方加總起來。如果 R2 分數很高越接近 1，表示模型的解釋能力很高。

變異數分析表 (ANOVA table)

變異來源	平方和	自由度	均方和	F
組間變異	$SSR = (\hat{y}_i - \bar{y})^2$	1	$MSR = \frac{SSR}{1}$	$F = \frac{MSR}{MSE}$
組內變異	$SSE = (y_i - \hat{y}_i)^2$	$n-2$	$MSE = \frac{SSE}{n-2}$	
總變異	$TSS = (y_i - \bar{y})^2$	$n-1$		

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \quad (0 \le R^2 \le 1)$$

在學術研究上最直覺的觀念是 R2 分數愈接近 1 越好，也有些人透過一些手段來製造 R2 分數很高的假象，詳細內容可以參考這篇文章 *(http://amebse.nchu.edu.tw/new_page_535.htm)* 。其實只透過 R2 個評估指標就來決定一個模型的好壞是不太好的習慣。更進一步可以使用 MSE、MAE 等殘差的評估值標來看每筆資料實際值與預測值的誤差。或是使用相對誤差來觀察預測模型的可信度。此外筆者還建議可以試著把每筆資料的真實 y 與模型預測的 ŷ 繪製出來，若呈現一條明顯的由左下到右上斜直線，則表示模型所預測的結果與真實答案很相近。

$$\text{絕對誤差（Absolute error）} = \text{預測值-真值}$$

$$\text{相對誤差（Relative Error）} = \frac{\text{絕對誤差}}{\text{真值}}$$

10. 任何事情別急著想用 AI 解決

近幾年 AI 的發展想必大家有目共睹，從影像識別到物件辨識的技術有著重大的進展。此外 2016 年 Google Deepmind 團隊的 AlphaGo 首度打敗人類，這也在人機對弈上開啟了一項重要的里程碑。甚至在自然語言方面，歸功於新的模型架構與硬體資源的進步，使得自然語言有重大的突破。看到這麼多 AI 的美好讓大家再次對深度學習點燃希望！只不過 AI 並非萬能，切記！所有的問題並不是將資料收集好，並將資料丟給電腦學習就會得到你想要的結果。大家也許會陷入「為 AI 而 AI」 的迷思，很多的任務其實透過具有規則的專家系統或是傳統演算法就可以達到很不錯的結果。再者我們都對 AI 的技術感到特別歡喜與期待，但是 AI 的黑盒子人類往往不知道模型下一步會產生什麼不可預期的結果。其實 AI 有很多的限制與挑戰，除了建立機器學習模型以外，我們更需要關注的是模型在想什麼。可解釋人工智慧必然是我們要探討的一段課題。AI 與機器人的出現並不是要取代人類，我認為 AI 比較適合扮演輔助人類的重要角色。

> 本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 28] 儲存訓練好的模型

今日學習目標

- 使用 pickle + gzip 儲存模型
  - 將訓練好的模型打包並儲存
- 載入儲存的模型
  - 讀取打包好的模型並預測

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/2021-13th-ironman/blob/main/docs/28.儲存訓練好的模型/28.XGBoost(Classification-iris).ipynb)*

前言

今天的教學內容要教各位如何將訓練好的模型儲存，並提供下一次載入模型和預測。在本系列的教學中介紹了許多 Sklearn 的模型演算法。當模型訓練好了，可以將訓練結果儲存起來，並建立一個 API 接口提供模型預測。

模型儲存方法

常見的儲存模型的套件有 pickle *(https://docs.python.org/3/library/pickle.html)* 與 joblib *(https://joblib.readthedocs.io/en/latest/)*。其中在 [Day 20] 機器學習金手指 - Auto-sklearn *(https://ithelp.ithome.com.tw/articles/10276333)* 最後有使用 joblib 來儲存模型，操作方法也非常簡單。然而在今天的教學中則使用另一種方法 pickle 來儲存模型。由於 pickle 儲存模型後容量可能會有好幾百 MB 因此建議可以透過 gzip 來壓縮模型並儲存。另外在 Python 官方文件中有警告絕對不要利用 pickle 來 unpickle 來路不明的檔案。因為透過 pickle 打包模型會有安全性疑慮，包括 `arbitrary code execution` 的問題，詳細內容可以參考這篇 文章 *(http://*

www.benfrederickson.com/dont-pickle-your-data/)。如果要追求執行速度與安全性，建議可以採用 JSON 格式來存取模型的參數與設定。

> 後記：這幾年ONNX模型通用格式也非常流行，除了神經網路之外也支援sklearn的模型儲存。大家不妨也可以試試看！

1) 載入資料集

今日的範例還是拿鳶尾花朵資料集進行示範。首先我們先載入資料集並進行資料的切割。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

iris = load_iris()
df_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= ['SepalLengthCm','SepalWidthCm','PetalL
df_data
```

2) 切割訓練集與測試集

```
from sklearn.model_selection import train_test_split
X = df_data.drop(labels=['Species'],axis=1).values # 移除Species並取得剩
y = df_data['Species'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

print('train shape:', X_train.shape)
print('test shape:', X_test.shape)
```

訓練模型 - XGBoost

XGBoost 模型是目前最熱門的演算法模型之一，詳細的內容可以參考 [Day 15] 機器學習常勝軍 - XGBoost *(https://ithelp.ithome.com.tw/articles/10273094)*。裡面會有介紹詳細的模型說明與手把手實作。當然大家也可以試著用其他 Sklearn 的模型訓練看看，一樣可以透過 `pickle` 來儲存訓練好的模型。

