

機器學習 百日馬拉松

2018.12.15

—

2019.04.15



初探深度學習

機器學習百日馬拉松課程 複習筆記

目錄

程式目錄	IV
第一章 Python 安裝及快速入門	1
1-1 Python 安裝	1
1-2 Python 快速入門	1
第二章 資料匯入、預視及 Pandas	6
2-1 Pandas DataFrame	6
2-2 Pandas 建立/讀取資料	7
2-2.1 利用 pd.DataFrame 函式創立 dataframe	7
2-2.2 使用 Pandas 匯入資料	7
2-3 預視資料 及 繪圖	8
DataFrame -- Pandas	8
2-3.1 連續型變數 預視 (含基礎迴圈 for)	10
2-3.2 類別型變數 預視 (含基礎聚集 def)	12
2-3.3 資料分組 預視	14
2-4 多個資料合併 merge / transform	20
第三章 格式化資料【資料預視、預處理及 Sklearn】	22
3-1 建立/讀取資料補充- Copy	28
3-2 特徵工程範例	29
3-2.1 範例 1：鐵達尼號的乘客生存預測 (binary classification: LogisticRegression)	29
Day017 什麼是 Titanic dataset?	29
數值型特徵	31
類別型特徵	37
特徵選擇	44
補充	51
3-2.2 範例 2：計程車費率預測 (LinearRegression)	55
3-2.3 範例 3：鳶尾花辨識(multi-classification: KNeighborsClassifier & 樹狀模型)	60
3-3 超參數 (Hyper-paramter)調整與優化	68
實戰練習 - Enron Fraud Dataset 安隆公司詐欺案資料集	69
3-4 非監督式學習及 Sklearn	78
3-4.1 聚類算法 Clustering: K-means	78
3-4.2 類算法 Clustering: 階層式分群 Hierarchical Clustering	88
3-4.3 降維方法 - 主成份分析 (PCA)	94

3-4.4 降維方法 - t-SNE	99
第四章 圖片資料【初探深度學習使用 Keras】	103
4-1 讀取檔案：txt 檔 & 圖片檔	103
讀圖檔的三種方法	104
圖檔的資料結構：矩陣中的數字代表的意義	105
情境練習：讀取 txt 檔案中的圖片連結	106
補充說明: resize	108
4-2 Keras 資料庫	109
4-3 資料前處理	111
STEP1. 載入套件	111
STEP2.1 資料預視	111
STEP2.2 資料預處理之概念	112
STEP3 設定模型與運算 (DNN vs. CNN / plus ImageDataGenerator)	113
實務情境上補充說明 – 資料量過大或過小的處理技巧	116
4-4 深度神經網路及多層感知(Multi-layer Perception, MLP)	123
執行程序藍圖與體驗	125
4-4.1 步驟一：選擇模型	128
4-4.1.1 序列模型搭建網路(Keras Sequential API)	128
4-4.1.2 函數式模型搭建網路(Functional API (or Model))	129
4-4.1.3 多層感知(Multi-layer Perception, MLP)	131
4-4.2 步驟二：建構網路層	132
4-4.2.1 激活函數(Activation Function)	132
4-4.2.2 正規化(Regularization)	134
4-4.3 步驟三：編譯	136
4-4.3.1 損失函數(Loss Function)	136
4-4.3.2 優化(Optimizer)	139
4-4.3.3 梯度調整(Gradient Descent)：優化器的參數調整	143
4-4.4 訓練、預測模型範例	149
4-4.4.1 Optimizers & Learning rates (D080 練習)	149
4-4.4.2 正規化(D084 練習)	152
4-4.4.3 callbacks 函數	154
4-5 傳統電腦視覺提取特徵的方法：color histogram 及 HOG 特徵	157
4-6 卷積神經網路 Convolution Neural Network_CNN	158
濾波器 (filter)的概念	158
卷積網路的組成	159

1. Convolution Layer 卷積層	159
2. Pooling Layer 池化層	161
3. Flatten Layer 平坦層	162
4. Fully connection Layer 全連接層	162
5. 回顧--卷積神經網路(CNN)特性	162
程式架構範例	163
隨堂小考	164
4-7 Keras 中的 CNN layers -- Conv2D	165
4-8 遷移學習(Transfer Learning) -- ResNet	166
4-9 Kaggle 資料實戰練習 -- 花朵辨識	168
範例解答 - 方法一 (ImageDataGenerator)	169
範例解答 - 方法二 (ImageDataGenerator + ResNet50)	173
附錄 網路學習資源	177

程式目錄

↪ Code 1: Day000_Start.ipynb	1
↪ Code 2: Day_003_build_dataframe.ipynb	6
↪ Code 3: Day017-030 Feature Engineering_ReviewPART I.ipynb	29
↪ Code 4: Day017-030 Feature Engineering_ReviewPART II.ipynb	55
↪ Code 5: Day017-030 Feature Engineering_ReviewPART III_Iris.ipynb	60
↪ Code 6: Day049_EnronFraudDataset+hyper_parameter.ipynb	69
↪ Code 7: Day036_evaluation_metricsReview.ipynb	77
↪ Code 8: Day055_K-means Review.ipynb	80
↪ Code 9: Day056_kmean Review DATA_iris.ipynb	80
↪ Code 10: Day057_hierarchical_clustering_Review.ipynb	88
↪ Code 11: Day058_hierarchical_clustering_Ans_noisy_moons.ipynb	88
↪ Code 12: Day059_PCA_Review.ipynb	94
↪ Code 13: Day060_PCA 原檔.ipynb	94
↪ Code 14: Day061_tsne_Review.ipynb	99
↪ Code 15: Day062_tsne_原檔-S 曲線.ipynb	99
↪ Code 16: Day062_tsne_雙同心圓.ipynb	99
↪ Code 17: Day003-3_ImportImage_Review.ipynb	103
↪ Code 18: Day003-5_Day064_OPENCV_.ipynb	104
↪ Code 19: Day003-4_ImportImage2_Review.ipynb	106
↪ Code 20: Day067_Keras_Dataset_Review.ipynb	111
↪ Code 21: Day097_Keras_CNN_vs_DNN_Review.ipynb	114
↪ Code 22: Day098_Python_generator_Ans.ipynb	116
↪ Code 23: Day099_data_augmentation_Review.ipynb	119
↪ Code 24: Day072-Activation_function.ipynb	132
↪ Code 25: Day067_Keras_Dataset_Review.ipynb + Day71-LossFunction 比較	137
↪ Code 26: Day089.1_CustomizedLoss.ipynb + Day089.9_Ans.ipynb	138
↪ Code 27: Day075-Back_Propagation_PPTexample_.ipynb	146
↪ Code 28: Day075-Back_Propagation.ipynb	146
↪ Code 29: Day074-Gradient_Descent_Mathexplian.ipynb	148
↪ Code 30: Day080_AnsReview.ipynb	149
↪ Code 31: Day084_AnsReview.ipynb	152
↪ Code 32: Day085_CB_EarlyStop_Ans.ipynb	154
↪ Code 33: Day086_CB_ModelCheckPoint_Ans.ipynb	154
↪ Code 34: Day087_CB_ReduceLR_Ans.ipynb	155
↪ Code 35: Day088_CB_CustomizedCallbacks_Ans.ipynb	156
↪ Code 36: Day094.1-CNN_Convolution.ipynb + Day095.1-CNN_Pooling.Padding.ipynb	160
↪ Code 37: Day096.9_Keras_CNN_layers_Ans.ipynb	165
↪ Code 38: Day100.1_transfer_learning.ipynb	167
↪ Code 39: Day101-2_Final_KerasCNN_Solution_basic.ipynb	169
↪ Code 40: Day101-3_Final_Keras_PreTrain_SolutionFixed.ipynb	173

第一章 Python 安裝及快速入門

1-1 Python 安裝

在 Windows，除了在 [Python 官網](https://www.python.org/downloads/)(<https://www.python.org/downloads/>) 下載安裝，另外也可以安裝 [Anaconda](https://www.anaconda.com/download/)(<https://www.anaconda.com/download/>)，這是 Python 的懶人包，除了 Python 本身(python2, 3) 還包含了 Python 常用的資料分析、機器學習、視覺化的套件。這次會使用到的深度學習套件"TensorFlow"，可以直接使用 jupyter notebook 安裝套件 TensorFlow；但是 Anaconda 內容齊全而致缺點容量太龐大，所以我們也可以利用命令提示字元視窗進行原生安裝(native installation)，文章參考<Windows 安裝深度學習框架 TensorFlow 與 Keras>[#1.1](#)。

這邊複習全部是使用 Ananconda、Python3 和 Jupyter Notebook 開發環境進行。

模型分析即在第二章之後的執行步驟都是



STEP1. 載入套件

STEP2. 資料預視及預處理

STEP3. 設定模型與運算、訓練結果並輸出預測



因此會每一章節會慢慢導入所需要的套件，第二章先進行資料預視及預處理，之後就會使用這些概念整理好的資料，進入以後的分析。

事前準備(Optional)：若尚未安裝相關套件，執行下一行，然後 restart kernel

```
!pip3 install --user sklearn  
!pip3 install --user --upgrade matplotlib
```

1-2 Python 快速入門

Python 是一個簡潔易讀的語言，這邊直接參考文章<一小時 Python 入門>[#1.2](#)，使用 jupyter notebook 執行基礎語法。

Code 1: Day000_Start.ipynb

1. print 語法 與 運算

```
print  
print('Hello world')  
print("Hello World!\n") #"\n"表示斷行  
print('Good', 'Bad', 'Ugly', sep = "/") #空白以"/"連接  
Hello world  
Hello World!  
Good/Bad/Ugly
```

#1.1. <Windows 安裝深度學習框架 TensorFlow 與 Keras> <http://yhuang1966.blogspot.com/2018/02/windows-tensorflow-keras.html>

#1.2. <一小時 Python 入門> <https://kopu.chat/2017/01/18/%E4%B8%80%E5%B0%8F%E6%99%82python%E5%85%A5%E9%96%80-part-1/>

```
#format 設定
# 使用{:型別}，想要放入字串變數是{:s}，整數是{:d}，而浮點數則是{:f}
print('Hello, {:s}. You have {:d} messages.'.format('Demo', 999))
# % 操作 http://www.cnblogs.com/vamei/archive/2013/03/12/2954938.html
print('Hello, %s. You have %d messages.' %('Demo2', 888))

# f-string 的一種是格式化字串 (formatting string) 的用法
# 像f...' 這種在字串符號''前面加上指定形式稱為 Prefix String Literal 前綴字串，類似的用法還有r'...'（原始字串）`b'...`（位元組字串）。
# 格式化字串的目的是用於在字串中嵌入變數，可以將字串與變數分離，這邊補充一下幾種類似的寫法：
# 參考Day18 問與答
# %formatting: 'Hello %s' % ('World')
# str.format(): 'Hello {name}'.format(name='foobar')
# f-string: f'Hello {name}'
```

Hello, Demo. You have 999 messages.
Hello, Demo2. You have 888 messages.

```
# 列印變數: 三大資料型態(Type)
a = 123 ; print(type(a), type(a)); #type(a)這是一個init(整數)。
b = 4.56 ; print(type(b), type(b)); #type(b)這是一個str(字串)。
c = '789'; print(type(c), type(c)); #type(c)這是一個float(浮點數)。
print("%+10x" % 10) # %ox 十六進位整數 "+" 結果會顯示+
print("%04d" % 5) # %d 十進位整數，"0"4d 整數前位數不足則補零
print("%6.3f" % 2.3) # %f 浮點數，6位數和小數點後三位
print("%.2f" % (4, 1.2)) #同上，增加*參數，這邊設定在刮弧內第一個數字4
type(a): <class 'int'>
type(b): <class 'float'>
type(c): <class 'str'>
+a
0005
2.300
1.2000
```

運算 http://tw.gitbook.net/python/python_basic_operators.html

```
a=5
b=2
print('a=', a)
print('b=', b)
print('a+b=', a+b)
print('a-b=', a-b)
print('a*b=', a*b)
print('a/b=', a/b) #除法
print('a//b=', a//b) #無條件捨去除法
print('a**b=', a**b) # a 的b 次方
```

a= 5
b= 2
a+b= 7
a-b= 3
a*b= 10
a/b= 2.5
a//b= 2
a**b= 25

```
c=-3
d=4
print('c=', c)
print('d=', d)
```

```

print('abs(c)=' , abs(c)) #取 a 絕對值
print('max(a, b)=' , max(a, b)) #取 a, b 的最大值
print('min(a, b)=' , min(a, b)) #取 a, b 的最小值
c= -3
d= 4
abs (c)= 3
max (a, b)= 5
min (a, b)= 2
# 不同的形態運算，可以用 int(), str(), float() 來轉換變數型態。
e=123
f='456'
print('e=' , e)
print('f=' , f)
print('e+int(f)=' , e+int(f))
print('str(e)+f=' , str(e)+f)
e= 123
f= 456
e+int(f)= 579
str(e)+f= 123456

```

2. If- Else 語法 [注意縮排]

```

if 判斷條件 1:
    執行語句 1.....
elif 判斷條件 2:
    執行語句 2.....
else:
    執行語句 3.....

```

```

num = 5
if num == 3:
    print ('boss')
elif num == 2:
    print ('user')
elif num == 1:
    print ('worker')
elif num < 0:
    print ('error')
else:
    print ('roadman')

```

```

# if-else 主要用在邏輯運算的判斷上：
# > 大於, < 小於, >= 大於等於, <= 小於等於, == 等於, != 不等於
# 或是布林值(Boolean) : true 真的, false 假的, and 且, or 或, not 非
roadman

```

3. For Loop 迴圈

```
for [變數名稱] in range(n): (縮排) print([變數名稱])
```

#range(n) = range(0, n) , 簡單來說就是個產生器，在這個 range 範圍中拿資料、拿完到不包含n的資料為止。

```
for i in range(2,10): #從 2 開始印到 9
```

```
    print(i)
```

```
print("")
```

#range(起點, 終點, 間距) , 一般間距預設為 1 、比如 range(0, 6) = range(0, 6, 1) 。代表從 0 到 5 , 每次加 1 的意思。

```
for i in range(2,10, 2): #從 2 開始每兩格數字印出來
```

```
    print(i)
```

```
2  
3  
4  
5  
6  
7  
8  
9
```

```
2  
4  
6  
8
```

4. List : 能無限儲存資料的格子

```
#List 有多少元素呢? len([list 變數名稱])。
```

```
a = ['Lynn', 0.87, 1234, True]  
len(a)  
for i in a:  
    print(i)
```

```
Lynn  
0.87  
1234  
True
```

```
#增減資料方法
```

```
singers=["Justin", "Taylor", "Lady Gaga"]  
singers.append("Sia")  
print("增加 Sia 壓軸後的名單:", singers)  
singers.insert(2, "Britney Spears")  
print("增加小甜甜布萊妮在第三位後的名單:", singers)  
singers.pop()  
print("不含最後一筆(壓軸)的名單:", singers)  
singers.pop(1)  
print("Taylor 退出後，不含最後一筆(壓軸)的名單:", singers)
```

```
#list.remove(x): 會把第一個出現的變數 x 拿掉
```

```
ABC=["A", "B", "C", "A", "B", "C"]  
ABC.remove("A")  
print("移除第一個 A 的列表:", ABC)  
ABC.remove("A")  
print("移除第二個 A 的列表:", ABC)
```

```
#List 還有切片 (Slice) 的功能
```

```
ABCDE=["A", "B", "C", "D", "E"]  
print("ABCDE[0:2] :", ABCDE[0:2])  
print("ABCDE[2:4] :", ABCDE[2:4])  
print("ABCDE[ :4] :", ABCDE[ :4])  
print("ABCDE[2: ] :", ABCDE[2: ])  
print("ABCDE[ : ] :", ABCDE[ : ])
```

```
增加 Sia 壓軸後的名單: ['Justin', 'Taylor', 'Lady Gaga', 'Sia']
```

```
增加小甜甜布萊妮在第三位後的名單: ['Justin', 'Taylor', 'Britney Spears', 'Lady Gaga', 'Sia']
```

```
不含最後一筆(壓軸)的名單: ['Justin', 'Taylor', 'Britney Spears', 'Lady Gaga']
```

```
Taylor 退出後，不含最後一筆(壓軸)的名單: ['Justin', 'Britney Spears', 'Lady Gaga']
```

```
移除第一個 A 的列表: ['B', 'C', 'A', 'B', 'C']
```

```
移除第二個 A 的列表: ['B', 'C', 'B', 'C']
```

```
ABCDE [0:2] : ['A', 'B']
```

```
ABCDE [2:4] : ['C', 'D']
```

```
ABCDE [ :4] : ['A', 'B', 'C', 'D']
```

```
ABCDE [2: ] : ['C', 'D', 'E']
```

```
ABCDE [ : ] : ['A', 'B', 'C', 'D', 'E']
```

5. Dict 一個靠標籤能查到東西的字典：

#dict 由大括號{}包住、元素以 key:value 的 key-value pair 組成 (key 後面接冒號和變數 value)，並以逗點 隔開。

```
d = {123:'Snoopy', 'cat':['Pusheen', 'Kitty']}
```

```
print(d)
```

```
print(d[123])
```

```
print(d["cat"])
```

#增加與更新 dict[key]=value: 如果 key 不存在，會增加這組 K-V；如果 key 已存在，會更新這組 K-V。

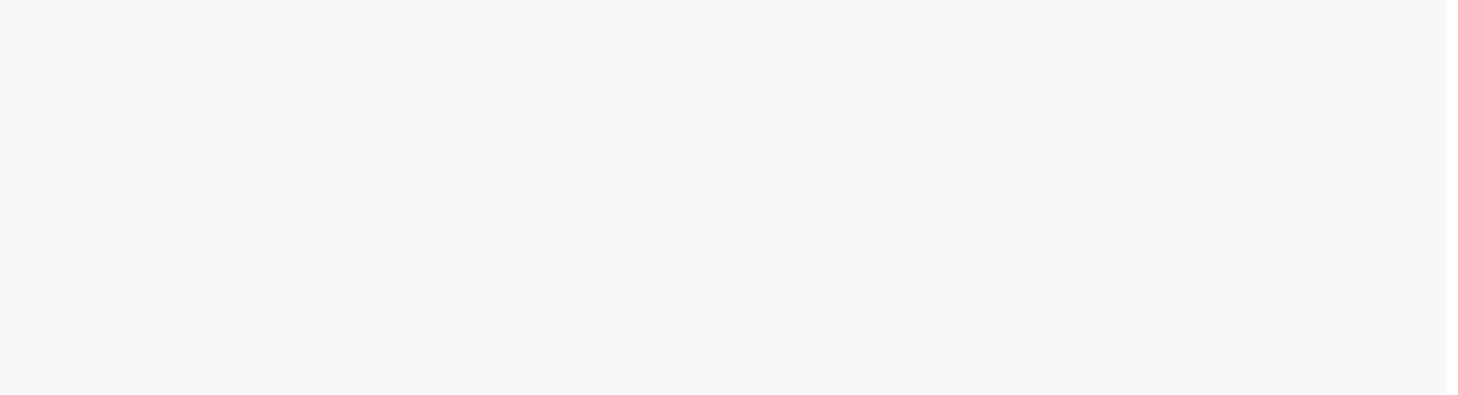
```
d["demo_new"]=[2, 3]
```

```
print(d)
```

#del dict[key]: 刪除特定的 key-value pari

```
del d["demo_new"]
```

```
print(d)
```

```
{123: 'Snoopy', 'cat': ['Pusheen', 'Kitty'] }  
Snoopy  
['Pusheen', 'Kitty']  
{123: 'Snoopy', 'cat': ['Pusheen', 'Kitty'], 'demo_new': [2, 3]}  
{123: 'Snoopy', 'cat': ['Pusheen', 'Kitty']}  
  

```

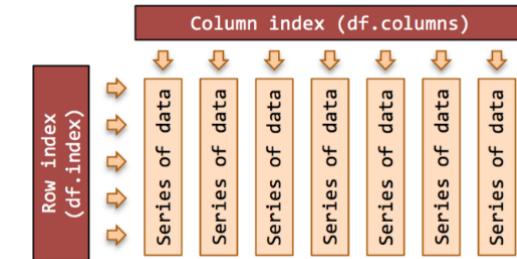
第二章 資料匯入、預視及 Pandas

資料來源很多元，在 Python 最快速學習的方法是直接使用套件中內建的資料庫，本章後面會用到 SKlearn 套件中內建的 Iris 資料庫及第四章會用到 Keras 套件中內建的 cifar10 資料庫來做範例。當然也可以自己用程式語法一筆一筆輸入創立資料清單，建立資料庫；或是自己已經有 Excel 或其它資料庫軟體創建的資料，直接匯入使用整理分析。後者將使用以下介紹的 Pandas 進行處理。

2-1 Pandas DataFrame

Pandas DataFrame 是一個表格型的資料結構，有行(Row)索引也有列(Column)索引，如右圖，參考文章[#2.1](#)有完整的行與列的整理方式，下面文章也會用不同角度說明。

一個 DataFrame 可以解構為多個，ndarray 可以再解構取得其中的數字、布林或文字。而 Panel 則是能儲存多個 DataFrame 資料結構。



<資料操作 (Pandas) >https://bookdata.readthedocs.io/en/latest/base/01_pandas.html

在介紹 Pandas 之前有許多書籍會提到 Numpy，主要原因是因為 Pandas 背後的數值型態都是 Numpy，Numpy 的資料結構可以幫助 Pandas 在執行運算上更有效率以及更省記憶體。舉例來說 Python 的內建資料結構 list 可以塞好幾種不同 type 的資料進去，如下所示，這個 list 裡面的資料有 string, int, float，但對於機器來說，要提升效能或是提升記憶體省用效率最好有一致的型別會比較好[#2.2](#)。numpy 套件可以將 list 轉換成 ndarray，常用的如 array() 及 zeros()、empty()、arange() 方法[#2.3](#)。

Code 2: Day_003_build_dataframe.ipynb

```
#匯入套件
import pandas as pd
import numpy as np

list2 = ["1", 2, 3., 4, 5]
list2
['1', 2, 3.0, 4, 5]
#使用 numpy 的 array 資料結構會強迫把裡面的資料都轉成同一型態
np.array(list2)
array(['1', '2', '3.0', '4', '5'], dtype='<U3')
```

#2.1. <資料操作 (Pandas) >https://bookdata.readthedocs.io/en/latest/base/01_pandas.html

#2.2. <[資料分析&機器學習] 第 2.3 講：Pandas 基本 function 介紹(Series, DataFrame, Selection, Grouping)>

<https://medium.com/jamelearningnote/%E8%83%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A0%0F%E5%99%A8%E5%AD%BB%E7%BF%92-%E7%AC%AC2-3-ER%AC%9B-pandas-%E5%9F%BA%E6%9C%ACfunction%E4%BB%8B%E7%B4%8B-9-series-dataframe-selection-grouping-447a3fa90b60>

#2.3. <[第 13 天] 常用屬性或方法 (2) ndarray> <https://ithelp.ithome.com.tw/articles/10185819>

2-2 Pandas 建立/讀取資料

2-2.1 利用 pd.DataFrame 函式創立 dataframe

方法一

```
data = {'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],
        'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],
        'visitor': [139, 237, 326, 456]}
```

```
visitors_1 = pd.DataFrame(data)
print(visitors_1)
```

	weekday	city	visitor
0	Sun	Austin	139
1	Sun	Dallas	237
2	Mon	Austin	326
3	Mon	Dallas	456

方法二

```
cities = ['Austin', 'Dallas', 'Austin', 'Dallas']
weekdays = ['Sun', 'Sun', 'Mon', 'Mon']
visitors = [139, 237, 326, 456]

list_labels = ['city', 'weekday', 'visitor']
list_cols = [cities, weekdays, visitors]

zipped = list(zip(list_labels, list_cols))
```

```
visitors_2 = pd.DataFrame(dict(zipped))
print(visitors_2)
```

	city	visitor	weekday
0	Austin	139	Sun
1	Dallas	237	Sun
2	Austin	326	Mon
3	Dallas	456	Mon