```
from xgboost import XGBClassifier

# 建立 XGBClassifier 模型
xgboostModel = XGBClassifier(n_estimators=100, learning_rate= 0.3)
# 使用訓練資料訓練模型
xgboostModel.fit(X_train, y_train)
```

```
# 使用訓練資料預測分類
predicted = xgboostModel.predict(X_train)
```

儲存 XGboost 模型

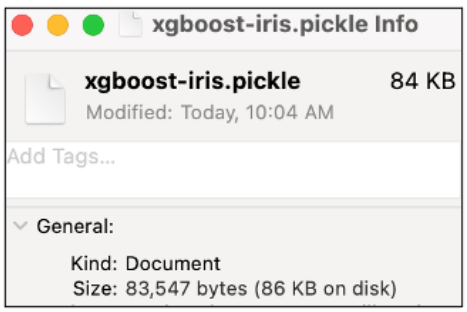
大家可以觀察 .pickle 與 .gzip 兩種不同副檔名儲存結果檔案大小有何差別?

1. 使用 pickle 儲存模型

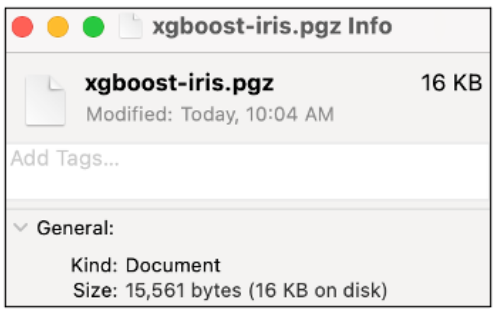
```
import pickle
with open('./model/xgboost-iris.pickle', 'wb') as f:
    pickle.dump(xgboostModel, f)
```

2. 使用 pickle 儲存模型並利用 gzip 壓縮

```
import pickle
import gzip
with gzip.GzipFile('./model/xgboost-iris.pgz', 'w') as f:
    pickle.dump(xgboostModel, f)
```



壓縮前



壓縮後

載入 XGboost 模型

試著載入兩種不同格式的模型, 並預測一筆資料。注意模型預測輸入必須為 numpy 型態, 且須為二維陣列格式。

1. 載入 gzip 格式模型

```
import pickle
import gzip

#讀取Model
```

```
with gzip.open('./model/xgboost-iris.pgz', 'r') as f:
    xgboostModel = pickle.load(f)
    pred=xgboostModel.predict(np.array([[5.5, 2.4, 3.7, 1. ]]))
    print(pred)
```

2. 載入 pickle 格式模型

```
#讀取Model
with open('./model/xgboost-iris.pickle', 'rb') as f:
    xgboostModel = pickle.load(f)
    pred=xgboostModel.predict(np.array([[5.5, 2.4, 3.7, 1. ]]))
    print(pred)
```

Reference

- How to save and load your Scikit-learn models in a minute *(https://medium.com/analytics-vidhya/save-and-load-your-scikit-learn-models-in-a-minute-21c91a961e9b)*
- Don't Pickle Your Data *(http://www.benfrederickson.com/dont-pickle-your-data/)*

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 29] 使用 Python Flask 架設 API 吧！

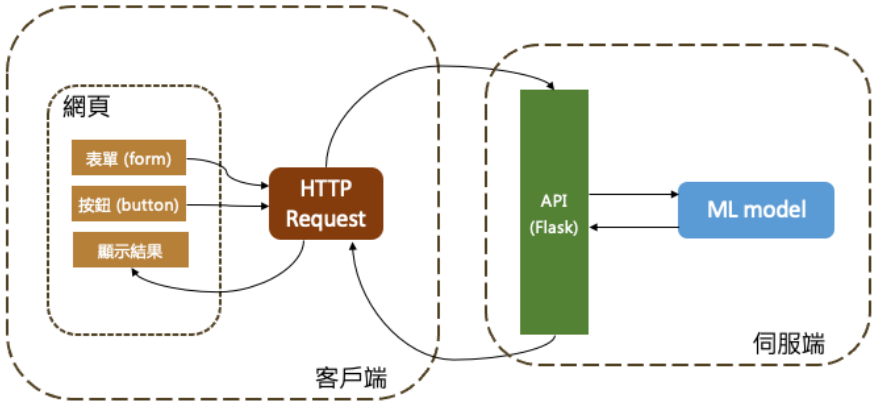
今日學習目標

- API 觀念講解
  - 什麼是 API？
  - RESTful API
  - HTTP Request 方法
- 手把手實作一個花朵分類器 API
  - 透過 Python Flask 建置一個後端預測模型 API

範例程式：Code *(https://github.com/andy6804tw/2021-13th-ironman/tree/main/29.使用Python-Flask架設API吧/Flask-API-example-with-ML-model)*

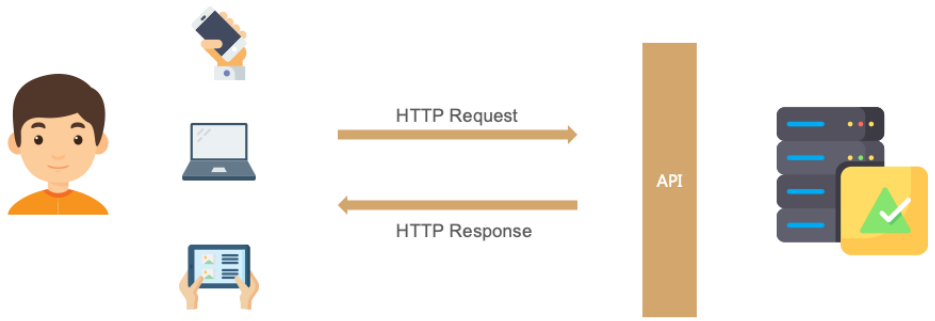
前言

當模型訓練完以後下一個步驟就是應用與落地。我們可以設計一個嵌入式系統與使用者互動，例如樹莓派、Jetson Nano、NeuroPilot...等硬體來協助 AI 模型的邊緣運算。或是設計一個手機 APP 以及網頁應用。很多人可能會有疑問模型訓練好然後下一個步驟該怎做？最常見的做法就是將訓練好的模型儲存起來並建立一個 API 部署在後端伺服器中，接著任何的終端設備都可以透過這一個 API 進行資料的存取與模型預測。下圖是一個簡單的模型落地的應用情境，我們可以在後端伺服器部署模型並建立一個 API 的接口與前端使用者互動。前端網頁的使用者透過 HTTP 的協定與後端伺服器進行通訊與資料交換，最終模型的預測結果會回傳到前端使用者並將結果選染在網頁上。我們延續昨天的內容 [Day 28] 儲存訓練好的模型 *(https://ithelp.ithome.com.tw/articles/10280076)*，目前已經成功的輸出模型。今天就來教各位如何透過 Python Flask 架設一個鳶尾花朵分類器的 API 吧！



什麼是 API？

所謂的 API 中文全名為應用程式介面 (Application Programming Interface) 是屬於客戶端與伺服端的溝通橋樑，它提供一個端口能夠進行資料交換。簡單來說是一個前端與後端的一個溝通介面。



另外大家可能聽過一個名詞叫做 RESTful API。所謂的 REST 為 Representational State Transfer 的縮寫是一種網路架構風格，近幾年來 REST 的概念已經被實作在大型網路系統中，而在 Web 服務中使用 REST 概念被實作出來的 API 就簡稱為 RESTful API 他是使用 HTTP 的協定完整定義 Web 服務在 HTTP Request 的各種流程。

HTTP Request 方法

透過網路協定 HTTP Request 不同的方法，可以實現不同的資料交換請求方式。HTTP 本身就是 REST 的實作，所謂的 HTTP Request *(https://developer.mozilla.org/zh-TW/docs/Web/HTTP/Methods)* 定義了八種請求方法分別為：

- GET *(https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET)*：此方法只能向指定的資源要求取得資料，並不會更動到內部資源。
- HEAD：HEAD 跟 GET 方法類似只差別在它並不會回傳你所請求的資源在 body 上，只回傳 HTTP header。
- POST *(https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)*：向指定的資源提交資料。
- PUT *(https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)*：向指定資源位置提交更新內容。
- DELETE *(https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE)*：向指定資源位置請求刪除內容。
- CONNECT：HTTP/1.1協議中預留給能夠將連接改為管道方式的代理服務器。
- OPTIONS：此方法可使服務器傳回該資源所支持的所有 HTTP 請求方法。
- TRACE：回顯服務器收到的請求，主要用於測試或診斷。

[程式實作] 鳶尾花朵分類器 API

建立 Python Flask API

Flask 是一個使用 Python 語言編寫的輕量級 Web 應用框架。在今日的文章中我們將延續昨天所儲存的鳶尾花朵分類器模型，建立一個花朵分類預測的 API。使用者可以透過 POST 協定從前端網頁發送四個數值分別為花萼的長與寬以及花瓣的長與寬。後端程式收到數值後送給事先打包好的模型，並將模型預測結果透過 JSON 格式回傳到前端使用者。以下為程式整個樹狀結構，其中在最外層資料夾有三個檔案分別有將模型封裝成函式的 `model.py` 與 Flask 主程式 `run.py` 以及負責管理專案套件的 `requirements.txt`。另外在 model 資料夾中負責儲存訓練好的模型壓縮檔。

```
.
├── model
│   └── xgboost-iris.pgz
├── model.py
├── requirements.txt
└── run.py
```

封裝預測模型 (model.py)

首先建立一個 `model.py` 檔案，在這個檔案中我們要載入事先訓練好的模型並將它封裝成一個 function 或是 class。在本範例程式中我們是建立一個 `predict()` 的函式並且允許接收一個 Numpy 的陣列，其中裡面允許夾帶四個花朵特徵的數值。最後將模型預測結果存放在 `pred` 變數中，並將預測的類別回傳。