2-2.2 使用 Pandas 匯入資料

pd.read_csv(csv_file), pd.read_excel(xlsx_file), or pd.read_table(txt_file)..... pandas 可以支援多種文字、二進位檔案與資料庫的資料載入，常見的 txt、csv、excel 試算表、MySQL 或 PostgreSQL 都難不倒.....

```
# 讀入 csv 文字檔
csv_file = "https://storage.googleapis.com/learn_pd_like_tidyverse/gapminder.csv"
gapminder = pd.read_csv(csv_file)
print(type(gapminder))
print("前五筆資料:\n", gapminder.head())
<class 'pandas.core.frame.DataFrame'>
前五筆資料:
   country continent  year  lifeExp      pop  gdpPercap
0  Afghanistan     Asia  1952  28.801  8425333  779.445314
1  Afghanistan     Asia  1957  30.332  9240934  820.853030
2  Afghanistan     Asia  1962  31.997  10267083  853.100710
3  Afghanistan     Asia  1967  34.020  11537966  836.197138
4  Afghanistan     Asia  1972  36.088  13079460  739.981106
```

#2.4 <練習資料：由 gapminder.org 提供的資料來練習使用 pandas 做資料載入、整理與合併。

<https://medium.com/datanpoint/%E5%BE%9E-pandas-%E9%96%8B%E5%A7%8B-python-%E8%88%87%E3%87%E6%96%99%E7%A7%91%E5%AD%B8%E4%B9%8B%E6%97%85-8dee36796d4a>

2-3 預視資料及繪圖

```
#載入 DataFrame 及 畫圖的套件
import pandas as pd #DataFrame
import matplotlib.pyplot as plt #畫圖
# import seaborn as sns #可美化作圖

#如果不想要每次在 jupyter notebook 顯示都寫.show()，可以在 jupyter notebook 加上%matplotlib inline
%matplotlib inline
```

DataFrame -- Pandas

pandas 有一些好用的屬性與方法可以快速瞭解一個 **DataFrame** 的外觀與內容：

- df.shape : 這個 DataFrame 有幾列有幾欄
- df.columns : 這個 DataFrame 的變數資訊
- df.index : 這個 DataFrame 的列索引資訊
- df.info() : 關於 DataFrame 的詳細資訊

Simple Aggregation in Pandas:

basic statistics

- count : Number of non-null observations
- sum : Sum of values
- mean : Mean of values
- mad : Mean absolute deviation
- median : Arithmetic median of values
- min : Minimum
- max : Maximum
- mode : Mode
- abs : Absolute Value
- prod : Product of values
- std : Unbiased standard deviation
- var : Unbiased variance
- sem : Unbiased standard error of the mean
- skew : Unbiased skewness (3rd moment)
- kurt : Unbiased kurtosis (4th moment)
- quantile : Sample quantile (value at %)
- cumsum : Cumulative sum
- cumprod : Cumulative product
- cummax : Cumulative maximum
- cummin : Cumulative minimum
- first(), last() : First and last item

These are all methods of DataFrame and Series objects.

```
# 因欄位眾多，先列印相關資訊:有幾筆、幾個欄位、包含那幾個欄位型態
print(f"關於 DataFrame 的詳細資訊:")
gapminder.info()
```

關於 DataFrame 的詳細資訊：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
country      1704 non-null object
continent    1704 non-null object
year         1704 non-null int64
lifeExp      1704 non-null float64
pop          1704 non-null int64
gdpPercap    1704 non-null float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
```

```
print(f"這個 DataFrame 有幾列有幾欄:\n {gapminder.shape}") #(有幾筆, 幾個欄位)
print(f"這個 DataFrame 的變數資訊:\n {gapminder.columns}") #list 出欄位名稱
print(f"這個 DataFrame 的列索引資訊:\n {gapminder.index}")
```

這個 DataFrame 有幾列有幾欄：

```
(1704, 6)
```

這個 DataFrame 的變數資訊：

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'],
      dtype='object')
```

這個 DataFrame 的列索引資訊：

```
RangeIndex(start=0, stop=1704, step=1)
```

```
# 進階處理：秀出資料欄位的類型與數量 df.dtypes.reset_index() 參考 複習 Day018
#print('dtype of each column:', df.dtypes.reset_index()) #語法尾端"()"可將結果列為 DataFrame: 列出各欄位的資料型態
#df.dtypes.reset_index #語法尾端不加"()" 與 df.info 有相同結果，只是一個 dtype: object，而非 DataFrame
```

```
dtype = gapminder.dtypes.reset_index()
dtype.columns = ["Count", "Column Type"]
dtype = dtype.groupby("Column Type").aggregate('count').reset_index()
print("\n秀出資料欄位的類型與數量")
print(dtype)
```

```
print("\n每個欄位有多少 coding: pd.Series.nunique ")
print(gapminder.apply(pd.Series.nunique, axis = 0))
```

```
print("\n各欄位 Missing 的量: \n{gapminder.isnull().sum()}") #觀察 missing 狀況
```

秀出資料欄位的類型與數量

	Column Type	Count
0	int64	2
1	float64	3
2	object	2

每個欄位有多少 coding: pd.Series.nunique

```
country      142
continent    5
year         12
lifeExp     1626
pop         1704
gdpPercap   1704
pop_10000   1704
dtype: int64
```

各欄位 Missing 的量:

```
country      0
continent    0
year         0
lifeExp     0
pop         0
gdpPercap   0
pop_10000   0
dtype: int64
```

2-3.1 連續型變數 預視 (含基礎迴圈 for)

- df.describe(): function is a useful summarisation tool that will quickly display statistics for any variable or group it is applied to.
- 盒鬚圖 (Box plot) plt.boxplot(df.[‘col_name’])
- 直方圖 (Histogram) plt.hist(col_samples)

```
# 1. 定義 數值欄位
numeric_columns = gapminder.select_dtypes(include=[“number”])
# 2. 列出各欄位有幾種 coding
gapminder.select_dtypes(include=[“number”]).apply(pd.Series.nunique, axis=0)
# 若只有2值(通常是0,1)的欄位是否為類別呢？因此先排除
numeric_cols = list(numeric_columns.columns[list(numeric_columns.apply(lambda x:len(x.unique())!=2))])
print(f原始的數字欄位數量有 {len(numeric_columns.columns)} ')
print(f經排除整理後純數字的欄位數量有 {len(numeric_cols)} ')
# 檢視這些欄位的數值範圍(原始的數字欄位)
gapminder[numeric_cols].describe() #count, mean, std, min, 25%, 50%, 75%, max
```

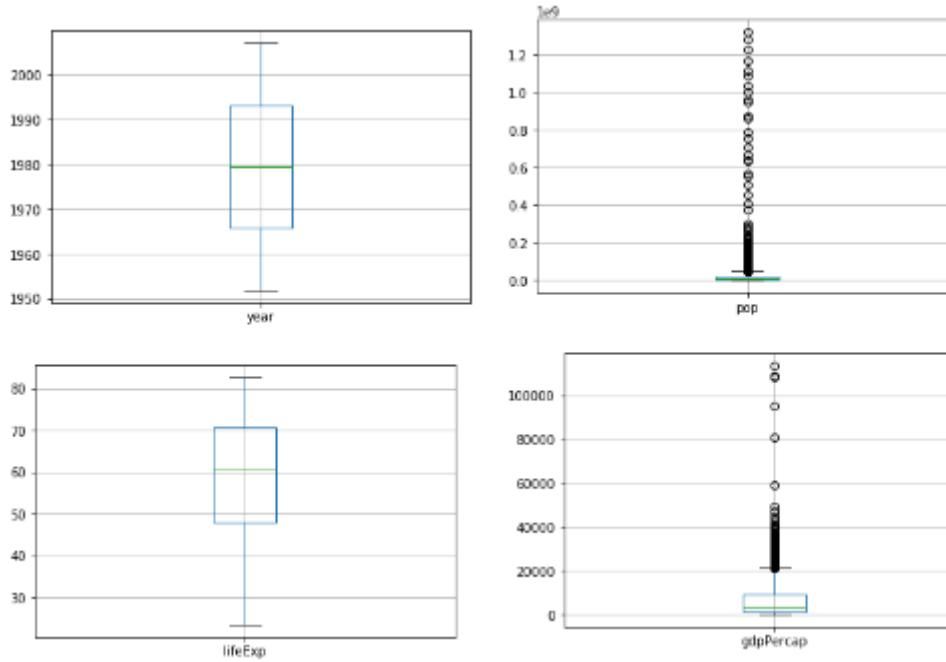
原始的數字欄位數量有 4

經排除整理後純數字的欄位數量有 4

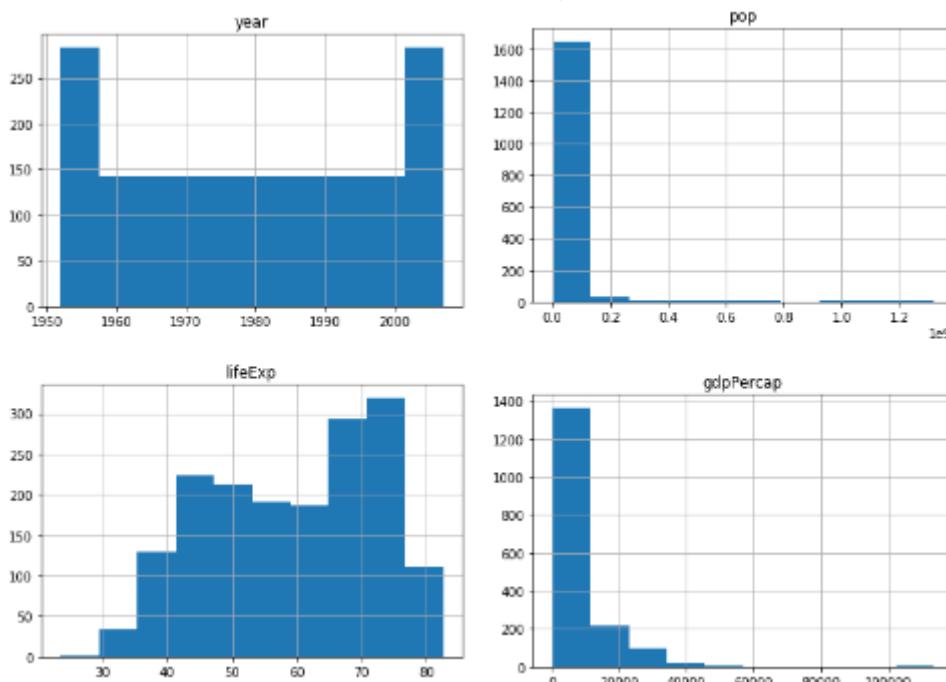
	year	lifeExp	pop	gdpPercap
count	1704.00000	1704.000000	1.704000e+03	1704.000000
mean	1979.50000	59.474439	2.960121e+07	7215.327081
std	17.26533	12.917107	1.061579e+08	9857.454543
min	1952.00000	23.599000	6.001100e+04	241.165877
25%	1965.75000	48.198000	2.793664e+06	1202.060309
50%	1979.50000	60.712500	7.023596e+06	3531.846989
75%	1993.25000	70.845500	1.958522e+07	9325.462346
max	2007.00000	82.603000	1.318683e+09	113523.132900

```
# 連續型資料視覺化 matplotlib :連結 https://ithelp.ithome.com.tw/articles/10186484
# 盒鬚圖 (Box plot) plt.boxplot(df.[‘col_name’])
# 直方圖 (Histogram) plt.hist(col_samples)
# 起一個簡單的迴圈
print("="*30, "箱型圖 迴圈", "="*30)
for col in numeric_cols: # 使用迴圈一欄一欄繪製 箱型圖
    numeric_columns[[col]].boxplot()
    plt.show()
print("="*30, "直方圖 迴圈", "="*30)
for col in numeric_cols: # 使用迴圈一欄一欄繪製 直方圖
    numeric_columns[[col]].hist()
    plt.show()
```

===== 箱型圖迴圈 =====



===== 直方圖迴圈 =====



2-3.2 類別型變數 預視 (含基礎聚集 def)

- df[df_name].value_counts()
- 柱狀圖 plt.bar(cat_value_counts.index, cat_value_counts)
- 圓餅圖 plt.pie(cat_value_counts, labels = cat_value_counts.index)

```
# 1.定義類別欄位: "category", "object" 等, 皇上第二點, 若數值欄位 coding 只有2值(通常是0,1)也有可能是類別欄位
category_columns = gapminder.select_dtypes(exclude=["number"])
# 2.列出各欄位有幾種coding #https://zhuanlan.zhihu.com/p/29986906
#gapminder['country'].value_counts()
#list(gapminder['country'].value_counts().index)
#DataFrame結構: pd.crosstab(index=gapminder['country'], columns="count")
#迴圈
#for cat in category_columns.columns.values:
#    print(gapminder[cat].value_counts())

#將簡單的迴圈定義一個簡單巨集
def cat_preview(cat_list):
    for cat in cat_list:
        print(gapminder[cat].value_counts())
    return

cat_preview(category_columns.columns.values)
Bulgaria           12
Iraq               12
Hong Kong, China  12
...
Serbia             12
Cote d'Ivoire     12
Morocco            12
Tanzania           12
Name: country, Length: 142, dtype: int64
Africa              624
Asia                396
Europe              360
Americas            300
Oceania              24
Name: continent, dtype: int64
```

```
# 類別型資料視覺化 matplotlib(柱狀圖、圓餅圖等): 連結 https://ithelp.ithome.com.tw/articles/10196410
```

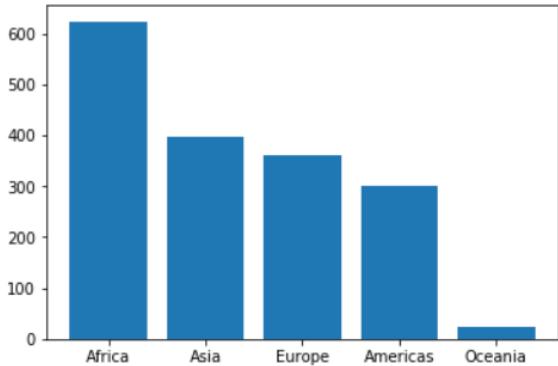
```
cat_value_counts=gapminder["continent"].value_counts()
print ("=" *30, "柱狀圖", "=" *30)
plt.bar(cat_value_counts.index, cat_value_counts)
plt.show()
print ("=" *30, "圓餅圖", "=" *30)
plt.pie(cat_value_counts, labels = cat_value_counts.index)
plt.show()
```

```
#將簡單的迴圈定義一個簡單巨集
```

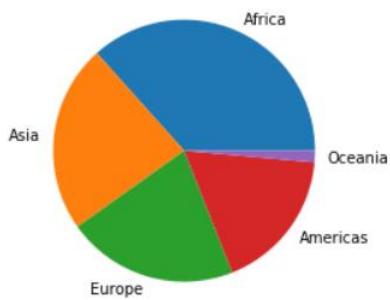
```
def cat_bar(cat_list):
    for cat in cat_list:
        gapminder[cat].value_counts().plot('bar')
        plt.show()
    return
```

```
print ("=" *30,"柱狀圖 迴圈", "=" *30)
cat_bar(category_columns.columns.values)
```

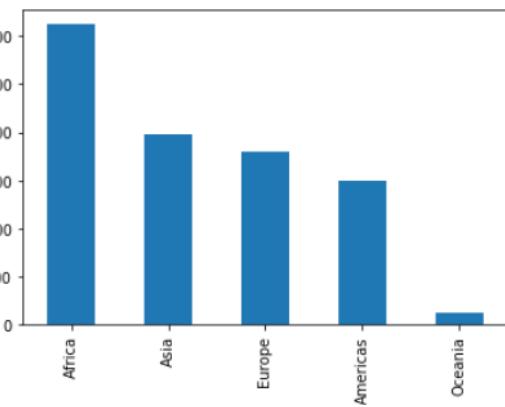
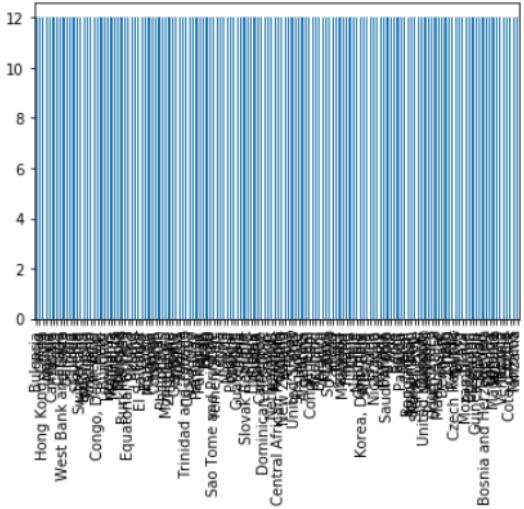
===== 柱狀圖 =====



===== 圓餅圖 =====



===== 柱狀圖 迴圈 =====



2-3.3 資料分組 預視

<https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/>

資料截取

- Subset1: 選擇觀察值
- Subset2: 選擇欄位
- Loc[]: 選擇觀察值、欄位
- filter() 多個條件的資料篩選

資料分析

- Grouping data: groupby()
- Aggregating data: aggregate()
- apply() 新增欄位
- lambda 函數計算

Subset 1.1

#撰寫布林判斷條件將符合條件的觀測值從資料框中篩選出，例如選出臺灣：
gapminder[gapminder['country'] == 'Taiwan'].head()

	country	continent	year	lifeExp	pop	gdpPercap
1500	Taiwan	Asia	1952	58.50	8550362	1206.947913
1501	Taiwan	Asia	1957	62.40	10164215	1507.861290
1502	Taiwan	Asia	1962	65.20	11918938	1822.879028
1503	Taiwan	Asia	1967	67.50	13648692	2643.858681
1504	Taiwan	Asia	1972	69.39	15226039	4062.523897

Subset 1.2

#如果有多個條件，可以使用 | 或 & 符號連結，例如選出 2007 年的亞洲國家：
gapminder[(gapminder['year'] == 2007) & (gapminder['continent'] == 'Asia')].head()

	country	continent	year	lifeExp	pop	gdpPercap
11	Afghanistan	Asia	2007	43.828	31889923	974.580338
95	Bahrain	Asia	2007	75.635	708573	29796.048340
107	Bangladesh	Asia	2007	64.062	150448339	1391.253792
227	Cambodia	Asia	2007	59.723	14131858	1713.778686
299	China	Asia	2007	72.961	1318683096	4959.114854

Subset 2.1

#用 list 標註變數名稱可以將變數從資料框中選出，實踐 select() 函數的功能，例如選出 country 與 continent 變數：
gapminder[['country', 'continent']].head()

	country	continent
0	Afghanistan	Asia
1	Afghanistan	Asia
2	Afghanistan	Asia
3	Afghanistan	Asia
4	Afghanistan	Asia

```
# Subset 2.2
#如果只選一個變數且沒有以 list 標註，同樣能選出變數，但是型別會變為 Series：
country = gapminder['country']
print(type(country))
<class 'pandas.core.series.Series'>
```

```
# Loc[]
gapminder.loc[0:20,["pop", "year", "continent"]]
```

	pop	year	continent
0	8425333	1952	Asia
1	9240934	1957	Asia
2	10267083	1962	Asia
3	11537966	1967	Asia
4	13079460	1972	Asia
.....			
20	3326498	1992	Europe

#filter() + groupby() 多個條件的資料篩選

```
def filter_func(x):
    return x["pop"].sum() < 6185000000
#display DEMO DataFrame
display('gapminder', gapminder.head(),
        "gapminder.groupby('continent').sum()", gapminder.groupby('continent').sum(),
        "gapminder.groupby('continent').filter(filter_func)", gapminder.groupby('continent').filter(filter_func).head())
'gapminder'
```

	country	continent	year	lifeExp	pop	gdpPerCap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

"gapminder.groupby('continent').sum()"

continent	year	lifeExp	pop	gdpPerCap
Africa	1235208	30491.96600	6187585961	1.368903e+06
Americas	593850	19397.62100	7351438499	2.140833e+06
Asia	783882	23785.70168	30507333901	3.129252e+06
Europe	712620	25885.32700	6181115304	5.209011e+06
Oceania	47508	1783.82900	212992136	4.469186e+05

"gapminder.groupby('continent').filter(filter_func)"

	country	continent	year	lifeExp	pop	gdpPerCap
12	Albania	Europe	1952	55.23	1282697	1601.056136
13	Albania	Europe	1957	59.28	1476505	1942.284244
14	Albania	Europe	1962	64.82	1728137	2312.888958
15	Albania	Europe	1967	66.22	1984060	2760.196931
16	Albania	Europe	1972	67.69	2263554	3313.422188

```
# groupby() partI: 每一組的數量
gapminder.groupby(["continent"]).size()
```

```
continent
Africa      624
Americas    300
Asia        396
Europe      360
Oceania     24
dtype: int64
```

```
# groupby() partII: 每一組的連續描述
```

```
gapminder.groupby(["continent"])['pop'].describe()
```

	count	mean	std	min	25%	50%	75%	max
continent								
Africa	624.0	9.916003e+06	1.549092e+07	60011.0	1342075.00	4579311.0	10801489.75	1.350312e+08
Americas	300.0	2.450479e+07	5.097943e+07	662850.0	2962358.75	6227510.0	18340309.00	3.011399e+08
Asia	396.0	7.703872e+07	2.068852e+08	120447.0	3844393.00	14530830.5	46300348.00	1.318683e+09
Europe	360.0	1.716976e+07	2.051944e+07	147962.0	4331500.00	8551125.0	21802867.00	8.240100e+07
Oceania	24.0	8.874672e+06	6.506342e+06	1994794.0	3199212.50	6403491.5	14351625.00	2.043418e+07

```
# groupby() partIII: produces Pandas Series or DataFrame
```

```
print("Produces Pandas DataFrame")
gapminder.groupby(["continent"])[['pop']].sum()
```

Produces Pandas DataFrame

pop

continent
Africa 6187585961
Americas 7351438499
Asia 30507333901
Europe 6181115304
Oceania 212992136

```
# aggregate() partI 一個變數的指定統計值
```

```
grouped = gapminder.agg({"pop": [min, max, sum]})
```

```
grouped.head()
```

pop

min	60011
max	1318683096
sum	50440465801

```
# aggregate() partII 將每個統計值命名
```

```
grouped = gapminder.groupby(["continent"]).agg({"pop": [min, max, sum]})
```

```
grouped.columns = ["_".join(x) for x in grouped.columns.ravel()]
```

```
grouped.head()
```

pop_min pop_max pop_sum

continent	pop_min	pop_max	pop_sum
Africa	60011	135031164	6187585961
Americas	662850	301139947	7351438499
Asia	120447	1318683096	30507333901
Europe	147962	82400996	6181115304
Oceania	1994794	20434176	212992136

```
# aggregate() partIII 多個變數的指定統計值
gapminder.groupby("continent").aggregate({"pop": 'min',
                                         "year": 'max'})
```

pop year

continent	pop	year
Africa	60011	2007
Americas	662850	2007
Asia	120447	2007
Europe	147962	2007
Oceania	1994794	2007

```
# aggregate() partIV --- lambda 與多個變數及統計值
# Define the aggregation procedure outside of the groupby operation
aggregations = {
    'pop':{
        'pop_min':'min',
        'pop_max':'max',
        'pop_sum':'sum'},
    'year': {
        'year_range': lambda x: sum(x)-x.mean()
    }
}
gapminder.groupby("continent").agg(aggregations)
```

pop year

	pop_min	pop_max	pop_sum	year_range
--	----------------	----------------	----------------	-------------------

continent	pop_min	pop_max	pop_sum	year_range
Africa	60011	135031164	6187585961	1233228.5
Americas	662850	301139947	7351438499	591870.5
Asia	120447	1318683096	30507333901	781902.5
Europe	147962	82400996	6181115304	710640.5
Oceania	1994794	20434176	212992136	45528.5

```
# apply() 新增欄位 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.apply.html
gapminder['pop_10000'] = gapminder['pop'].apply(lambda x: x/10000)
gapminder.head()
```

	country	continent	year	lifeExp	pop	gdpPercap	pop_10000
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	842.5333
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	924.0934
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	1026.7083
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	1153.7966
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	1307.9460

```
# 匿名函數 lambda 運算式來定義函式，執行運算式時將會產生函式物件。
#例: 資料轉換
subset=gapminder.loc[0:1000, ["pop", "year", "continent"]]
subset.groupby(["continent"]).transform(lambda x: x / x.mean())
#等同於 subset.groupby(["continent"]).apply(lambda x: x / x.mean())
```

	pop	year
0	0.071355	0.986305
1	0.078262	0.988831
2	0.086953	0.991358
3	0.097716	0.993884
4	0.110771	0.996410
...
996	0.006781	0.986305
997	0.007471	0.988831
998	0.008556	0.991358
999	0.009735	0.993884
1000	0.011183	0.996410