```
# -*- coding: UTF-8 -*-
import pickle
import gzip

# 載入模型
with gzip.open('./model/xgboost-iris.pgz', 'rb') as f:
    xgboostModel = pickle.load(f)

# 將模型預測寫成一個 function
def predict(input):
    pred=xgboostModel.predict(input)[0]
    return pred
```

建立 Flask API (run.py)

接著我們要透過 Flask 建立一個 API，首先要設定開放跨網域 CORS 權限。所謂的跨來源資源共享 (Cross-Origin Resource Sharing, CORS) 是一種使用額外 HTTP 標頭來讓目前瀏覽網站的使用者能訪問不同來源網域的伺服器。當使用者請求一個來自於不同網域、通訊協定或通訊埠的資源時，會建立一個跨來源 HTTP 請求。所以在撰寫程式的時候必須透過 `flask_cors` 裡面所提供的 CORS 添加跨來源資源共享。這樣前端使用者在不同網域利用 ajax 或 fetch 存取 API 時就會有讀取權限。

```
# -*- coding: UTF-8 -*-
import numpy as np
import model

from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
```

接著我們先示範建立一個 GET 的路由 `@app.route('/')`，單引號內的內容即代表使用者在呼叫 API 的路徑位置 `/` 代表是 root 的意思。在這一個測試的路由中我們直接回傳一個 `hello!!` 的字串。稍後將會教各位如何透過 Postman 這個軟體來測試 API。另外第二個路由是負責接收花朵四個數值，並將這四個數值放到 Numpy 陣列中送到稍早以封裝好的 `mpdel.py` 中的 `predict()` 方法。並將預測的結果透過 JSON 格式回應給前端使用者。值得注意的是這一個

路由我們是設定他的路徑為 `/predict` 以及 HTTP Requesst 的方法指定 `methods=['POST']`。

```
@app.route('/')
def index():
    return 'hello!!'

@app.route('/predict', methods=['POST'])
def postInput():
    # 取得前端傳過來的數值
    insertValues = request.get_json()
    x1=insertValues['sepalLengthCm']
    x2=insertValues['sepalWidthCm']
    x3=insertValues['petalLengthCm']
    x4=insertValues['petalWidthCm']
    input = np.array([[x1, x2, x3, x4]])

    result = model.predict(input)

    return jsonify({'return': str(result)})
```

最後我們透過 `app.run()` 將此 API 部署在伺服器的 3000 PORT 當中。`host='0.0.0.0'` 表示預設路由將會自動幫你取得目前伺服器的固定 IP 位置。由於我們目前在本機開發等等測試時可以直接使用 `http://localhost:3000` 進行測試。另外參數 `debug` 設定為 True 即表示 API 被啟動時會自動監聽程式是否有變動，如果有更新內容即會立刻重新啟動 API。此設定適合在開發時候使用，而真正上線時再調成 False。

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=3000, debug=True)
```

管理套件版本 (requirements.txt)

`requirements.txt` 這一支檔案是負記錄了當前專案資料夾下程式所有依賴的套件及相對應的版本。下列五個是在本實作中將會使用到的套件，若套件後面沒有特別指定版本號，安裝時將會自動安裝最新的版本。

```
Flask
Flask-Cors
numpy
scikit-learn
xgboost
```

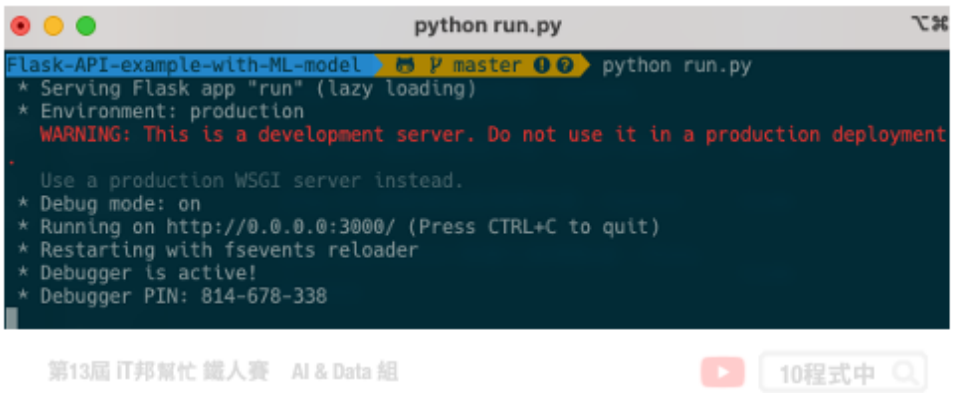
假設程式在另一台電腦上執行時，要一個一個安裝套件很麻煩。因此可以直接透過 `requirements.txt` 紀錄專案中依賴的套件。並且輸入以下指令即可一次安裝所有指定的套件。

```
pip install -r requirements.txt
```

執行 API

在本機或開發環境中測試執行 API 的方式很簡單。只要開啟終端機並輸入以下指令即可：

```
python run.py
```



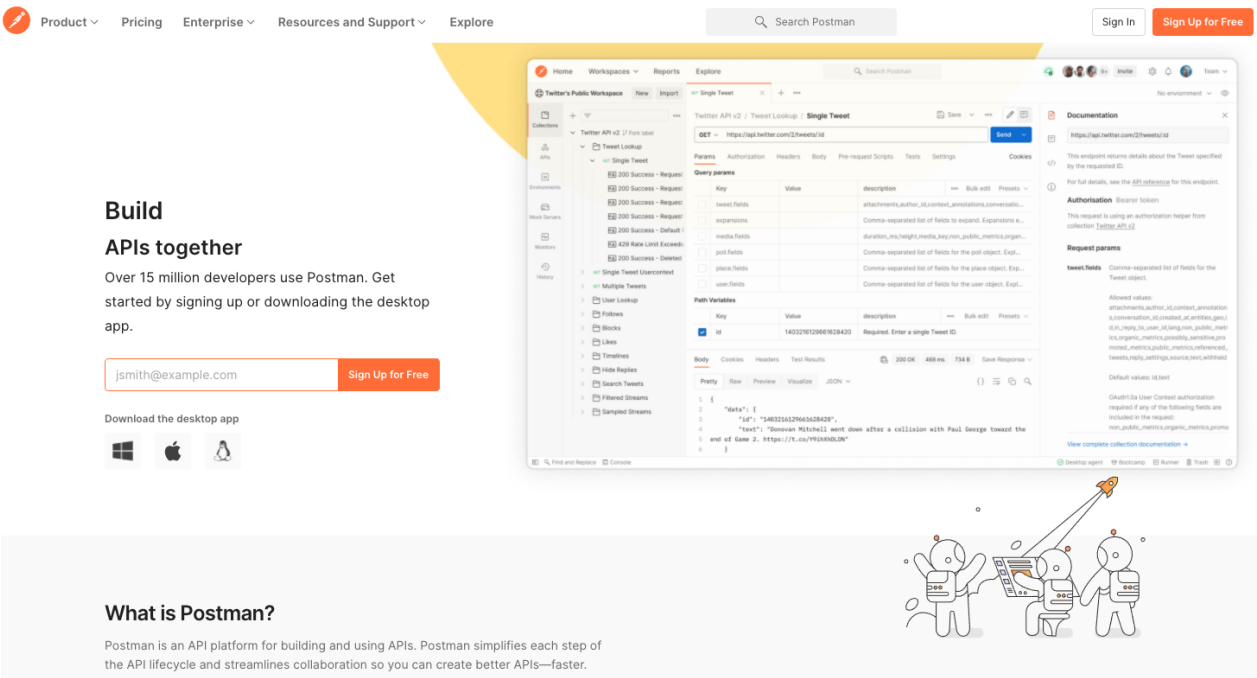
程式真正上線時建議使用 gunicorn 或 Waitress 來產生 WSGI 服務，並於背景運行

還記得我們有寫一個 GET 方法的測試路由嗎？這時候大家可以開啟電腦中的瀏覽器並在網址列輸入 `http://localhost:3000` 即可立即看到 API 在指定的路徑下所回應的內容。如果出現以下畫面即代表 API 已經正常的被運行囉。

那你可能會問我該怎麼測試另一個 POST 方法呢？由於 GET 方法比較好處理，我們直接在瀏覽器輸入路徑就能立即觀看結果。那當我們要測試 POST、PUT、DELETE 等方法時就必須依靠第三方軟體 Postman 來協助模擬 HTTP Request 完成 API 測試。

測試 API 的好工具 Postman

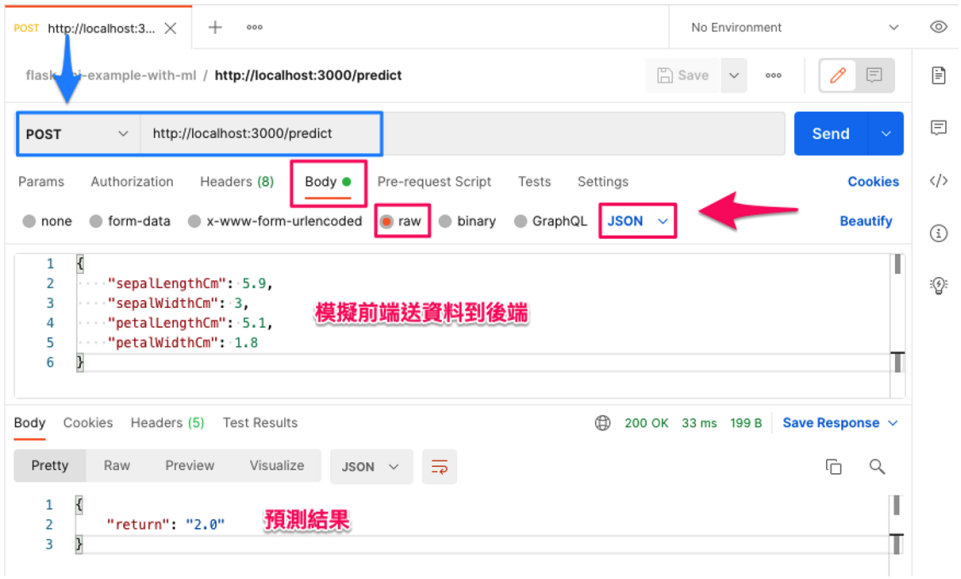
當你寫好一支 API 時要馬上測試看看你寫的程式邏輯是否正確，就可以使用 Postman 這個軟體來做 API 測試。Postman 他是一個能夠模擬 HTTP Request 的工具能夠讓你簡單快速的測試你的 API，並且內建包含許多 HTTP 的請求方式，例如常見的 GET(取得)、POST(新增)、PUT(修改)、DELETE(刪除)。首先大家可以到官網 *(https://www.postman.com/)*下載與安裝。