1001 rows × 2 columns

```
# 較複雜的邏輯你可以使用 def 來定義。
def func(x, y):
    return x * y
gapminder.apply(lambda gapminder: func(gapminder["pop"], gapminder["year"])), axis=1)
0      16446250016
1      18084507838
2      20144016846
3      22695179122
...
1700    21323045280
1701    22775681156
1702    23876979126
1703    24708464001
Length: 1704, dtype: int64
```

```
#畫圖
#時間與數值：線圖
print("將臺灣資料篩選出來並繪製從 1952 年至 2007 年的人口變化：")
gapminder_twn = gapminder[gapminder['country'] == 'Taiwan']
gapminder_twn[['year', 'pop']].plot(kind = 'line', x = 'year', y = 'pop', title = 'Pop vs. Year in Taiwan', legend = False)
plt.show()
print("將中國、日本、南韓與臺灣資料篩選出來並繪製從 1952 年至 2007 年的平均壽命變化：")
gapminder_northasia = gapminder.loc[gapminder['country'].isin(['China', 'Japan', 'Korea, Rep.', 'Taiwan'])]
gapminder_northasia_pivot = gapminder_northasia.pivot_table(values = 'lifeExp', columns = 'country', index = 'year')
gapminder_northasia_pivot.plot(title = 'Life Expectancies in North Asia')
plt.show()
#類別與數值：直方圖、箱型圖、柱狀圖
print("人均所得直方圖依照不同洲別以不同顏色繪製：")
gapminder_2007 = gapminder[gapminder['year'] == 2007]
gapminder_continent_pivot = gapminder_2007.pivot_table(values = 'gdpPercap', columns = 'continent', index = 'country')
gapminder_continent_pivot.plot(kind = 'hist', alpha=0.5, bins = 20, title = 'GDP Per Capita by Continent')
plt.show()
print("依照不同洲別，將人均所得以盒鬚圖繪製：")
gapminder_continent_pivot.plot(kind = 'box', title = 'GDP Per Capita by Continent')
plt.show()
print("繪製 2007 年各洲平均壽命、平均財富：")
```

```

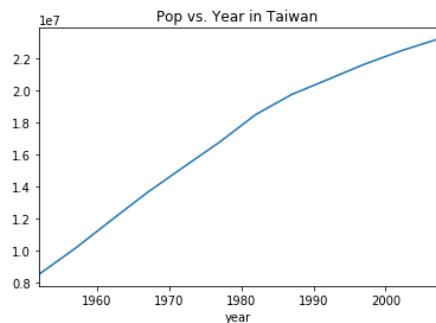
summarized_df = gapminder[gapminder['year'] == 2007].groupby(by = 'continent')[['lifeExp', 'gdpPercap']].mean()
summarized_df.plot(kind = 'barh', subplots = True, layout = (1, 2), sharex = False, sharey = True, legend = False)
plt.show()

# 相關性：散佈圖、hexbin plot
print("2007 年各國人均所得與平均壽命的散佈圖：")
gapminder_2007.plot(kind = 'scatter', x = 'gdpPercap', y = 'lifeExp', title = 'Wealth vs. Health in 2007')
plt.show()

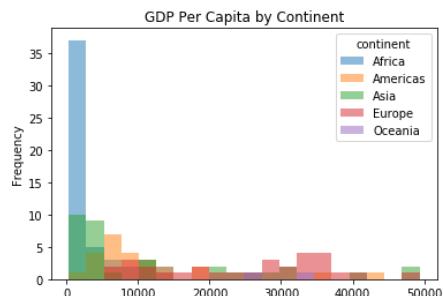
print("2007 年各國人均所得與平均壽命的 hexbin plot：")
gapminder_2007.plot(kind = 'hexbin', x = 'gdpPercap', y = 'lifeExp', title = 'Wealth vs. Health in 2007', gridsize = 20)
plt.show()

```

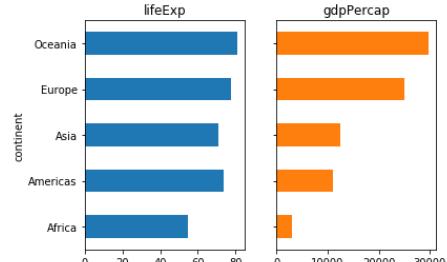
將臺灣資料篩選出來並繪製
從 1952 年至 2007 年的人口變化：



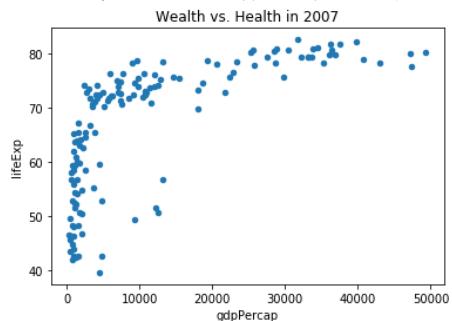
人均所得直方圖依照不同洲別以不同顏色繪製：



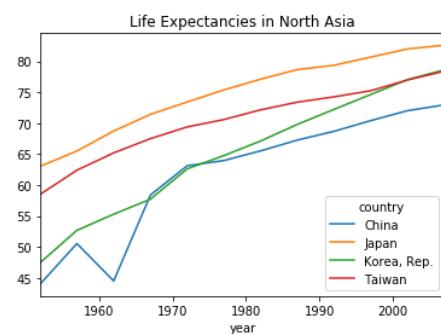
繪製 2007 年各洲平均壽命、平均財富：



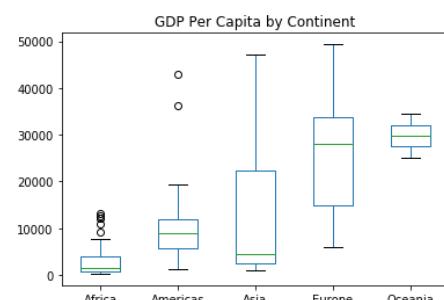
2007 年各國人均所得與平均壽命的散佈圖：



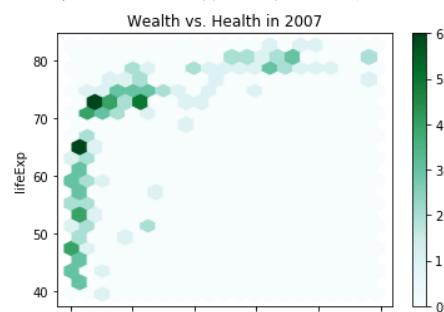
將中國、日本、南韓與臺灣資料篩選出來並繪製
從 1952 年至 2007 年的平均壽命變化：



依照不同洲別，將人均所得以盒鬚圖繪製：



2007 年各國人均所得與平均壽命的 hexbin plot：



2-4 多個資料合併 merge / transform

本小節回顧課程 Day_008。

```
# 生成範例用的資料()
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3'],
                     'C': ['C0', 'C1', 'C2', 'C3'],
                     'D': ['D0', 'D1', 'D2', 'D3']},
                     index=[0, 1, 2, 3])
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                     'B': ['B4', 'B5', 'B6', 'B7'],
                     'C': ['C4', 'C5', 'C6', 'C7'],
                     'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                     'B': ['B8', 'B9', 'B10', 'B11'],
                     'C': ['C8', 'C9', 'C10', 'C11'],
                     'D': ['D8', 'D9', 'D10', 'D11']},
                     index=[8, 9, 10, 11])
df4 = pd.DataFrame({'B': ['B2', 'B3', 'B6', 'B7'],
                     'D': ['D2', 'D3', 'D6', 'D7'],
                     'F': ['F2', 'F3', 'F6', 'F7']},
                     index=[2, 3, 6, 7])
```

```
# 沿縱軸合併
result = pd.concat([df1, df2, df3])
result
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
# 沿橫軸合併
result = pd.concat([df1, df4], axis=1)
result
```

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

```
# 沿橫軸合併
result = pd.concat([df1, df4], axis=1, join='inner')
# 硬串接
print(result)

result = pd.merge(df1, df4, how='inner')
print(result)
```

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
	A	B	C	D	F		
0	A2	B2	C2	D2	F2		
1	A3	B3	C3	D3	F3		

```
# 將欄-列逐一解開
```

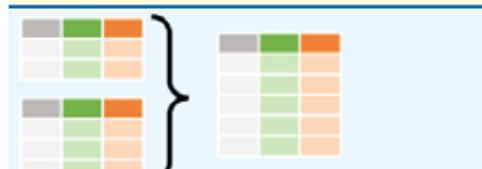
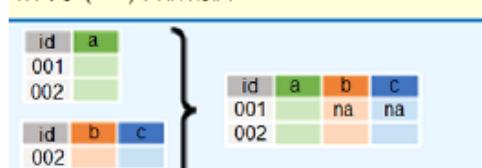
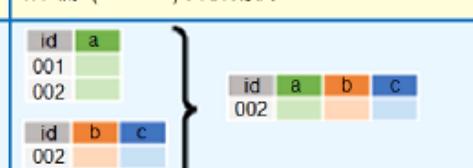
```
print(df1)
```

```
df1.melt()
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	variable	value
0	A	A0
1	A	A1
2	A	A2
3	A	A3
4	B	B0
5	B	B1
6	B	B2
7	B	B3
8	C	C0
9	C	C1
10	C	C2
11	C	C3
12	D	D0
13	D	D1
14	D	D2
15	D	D3

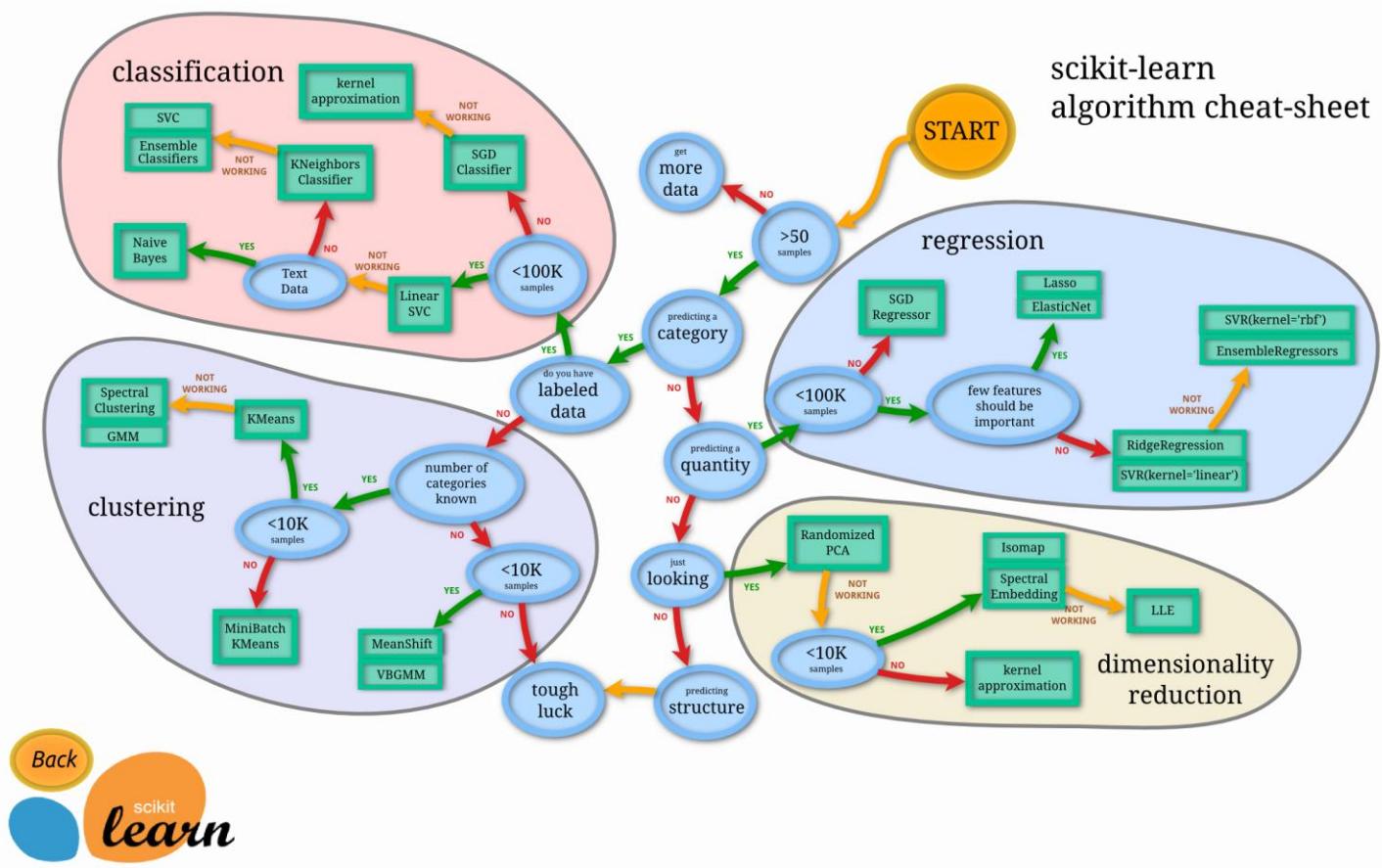
本章的總複習可以參考官方 PDF 簡介^{#2.5}。

 pd.melt(df) 將"欄" (column) 轉成"列" (row)	 pd.pivot(columns='var', values='val') 將"列" (row) 轉成 "欄" (column)
 pd.concat([df1, df2]) 沿"列" (row) 合併兩個 dataframe	 pd.concat([df1, df2], axis = 1) 沿"欄" (column) 合併兩個 dataframe
 pd.merge(df1, df2, on = 'id', how = 'outer') 將 df1, df2 以 "id" 這欄做全合併 (遺失以 na 補)	 pd.merge(df1, df2, on = 'id', how = 'inner') 將 df1, df2 以 "id" 這欄做部分合併

第三章 格式化資料

【資料預視、預處理及 Sklearn】

Scikit-learn 簡稱 **SKlearn**，SKlearn 已內建在一開始安裝的 Anaconda 套件中，可以直接使用。除了包含許多知名的 Machine learning 的算法之外，其內建許多的知名的 dataset (如鳶尾花 Iris 以及手寫辨識數字 digits 的資料)，可以快速用兩三行的程式碼就將資料轉成 pandas 的格式。Sklearn 在官網把功能分為六大塊，分別是監督式學習(Supervised learning)的分類(Classification)以及迴歸(Regression)演算法、非監督式的分群演算法(Clustering)、還有降低維度(Dimensionality reduction)、模型選擇(Model selection)以及資料的前處理(Preprocess)。



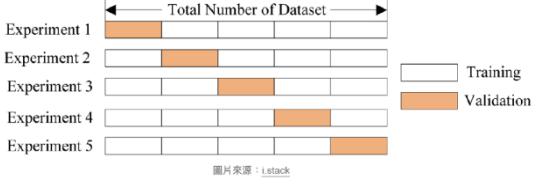
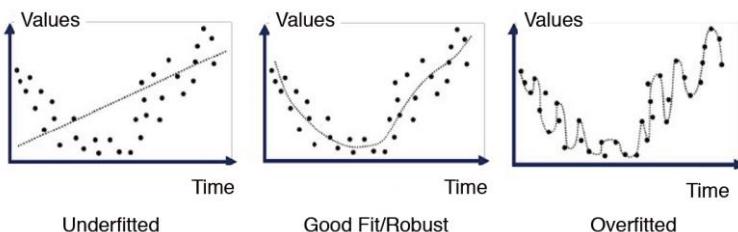
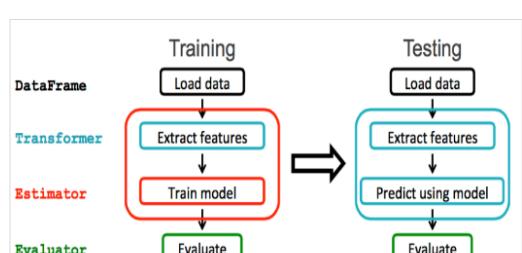
首先總結回顧 Day_042 內容，機器學習基礎模型建立的架構列為表。第二章已先介紹資料預視，本章再進入資料整理，這部分即是「前處理(Preprocess)」，稱之為**特徵工程**。接下來，先以「**分類 (Classification)**」與「**迴歸 (Regression)**」的監督式學習(Supervised learning)做範例，這兩個前者預測類別目標變數，後者預測連續型目標變數。前者又依二元分類 (Binary classification) 的演算法為 Logistic 邏輯斯迴歸 分類器(Classifier)、

多元分類 (Multiclass classification) 演算法為 [k-Nearest Neighbors](#) 分類器或 [決策樹](#) 分類器(Decision Tree Classifiers)。

依照表中步驟依序進行，檢視/整理資料(步驟 2)後，進行模型估計(步驟 3)再評估模型好壞(步驟 4)，重覆嘗試不同設定找出適當的模型，重覆的動作就是特徵篩選，也可以另依一些指標或方法進行模型篩選(feature/model selection) (步驟 5)。文中共有三個範例，各有不同情境來解釋各步驟的處理方法。範例一為「鐵達尼號的乘客生存預測」，預測因子為類別二分類的生或死，使用 Logistic regression，在這例子，會針對連續型及類別型的預測因子進行資料整理，利用評估的指標比較各種資料處理的方式。範例二為「計程車費率預測」，預測因子為連續型的費率，使用 liner regression，這例子針對時間及距離的特徵資料整理進行整理範例。範例三為 Sklearn 內建的「鳶尾花 Iris 辨識」資料，預測多類別分類問題，並針對資料分割，分為訓練集測試集，保留部分的資料進行步驟 4 的模型評估，可以避免過擬合 (Over-fitting) 的情形發生。

依 Day_042 內容，機器學習基礎模型建立的架構列為下表：

STEP1:	匯入套件
STEP2.1: 匯入/預視 資料	承第二章，回顧課程 Day_003_build_dataframe 讀進資料，並檢查資料的 shape (有多少 samples (rows), 多少 features (columns), label 的型態是什麼？)，搭配資料分布圖增加視覺上的判斷。 <pre>import pandas as pd import numpy as np from sklearn import datasets # 使用 pandas 讀取 .csv 檔 : pd.read_csv # 使用 numpy 讀取 .txt 檔 : np.loadtxt # 使用 Scikit-learn 內建的資料集 : sklearn.datasets.load_XXX 檢查資料數量 : data.shape (data should be np.array or dataframe)</pre>
STEP2.2: 特徵工程處 理	本章開始回顧課程 Day_017-030_Feature_Engineering，資料除了數值型特徵、類別型特徵；另外還有二元特徵、排序型特徵、時間型特徵。「特徵工程」是將這些特徵的原始資料(可能是文字、數值等)轉換成對應的數值(調整過的分數、數字分組標籤等)。 <pre>from sklearn.preprocessing import MinMaxScaler, StandardScaler from sklearn.preprocessing import LabelEncoder # D19_數值型特徵-缺值填補 (df.fillna(-1), df.fillna(0), df.fillna(df.mean())) # D19_數值型特徵-標準化 (MinMaxScaler().fit(), StandardScaler().fit()) # D20_數值型特徵-捨棄群值 # D21_數值型特徵-去除偏態 (log1p, boxcox (lambda=0)) # D22_類別型特徵(有序離散值)-標籤編碼 (LabelEncoder().fit_transform(df_obj[c])) # D22_類別型特徵(無序離散值)-獨熱編碼 (pd.get_dummies(df_obj)) # from sklearn.preprocessing import OneHotEncoder # D23_類別型特徵(類別數少量)-均值編碼 (groupby(['Y']).mean().reset_index()) # D23_類別型特徵 - Counting 計數編碼 (count_df['X_Count']) # D23_類別型特徵 - Feature Hash 特徵雜湊 (count_df['X'].map(lambda x:hash(x) % 5)) # Day_027_特徵組合 - 類別與數值組合「群聚編碼」 # Day_028_特徵選擇: 有 三大類 方法 # • 過濾法 (Filter) : 《相關係數過濾法》 # • 包裝法 (Wrapper) : 《Stepwise》 # • 嵌入法 (Embedded) : 《L1(Lasso)嵌入法》、《GDBT(梯度提升樹)嵌入法》 # Day_025, 026_DayTime_Features_時間及經緯度資料 - # datetime.strptime("21/11/06 16:30", "%d/%m/%y %H:%M") # df['distance_2D'] = (df['longitude_diff']**2 + df['latitude_diff']**2)**0.5</pre>

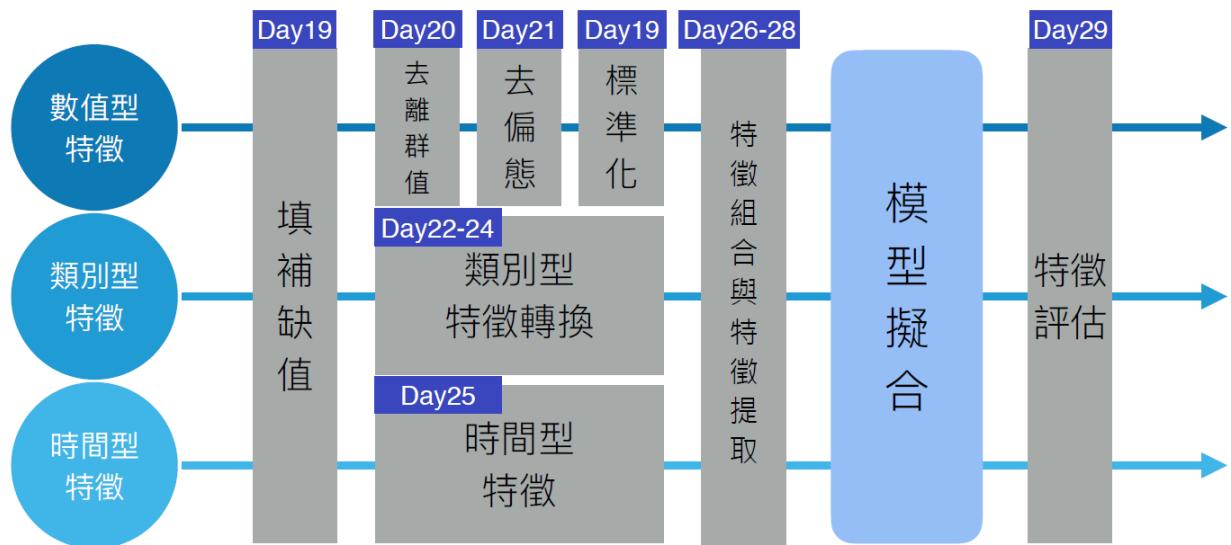
STEP2.3: 分割資料	<p>回顧課程 Day_034_train_test_split。良好的資料輸入取決於資料預處理與特徵工程，而好的輸入將大大影響到模型是否可以發揮其理論正常水準，真正在進行機器學習工作流程的時候資料預處理往往是最花時間的部份。同時為了方便模型的校調，因此會把資料集分為訓練資料、驗證資料。</p> <p>將資料切為訓練 (train) / 測試 (test)</p> <pre>from sklearn.model_selection import train_test_split train_test_split(data)</pre>  <p>圖片來源：slack</p> <p>若僅做一次訓練/測試集切分，有些資料會沒有被拿來訓練過，因此後續就有 cross-validation 的方法，可以讓結果更為穩定，K 為 fold 數量，即 K-fold Cross-validation。</p> <p>圖為 5-fold cross-validation，每筆資料都曾經當過一次驗證集，再取平均得到最終結果。</p>		
STEP3: 建立模型	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; vertical-align: top;"> <p>連續型：預測的目標值為實數 (-∞至 ∞) 回歸(Regression)</p> <pre>from sklearn.linear_model import LinearRegression from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor</pre> <p>將資料 fit 進模型開始訓練</p> <pre>#一般線性模型 #model = LinearRegression().fit(X,y) #樹狀線性模型 #model = DecisionTreeRegressor() model = RandomForestRegressor() model.fit(x_train, y_train)</pre> </td><td style="padding: 5px; vertical-align: top;"> <p>類別型：預測的目標值為類別(二分類/多分類) 分類(Classification)</p> <pre>from sklearn.linear_model import LogisticRegression #一般分類模型-二分類 # model = LogisticRegression().fit(X,y) from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import GradientBoostingClassifier #一般分類模型-二分或多分類 # model = KNeighborsClassifier() #樹狀分類模型 #model = DecisionTreeClassifier() #model = RandomForestClassifier() model = GradientBoostingClassifier() model.fit(x_train, y_train)</pre> </td></tr> </table>	<p>連續型：預測的目標值為實數 (-∞至 ∞) 回歸(Regression)</p> <pre>from sklearn.linear_model import LinearRegression from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor</pre> <p>將資料 fit 進模型開始訓練</p> <pre>#一般線性模型 #model = LinearRegression().fit(X,y) #樹狀線性模型 #model = DecisionTreeRegressor() model = RandomForestRegressor() model.fit(x_train, y_train)</pre>	<p>類別型：預測的目標值為類別(二分類/多分類) 分類(Classification)</p> <pre>from sklearn.linear_model import LogisticRegression #一般分類模型-二分類 # model = LogisticRegression().fit(X,y) from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import GradientBoostingClassifier #一般分類模型-二分或多分類 # model = KNeighborsClassifier() #樹狀分類模型 #model = DecisionTreeClassifier() #model = RandomForestClassifier() model = GradientBoostingClassifier() model.fit(x_train, y_train)</pre>
<p>連續型：預測的目標值為實數 (-∞至 ∞) 回歸(Regression)</p> <pre>from sklearn.linear_model import LinearRegression from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor</pre> <p>將資料 fit 進模型開始訓練</p> <pre>#一般線性模型 #model = LinearRegression().fit(X,y) #樹狀線性模型 #model = DecisionTreeRegressor() model = RandomForestRegressor() model.fit(x_train, y_train)</pre>	<p>類別型：預測的目標值為類別(二分類/多分類) 分類(Classification)</p> <pre>from sklearn.linear_model import LogisticRegression #一般分類模型-二分類 # model = LogisticRegression().fit(X,y) from sklearn.neighbors import KNeighborsClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import GradientBoostingClassifier #一般分類模型-二分或多分類 # model = KNeighborsClassifier() #樹狀分類模型 #model = DecisionTreeClassifier() #model = RandomForestClassifier() model = GradientBoostingClassifier() model.fit(x_train, y_train)</pre>		
STEP4: 評估 validation	<p>回顧課程 Day_33 overfitting 的概念</p>  <p>交叉驗證(Cross validation)，交叉驗證用於防止模型過於複雜而引起的過擬合。有時亦稱迴圈估計，是一種統計學上將資料樣本切割成較小子集的實用方法(STEP2.3)。於是可以在一個子集上做分析，而其它子集則用來做後續對此分析的確認及驗證。一開始的子集被稱為訓練集。而其它的子集則被稱為驗證集或測試集。交叉驗證是一種評估統計分析、機器學習演算法對獨立於訓練資料集的<u>泛化能力</u> (<u>generalize</u>)。</p> <p>機器學習模型的目標函數中有兩個非常重要的元素</p> <ul style="list-style-type: none"> • 損失函數 (Loss function)：衡量預測值與實際值的差異，讓模型能往正確的方向學習 • 正則化 (Regularization)：是避免模型變得過於複雜，造成過擬合 (Over-fitting) <div style="text-align: right; margin-top: 20px;">  <pre> graph LR subgraph Training [Training] direction TB T1[Load data] --> T2[Extract features] T2 --> T3[Train model] T3 --> T4[Evaluate] end subgraph Testing [Testing] direction TB T5[Load data] --> T6[Extract features] T6 --> T7[Predict using model] T7 --> T8[Evaluate] end T3 <--> T6 </pre> </div>		

目標函數 1 指標函數 / 損失函數	<p>回顧課程 Day_036 evaluation metrics 將測試資料 (features) 放進訓練好的模型中，得到 prediction，與測試資料的 label (y_test) 做評估</p> <pre>From sklearn.model_selection import cross_val_score estimator = LogisticRegression() cross_val_score(estimator, train_X, train_Y, cv=5).mean()</pre> <ul style="list-style-type: none"> ● cross_val_score 這個函數驗證的計分方式(scoring) 是可以由 scoring 指定的。 ● 如果沒有指定，則回歸問題會選擇使用 r2，分類問題會選擇 accuracy。 ● accuracy 當然分數就不可能為負，但是 r2 就有可能為負的喔。 <table border="1" data-bbox="270 418 1529 487"> <thead> <tr> <th data-bbox="270 418 869 487">連續型：預測的目標值為實數 (-∞至 ∞) 回歸(Regression)</th><th data-bbox="869 418 1529 487">類別型：預測的目標值為類別(二分類/多分類) 分類(Classification)</th></tr> </thead> <tbody> <tr> <td data-bbox="270 487 869 1233"> <pre>from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score</pre> <ul style="list-style-type: none"> ● MAE ● MSE ● R-square <pre>prediction = model.predict(x_test) mean_absolute_error(prediction, y_test) mean_squared_error(prediction, y_test) r2_score(prediction, y_test)</pre> </td><td data-bbox="869 487 1529 1233"> <pre>from sklearn.metrics import roc_auc_score, f1accuracy_score, f1_score, precision_score, recall_score, fbeta_score</pre> <ul style="list-style-type: none"> ● Accuracy ● ROC (Receiver Operating Curve)/ AUC (Area Under Curve) ● F1-Score, Precision, Recall ● F2-Score (詳見語法) <pre># 使用 roc_auc_score 來評估特別注意 y_pred 必須要放機率值進去，得到結果 0.5，表示無預測能力；結果 1 或是 0 表示可二分類。 Roc_auc_score(y_test, y_pred)</pre> <pre># 其它的 y_pred 放入二分類的數值即可，使用 np.where 函數，將 y_pred > 0.5 的值變為 1，小於 0.5 的為 0 y_pred_binarized = np.where(y_pred>threshold, 1, 0) f1_score(y_test, y_pred_binarized) precision_score(y_test, y_pred_binarized) recall_score(y_test, y_pred_binarized)</pre> </td></tr> </tbody> </table>	連續型：預測的目標值為實數 (-∞至 ∞) 回歸(Regression)	類別型：預測的目標值為類別(二分類/多分類) 分類(Classification)	<pre>from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score</pre> <ul style="list-style-type: none"> ● MAE ● MSE ● R-square <pre>prediction = model.predict(x_test) mean_absolute_error(prediction, y_test) mean_squared_error(prediction, y_test) r2_score(prediction, y_test)</pre>	<pre>from sklearn.metrics import roc_auc_score, f1accuracy_score, f1_score, precision_score, recall_score, fbeta_score</pre> <ul style="list-style-type: none"> ● Accuracy ● ROC (Receiver Operating Curve)/ AUC (Area Under Curve) ● F1-Score, Precision, Recall ● F2-Score (詳見語法) <pre># 使用 roc_auc_score 來評估特別注意 y_pred 必須要放機率值進去，得到結果 0.5，表示無預測能力；結果 1 或是 0 表示可二分類。 Roc_auc_score(y_test, y_pred)</pre> <pre># 其它的 y_pred 放入二分類的數值即可，使用 np.where 函數，將 y_pred > 0.5 的值變為 1，小於 0.5 的為 0 y_pred_binarized = np.where(y_pred>threshold, 1, 0) f1_score(y_test, y_pred_binarized) precision_score(y_test, y_pred_binarized) recall_score(y_test, y_pred_binarized)</pre>
連續型：預測的目標值為實數 (-∞至 ∞) 回歸(Regression)	類別型：預測的目標值為類別(二分類/多分類) 分類(Classification)				
<pre>from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score</pre> <ul style="list-style-type: none"> ● MAE ● MSE ● R-square <pre>prediction = model.predict(x_test) mean_absolute_error(prediction, y_test) mean_squared_error(prediction, y_test) r2_score(prediction, y_test)</pre>	<pre>from sklearn.metrics import roc_auc_score, f1accuracy_score, f1_score, precision_score, recall_score, fbeta_score</pre> <ul style="list-style-type: none"> ● Accuracy ● ROC (Receiver Operating Curve)/ AUC (Area Under Curve) ● F1-Score, Precision, Recall ● F2-Score (詳見語法) <pre># 使用 roc_auc_score 來評估特別注意 y_pred 必須要放機率值進去，得到結果 0.5，表示無預測能力；結果 1 或是 0 表示可二分類。 Roc_auc_score(y_test, y_pred)</pre> <pre># 其它的 y_pred 放入二分類的數值即可，使用 np.where 函數，將 y_pred > 0.5 的值變為 1，小於 0.5 的為 0 y_pred_binarized = np.where(y_pred>threshold, 1, 0) f1_score(y_test, y_pred_binarized) precision_score(y_test, y_pred_binarized) recall_score(y_test, y_pred_binarized)</pre>				
目標函數 2 正規化	<p>在這會給範例，特徵選擇中嵌入法 (Embedded) 的 L1(Lasso) 嵌入法，即使用機器學習模型，根據擬合後的係數，刪除係數低於門檻的特徵。</p> <ul style="list-style-type: none"> ● L1、L2 可調整 alpha 值決定正則化的強度，印出模型參數的數值都明顯小於單純的線性回歸。 <table border="1" data-bbox="270 1347 1529 1495"> <tr> <td data-bbox="270 1347 869 1495"> <pre>from sklearn.linear_model import Lasso reg = Lasso(alpha=0.1) reg.fit(X, y) print(reg.coef_) # 印出訓練後的模型參數</pre> </td><td data-bbox="869 1347 1529 1495"> <pre>from sklearn.linear_model import Ridge reg = Ridge(alpha=0.1) reg.fit(X, y) print(reg.coef_) # 印出訓練後的模型參數</pre> </td></tr> </table>	<pre>from sklearn.linear_model import Lasso reg = Lasso(alpha=0.1) reg.fit(X, y) print(reg.coef_) # 印出訓練後的模型參數</pre>	<pre>from sklearn.linear_model import Ridge reg = Ridge(alpha=0.1) reg.fit(X, y) print(reg.coef_) # 印出訓練後的模型參數</pre>		
<pre>from sklearn.linear_model import Lasso reg = Lasso(alpha=0.1) reg.fit(X, y) print(reg.coef_) # 印出訓練後的模型參數</pre>	<pre>from sklearn.linear_model import Ridge reg = Ridge(alpha=0.1) reg.fit(X, y) print(reg.coef_) # 印出訓練後的模型參數</pre>				
	<ul style="list-style-type: none"> ● 複習課程 D39_regression model_LASSOvsRidge 的問與答： <p>1. LASSO (L1) 回歸可以被用來作為 Feature selection 的工具，請了解 LASSO 模型為什麼可用來作 Feature selection？</p> <p>當模型的目標函數加入 L1 正則化後，會傾向讓用不到的特徵的權重降為 0，因此實際上可以替我們做到 feature selection 的功用。詳細的數學原理可以參考台大資工林軒田教授的影片推導 https://www.youtube.com/watch?time_continue=1&v=PeQeKeeGu3A</p> <p>2. 當自變數 (X) 存在高度共線性時，Ridge Regression (L2) 可以處理這樣的問題嗎？</p> <p>可以。Ridge Regression 就是透過 L2 進行權重的 decay，進而減少特徵的共線性帶來的 Over-fitting 情形。後面的課程會提到一種非監督式的方法叫做 PCA (Principal Component Analysis) 主成分分析，是一種用來消除特徵具有共線性的方法，而 Ridge Regression 有些地方其實與 PCA 非常類似，可參考以下網址文章。</p> <p>PCA 與 Ridge Regression 的關係 https://tawehuang.hpd.io/2018/06/10/pca-%E8%88%87-ridge-regression-%E7%9A%84%E9%97%9C%E4%BF%82/</p>				

STEP5:	<p>重覆 STEP2- STEP4 多次進行特徵選擇、模型篩選，找出在測試集和訓練集最佳的模型。</p> <p>本章使用損失函數的 <code>cross_val_score(estimator, train_X, train_Y, cv=5).mean()</code> 在範例一分類問題用 accuracy、範例二回歸問題用 r2。另外在範例一，也會有正規化的 L1(Lasso) 嵌入法範例。</p> <p>如何解決過擬合或欠擬合</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; vertical-align: top;"> 過擬合 <ul style="list-style-type: none"> • 增加資料量 • 降低模型複雜度 • 使用正規化 (Regularization) </td><td style="padding: 5px; vertical-align: top;"> 欠擬合 <ul style="list-style-type: none"> • 增加模型複雜度 • 減輕或不使用正規化 </td></tr> </table>	過擬合 <ul style="list-style-type: none"> • 增加資料量 • 降低模型複雜度 • 使用正規化 (Regularization) 	欠擬合 <ul style="list-style-type: none"> • 增加模型複雜度 • 減輕或不使用正規化
過擬合 <ul style="list-style-type: none"> • 增加資料量 • 降低模型複雜度 • 使用正規化 (Regularization) 	欠擬合 <ul style="list-style-type: none"> • 增加模型複雜度 • 減輕或不使用正規化 		

特徵工程 - 學習地圖

接下來會以此圖為主，逐步為各位解說知識點



3-1 建立/讀取資料補充- Copy

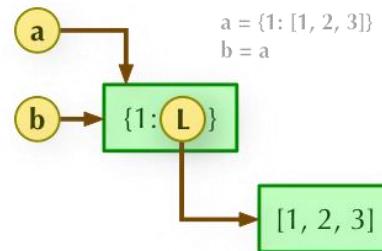
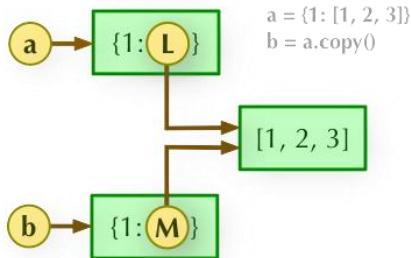
在進入整理資料之前，先補充一下資料拷貝複製的方法，Python 中的三種方法：直接賦值、淺拷貝和深度拷貝，以下說明^{#3.1}。在本章範例會用深度拷貝來避免修改到原始資料。

一、直接賦值：其實就是物件的引用（別名）。

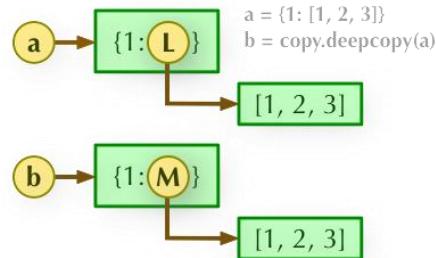
賦值引用的 a 和 b 都指向同一個對象。

```
import copy
```

二、淺拷貝 Copy：a 和 b 是一個獨立的對象，但他們的子物件還是指向統一對象（是引用）。



三、深度拷貝 Deepcopy：a 和 b 完全拷貝了父物件及其子物件，兩者是完全獨立的。



```
a = {1: [1, 2, 3]}
b = a.copy()
print(a, b)

a[1].append(4)
print(a, b)
```

```
{1: [1, 2, 3]} {1: [1, 2, 3]}
{1: [1, 2, 3, 4]} {1: [1, 2, 3, 4]}
```

```
import copy
c = copy.deepcopy(a)
print(a, c)

a[1].append(5)
print(a, c)
```

```
{1: [1, 2, 3, 4]} {1: [1, 2, 3, 4]}
{1: [1, 2, 3, 4, 5]} {1: [1, 2, 3, 4]}
```

#更多實例

```
a = [1, 2, 3, 4, ['a', 'b']] #原始對象
b = a #賦值，傳對象的引用
c = copy.copy(a) #物件拷貝，淺拷貝
d = copy.deepcopy(a) #物件拷貝，深拷貝

a.append(5) #修改對象 a
a[4].append('c') #修改物件 a 中的['a', 'b']陣列物件

print('a = ', a)
print('b = ', b)
print('c = ', c)
print('d = ', d)

a = [1, 2, 3, 4, ['a', 'b', 'c'], 5]
b = [1, 2, 3, 4, ['a', 'b', 'c'], 5]
c = [1, 2, 3, 4, ['a', 'b', 'c']]
d = [1, 2, 3, 4, ['a', 'b']]
```

#3.1. <Python 直接賦值、淺拷貝和深度拷貝解析><http://www.runoob.com/w3cnote/python-understanding-dict-copy-shallow-or-deep.html>

3-2 特徵工程範例

特徵分為數值型特徵、類別型特徵；另外還有二元特徵、排序型特徵、時間型特徵。「特徵工程」是將這些特徵的原始資料(可能是文字、數值等)轉換成對應的數值(調整過的分數、數字分組標籤等)。以下三大範例說明：

- 連續型資料及類別型資料處理：

範例-鐵達尼號的乘客生存預測 (binary classification: LogisticRegression)

- 時間及距離資料處理：

範例-計程車費率預測 (LinearRegression)

- 分割資料集處理-套件資料：

範例-鳶尾花辨識 (multi-classification: KNeighborsClassifier & 樹狀模型)

3-2.1 範例 1：鐵達尼號的乘客生存預測 (binary classification: LogisticRegression)

鐵達尼生存預測 Titanic dataset 是資料科學競賽平台 Kaggle 上的一開放資料集(<https://www.kaggle.com/c/titanic>)，幾乎所有的機器學習入門者都會用這個資料集開始探索機器學習的世界。資料內容是 Titanic 上的乘客資料。以下為各欄位的名稱及代表意義：

欄位名稱	定義
survival	存歿
pclass	購票分級
sex	性別
Age	年紀
sibsp	鐵達尼號上的兄弟姊妹/配偶數量
parch	鐵達尼號上的雙親/子女數量
ticket	票根號碼
fare	票價
cabin	船艙編號
embarked	登船港口

接下來會依照連續型資料及類別型資料進行特徵工程的處理：

- 數值型特徵
- 類別型特徵
- 特徵選擇
- 補充

Code 3: Day017-030 Feature Engineering_ReviewPART I.ipynb

Day017 什麼是 Titanic dataset? <https://ithelp.ithome.com.tw/articles/10200327>

準備資料

import pandas as pd

```

data_path = 'data/'
df_train = pd.read_csv(data_path + 'titanic_train.csv')
df_test = pd.read_csv(data_path + 'titanic_test.csv')

train_Y = df_train['Survived']
ids = df_test['PassengerId']
df_train = df_train.drop(['PassengerId', 'Survived'], axis=1)
df_test = df_test.drop(['PassengerId'], axis=1)
df = pd.concat([df_train, df_test]) #前半段為訓練資料, 後半段為測試資料, 合併一起進行資料整理
print("df_train.shape", df_train.shape)
print("df_test.shape", df_test.shape)
print(f'df.shape: {df.shape}, NOTE: 前 {df_train.shape[0]} 筆為訓練資料, \
接下來 {df_test.shape[0]} 筆為測試資料, 共有 {df.shape[1]} 個欄位(不包含 train_Y 和 ID)')
print(df.head())

```

```

df_train.shape (891, 10)
df_test.shape (418, 10)
df.shape: (1309, 10) , NOTE: 前 891 筆為訓練資料, 接下來 418 筆為測試資料, 共有 10 個欄位(不包含
train_Y 和 ID)

      Pclass          Name     Sex   Age \
0            3  Braund, Mr. Owen Harris    male  22.0
1            1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
2            3           Heikkinen, Miss. Laina  female  26.0
3            1  Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
4            3           Allen, Mr. William Henry    male  35.0

   SibSp  Parch       Ticket     Fare Cabin Embarked
0      1      0         A/5 21171  7.2500   NaN        S
1      1      0          PC 17599  71.2833  C85        C
2      0      0  STON/O2. 3101282  7.9250   NaN        S
3      1      0         113803  53.1000  C123        S
4      0      0         373450  8.0500   NaN        S

```

```

#D18 檢視不同資料類型列表
#秀出資料欄位的類型與數量
dtype_df = df.dtypes.reset_index()
dtype_df.columns = ["Count", "Column Type"]
dtype_df = dtype_df.groupby("Column Type").aggregate('count').reset_index()
dtype_df

#確定只有 int64, float64, object 三種類型後, 分別將欄位名稱存於三個 list 中
int_features = []
float_features = []
object_features = []
for dtype, feature in zip(df.dtypes, df.columns):
    if dtype == 'float64':
        float_features.append(feature)
    elif dtype == 'int64':
        int_features.append(feature)
    else:
        object_features.append(feature)
print(f'{len(int_features)} Integer Features : {int_features}\n')
print(f'{len(float_features)} Float Features : {float_features}\n')
print(f'{len(object_features)} Object Features : {object_features}\n')

print(f'\n每個欄位有多少 coding: pd.Series.nunique \n')
print(df.apply(pd.Series.nunique, axis = 0))
3 Integer Features : ['Pclass', 'SibSp', 'Parch']

2 Float Features : ['Age', 'Fare']

```

```
5 Object Features : ['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']
```

每個欄位有多少 coding: pd.Series.nunique

```
Pclass      3  
Name     1307  
Sex       2  
Age      98  
SibSp      7  
Parch      8  
Ticket    929  
Fare      281  
Cabin    186  
Embarked    3  
dtype: int64
```

```
#####
```

數值型特徵

```
#####
```

#D19 數值型特徵 - 補缺失值與標準化

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler  
from sklearn.model_selection import cross_val_score  
from sklearn.linear_model import LogisticRegression
```

削減文字型欄位, 只剩數值型欄位

```
df_num = pd.concat((df[int_features], df[float_features]), axis=1)  
print(f'各欄位 Missing 的量: \n{df_num.isnull().sum()}') # 觀察 missing 狀況, 只有Age 和 Fare 有missing
```

#看分佈 BoxPlot

```
import matplotlib.pyplot as plt  
# 檢視這些欄位的數值範圍  
for col in df_num.columns:  
    df_num[[col]].boxplot()  
    plt.show()  
print(df_num.describe())  
print("==" * 30)  
#####
```

#試著兩種以上填補的缺值, 看看何者比較好?

```
#df_m1 = df.fillna(-1) # 空值補 -1  
#df_0 = df.fillna(0) # 空值補 0  
#df_mn = df.fillna(df.mean()) # 空值補平均值 mean、中位數 median、眾數 mode  
#使用不同的標準化方式(原值 / 最小最大化 / 標準化), 何者效果最好?  
#df_MinMax = MinMaxScaler().fit_transform(df) # 搭配最大最小化  
#df_Std = StandardScaler().fit_transform(df) # 搭配標準化
```

利用 LogisticRegression 和 cross_val_score 預視結果:

```
#空值補平均值  
df_train_mn = df_num.fillna(df_num.mean()) # train_X = df_mn[:train_num]  
train_num = train_Y.shape[0]  
train_X = df_train_mn[:train_num] #取訓練料集 建模型  
estimator = LogisticRegression()  
print('調整 mn 後 cross_val_score:', cross_val_score(estimator, train_X, train_Y, cv=5).mean())
```

#空值補平均值, 搭配最大最小化, 有較好結果

```
df_mn = df_num.fillna(df_num.mean()) # train_X = df_mn[:train_num]
```

```

df_MinMax = MinMaxScaler().fit_transform(df_mn) #有一些特殊變數，如具有特殊意義，這類變數並不適合做標準化或其他 Scaler，一般模型中需要統一使用同一種特徵縮放的方法
df_train_mnMinMax = pd.DataFrame(df_MinMax, columns=df_num.columns)
train_X = df_train_mnMinMax[:train_num] #取訓練料集 建模型
estimator = LogisticRegression()
print('調整 mn_MinMax 後 cross_val_score:', cross_val_score(estimator, train_X, train_Y, cv=5).mean())

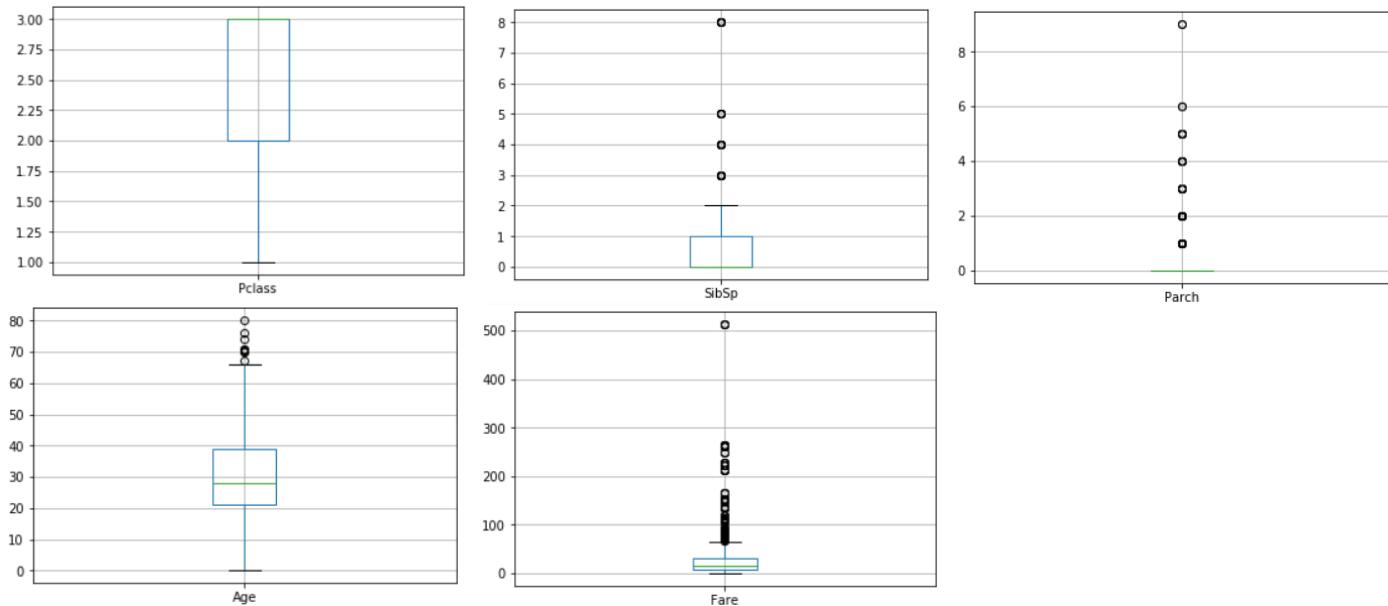
```

各欄位 Missing 的量：

```

Pclass      0
SibSp       0
Parch       0
Age        263
Fare        1
dtype: int64

```



	Pclass	SibSp	Parch	Age	Fare
count	1309.000000	1309.000000	1309.000000	1046.000000	1308.000000
mean	2.294882	0.498854	0.385027	29.881138	33.295479
std	0.837836	1.041658	0.865560	14.413493	51.758668
min	1.000000	0.000000	0.000000	0.170000	0.000000
25%	2.000000	0.000000	0.000000	21.000000	7.895800
50%	3.000000	0.000000	0.000000	28.000000	14.454200
75%	3.000000	1.000000	0.000000	39.000000	31.275000
max	3.000000	8.000000	9.000000	80.000000	512.329200

=====

調整 mn 後 cross_val_score: 0.6981761033723469

調整 mn_MinMax 後 cross_val_score: 0.6993501991462476

複習：填補缺值的方式

- 填補統計值
 - 填補平均值(Mean)：數值型欄位，偏態不明顯
 - 填補中位數(Median)：數值型欄位，偏態很明顯
 - 填補眾數.Mode)：類別型欄位
- 填補指定值 - 需對欄位領域知識已有了解
 - 補 0：空缺原本就有 0 的含意，如前頁的房間數
 - 補不可能出現的數值：類別型欄位，但不適合用眾數時
- 填補預測值 - 速度較慢但精確，從其他資料欄位學得填補知識
 - 若填補範圍廣，且是重要特徵欄位時可用本方式
 - 本方式須提防overfitting：可能退化成為其他特徵的組合

複習：標準化 / 最小最大化

- 標準化 (Standard Scaler)：假定數值為常態分佈，適合本方式平衡特徵
- 最小最大化 (MinMax Scaler)：假定數值為均勻分佈，適合本方式平衡特徵



如果離群值有處理 (處理方式將於 Day20 課程介紹)，兩者差異不太大

標準化 / 最小最大化適用場合

- 樹狀模型或非樹狀模型(參考今日練習題)
 - **非樹狀模型**：如線性迴歸, 羅吉斯迴歸, 類神經...等，標準化 / 最小最大化後對預測會有影響
 - **樹狀模型**：如決策樹, 隨機森林, 梯度提升樹...等，標準化 / 最小最大化後對預測不會有影響
- 標準化 / 最小最大化 使用上的差異
 - 標準化：轉換不易受到極端值影響
 - 最小最大化：轉換容易受到極端值影響

註：因此，去過離群值的特徵，比較適用最大最小化

#D20_數值型特徵-去除離群值

#試著限制'鐵達尼號上的兄弟姊妹/配偶數量'(SibSp)欄位的上下限,看看能否再進一步提高分數?