安裝好之後可以打開程式並點選 POST 並貼上 API 網址。由於我們現在要測試另一個預測的路徑 `predict`，因此在網址列貼上 `http://localhost:3000/predict`。點選 Body-> raw -> JSON 並將花朵的四個參數以 JSON 格式進行描述。

```
{
    "sepalLengthCm": 5.9,
    "sepalWidthCm": 3,
    "petalLengthCm": 5.1,
    "petalWidthCm": 1.8
}
```

點選 `send` 後即可將模擬的四個數值透過 JSON 格式使用 POST 方法傳送到後端 API 中的 `predict` 路徑。該 API 透過 POST 接收到前端使用者所發送的訊息後，解析這四個數值並依序放在陣列中並進行模型預測。最終預測結果會將花的種類以 JSON 格式回傳到前端使用者。此時前端的網頁設計師就可以將拿到的預測結果進行前端的畫面渲染與更新。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！

[Day 30] 使用 Heroku 部署機器學習 API

今日學習目標

- 動手部署自己的機器學習 API
- 使用 Heroku 雲端平台部署應用程式

範例程式：Code *(https://github.com/1010code/Flask-API-example-with-ML-model-heroku)*

前言

開發的最後一哩路部署應用。部署 API 必須在一個穩定的伺服器上運行，大多數企業可能會租用雲端的虛擬伺服器。常見的雲端平台三巨頭有 Google Cloud Platform (GCP)、Amazon Web Service (AWS) 以及 Microsoft Azure。以上三家供應商都有提供免費的試用額度以及部署的教學，另外雲端伺服器計費的方式是採用多少付多少的概念收費。若有 GCP 使用需求可以參考我過去所錄製的系列教學影片 GCP教學-Python *(https://www.youtube.com/watch?v=z3bU_MYPlOs&list=PLXSkku8eiD-hcW9N9A6M8Y6Hma8DHBsEJ)*。

Heroku 雲端平台

Heroku *(https://www.heroku.com)* 是一個支援多種程式語言的雲平台即服務。並且提供一個~~免費~~(現在要付費了)的雲端服務，這個雲端平台~~一個帳號可以免費建立五個專案~~，雖然是~~免費當然也有使用上的限制。例如：(1) 超過30分鐘閒置將會進入睡眠狀態，之後重新啟動 API 時會需要等待一些時間才有回應。(2) 500MB的儲存空間限制。當然 Heroku 也提供多種語言的部署環境像是 Ruby、Node.js、PHP、Go、Python ...等。~~ 本篇文章會教你如何部署 Python 的 Flask API。

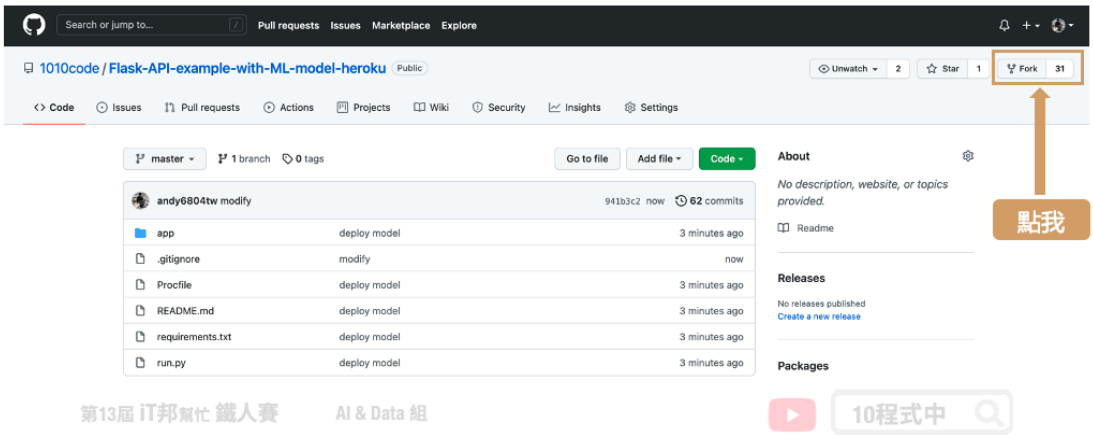
1. 前置作業

1.1) 範例程式碼

這一篇文章將以花朵分類 API 為例，拿一個先已經訓練好的模型進行 Python Flask API 的開發與部署。至於模型的訓練和 Flask API 的詳細內容這邊就不細提，若各位想了解的可以參考昨天的內容 [Day 29] 使用 Python Flask 架設 API 吧！ *(https://ithelp.ithome.com.tw/articles/*