#顯示1stFlrSF與目標值的散佈圖

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
train_num = train_Y.shape[0]
```

```
sns.regplot(x = df_mn['SibSp'][train_num], y=train_Y)
```

```
plt.show()
```

#方法一：將SibSp限制在0到8以內，調整極端值在範圍內 [numpy.clip(a, a_min, a_max, out=None)] 也就是說 clip這個函數將將陣列中的元素x限制在 a_min, a_max 之間，大於 a_max 的就使得它等於 a_max，小於 a_min,的就使得它等於 a_min。]

```
#df['SibSp'] = df['Fare'].clip(0, 8)
```

#方法二：將SibSp保留在0到8以內，捨棄離群值

```
#keep_indexs = (df['SibSp'] > 0) & (df['SibSp'] < 8)
```

```
#df_keep = df[keep_indexs]
```

```
#train_Y = train_Y[keep_indexs]
```

#做迴歸，觀察分數：

#以平均值補值，並將SibSp保留在0到8以內，捨棄離群值

```
df_mn = df_num.fillna(df_num.mean())
```

```
df_train_mn=df_mn[:train_num]
```

```
print("原始資料筆數:", df_train_mn.shape)
```

```
keep_indexs = (df_train_mn['SibSp'] >= 0) & (df_train_mn['SibSp'] < 8)
```

```
df_keep = df_train_mn[keep_indexs]
```

```
print("捨棄離群值後筆數:", df_keep.shape)
```

```
train_Y_keep = train_Y[keep_indexs]
```

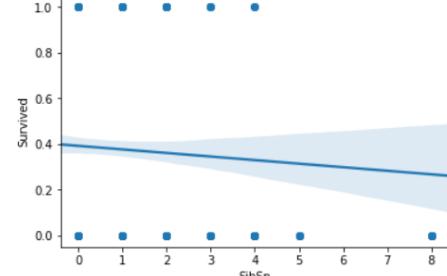
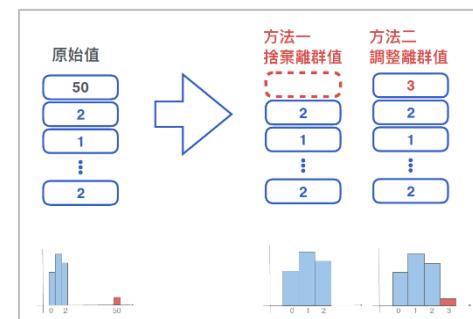
```
sns.regplot(x = df_keep['SibSp'], y=train_Y_keep)
```

```
plt.show()
```

train_X_keep = df_keep #取訓練料集 建模型

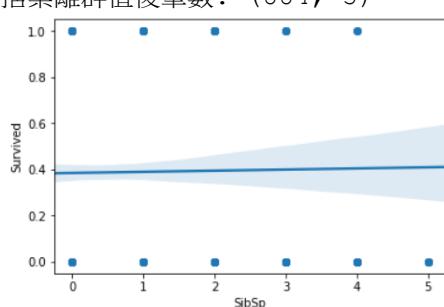
```
estimator = LogisticRegression()
```

```
print('調整*捨棄離群值*後 cross_val_score:', cross_val_score(estimator, train_X_keep, train_Y_keep, cv=5).mean())
```



原始資料筆數: (891, 5)

捨棄離群值後筆數: (884, 5)



調整*捨棄離群值*後 cross_val_score: 0.6924284984678243

```

#D21_數值型特徵-去除偏態
# 顯示 Fare 與目標值的散佈圖
import numpy as np
import copy
import seaborn as sns
import matplotlib.pyplot as plt
sns.distplot(df['Fare'][:train_num])
plt.show()

# 將 Fare 取 log1p [[ log1p = log (x+1) ]] 後, 看散佈圖, 並計算分數:
#df_fixed['Fare'] = np.log1p(df_fixed['Fare'])
# 將 Fare 取 boxcox 後, 看散佈圖, 並計算分數
#面對有數值=0 的情況, 修正方式: 加入下面這一行, 使最小值大於 0, 類似 log1p 的概念
#df_fixed['Fare'] = df_fixed['Fare'] + 1
#df_fixed['Fare'] = stats.boxcox(df_fixed['Fare'])[0]

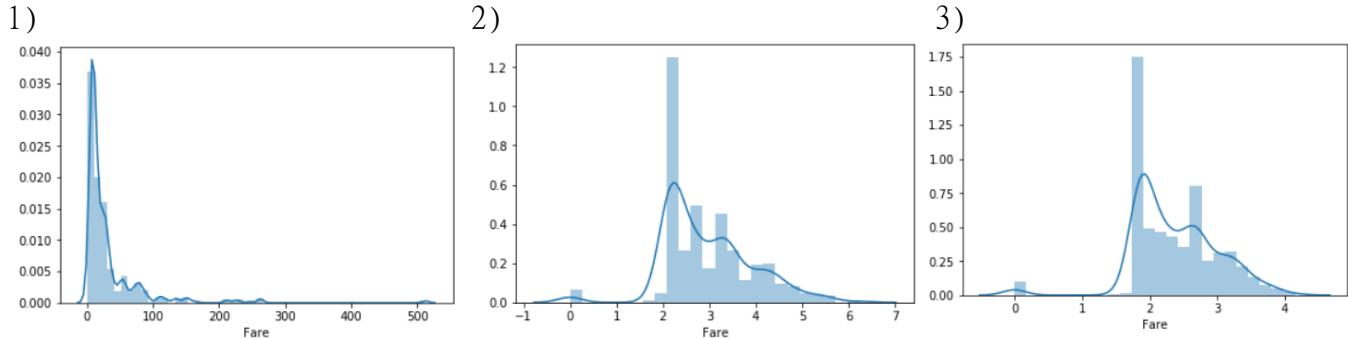
# 將 Fare 取 log1p [[ log1p = log (x+1) ]] 後, 看散佈圖, 並計算分數:
# 以平均值 補值,
df_mn = df_num.fillna(df_num.mean())
df_mn_log1p = copy.deepcopy(df_mn)      #複製獨立檔案出來 整理
df_mn_log1p['Fare'] = np.log1p(df_mn_log1p['Fare'])
sns.distplot(df_mn_log1p['Fare'][:train_num])
plt.show()

train_X = df_mn_log1p[:train_num]
estimator = LogisticRegression()
print('調整 log1p 後 cross_val_score:', cross_val_score(estimator, train_X, train_Y, cv=5).mean())

# 將 Fare 取 boxcox (lambda=0) [[ 即 log]] 後, 看散佈圖, 並計算分數
# 以平均值 補值, boxcox 說明見 https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html
df_mn = df_num.fillna(df_num.mean())
df_mn_boxcox = copy.deepcopy(df_mn)      #複製獨立檔案出來 整理
from scipy import stats
df_mn_boxcox['Fare'] = df_mn_boxcox['Fare'] + 1 #Data must be positive. 面對有數值=0 的情況, 修正方式: 加入下面這一行, 使最小值大於 0, 類似 log1p 的概念
df_mn_boxcox['Fare'] = stats.boxcox(df_mn_boxcox['Fare'])[0] # Box-Cox power transformed array 取第一筆的統計資料
#If lmbda is None, find the lambda that maximizes the log-likelihood function and return it as the second output argument.
sns.distplot(df_mn_boxcox['Fare'][:train_num])
plt.show()

train_X = df_mn_boxcox[:train_num]
estimator = LogisticRegression()
print('調整 boxcox0 後 cross_val_score:', cross_val_score(estimator, train_X, train_Y, cv=5).mean())

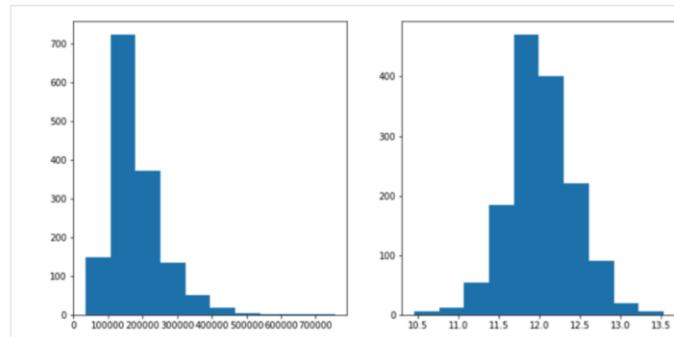
```



- 2) 調整 log1p 後 cross_val_score: 0.7038383040988557
 3) 調整 boxcox0 後 cross_val_score: 0.7027336816575154

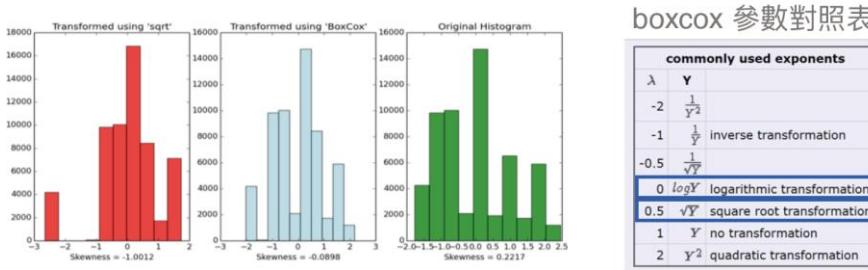
複習：對數去偏(log1p)

對數去偏就是使用**自然對數**去除偏態
 常見於計數 / 價格這類**非負且可能為 0**的欄位
 因為需要將 0 對應到 0，所以先加一 (plus one) 再取對數 (log)
 還原時使用 expm1，也就是先取指數 (exp) 後再減一 (minus one)



方根去偏(sqrt) / 分布去偏(coxbox)

- 方根去偏(sqrt) 就是將數值減去最小值後開根號，最大值有限時適用 (例：成績轉換)
- 分布去偏(boxcox) 是採用boxcox轉換函數(詳見右表)，函數的 lambda(λ) 參數為 0 時等於 log 函數，lambda(λ) 為 0.5 時等於開根號 (即sqrt)，因此可藉由參數的調整更靈活地轉換數值，但要特別注意Y的輸入數值必須要為正 (不可為0)



思考: 下三個整數變數，是否為類別變數呢？是否考慮分組來分析？

```
# 3 Integer Features : ['Pclass', 'SibSp', 'Parch']
# pclass 購票分級
# sibsp 鐵達尼號上的兄弟姊妹/配偶數量
# parch 鐵達尼號上的雙親/子女數量

print(f"df['Pclass'] \n{df['Pclass'].value_counts()}")
print(f"df['SibSp'] \n{df['SibSp'].value_counts()}")
print(f"df['Parch'] \n{df['Parch'].value_counts()}")
```

思考: 這三個整數變數，是否為類別變數呢？是否考慮分組來分析？

```
df['Pclass']
3      709
1      323
2      277
Name: Pclass, dtype: int64
df['SibSp']
0      891
1      319
2       42
4       22
3       20
8       9
5       6
Name: SibSp, dtype: int64
df['Parch']
0     1002
1      170
2      113
3       8
5       6
4       6
9       2
6       2
Name: Parch, dtype: int64
```

#####

類別型特徵

#####

```
#D22_類別型特徵 - 標籤編碼、獨熱編碼
# 只留類別型欄位
df_obj = df[object_features]
print(f'各欄位 Missing 的量: \n{df_obj.isnull().sum()}') # 觀察 missing 狀況，只有 Cabin 和 Embarked 有 missing
# 補值
df_obj = df_obj.fillna('None')
train_num = train_Y.shape[0]
print(df_obj.head())

from sklearn.preprocessing import LabelEncoder
# LabelEncoder 標籤編碼 + 羅吉斯迴歸
print(f'\n ===== \n LabelEncoder 標籤編碼 + 羅吉斯迴歸')
df_temp = pd.DataFrame()
for c in df_obj.columns:
    df_temp[c] = LabelEncoder().fit_transform(df_obj[c])
train_X = df_temp[:train_num]
estimator = LogisticRegression()
print(f'shape : {train_X.shape}')
print(f'score : {cross_val_score(estimator, train_X, train_Y, cv=5).mean()}')
```

```

# One Hot Encoding 獨熱編碼 + 羅吉斯迴歸
print(f'\n ===== \n One Hot Encoding 獨熱編碼 + 羅吉斯迴歸')
df_temp = pd.get_dummies(df_obj)
train_X = df_temp[:train_num]
estimator = LogisticRegression()
print(f'shape : {train_X.shape}')
print(f'score : {cross_val_score(estimator, train_X, train_Y, cv=5).mean()}')


#D23_類別型特徵 - 均值編碼
# Mean Encoding 均值編碼 + 羅吉斯迴歸
print(f'\n ===== \n Mean Encoding 均值編碼 + 羅吉斯迴歸')
df_temp = pd.concat([df_obj[:train_num], train_Y], axis=1)
for c in df_obj.columns:
    mean_df = df_temp.groupby([c])['Survived'].mean().reset_index()
    mean_df.columns = [c, f'{c}_mean']
    df_temp = pd.merge(df_temp, mean_df, on=c, how='left')
    df_temp = df_temp.drop([c], axis=1)
train_X = df_temp.drop(['Survived', 'Name_mean', 'Ticket_mean'], axis=1)
estimator = LogisticRegression()
print(f'shape : {train_X.shape}')
print(f'score : {cross_val_score(estimator, train_X, train_Y, cv=5).mean()}')
print(f'Note: 因為 Name 與 Ticket 兩個欄位的種類數眾多, 如果均值編碼時加入這裡個欄位的均值會過擬合(overfitting), 因此程式碼中, 需要先移除這兩個欄位, 再做均值編碼')


#D23_類別型特徵 - 計數編碼、特徵雜湊
# 加上 'Cabin' 欄位的計數編碼
print(f'\n ===== \n 加上 Cabin 欄位的計數編碼 \n')
count_df = df_obj.groupby(['Cabin'])['Name'].agg({'Cabin_Count': 'size'}).reset_index()
count_df = pd.merge(df_obj, count_df, on=['Cabin'], how='left')
print(count_df.sort_values(by=['Cabin_Count'], ascending=False).head(10))


# Counting 計數編碼 + 邏輯斯迴歸
print(f'\n ===== \n Counting 計數編碼 + 羅吉斯迴歸')
df_temp = pd.DataFrame()
for c in object_features:
    df_temp[c] = LabelEncoder().fit_transform(count_df[c])
df_temp['Cabin_Count'] = count_df['Cabin_Count']
train_X = df_temp[:train_num]
estimator = LogisticRegression()
print(cross_val_score(estimator, train_X, train_Y, cv=5).mean())


# Feature Hash 特徵雜湊 + 邏輯斯迴歸
df_temp = pd.DataFrame()
print(f'\n ===== \n Feature Hash 特徵雜湊 + 羅吉斯迴歸')
for c in object_features:
    df_temp[c] = LabelEncoder().fit_transform(count_df[c])
df_temp['Cabin_Hash'] = count_df['Cabin'].map(lambda x:hash(x) % 5)
train_X = df_temp[:train_num]
estimator = LogisticRegression()
print(cross_val_score(estimator, train_X, train_Y, cv=5).mean())


# 計數編碼 + 特徵雜湊 + 邏輯斯迴歸
print(f'\n ===== \n 計數編碼 + 特徵雜湊 + 羅吉斯迴歸')

```

```

df_temp = pd.DataFrame()
for c in object_features:
    df_temp[c] = LabelEncoder().fit_transform(count_df[c])
df_temp['Cabin_Hash'] = count_df['Cabin'].map(lambda x:hash(x) % 5)
df_temp['Cabin_Count'] = count_df['Cabin_Count']
train_X = df_temp[:train_num]
estimator = LogisticRegression()
print(cross_val_score(estimator, train_X, train_Y, cv=5).mean())
print(f'\n計數編碼 + 特徵雜湊 DATA \n {df_temp.head(10)}')

```

各欄位 Missing 的量：

```

Name      0
Sex       0
Ticket    0
Cabin     1014
Embarked  2
dtype: int64

```

			Name	Sex	\
0		Braund, Mr. Owen Harris	male		
1	Cumings, Mrs. John Bradley (Florence Briggs Th... 2	Heikkinen, Miss. Laina 3	female		
4	Futrelle, Mrs. Jacques Heath (Lily May Peel) Allen, Mr. William Henry		female	male	

	Ticket	Cabin	Embarked
0	A/5 21171	None	S
1	PC 17599	C85	C
2	STON/O2. 3101282	None	S
3	113803	C123	S
4	373450	None	S

=====

LabelEncoder 標籤編碼 + 羅吉斯迴歸

shape : (891, 5)

score : 0.780004837244799

=====

One Hot Encoding 獨熱編碼 + 羅吉斯迴歸

shape : (891, 2429)

score : 0.8013346043513216

=====

Mean Encoding 均值編碼 + 羅吉斯迴歸

shape : (891, 3)

score : 0.8350366889413987

Note：因為 Name 與 Ticket 兩個欄位的種類數眾多，如果均值編碼時加入這裡個欄位的均值會過擬合 (overfitting)，因此程式碼中，需要先移除這兩個欄位，再做均值編碼

===== =====

===== =====

加上 Cabin 欄位的計數編碼

		Name	Sex	Ticket	Cabin	\
0	Braund, Mr. Owen Harris	male	A/5 21171	None		
822	Reuchlin, Jonkheer. John George	male		19972	None	
830	Yasbeck, Mrs. Antoni (Selini Alexander)	female		2659	None	
828	McCormack, Mr. Thomas Joseph	male		367228	None	
827	Mallet, Master. Andre	male	S.C./PARIS	2079	None	
826	Lam, Mr. Len	male		1601	None	
825	Flynn, Mr. John	male		368323	None	

824	Panula, Master.	Urpo Abraham	male	3101295	None
821	Lulic, Mr.	Nikola	male	315098	None
832	Saad, Mr.	Amin	male	2671	None

	Embarked	Cabin_Count
0	S	1014
822	S	1014
830	C	1014
828	Q	1014
827	C	1014
826	S	1014
825	Q	1014
824	S	1014
821	S	1014
832	C	1014

=====

Counting 計數編碼 + 羅吉斯迴歸

0.7856230275549181

=====

Feature Hash 特徵雜湊 + 羅吉斯迴歸

0.7766593008620863

=====

計數編碼 + 特徵雜湊 + 羅吉斯迴歸

0.7833632114766116

計數編碼 + 特徵雜湊 DATA

	Name	Sex	Ticket	Cabin	Embarked	Cabin_Hash	Cabin_Count
0	155	1	720	185	3	0	1014
1	286	0	816	106	0	0	2
2	523	0	914	185	3	0	1014
3	422	0	65	70	3	4	2
4	22	1	649	185	3	0	1014
5	818	1	373	185	2	0	1014
6	767	1	109	163	3	3	2
7	914	1	541	185	3	0	1014
8	605	0	477	185	3	0	1014
9	847	0	174	185	0	0	1014

標籤編碼 / 獨熱編碼的比較

	大小有無意義	儲存空間/計算時間	適用模型
標籤編碼 Label Encoding	無意義	小	樹狀模型
獨熱編碼 One Hot Encoding	有意義	較大	非樹狀模型

綜合建議

- 類別型特徵建議預設採用標籤編碼
- 除非該特徵重要性高，且可能值較少(獨熱編碼時負擔較低) 時，才應考慮使用獨熱編碼

複習

基礎編碼 1：標籤編碼 (Label Encoding)

- 類似於流水號，依序將新出現的類別依序編上新代碼，已出現的類別編上已使用的代碼
- 確實能轉成分數，但缺點是分數的大小順序沒有意義



額外線索：如果類別特徵看起來與目標值有顯著相關，應該如何編碼？

均值編碼

均值編碼 (Mean Encoding)：使用目標值的平均值，取代原本的類別型特徵
*在部分模型中，使用均值編碼作為類別型特徵預設編碼方式



基礎編碼 2：獨熱編碼 (One Hot Encoding)

- 為了改良數字大小沒有意義的問題，將不同的類別分別獨立為一欄
- 缺點是需要較大的記憶空間與計算時間，且類別數量越多時越嚴重



如果交易樣本非常少，且剛好抽到極端值，平均結果可能會有誤差很大：

平滑化 (Smoothing)

因此，均值編碼還需要考慮紀錄筆數，當作可靠度的參考



- 當平均值的可靠度低時，我們會傾向相信全部的總平均
- 當平均值的可靠度高時，我們會傾向相信類別的平均
- 依照紀錄筆數，在這兩者間取折衷

$$\text{新類別均值} = \frac{\text{原類別平均} * \text{類別樣本數} + \text{全部的總平均} * \text{調整因子}}{\text{類別樣本數} + \text{調整因子}}$$

*調整因子用來調整平滑化的程度，依總樣本數調整

小提醒：均值編碼容易 overfitting

雖然均值編碼符合直覺，並且也是強大的編碼方式
但實際上使用時很容易 overfitting (即使使用了平滑化)
所以需確認是否適合再使用 (用 cross validation 確認使用前後分數)

計數編碼 (Counting)

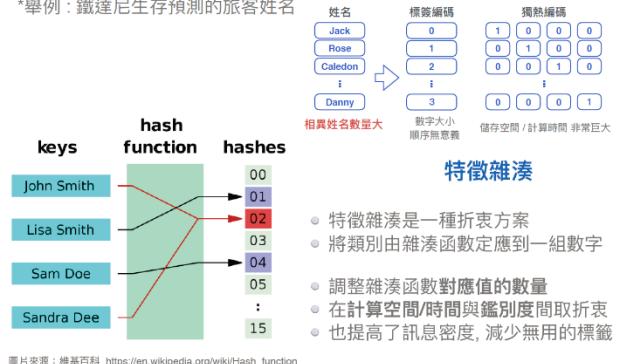
如果類別的目標均價與類別筆數呈正相關（或負相關），也可以將筆數本身當成特徵
例如：購物網站的消費金額預測



*自然語言處理時，字詞的計數編碼又稱詞頻，本身就是一個很重要的特徵

特徵雜湊 (Feature Hash)

類別型特徵最麻煩的問題：相異類別的數量非常龐大，該如何編碼？
*舉例：鐵達尼生存預測的旅客姓名



圖片來源：維基百科 https://en.wikipedia.org/wiki/Hash_function

```

# Day_027_特徵組合 - 類別與數值組合「群聚編碼」
# 試著使用鐵達尼號的例子，創立兩種以上的群聚編碼特徵( mean、median、mode、max、min、count 均可)
import copy
df_Copy=copy.deepcopy(df)
# 取船票票號(Ticket), 對乘客年齡(Age)做群聚編碼
df_Copy['Ticket'] = df_Copy['Ticket'].fillna('None')
df_Copy['Age'] = df_Copy['Age'].fillna(df_Copy['Age'].mean())

mean_df = df_Copy.groupby(['Ticket'])['Age'].mean().reset_index()
mode_df = df_Copy.groupby(['Ticket'])['Age'].apply(lambda x: x.mode()[0]).reset_index()
median_df = df_Copy.groupby(['Ticket'])['Age'].median().reset_index()
max_df = df_Copy.groupby(['Ticket'])['Age'].max().reset_index()
min_df = df_Copy.groupby(['Ticket'])['Age'].min().reset_index()
temp = pd.merge(mean_df, mode_df, how='left', on=['Ticket'])
temp = pd.merge(temp, median_df, how='left', on=['Ticket'])
temp = pd.merge(temp, max_df, how='left', on=['Ticket'])
temp = pd.merge(temp, min_df, how='left', on=['Ticket'])
temp.columns = ['Ticket', 'Age_Mean', 'Age_Mode', 'Age_Median', 'Age_Max', 'Age_Min']
temp.head()

```

	Ticket	Age_Mean	Age_Mode	Age_Median	Age_Max	Age_Min
0	110152	26.333333	16.000000	30.000000	33.0	16.000000
1	110413	36.333333	18.000000	39.000000	52.0	18.000000
2	110465	38.440569	29.881138	38.440569	47.0	29.881138
3	110469	30.000000	30.000000	30.000000	30.0	30.000000
4	110489	42.000000	42.000000	42.000000	42.0	42.000000

```

df_Copy = pd.merge(df_Copy, temp, how='left', on=['Ticket'])
df_Copy = df_Copy.drop(['Ticket'], axis=1)
df_Copy.head()

```

Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked	Age_Mean	Age_Mode	Age_Median	Age_Max	Age_Min
0	Braund, Mr. Owen	male	22.0	1	0	7.2500	NaN	S	22.0	22.0	22.0	22.0	22.0
1	Cumings, Mrs. John	female	38.0	1	0	71.2833	C85	C	38.5	38.0	38.5	39.0	38.0
2	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	NaN	S	26.0	26.0	26.0	26.0	26.0
3	Futrelle, Mrs. Jacques	female	35.0	1	0	53.1000	C123	S	36.0	35.0	36.0	37.0	35.0
4	Allen, Mr. William	male	35.0	0	0	8.0500	NaN	S	35.0	35.0	35.0	35.0	35.0

```

# 只取 int64, float64 兩種數值型欄位, 存於 num_features 中
num_features = []
for dtype, feature in zip(df_Copy.dtypes, df_Copy.columns):
    if dtype == 'float64' or dtype == 'int64':
        num_features.append(feature)
print(f'{len(num_features)} Numeric Features : {num_features}\n')

```

```

# 刪減文字型欄位, 只剩數值型欄位
df_Copy_num = df_Copy[num_features]
df_Copy_1 = df_Copy_num.fillna(-1)
df_Copy_1.head()

```

10 Numeric Features : ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Age_Mean', 'Age_Mode', 'Age_Median', 'Age_Max', 'Age_Min']

Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	Braund, Mr. Owen	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	Cumings, Mrs. John	female	38.0	1	0	PC 17599	71.2833	C85	C
2	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	Futrelle, Mrs. Jacques	female	35.0	1	0	113803	53.1000	C123	S
4	Allen, Mr. William	male	35.0	0	0	373450	8.0500	NaN	S

```

# 新特徵 + 邏輯斯迴歸
df_CopyMinMax = MinMaxScaler().fit_transform(df_Copy_1)
train_num = train_Y.shape[0]
train_X = df_CopyMinMax[:train_num]
estimator = LogisticRegression()

print(f'\n ===== \n 調整類別與數值組合「群聚編碼」 + 羅吉斯迴歸')
print(f' 取船票票號(Ticket), 對乘客年齡(Age)做群聚編碼後 cross_val_score:\n', cross_val_score(estimator, train_X,
train_Y, cv=5).mean())
=====
```

調整類別與數值組合「群聚編碼」 + 羅吉斯迴歸
取船票票號(Ticket), 對乘客年齡(Age)做群聚編碼後 cross_val_score:
0.7060727991156779

小結 Q&A：將上述的新特徵，合併原有的欄位做生存率預估，結果是否有改善？

結果可以發現：聚類編碼都在正確率上有穩定提升，即是課堂所述：均值編碼容易 overfitting/ 聚類編碼不容易 overfitting 的效果

NOTE：不過這邊的數值型特徵與類別型特徵，是有特別用特徵重要性挑選過的，因此同學自行挑選的特徵可能未必提升，至於特徵重要性如何使用，另外複習參考 Day29 內容。

複習

群聚編碼 (Group by Encoding)

既然數值型特徵之間能合成新特徵，那麼類別與數值型之間也能嗎？

例：保費續約預測中，已知險種(類別型)與前一年度保費(數值型)都是重要特徵
是否可以組合兩者，形成新特徵？

類似均值編碼的概念，可以取類別平均值 (Mean) 取代險種作為編碼
但因為比較像性質描寫，因此還可以取其他統計值，如中位數 (Median)，眾數 (Mode)，最大值 (Max)，最小值 (Min)，次數 (Count)...等

險種	前一年保費	平均值	最大值	次數
第三責任險	2600	2466.7	2600	3
竊盜險	1600	1500	1600	2
第三責任險	2500	2466.7	2600	3
第三責任險	2300	2466.7	2600	3
竊盜險	1400	1500	1600	2

- 數值型特徵對文字型特徵最重要的特徵組合方式
- 常見的有 mean, median, mode, max, min, count 等
- 與均值編碼 (Mean Encoding) 的比較

名稱	均值編碼 Mean Encoding	群聚編碼 Group by Encoding
平均對象	目標值	其他數值型特徵
過擬合 (Overfitting)	容易	不容易

Q : 亂槍打鳥？不會造出無用的特徵嗎？

Ans : 機器學習的特徵是寧謐勿缺的，

因為以前以非樹狀模型為主，為了避免共線性，會很注意類似的特徵不要增加太多
但現在強力的模型都是樹狀模型，所以只要有可能就通通做特徵囉

```
#####
```

特徵選擇

```
#####
```

概念簡介

為了提升精確度並減少計算時間，特徵需要適當的增加與減少，如上述群聚編碼 (Day 27)或下一個範例將特徵額外加減乘除的特徵組合 (Day 26)，都是額外增加特徵；若想要減少特徵，就需要做特徵選擇 (Day 28)，特徵選擇有三大類方法：

- 過濾法 (Filter)：選定統計數值與設定門檻，刪除低於門檻的特徵 《相關係數過濾法》
- 包裝法 (Wrapper)：根據目標函數，逐步加入特徵或刪除特徵
- 嵌入法 (Embedded)：使用機器學習模型，根據擬合後的係數，刪除係數低於門檻的特徵 《L1(Lasso)嵌入法》、使用梯度提升樹擬合後，以特徵在節點出現的頻率當作特徵重要性，以此刪除重要性低於門檻的特徵 《GDBT(梯度提升樹)嵌入法》

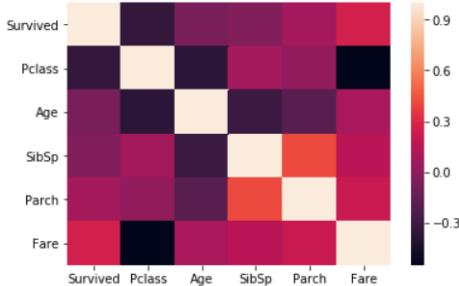
一、過濾法 (Filter)：選定統計數值與設定門檻，刪除低於門檻的特徵 《相關係數過濾法》

本例下圖顏色越淺表示越正相關，因此要刪除紅框中顏色較深的特徵：訂出相關係數門檻值，特徵相關係數絕對值低於門檻者刪除

```
#Day_028_Feature_Selection_Ans
data_path = 'data/'
df_train = pd.read_csv(data_path + 'titanic_train.csv')
df_train_Copy=copy.deepcopy(df_train)

df_train_Copy = df_train_Copy.drop(['PassengerId'], axis=1)
# 計算 df 整體相關係數, 並繪製成熱圖
import seaborn as sns
import matplotlib.pyplot as plt
corr = df_train_Copy.corr()
sns.heatmap(corr)

plt.show()
```



```
# 記得刪除 Y
df_train_Copy = df_train_Copy.drop(['Survived'], axis=1)
df_Copy2 = pd.concat([df_train_Copy, df_test]) #前半段為訓練資料, 後半段為測試資料, 合併一起進行資料整理
```

```
#這邊沿用 Day_027 的檔案
#只取 int64, float64 兩種數值型欄位, 存於 num_features 中
num_features = []
for dtype, feature in zip(df_Copy2.dtypes, df_Copy2.columns):
    if dtype == 'float64' or dtype == 'int64':
        num_features.append(feature)
print(f'{len(num_features)} Numeric Features : {num_features}\n')
```

```
#削減文字型欄位, 只剩數值型欄位
df_Copy2 = df_Copy2[num_features]
df_Copy2 = df_Copy2.fillna(-1)
df_Copy2.head()
```

```
5 Numeric Features : ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

Pclass	Age	SibSp	Parch	Fare
0	3	22.0	1	0
1	1	38.0	1	0
2	3	26.0	0	0
3	1	35.0	1	0
4	3	35.0	0	0

```
# 原始特徵 + 邏輯斯迴歸
print(f'\n ===== \n 原始特徵 + 邏輯斯迴歸')
df_CopyMinMax2 = MinMaxScaler().fit_transform(df_Copy2)
train_num = train_Y.shape[0]
train_X = df_CopyMinMax2[:train_num]
estimator = LogisticRegression()
print(f' cross_val_score:\n', cross_val_score(estimator, train_X, train_Y, cv=5).mean())

# 篩選相關係數大於 0.05 或小於 -0.05 的特徵
print(f'\n ===== \n 0.05 特徵 + 邏輯斯迴歸')
high_list = list(corr[(corr['Survived']>0.05) | (corr['Survived']<-0.05)].index)
high_list.pop(0)
print(' 篩選後留下的變數 : ', high_list)
# 0.05 特徵 + 邏輯斯迴歸
df_CopyMinMax2 = MinMaxScaler().fit_transform(df_Copy2[high_list])
train_num = train_Y.shape[0]
train_X = df_CopyMinMax2[:train_num]
estimator = LogisticRegression()
print(f' 篩選相關係數大於 0.05 或小於 -0.05 特徵的 cross_val_score:\n', cross_val_score(estimator, train_X, train_Y, cv=5).mean())


```

```
# 篩選相關係數大於 0.1 或小於 -0.1 的特徵
print(f'\n ===== \n 0.1 特徵 + 邏輯斯迴歸')
high_list = list(corr[(corr['Survived']>0.1) | (corr['Survived']<-0.1)].index)
high_list.pop(0)
print(' 篩選後留下的變數 : ', high_list)
# 0.1 特徵 + 邏輯斯迴歸
df_CopyMinMax2 = MinMaxScaler().fit_transform(df_Copy2[high_list])
train_num = train_Y.shape[0]
train_X = df_CopyMinMax2[:train_num]
estimator = LogisticRegression()
print(f' 篩選相關係數大於 0.1 或小於 -0.1 特徵的 cross_val_score:\n', cross_val_score(estimator, train_X, train_Y, cv=5).mean())
=====
```

原始特徵 + 邏輯斯迴歸
cross_val_score:
0.6971346062663598

=====
0.05 特徵 + 邏輯斯迴歸
篩選後留下的變數： ['Pclass', 'Age', 'Parch', 'Fare']
篩選相關係數大於 0.05 或小於 -0.05 特徵的 cross_val_score:
0.6993817972775959

=====
0.1 特徵 + 邏輯斯迴歸
篩選後留下的變數： ['Pclass', 'Fare']
篩選相關係數大於 0.1 或小於 -0.1 特徵的 cross_val_score:
0.6791567235397566

小結 Q&A：鐵達尼生存率預測中，試著變更兩種以上的相關係數門檻值，觀察預測能力是否提升？

由本例不同門檻可以發現，相關係數有時候確實能成功提升準確度，但篩選過頭會有反效果，

門檻的決定沒有一定的準則，因此這個方法實際上並不那麼好用

二、包裝法 (Wrapper)：根據目標函數，逐步加入特徵或刪除特徵 Stepwise

參考資料：[R 筆記 – \(18\) Subsets & Shrinkage Regression \(Stepwise & Lasso\)](https://rpubs.com/skydome20/R-Note18-Subsets_Shinkage_Methods) https://rpubs.com/skydome20/R-Note18-Subsets_Shinkage_Methods

Stepwise 的中文叫「逐步回歸法」，核心概念很簡單，都是想說能不能用比較簡單的模型，就能達成跟原模型差不多的表現與效果。

保留不同的變數創造不同的模型，這些模型根據 AIC 或 BIC 指標，選取一個表現最佳的模型！可以想見的，這樣的做法是會耗費大量的時間。所以 Stepwise Regression 改善了這種情況：只需要建構「一個模型」，然後在上面直接新增(或減少)變數。一般有兩種方法：向前選取法(Forward)跟向後選取法(Backward)。

在這次課程無特別列出範例，之後有機會再補。

三、嵌入法 (Embedded)：使用機器學習模型，根據擬合後的係數，刪除係數低於門檻的特徵 《L1(Lasso) 嵌入法》

配合 Day_029 及 Day_039 複習

另一種迴歸的變形是引入正則化 regularization (i.e., shrinkage)的技巧：

LASSO 為 Linear Regression 加上 $L1: \alpha \sum |weights|$

Ridge 為 Linear Regression 加上 $L2: \alpha \sum (weights)^2$

其中有個超參數 α 可以調整正則化的強度

簡單來說，LASSO 與 Ridge 就是迴歸模型加上不同的正則化函數，這兩種都是希望模型的參數值不要太大，原因是參數的數值變小，噪音對最終輸出的結果影響越小，提升模型的泛化能力，但也讓模型的擬合能力下降。由於 L1 跟 L2 形成的可行解域並不同，因此在收縮變數上面，Ridge 跟 Lasso 的表現也不一樣。可參考[網址資料](#)得知 [32](#)，隨著 lambda(懲罰值)增加時，Lasso 的變數係數會陸續變為 0；但 Ridge 却不一樣，直到某個瞬間才會全部一起變成 0。正因為這樣的特性，只要選取一個恰當的 lambda，便可以在 Lasso 上找出係數尚未為 0 的變數，以此來進行變數挑選。使用 Lasso Regression 時，調整不同的正規化程度，就會自然使得一部分的特徵係數為 0，因此刪除的是係數為 0 的特徵，不須額外指定門檻，但需調整正規化程度。

```
# Day_028 嵌入法 (Embedded)：使用機器學習模型，根據擬合後的係數，刪除係數低於門檻的特徵 《L1(Lasso) 嵌入法》
from sklearn.linear_model import Lasso
L1_Reg = Lasso(alpha=0.005)
df_CopyMinMax2 = MinMaxScaler().fit_transform(df_Copy2)
train_num = train_Y.shape[0]
train_X = df_CopyMinMax2[:train_num]
L1_Reg.fit(train_X, train_Y)
print(f'根據目標函數，逐步加入特徵或刪除特徵 L1_Reg.coef_, L1_Reg.coef_')

from itertools import compress
L1_mask = list((L1_Reg.coef_>0) | (L1_Reg.coef_<0))
L1_list = list(compress(list(df_Copy2), list(L1_mask)))
print('篩選後留下的變數：', L1_list)
```

L1_EMBEDDING 特徵 + 線性迴歸

```
df_CopyMinMax2 = MinMaxScaler().fit_transform(df_Copy2[L1_list])
train_num = train_Y.shape[0]
train_X = df_CopyMinMax2[:train_num]
estimator = LogisticRegression()
print(f'L1_EMBEDDING 特徵 + 線性迴歸的 cross_val_score:\n', cross_val_score(estimator, train_X, train_Y, cv=5).mean())
```

根據目標函數，逐步加入特徵或刪除特徵 L1_Reg.coef_ [-0.39123864 -0.13640466 -0. 0. 0.]
篩選後留下的變數： ['Pclass', 'Age']
L1_EMBEDDING 特徵 + 線性迴歸的 cross_val_score:
0.6847938159868103

小結 Q&A：承上，使用 L1_EMBEDDING 做特徵選擇(自訂門檻)，觀察預測能力是否提升？

由範例中可見，使用 L1_EMBEDDING 效果並不如顯著，而且與相關係數法類似，L1_EMBEDDING 也需要調整 alpha 值，且沒有一定法則，所以也並非好用的特徵篩選方式

四、嵌入法 (Embedded)：《GDBT(梯度提升樹)嵌入法》

使用梯度提升樹擬合後，以特徵在節點出現的頻率當作特徵重要性，以此刪除重要性低於門檻的特徵，這種作法也稱為 **GDBT 嵌入法**。由於特徵重要性不只可以刪除特徵，也是增加特徵的關鍵參參考。以下是 Day_028 講義提到的三種特徵選取比較，看似各有長處，但是近年來 GDBT 的改良版本：Xgboost...等幾種算法，大幅改良了計算時間，因此成為了特徵選擇的主流。

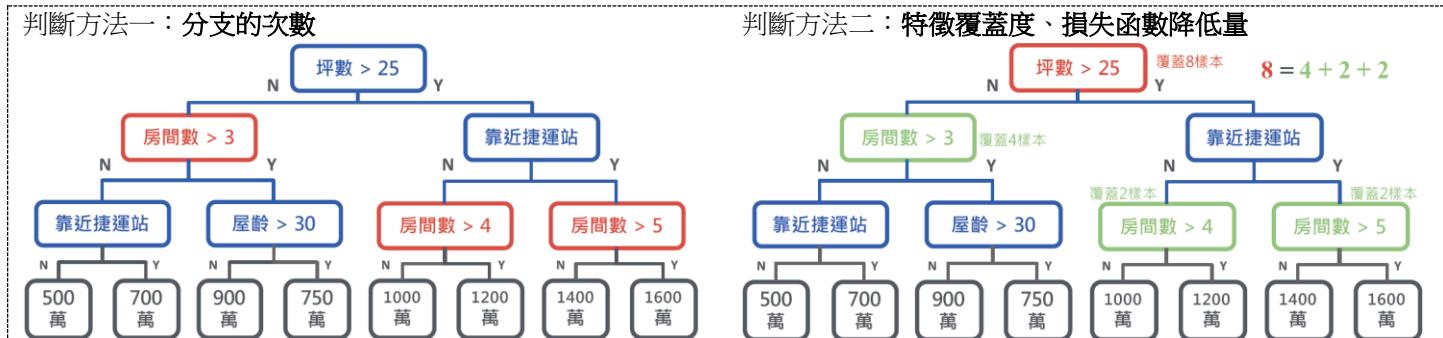
	計算時間	共線性	特徵穩定性
相關係數過濾法	快速	無法排除	穩定
Lasso 嵌入法	快速	能排除	不穩定
GDBT 嵌入法	較慢	能排除	穩定

特徵重要性

回顧 Day_029 講義範例，讓我們先來來看什麼是特徵重要性：範例是房價預估決策樹的預測圖，四個特徵(坪數、房間數、屋齡、是否靠近捷運站)之中，請問你覺得哪一個特徵比較重要？

判斷方法一：特徵重要性預設方式是取特徵決定分支的次數，此例例而言：坪數 x1 次、房間數 x3 次、靠近捷運站 x2 次、屋齡 x1 次，所以最重要的特徵是 房間數

判斷方法二：但分支次數以外，還有兩種更直覺的特徵重要性，及特徵覆蓋度、損失函數降低量，本例的特徵覆蓋度(假定八個結果樣本數量量一樣多)：坪數與房間數的覆蓋度相同(都是 8)，而損失函數降低量，則是要看損失函數(loss function)決定



套件中的特徵重要性

sklearn 當中的樹狀模型，都有特徵重要性這項方法(.feature_importances_)，實際上都是分支次數。

進階版的 GDBT 模型(xgboost, lightgbm, catboost) 中，才有上述三種不同的重要性。

Xgboost 對應參數 (importance_type)	計算時間	估計精確性	sklearn 有此功能
分支次數	weight	最快	最低 ○
分支覆蓋度	cover	快	中 X
損失降低量 (資訊增益度)	gain	較慢	最高 X

```
#Day_029_Feature_Importance
df_Copy3=copy.deepcopy(df)

# 因為需要把類別型與數值型特徵都加入，故使用最簡版的特徵工程
for c in df_Copy3.columns:
    df_Copy3[c] = df_Copy3[c].fillna(-1)
    if df_Copy3[c].dtype == 'object':
        df_Copy3[c] = LabelEncoder().fit_transform(list(df_Copy3[c].values))
    df_Copy3[c] = MinMaxScaler().fit_transform(df_Copy3[c].values.reshape(-1, 1))

df_Copy3.head()
```

Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1.0	0.118683	1.0	0.283951	0.125	0.0	0.775862	0.016072	0.000000
1	0.0	0.218989	0.0	0.481481	0.125	0.0	0.879310	0.140813	0.575269
2	1.0	0.400459	0.0	0.333333	0.000	0.0	0.984914	0.017387	0.000000
3	0.0	0.323124	0.0	0.444444	0.125	0.0	0.070043	0.105390	0.381720
4	1.0	0.016845	1.0	0.444444	0.000	0.0	0.699353	0.017630	0.000000

```
from sklearn.ensemble import GradientBoostingClassifier
# 梯度提升樹擬合後，將結果依照重要性由高到低排序 (note : D27 作業中'Ticket'是第一名特徵, 'Age'是數值特徵中排名最高者)
train_num = train_Y.shape[0]
train_X = df_Copy3[:train_num]
estimator = GradientBoostingClassifier()
estimator.fit(df_Copy3[:train_num].values, train_Y)
```

```

feats = pd.Series(data=estimator.feature_importances_, index=df.columns)
feats = feats.sort_values(ascending=False)
feats

Sex          0.444025
Pclass        0.111766
Ticket        0.098616
Fare          0.088184
Name          0.080810
Age           0.073982
Cabin         0.071090
Embarked      0.012246
Parch         0.011531
SibSp          0.007750
dtype: float64

```

```

# 原始特徵 + 梯度提升樹
train_num = train_Y.shape[0]
train_X = df_Copy3[:train_num]
estimator = GradientBoostingClassifier()
print(f'原始特徵 + 梯度提升樹的 cross_val_score:\n', cross_val_score(estimator, train_X, train_Y, cv=5).mean())

# 高重要性特徵 + 梯度提升樹
high_feature = list(feats[:5].index)
print(f'高重要性特徵:', high_feature)
train_num = train_Y.shape[0]
train_X = df_Copy3[high_feature][:train_num]
estimator = GradientBoostingClassifier()
print(f'高重要性特徵 + 梯度提升樹 + 梯度提升樹的 cross_val_score:\n', cross_val_score(estimator, train_X, train_Y, cv=5).mean())

```

原始特徵 + 梯度提升樹的 cross_val_score:
0.8238381835223724
高重要性特徵: ['Sex', 'Pclass', 'Ticket', 'Fare', 'Name']
高重要性特徵 + 梯度提升樹 + 梯度提升樹的 cross_val_score:
0.8137131279773969