10280422)。另外建議大家可以參考下面這份程式碼進行今天的內容實作，使用 GitHub 並將程式 fork 到自己的帳號中。

- 範例程式碼 *(https://github.com/1010code/Flask-API-example-with-ML-model-heroku)*



以下簡單說明專案內部署 Heroku 的重要檔案。

1.2 Procfile 設定檔

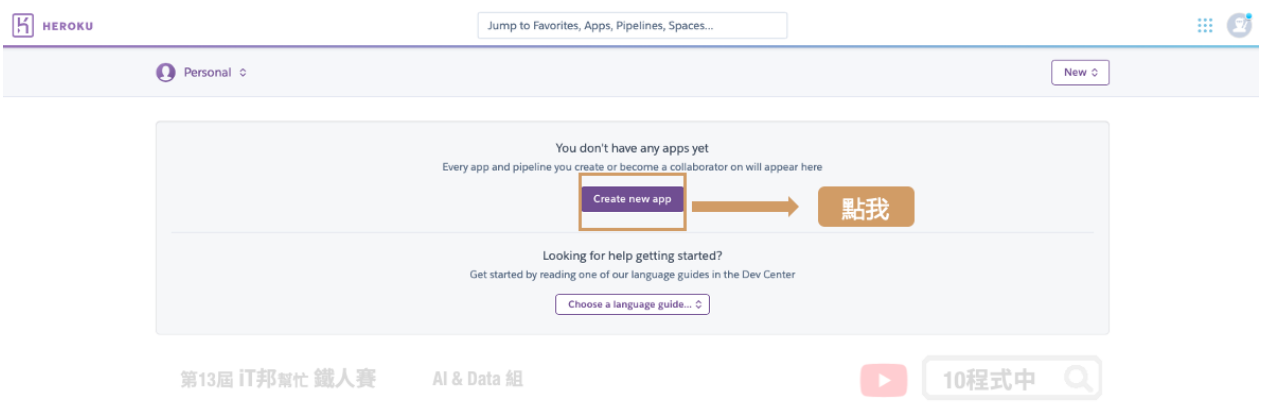
Procfile 這個檔案是要告訴 Heroku 要如何啟動這個 web app，在 Heroku 裡，執行 Python 要使用 Gunicorn 來啟動 web server。所以在 requirements.txt 裡，請記得要輸入 gunicorn。Procfile 檔案，的內容如下：

```
web gunicorn run:app
```