小結 Q&A：將特徵重要性較低的一半特徵刪除後，再做生存率預估，正確率是否有變化？

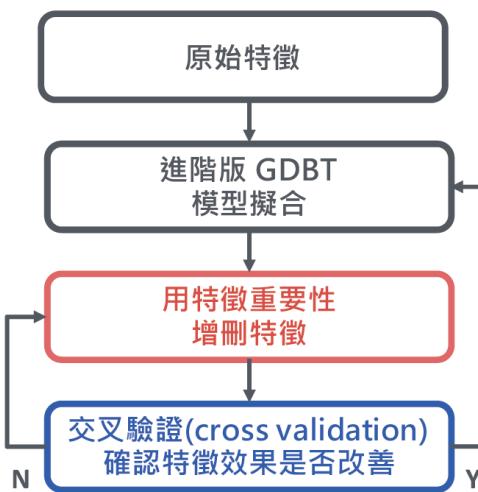
雖然範例中，看似高重要性特徵的預估正確率較差。

但這是只使用一半特徵的結果，表示高重要特徵的資訊密度是高的，使用較少特徵，泛化能力(對外部資料預測力)也會提升，因此只要不要刪到一半那麼多，仍可能是適當的特徵選擇方式。

另一個原因也可能是這邊的模型(GradientBoostingClassifier)並未經過調參，所以特徵重要性排名較不準確導致。

機器學習中的優化循環

- 機器學習特徵優化，循環方式如圖
- 其中增刪特徵指的是
 - 特徵選擇(刪除)
 - 挑選門檻，刪除一部分特徵重要性較低的特徵
 - 特徵組合(增加)
 - 依領域知識，對前幾名特徵做特徵組合或群聚編碼，形成更強力特徵
- 由交叉驗證確認特徵是否有改善，若沒有改善則回到上一輪重選特徵增刪
- 這樣的流程圖綜合了 PART 2 : 特徵工程的主要內容，是這個部分的**核心知識**



套件中的特徵重要性

- sklearn 當中的樹狀模型，都有特徵重要性這項方法 (.feature_importances_)，而實際上都是**分支次數**
- 進階版的 GDBT 模型(xgboost, lightgbm, catboost) 中，才有上述三種不同的重要性

Xgboost 對應參數 (importance_type)		計算時間	估計精確性	sklearn 有此功能
分支次數	weight	最快	最低	○
分支覆蓋度	cover	快	中	✗
損失降低量 (資訊增益度)	gain	較慢	最高	✗

排列重要性 (permutation Importance)

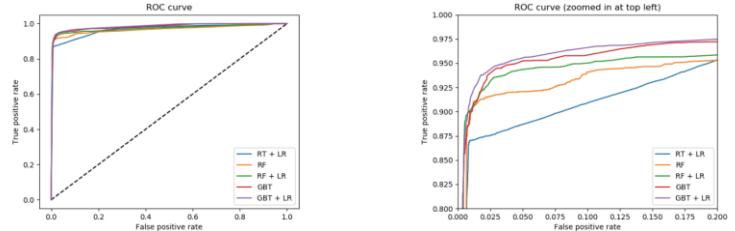
- 雖然特徵重要性相當實用，然而計算原理必須基於樹狀模型，於是有了可延伸至非樹狀模型的**排序重要性**
- 排序重要性計算，是打散單一特徵的資料排序順序，再用原本模型重新預測，觀察打散前後誤差會變化多少

特徵重要性 Feature Importance		排序重要性 Permutation Importance
適用模型	限定樹狀模型	機器學習模型均可
計算原理	樹狀模型的分歧特徵	打散原始資料中單一特徵的排序
額外計算時間	較短	較長

補充

葉編碼 (leaf encoding) + 邏輯斯迴歸

- 葉編碼需要先對樹狀模型擬合後才能生成，如果這步驟挑選了較佳的參數，後續處理效果也會較好，這點與特徵重要性類似
- 實際結果也證明，在分類預測中使用樹狀模型，再對這些擬合完的樹狀模型進行葉編碼+邏輯斯迴歸，通常會將預測效果再進一步提升



圖片來源：scikit-learn

複習 Day30 葉編碼- 只有分類問題比較適合葉編碼

```
# Day_30 葉編碼 只有分類問題比較適合葉編碼
# 做完特徵工程前的所有準備

import pandas as pd
import numpy as np
import copy
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
# 因為擬合(fit)與編碼(transform)需要分開, 因此不使用get_dummies, 而採用sklearn的OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve

df_Copy4=copy.deepcopy(df)
# 因為需要把類別型與數值型特徵都加入, 故使用最簡版的特徵工程
for c in df_Copy4.columns:
    df_Copy4[c] = df_Copy4[c].fillna(-1)
    if df_Copy4[c].dtype == 'object':
        df_Copy4[c] = LabelEncoder().fit_transform(list(df_Copy4[c].values))
    df_Copy4[c] = MinMaxScaler().fit_transform(df_Copy4[c].values.reshape(-1, 1))
df_Copy4.head()

train_num = train_Y.shape[0]
train_X = df_Copy4[:train_num].values
# 因為訓練邏輯斯迴歸時也要資料, 因此將訓練及切成三部分 train / val / test, 採用 test 驗證而非 k-fold 交叉驗證
# train 用來訓練梯度提升樹, val 用來訓練邏輯斯迴歸, test 驗證效果
train_X, test_X, train_Y, test_Y = train_test_split(train_X, train_Y, test_size=0.5)
train_X, val_X, train_Y, val_Y = train_test_split(train_X, train_Y, test_size=0.5)
```

```
# 梯度提升樹調整參數並擬合後, 再將葉編碼(*.apply)結果做獨熱 / 邏輯斯迴歸
# 調整參數的方式採用 RandomSearchCV 或 GridSearchCV, 以後的進度會再教給大家, 本次先直接使用調參結果
gdbt = GradientBoostingClassifier(subsample=0.93, n_estimators=320, min_samples_split=0.1, min_samples_leaf=0.3,
                                   max_features=4, max_depth=4, learning_rate=0.16)
onehot = OneHotEncoder()
lr = LogisticRegression(solver='lbfgs', max_iter=1000)

gdbt.fit(train_X, train_Y)
onehot.fit(gdbt.apply(train_X)[:, :, 0])
lr.fit(onehot.transform(gdbt.apply(val_X)[:, :, 0]), val_Y)

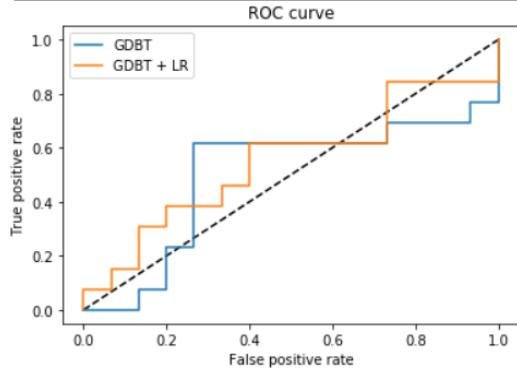
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=1000, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
                    tol=0.0001, verbose=0, warm_start=False)
```

```

# 將梯度提升樹+葉編碼+邏輯斯迴歸結果輸出
pred_gdbt_lr = lr.predict_proba(onehot.transform(gdbt.apply(test_X)[:, :, 0]))[:, 1]
fpr_gdbt_lr, tpr_gdbt_lr, _ = roc_curve(test_Y, pred_gdbt_lr)
# 將梯度提升樹結果輸出
pred_gdbt = gdbt.predict_proba(test_X)[:, 1]
fpr_gdbt, tpr_gdbt, _ = roc_curve(test_Y, pred_gdbt)

import matplotlib.pyplot as plt
# 將結果繪圖
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_gdbt, tpr_gdbt, label='GDBT')
plt.plot(fpr_gdbt_lr, tpr_gdbt_lr, label='GDBT + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()

```



```

#完成隨機森林的鐵達尼生存率預測，以及對應的葉編碼+邏輯斯迴歸
rf = RandomForestClassifier(n_estimators=20, min_samples_split=10, min_samples_leaf=5,
                           max_features=4, max_depth=3, bootstrap=True)
onehot = OneHotEncoder()
lr = LogisticRegression(solver='lbfgs', max_iter=1000)

rf.fit(train_X, train_Y)
onehot.fit(rf.apply(train_X))
lr.fit(onehot.transform(rf.apply(val_X)), val_Y)

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=1000, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
                    tol=0.0001, verbose=0, warm_start=False)

```

```

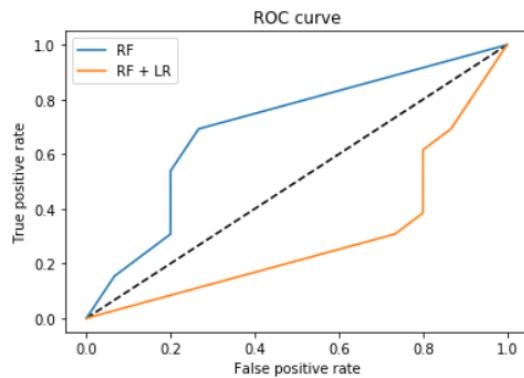
# 將隨機森林+葉編碼+邏輯斯迴歸結果輸出
pred_rf_lr = lr.predict_proba(onehot.transform(rf.apply(test_X)))[:, 1]
fpr_rf_lr, tpr_rf_lr, _ = roc_curve(test_Y, pred_rf_lr)
# 將隨機森林結果輸出
pred_rf = rf.predict_proba(test_X)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(test_Y, pred_rf)

```

```

import matplotlib.pyplot as plt
# 將結果繪圖
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.plot(fpr_rf_lr, tpr_rf_lr, label='RF + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()

```



小結 Q&A：上述的結果，葉編碼是否有提高預測的正確性呢？

以 `roc_auc` 而言，看起來效果不是很明顯，範例中的結果甚至反過來，建議可以嘗試多跑幾次，結果可能都有些差異，所以葉編碼效果仍要看資料而定，未必一定比較好，而是提供多一個方式讓大家選擇

分類預測的集成

由於分類預測的集成，概念上與迴歸預測的集成有所不同，所以在最後做個補充

分類的預測結果，意義上是對機率的預估，而不同特徵表示不同的判斷條件

想一想：假如要估計鐵達尼號上的生存機率

已知來自法國的旅客生存機率是 0.8，且年齡 40 到 50 區間的生存機率也是 0.8

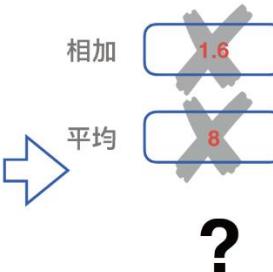
假如當作兩個預估模型，迴歸預測要集成兩種預測的做法有兩種：相加或平均

但是相加 $0.8 + 0.8 = 1.6$ ，機率會超過 1，不合理

平均 $(0.8 + 0.8) / 2 = 0.8$ ，法國機率已是 0.8，加上正向的事件居然還更低，也不合理

應該要比 0.8 更高，但又不能到 1

那麼，該如何集成才合理呢



解法：邏輯斯迴歸(logistic regression)與其重組

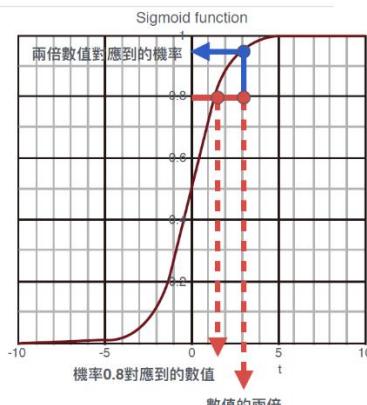
我們可以將邏輯斯迴歸理解成「線性迴歸 + Sigmoid 函數」

而 sigmoid 函數理解成「成功可能性與機率的互換」

這裡的成功可能性正表示更可能，負表示較不可能

所以當我們使用 sigmoid 的反函數
就可以將機率重新轉為成功可能性
加完後再用 sigmoid 轉回機率

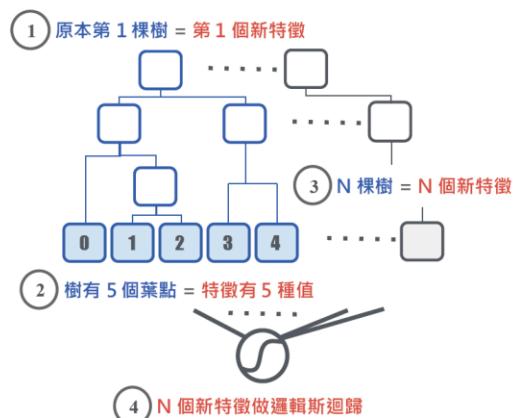
以此例而言，我們可以看到最後加成的結果是一個
介於 0.9 到 1 之間的機率



底圖圖片來源：[wikipedia](#)

葉編碼 (leaf encoding) 原理

- 樹狀模型 (只要樹的數量大於 1) 做出預測時，模型預測時就免不了要用到相加 (梯度提升樹) 或平均 (隨機森林) 等集成方式
- 所以在分類預測時，理論上我們也必須將每棵樹對應的機率用 sigmoid 反函數反轉成成功可能性後相加 / 平均，但實際上可以使用葉編碼再加上邏輯斯迴歸，就可以達成上述結果
- 葉編碼 (leaf encoding) 顧名思義，是採用決策樹的葉點作為編碼依據重新編碼
- 每棵樹視為一個新特徵，每個新特徵均為分類型特徵，決策樹的葉點與該特徵標籤一一對應
- 最後再以邏輯斯迴歸合併預測



衍伸討論：有關樹狀模型與模型可解釋性

- 經由課程我們知道：樹狀模型有幾個重要的應用
 - 特徵重要性(feature importance)：目前是特徵選擇的最主流作法
 - 葉編碼：將特徵打散，完全依照樹狀模型的葉點重新編碼，再加上邏輯斯迴歸，可以再進一步提升分類預測能力
- 上述樹狀模型的獨特應用，都是基於人們對決策樹的理解與可解釋性(explainable)而有的設計
- 但目前深度學習的基礎：類神經網路，最缺乏的就是可以解釋性，若類神經網路能在可解釋性上更進一步，則可以想見也可以有更多的衍伸應用例如：capsule 模型)

3-2.2 範例 2：計程車費率預測 (LinearRegression)

<https://www.kaggle.com/c/new-york-city-taxi-fare-prediction>

使用程車費率預測競賽練習時間欄位處理

- 時間特徵分解 + 特徵組合
- LinearRegression
- GradientBoostingRegressor(GDBT)

Code 4: Day017-030 Feature Engineering_ReviewPART II.ipynb

```
# 做完特徵工程前的所有準備
```

```
import pandas as pd
import numpy as np
import datetime
data_path = 'data/'
df = pd.read_csv(data_path + 'taxi_data1.csv')

train_Y = df['fare_amount']
df = df.drop(['fare_amount'], axis=1)
df.head()
```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2011-10-21 23:54:10 UTC	-73.990580	40.761071	-73.981128	40.758634	2
1	2015-02-03 10:42:03 UTC	-73.988403	40.723431	-73.989647	40.741695	1
2	2014-03-16 18:58:58 UTC	-74.015785	40.715110	-74.012029	40.707888	2
3	2009-06-13 16:10:54 UTC	-73.977322	40.787275	-73.958030	40.778838	3
4	2014-06-12 03:25:56 UTC	-73.989683	40.729717	-73.982490	40.761887	3

```
df.describe()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	-72.635174	39.981148	-72.619979	39.972626	1.694400
std	10.013060	5.989760	10.065420	6.016378	1.315962
min	-82.325968	-73.999735	-82.299888	-74.002015	0.000000
25%	-73.992029	40.735483	-73.991756	40.733386	1.000000
50%	-73.981477	40.752824	-73.979938	40.752687	1.000000
75%	-73.966317	40.767127	-73.963018	40.767461	2.000000
max	40.763382	41.366138	40.755657	41.366138	6.000000

```
#####
# Day_025_DayTime_Features_日週期
```

```
# 時間特徵分解方式: 使用 datetime
```

```
df['pickup_datetime'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S UTC'))
df['pickup_year'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%Y')).astype('int64')
df['pickup_month'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%m')).astype('int64')
df['pickup_day'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%d')).astype('int64')
df['pickup_hour'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%H')).astype('int64')
df['pickup_minute'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%M')).astype('int64')
df['pickup_second'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%S')).astype('int64')
df.head()
```

```
# 語法參閱 https://docs.python.org/3/library/datetime.html
```

```
# 加入星期幾 (day of week) 與第幾周 (week of year) 這兩項特徵
```

```
# df['pickup_dow'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%w')).astype('int64')
# df['pickup_woy'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%W')).astype('int64')
# df.head()
```

```
# 加上"日週期"特徵
```

```
#import math
```

```

#df['day_cycle'] = df['pickup_hour']/12 + df['pickup_minute']/720 + df['pickup_second']/43200
#df['day_cycle'] = df['day_cycle'].map(lambda x:math.sin(x*math.pi))
#df.head()
# 加上"年週期"與"周週期"特徵(可自訂公式)
#df['year_cycle'] = df['pickup_month']/6 + df['pickup_day']/180
#df['year_cycle'] = df['year_cycle'].map(lambda x:math.cos(x*math.pi))
#df['week_cycle'] = df['pickup_dow']/3.5 + df['pickup_hour']/84
#df['week_cycle'] = df['week_cycle'].map(lambda x:math.sin(x*math.pi))
#df.head()

```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_day	pickup_hour	pickup_minute	pickup_second
0	2011-10-21 23:54:10	-73.990580	40.761071	-73.981128	40.758634	2	2011	10	21	23	54	10
1	2015-02-03 10:42:03	-73.988403	40.723431	-73.989647	40.741695	1	2015	2	3	10	42	3
2	2014-03-16 18:58:58	-74.015785	40.715110	-74.012029	40.707888	2	2014	3	16	18	58	58
3	2009-06-13 16:10:54	-73.977322	40.787275	-73.958030	40.778838	3	2009	6	13	16	10	54
4	2014-06-12 03:25:56	-73.989683	40.729717	-73.982490	40.761887	3	2014	6	12	3	25	56

```

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor

# 將結果使用線性迴歸/ 梯度提升樹分別看結果
df_temp = df.drop(['pickup_datetime'], axis=1)
scaler = MinMaxScaler()
train_X = scaler.fit_transform(df_temp)
Linear = LinearRegression()
print(f'Linear Reg Score : {cross_val_score(Linear, train_X, train_Y, cv=5).mean()}')
GDBT = GradientBoostingRegressor()
print(f'Gradient Boosting Reg Score : {cross_val_score(GDBT, train_X, train_Y, cv=5).mean()}')

Linear Reg Score : 0.026876871475642418
Gradient Boosting Reg Score : 0.7118163305669516

```

依參考資料，試著加入星期幾 (day of week) 與第幾周 (week of year) 這兩項特徵，看看結果會比原本只有時間特徵分解的結果更好或更差？

- 加入星期幾與第幾周的特徵後，看起來結果似乎更差了，可能表示"第幾周"這個特徵沒有提供多少資訊
- 但是搭配日週期之後效果有提升，可能是假日與非假日(星期幾)的作息(日週期)與費率有相關性

```

# 語法參閱 https://docs.python.org/3/library/datetime.html
df['pickup_dow'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%w')).astype('int64')
df['pickup_woy'] = df['pickup_datetime'].apply(lambda x: datetime.datetime.strftime(x, '%W')).astype('int64')
df.head()

```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_day	pickup_hour	pickup_minute	pickup_second	dow	woy
0	2011-10-21 23:54:10	-73.990580	40.76107	-73.981128	40.75863	2	2011	10	21	23	54	10	5	42
1	2015-02-03 10:42:03	-73.988403	40.72343	-73.989647	40.74169	1	2015	2	3	10	42	3	2	5
2	2014-03-16 18:58:58	-74.015785	40.71511	-74.012029	40.70788	2	2014	3	16	18	58	58	0	10
3	2009-06-13 16:10:54	-73.977322	40.78727	-73.958030	40.77883	3	2009	6	13	16	10	54	6	23
4	2014-06-12 03:25:56	-73.989683	40.72971	-73.982490	40.76188	3	2014	6	12	3	25	56	4	23

```

# 將結果使用線性迴歸/ 梯度提升樹分別看結果
df_temp = df.drop(['pickup_datetime'], axis=1)
train_X = scaler.fit_transform(df_temp)
print(f'Linear Reg Score : {cross_val_score(Linear, train_X, train_Y, cv=5).mean()}')
print(f'Gradient Boosting Reg Score : {cross_val_score(GDBT, train_X, train_Y, cv=5).mean()}')

Linear Reg Score : 0.028649006962517265
Gradient Boosting Reg Score : 0.7099576444627768

```

```
# 加上"日週期"特徵(參考講義"週期循環特徵")
```

```
import math
```

```
df['day_cycle'] = df['pickup_hour']/12 + df['pickup_minute']/720 + df['pickup_second']/43200
```

```
df['day_cycle'] = df['day_cycle'].map(lambda x:math.sin(x*math.pi))
```

```
df.head()
```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_day	pickup_hour	pickup_minute	pickup_second	pickup_dow	pickup_woy	day_cycle
0	2011-10-21 23:54:10	-73.990580	40.761071	-73.981128	40.758634	2	2011	10	21	23	54	10	5	42	-0.0254
1	2015-02-03 10:42:03	-73.988403	40.723431	-73.989647	40.741695	1	2015	2	3	10	42	3	2	5	0.3336
2	2014-03-16 18:58:58	-74.015785	40.715110	-74.012029	40.707888	2	2014	3	16	18	58	58	0	10	-0.9670
3	2009-06-13 16:10:54	-73.977322	40.787275	-73.958030	40.778838	3	2009	6	13	16	10	54	6	23	-0.8888
4	2014-06-12 03:25:56	-73.989683	40.729717	-73.982490	40.761887	3	2014	6	12	3	25	56	4	23	0.7824

```
# 將結果使用線性迴歸/梯度提升樹分別看結果
```

```
df_temp = df.drop(['pickup_datetime'], axis=1)
```

```
train_X = scaler.fit_transform(df_temp)
```

```
print(f'Linear Reg Score : {cross_val_score(Linear, train_X, train_Y, cv=5).mean()}')
```

```
print(f'Gradient Boosting Reg Score : {cross_val_score(GDBT, train_X, train_Y, cv=5).mean()}')
```

```
Linear Reg Score : 0.028274009668816637
```

```
Gradient Boosting Reg Score : 0.7133543824849025
```

依參考資料，試著完成年週期與周週期的特徵，看看結果會比範例中的結果更好或更差？

- 從結果來看，加上年週期與周週期，正確性有顯著的提升，這可能表示冷熱等確實與費率也有相關性(天氣熱比較搭計程車?)