2. 部署 Heroku 專案

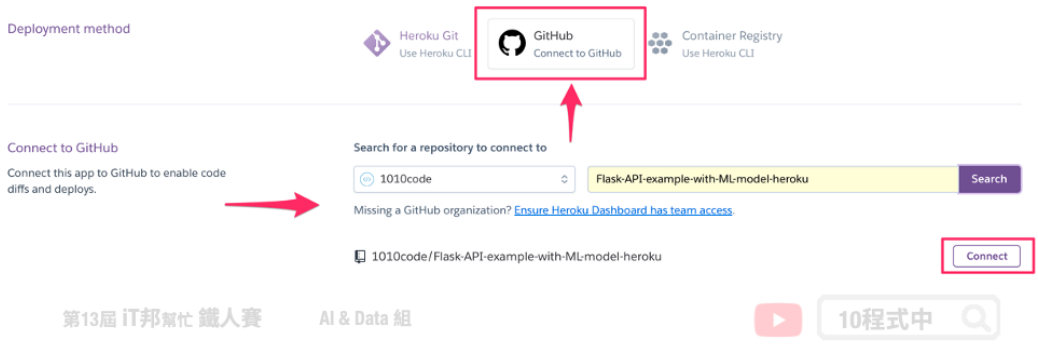
2.1 在 Heroku 建立應用程式

建立帳號後右上角「New」中的「Create new app」建立第一個應用程式：

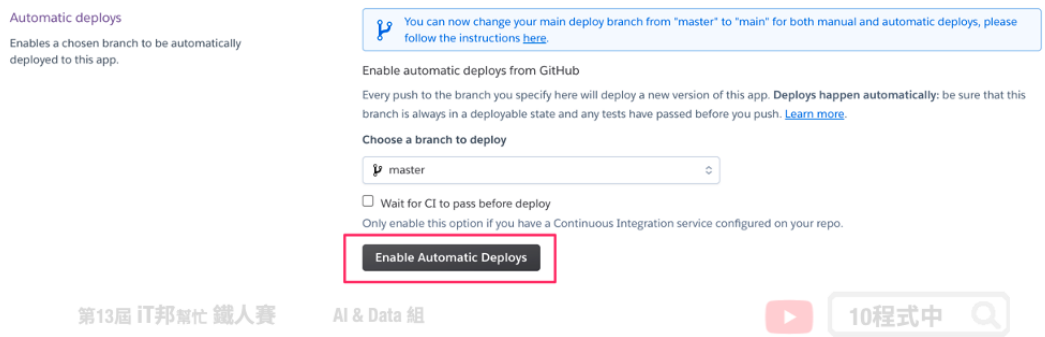


2.2 專案與 GitHub 連動

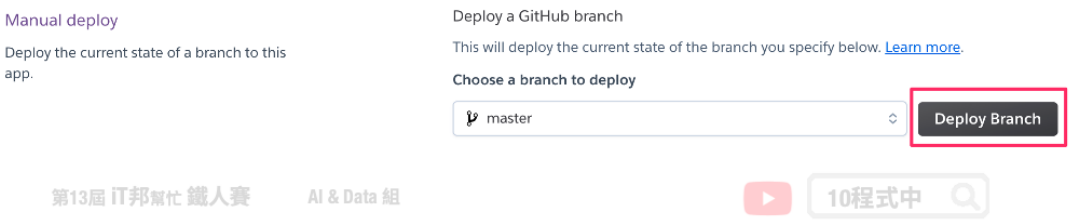
這一步驟是將 GitHub 上的專案直接與 Heroku 做連動，你也可以直接 Fork 這個專案直接實作。或是你也可以透過 Heroku CLI 直接將本機的程式碼部署到 Heroku 主機中。部署階段蠻吃大家 Git 版控的能力，基本的教學這裡就不贅述，想了解更多 Git 技巧可以參考 *(https://github.com/doggy8088/Learn-Git-in-30-days)*。



點選 Enable Automatic Deploys 連動後可以選擇自動部署。當你 GitHub 專案的程式碼有更新時他會自動幫你把更新的程式部署到 Heroku 中。



由於自動更新與部署會有上限次數，當你的專案在 GitHub 更新次數太頻繁。Heroku 就會停止自動發布，這時候你也可以試試手動部署。



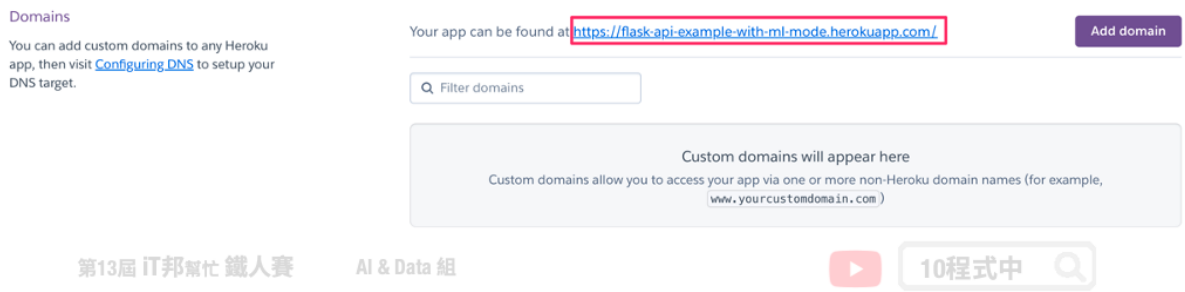
部署專案

確認以下事情都完成後就可以部署程式囉！記得我們有跟 GitHub 連動，當你的專案 `git push` 後 Heroku 就會幫你自動部署了。你可以從 Activity 內看到部署狀態，也能從右上角 More -> View logs 觀看後台 Log 訊息。或者你也可以從 Deploy 內手動部署也行。

- Python Flask API 程式撰寫 ✅
- 專案內建立 Procfile ✅
- Heroku 建立專案 ✅
- Heroku 與 GitHub連動 ✅

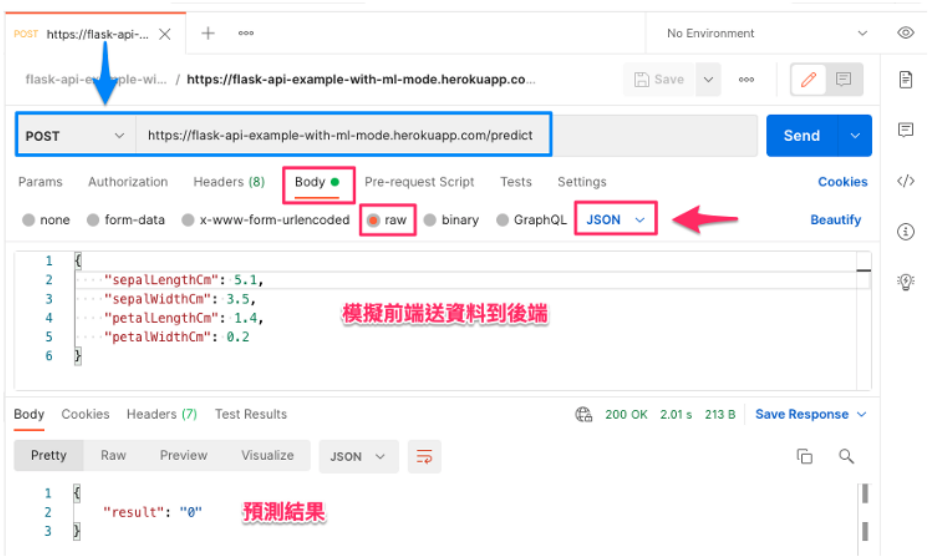
部署完成後你可以在 Settings 內的 Domains 看到你的雲端連結，這個連結點下去就能看到我們的API囉！

https://flask-api-example-with-ml-mode.herokuapp.com



測試 API

昨天已經跟大家介紹 Postman 的使用方式。今天我們就來試試部署在雲端伺服器的結果，基本上測試的方式跟昨天在本機測試的方法一模一樣。打開 Postman 點選 POST 並貼上 API 網址 `https://專案名稱.herokuapp.com/predict`。並模擬前端使用者發送數值 Body -> raw -> JSON 將花朵的四個參數以 JSON 格式傳給後端 API。



本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/2021-13th-ironman)* 取得！