```
# 加上"年週期"與"周週期"特徵
```

```
df['year_cycle'] = df['pickup_month']/6 + df['pickup_day']/180
```

```
df['year_cycle'] = df['year_cycle'].map(lambda x:math.cos(x*math.pi))
```

```
df['week_cycle'] = df['pickup_dow']/3.5 + df['pickup_hour']/84
```

```
df['week_cycle'] = df['week_cycle'].map(lambda x:math.sin(x*math.pi))
```

```
df.head()
```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_day	pickup_hour	pickup_minute	pickup_second	pickup_dow	pickup_woy	day_cycle	year_cycle	week_cycle
0	2011-10-21 23:54:10	-73.99058	40.76107	-73.98112	40.75863	2	2011	10	21	23	54	10	5	42	-0.0254	0.7771	-0.8045
1	2015-02-03 10:42:03	-73.98840	40.72343	-73.98964	40.74169	1	2015	2	3	10	42	3	2	5	0.3336	0.4539	0.8262
2	2014-03-16 18:58:58	-74.01578	40.71511	-74.01202	40.70788	2	2014	3	16	18	58	58	0	10	-0.9670	-0.2756	0.6234
3	2009-06-13 16:10:54	-73.97732	40.78727	-73.95803	40.77883	3	2009	6	13	16	10	54	6	23	-0.8888	-0.9743	-0.2947
4	2014-06-12 03:25:56	-73.98968	40.72971	-73.98249	40.76188	3	2014	6	12	3	25	56	4	23	0.7824	-0.9781	-0.5320

```
# 將結果使用線性迴歸/梯度提升樹分別看結果
```

```
df_temp = df.drop(['pickup_datetime'], axis=1)
```

```
train_X = scaler.fit_transform(df_temp)
```

```
print(f'Linear Reg Score : {cross_val_score(Linear, train_X, train_Y, cv=5).mean()}')
```

```
print(f'Gradient Boosting Reg Score : {cross_val_score(GDBT, train_X, train_Y, cv=5).mean()}')
```

```
Linear Reg Score : 0.028326599866692927
```

```
Gradient Boosting Reg Score : 0.7143342369441861
```

```

#####
# Day_026_特徵組合 - 數值與數值組合，見 PART II
# 增加緯度差, 經度差, 座標距離等三個特徵
df['longitude_diff'] = df['dropoff_longitude'] - df['pickup_longitude']
df['latitude_diff'] = df['dropoff_latitude'] - df['pickup_latitude']
df['distance_2D'] = (df['longitude_diff']**2 + df['latitude_diff']**2)**0.5
df[['distance_2D', 'longitude_diff', 'latitude_diff', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude']].head()

```

	distance_2D	longitude_diff	latitude_diff	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	0.009761	0.009452	-0.002437	-73.990580	40.761071	-73.981128	40.758634
1	0.018307	-0.001244	0.018265	-73.988403	40.723431	-73.989647	40.741695
2	0.008140	0.003756	-0.007222	-74.015785	40.715110	-74.012029	40.707888
3	0.021056	0.019292	-0.008437	-73.977322	40.787275	-73.958030	40.778838
4	0.032964	0.007193	0.032170	-73.989683	40.729717	-73.982490	40.761887

```

# 結果：準確度上升
df_temp = df.drop(['pickup_datetime'], axis=1)
train_X = scaler.fit_transform(df_temp)
Linear = LinearRegression()
print(f'Linear Reg Score : {cross_val_score(Linear, train_X, train_Y, cv=5).mean()}')
GDBT = GradientBoostingRegressor()
print(f'Gradient Boosting Reg Score : {cross_val_score(GDBT, train_X, train_Y, cv=5).mean()}')

```

Linear Reg Score : 0.027357389983710022
 Gradient Boosting Reg Score : 0.8062592717614099

依參考教材，試著使用經緯度一圓的長度比這一概念，組合出一個新特徵，再觀察原特徵加上新特徵是否提升了正確率？

- 結果準確率反而下降了，其原因並非是經緯度比例不正確，而是因為這些計程車地點在紐約
- 紐約河流與主要道路均為南北向，所以使得東西向的跨越往往路程比較崎嶇，計程車費反而較高（同學可以試著將經度差改成緯度差的 1.3 倍計算，反而會提升預測率）這告訴我們：不能只單靠領域知識，仍要以實際資料輔助驗證才行

```

import math
latitude_average = df['pickup_latitude'].mean()
latitude_factor = math.cos(latitude_average/180*math.pi)
df['distance_real'] = ((df['longitude_diff']*latitude_factor)**2 + df['latitude_diff']**2)**0.5

# 觀察結果：準確度下降
df_temp = df.drop(['pickup_datetime'], axis=1)
train_X = scaler.fit_transform(df_temp)
print(f'Linear Reg Score : {cross_val_score(Linear, train_X, train_Y, cv=5).mean()}')
print(f'Gradient Boosting Reg Score : {cross_val_score(GDBT, train_X, train_Y, cv=5).mean()}')

```

Linear Reg Score : 0.025282421299993675
 Gradient Boosting Reg Score : 0.8012370688880539

參考教材

時間特徵分解

最常見的特殊欄位是時間欄位，想想看應該怎樣編碼？

時間戳記	年	月	日	時	分	秒
2014-06-12 03:25:56	2014	6	12	3	25	56
2011-07-16 01:19:59	2011	7	16	1	19	59
2011-10-21 23:54:10	2011	10	21	23	54	10
2015-02-03 10:42:03	2015	2	3	10	42	3
2009-06-13 16:10:54	2009	6	13	16	10	54
2010-08-09 14:46:03	2010	8	9	14	46	3

週期循環特徵

時間也有週期的概念，可以用週期合成一些重要的特徵

聯想看看：有哪幾種時間週期，可串聯到一些可做特徵的性質？



- **年週期** 與春夏秋冬季節溫度相關
- **月週期** 與薪水、繳費相關
- **周週期** 與周休、消費習慣相關
- **日週期** 與生理時鐘相關

前述的週期所需數值都可由時間欄位組成，但還首尾相接

因此週期特徵還需以正弦函數(**sin**)或餘弦函數(**cos**)加以組合

例如：年週期 (正 : **冷** / 負 : **熱**)

$$\cos((月/6 + 日/180)\pi)$$

周週期 (正 : **精神飽滿** / 負 : **疲倦**)

$$\sin((星期幾/3.5 + 小時/84)\pi)$$

日週期 (正 : **精神飽滿** / 負 : **疲倦**)

$$\sin((小時/12 + 分/720 + 秒/43200)\pi)$$

*註：此處小時是24小時制

針對 Python 時間模組 `datetime\time` 進行詳細探討。轉義符對應意義如下

`%a` 本地簡化星期名稱

`%A` 本地完整星期名稱

`%b` 本地簡化的月份名稱

`%B` 本地完整的月份名稱

`%c` 本地相應的日期表示和時間表示

`%d` 月內中的一天 (0-31)

`%H` 24 小時制小時數 (0-23)

`%I` 12 小時制小時數 (01-12)

`%j` 年內的一天 (001-366)

`%m` 月份 (01-12)

`%M` 分鐘數 (00=59)

`%p` 本地 A.M. 或 P.M. 的等價符

`%S` 秒 (00-59)

特徵組合

在計程車費預估中，有四個欄位分別表示起終點的經緯度
想想看，是否可以用這些組合出與車費更有相關的特徵？

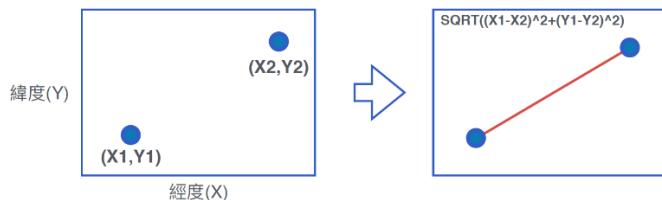
起點經度 起點緯度 終點經度 終點緯度

-73.99058	40.76107	-73.98112	40.75863
-73.98840	40.72343	-73.98964	40.74169
-74.01578	40.71511	-74.01202	40.70788
-73.97732	40.78727	-73.95803	40.77883



合理的想法是：將這四個特徵看成座標

因此用平面座標距離組合出來的特徵，更有預測力也非常合理



事實：經緯度每一度並不一樣長



圖片來源：worldatlas

觀察資料緯度集中在 40.75 度附近

可以算得經度與緯度代表的長度比為 $\cos(40.75\text{度}) : 1 = 0.75756 : 1$

由此校正後的兩地距離，預測正確度更高

`%U` 一年中的星期數 (00-53) 星期天為星期的開始

`%w` 星期 (0-6) · 星期天為星期的開始

`%W` 一年中的星期數 (00-53) 星期一為星期的開始

`%x` 本地相應的日期表示

`%X` 本地相應的時間表示

`%y` 兩位數的年份表示 (00-99)

`%Y` 四位數的年份表示 (000-9999)

`%Z` 當前時區的名稱

`%%` %號本身

3-2.3 範例 3：鳶尾花辨識(multi-classification: KNeighborsClassifier & 樹狀模型)

用 Python 自學資料科學與機器學習入門實戰：Scikit Learn 基礎入門 <https://www.happy coder.org/2017/10/14/python-data-science-and-machine-learning-scikit-learn-basic-tutorial/>

- 教學文章使用 scikit-learn 內建的資料集 Iris (鳶尾花) 的花萼、花蕊長寬進行花朵類別判別 (**setosa** 山鳶尾、**versicolor** 變色鳶尾、**virginica** 維吉尼亞鳶尾)。在這個資料集中已經幫我們標註好每筆資料對應的類別，所以我們可以視為多類別分類問題 (multi-classification)。



Code 5: Day017-030 Feature Engineering_ReviewPART III_Iris.ipynb

```
# 引入 numpy、pd 和 sklearn(scikit-learn) 模組
import numpy as np
import pandas as pd
from sklearn import datasets, metrics
# 引入 train_test_split 分割方法，注意在 sklearn v0.18 後 train_test_split 從 sklearn.cross_validation 子模組搬到
# sklearn.model_selection 中
from sklearn.model_selection import train_test_split
# 引入 KNeighbors 模型
from sklearn.neighbors import KNeighborsClassifier
```

```
# 引入 iris 資料集
raw_iris = datasets.load_iris()
# 探索性分析 Exploratory data analysis，了解資料集內容
# 先印出 key 值，列出有哪些值：['data', 'target', 'target_names', 'DESCR', 'feature_names']
print(raw_iris.keys())

# 印出 feature 值
print(raw_iris['data'])
# 印出目標值，分別對應的是三種花的類別：['setosa 山鳶尾' 'versicolor 變色鳶尾' 'virginica 維吉尼亞鳶尾']
print(raw_iris['target'])
# 印出目標標籤，三種花的類別：['setosa' 'versicolor' 'virginica']
print(raw_iris['target_names'])
# 印出資料集內容描述
print(raw_iris['DESCR'])
# 印出屬性名稱，['sepal length 花萼長度(cm)', 'sepal width 花萼寬度(cm)', 'petal length 花蕊長度(cm)', 'petal width 花蕊寬度(cm)']
print(raw_iris['feature_names'])

# 類別種類
print(np.unique(raw_iris.target))
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 :
 [6.7 3.3 5.7 2.5]
 [6.7 3. 5.2 2.3]
 [6.3 2.5 5. 1.9]
 [6.5 3. 5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3. 5.1 1.8]]]
```



```

# 將資料轉為 pandas DataFrame
# data 為觀察目標變數
df_X = pd.DataFrame(raw_iris.data)
# target 為預測變數
df_y = pd.DataFrame(raw_iris.target)

# 將資料切分為 training data 和 testing data，其中 random_state 若設為 0 或不設則即便實例不同但因種子相同產生同樣隨機編號，若設為 1 則每次隨機產生不同編號
# test_size 為切分 training data 和 testing data 的比例
X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.25, random_state=4)

# 印出所有資料集筆數
print(len(df_y))
print(len(y_train))# 印出切分 y_train 的數量為所有資料集的 70%，共 105 筆
print(len(y_test))# 印出切分的 y_test 資料為所有資料集的 30%，共 45 筆

```

150
112
38

```

#建立分類器 1:
# 初始化 KNeighborsClassifier 實例
knn = KNeighborsClassifier()
# 使用 fit 來建置模型，其參數接收 training data matrix, testing data array，所以進行 y_train.values.ravel() 轉換
knn.fit(X_train, y_train.values.ravel())

# 使用 X_test 來預測結果
print("predict:", knn.predict(X_test))
# 印出 testing data 預測標籤機率
print("predict_proba:", knn.predict_proba(X_test))
# 印出預測準確率
print("score", knn.score(X_test, y_test))

```

#延伸閱讀: 如何選擇 k

<https://ithelp.ithome.com.tw/articles/10187191>

```

predict: [2 0 2 2 2 1 2 0 0 2 0 0 0 1 2 0 1 0 0 2 0 2 1 0 0 0 0 0 0 2 1 0 2 0 1 2 2 1]
predict_proba: [[0. 0. 1. ]
 [1. 0. 0. ]
 [0. 0. 1. ]
 [0. 0. 1. ]
 [0. 1. 0. ]
 [0. 0.2 0.8]
 [1. 0. 0. ]
 :
 [0. 1. 0. ]]
score 0.9736842105263158

```

```

# Day_042_建立分類器2: DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
iris_clf = clf.fit(X_train, y_train.values.ravel())
# 預測測試集
y_pred = clf.predict(X_test)
# 預測
print("predict:", iris_clf.predict(X_test))
# 印出預測準確率
print("score", iris_clf.score(X_test, y_test))
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("iris.feature_names", raw_iris.feature_names)
print("Feature importance: ", clf.feature_importances_)
predict: [2 0 2 2 2 1 2 0 0 2 0 0 0 1 2 0 1 0 0 2 0 1 0 0 0 0 0 0 2 1 0 2 0 1 2 2 1]
score 0.9736842105263158
Accuracy: 0.9736842105263158
iris.feature_names ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Feature importance: [0.01796599 0. 0.05992368 0.92211033]

```

```

# 安裝pydotplus C:\Users\Administrator\Anaconda3\Scripts\pip.exe install pydotplus
# GraphViz's executables not found 錯誤 http://beanobody.blogspot.com/2017/04/graphviz-executables-not-found.html
import os
print(os.environ['PATH'])
os.environ['PATH'] = os.environ['PATH'] + (';c:\\Program Files (x86)\\Graphviz2.38\\bin\\')

```

```

from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz

```

```

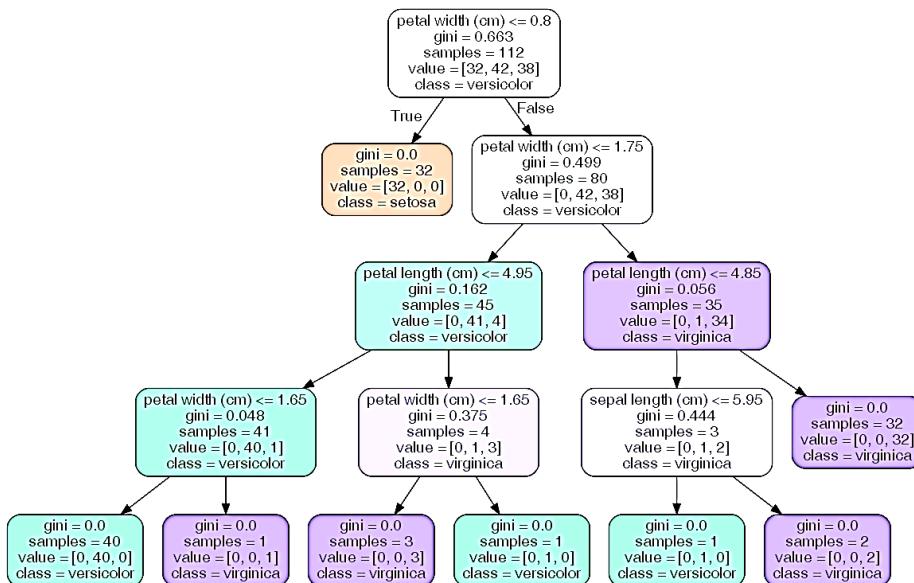
dot_data = export_graphviz(
    clf,
    filled=True,
    rounded=True,
    class_names=raw_iris.target_names,
    feature_names=raw_iris.feature_names,
    out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree.png')

```

```

from IPython.display import Image
Image(filename = 'tree.png', width=500)

```



```

# Day_044_建立分類器3: random_forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
iris_clf = clf.fit(X_train, y_train.values.ravel())
# 預測測試集
y_pred = clf.predict(X_test)
# 預測
print("predict:", iris_clf.predict(X_test))
# 印出預測準確率
print("score", iris_clf.score(X_test, y_test))
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("iris.feature_names", raw_iris.feature_names)
print("Feature importance: ", clf.feature_importances_)
predict: [2 0 2 2 2 1 2 0 0 1 0 0 0 1 2 0 1 0 0 2 0 2 1 0 0 0 0 0 0 2 1 0 2 0 1 2 2 1]
score 0.9473684210526315
Accuracy: 0.9473684210526315
iris.feature_names ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Feature importance: [0.03436959 0.02352723 0.44474541 0.49735777]

```

```

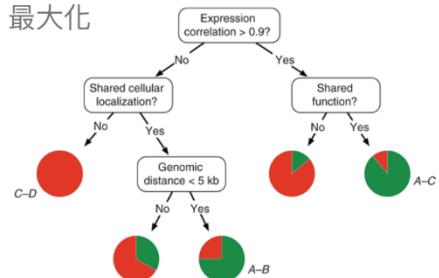
# Day_046_建立分類器4: gradient_boosting_machine
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier()
iris_clf = clf.fit(X_train, y_train.values.ravel())
# 預測測試集
y_pred = clf.predict(X_test)
# 預測
print("predict:", iris_clf.predict(X_test))
# 印出預測準確率
print("score", iris_clf.score(X_test, y_test))
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("iris.feature_names", raw_iris.feature_names)
print("Feature importance: ", clf.feature_importances_)
predict: [2 0 2 2 2 1 2 0 0 2 0 0 0 1 2 0 1 0 0 2 0 2 1 0 0 0 0 0 0 2 1 0 2 0 1 2 2 1]
score 0.9736842105263158
Accuracy: 0.9736842105263158
iris.feature_names ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Feature importance: [0.00942538 0.01722308 0.32758917 0.64576237]

```

Day 41-42

決策樹 (Decision Tree)

- 透過一系列的是非問題，幫助我們將資料進行切分
- 可視覺化每個決策的過程，是個具有非常高解釋性的模型
- 從訓練資料中找出規則，讓每一次決策能使訊息增益 (Information Gain) 最大化
- 訊息增益越大代表切分後的兩群資料，群內相似程度越高
- 例如使用健檢資料來預測性別，若使用頭髮長度超過 50 公分來切分，則切分後兩群資料很有可能多數都為男生或女生 (相似程度高) 這樣頭髮長度就是個很好的 feature。

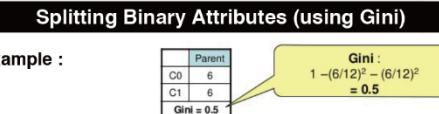


訊息增益 (Information Gain)

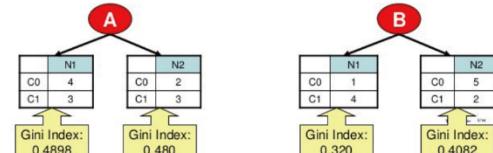
- 決策樹模型會用 features 切分資料，該選用哪個 feature 來切分則是由訊息增益的大小決定的。希望切分後的資料相似程度很高，通常使用吉尼係數來衡量相似程度
- 該怎麼衡量資料相似程度？通常使用吉尼係數 (gini-index) 或熵 (entropy) 來衡量，兩者都可使用

$$Gini = 1 - \sum_j p_j^2$$

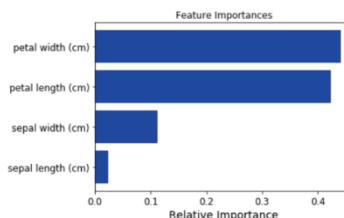
$$Entropy = - \sum_j p_j \log_2 p_j$$



Suppose there are two ways(A and B) to split the data into smaller subset.



By : Mohd Nor Abdul Hamid, Ph.D
(universiti Utara Malaysia)



決策樹的特徵重要性 (Feature importance)

- 我們可以從構建樹的過程中，透過 feature 被用來切分的次數，來得知哪些 features 是相對有用的
- 所有 feature importance 的總和為 1
- 實務上可以使用 feature importance 來了解模型如何進行分類

使用 Sklearn 建立決策樹模型

根據回歸/分類問題分別建立不同的 Classifier

```

from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
  
```

決策樹的超參數

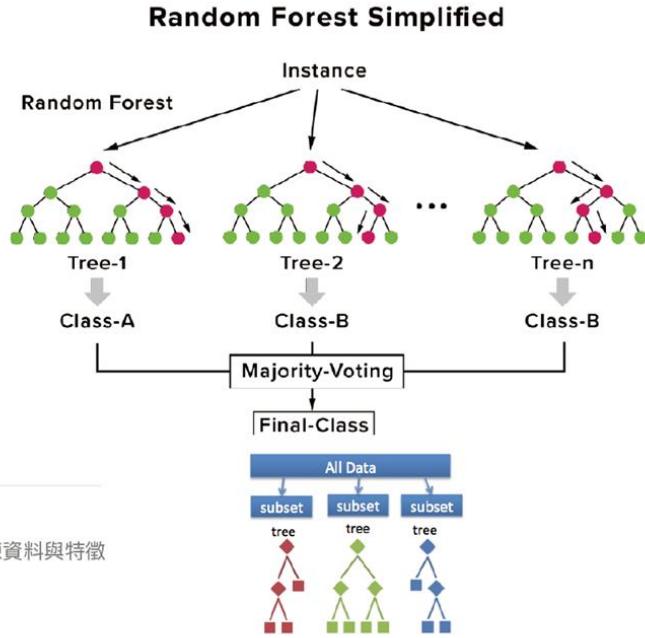
- Criterion: 衡量資料相似程度的 metric
- Max_depth: 樹能生長的最深限制
- Min_samples_split: 至少要多少樣本以上才進行切分
- Min_samples_leaf: 最終的葉子 (節點) 上至少要有多少樣本

```

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(
    criterion = "gini",
    max_depth = None,
    min_samples_split = 2,
    min_samples_leaf = 1,
)
# feature importance
clf.feature_importances_
  
```

集成模型 - 隨機森林 (Random Forest)

集成 (Ensemble) 是將多個模型的結果組合在一起，透過投票或是加權的方式得到最終結果



隨機森林 (Random Forest), 隨機在哪？

- 決策樹生成時，是考慮所有資料與特徵來做切分的
- 而隨機森林的每一棵樹在生成過程中，都是隨機使用一部份的訓練資料與特徵
代表每棵樹都是用隨機的資料訓練而成的

使用 Sklearn 中的隨機森林

- 如同決策樹的使用方式，根據不同問題 import 不同的模型
- 可以看到是從 `sklearn.ensemble` 這裏 import 的，代表隨機森林是個集成模型，透過多棵複雜的決策樹來投票得到結果，緩解原本決策樹容易過擬和的問題，實務上的結果通常都會比決策樹來得好

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor()
  
```

隨機森林的模型超參數

- 同樣是樹的模型，所以像是 `max_depth`, `min_samples_split` 都與決策樹相同
- 可決定要生成數的數量，越多越不容易過擬和，但是運算時間會變長

from `sklearn.ensemble` import `RandomForestClassifier`

```

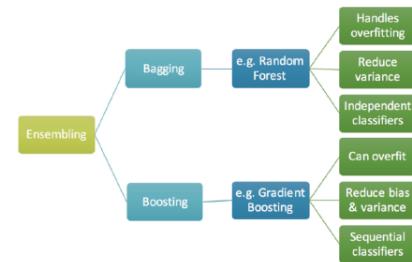
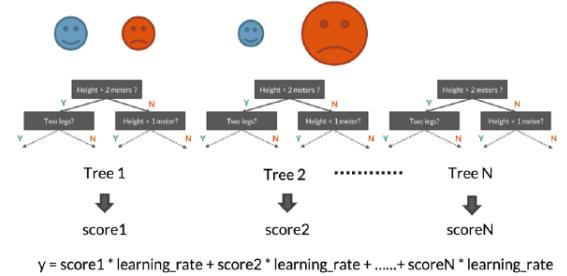
clf = RandomForestClassifier(
    n_estimators=10, #決策樹的數量
    criterion="gini",
    max_features="auto", #如何選取 features
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1
)
  
```

梯度提升機 (Gradient Boosting Machine)

- 隨機森林使用的集成方法稱為 Bagging (Bootstrap aggregating)，用抽樣的資料與 features 生成每一棵樹，最後再取平均
- Boosting 則是另一種集成方法，希望能夠由後面生成的樹，來修正前面樹學不好的地方
- 要怎麼修正前面學錯的地方呢？計算 Gradient！
- 每次生成樹都是要修正前面樹預測的錯誤，並乘上 learning rate 讓後面的樹能有更多學習的空間

Bagging 與 Boosting 的差別

- Bagging 是透過抽樣 (sampling) 的方式來生成每一棵樹，樹與樹之間是獨立生成的
- Boosting 是透過序列 (additive) 的方式來生成每一顆樹，每棵樹都會與前面的樹關聯，因為後面的樹要能夠修正



使用 Sklearn 中的梯度提升機

- 可以看到如同隨機森林，我們一樣從 sklearn.ensemble 這裏 import 進來，代表梯度提升機同樣是個集成模型，透過多棵決策樹依序生成來得到結果，緩解原本決策樹容易過擬和的問題，實務上的結果通常也會比決策樹來得好

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
clf = GradientBoostingClassifier()
```

使用 Sklearn 中的梯度提升機

- 同樣是樹的模型，所以像是 max_depth, min_samples_split 都與決策樹相同
- 可決定要生成數的數量，越多越不容易過擬和，但是運算時間會變長

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
clf = GradientBoostingClassifier()
loss="deviance", #Loss 的選擇，若改為 exponential 則會變成 Adaboosting 演算法，概念相同但實作稍微不同
```

```
learning_rate=0.1, #每棵樹對最終結果的影響，應與 n_estimators 成反比
n_estimators=100 #決策樹的數量
```

)

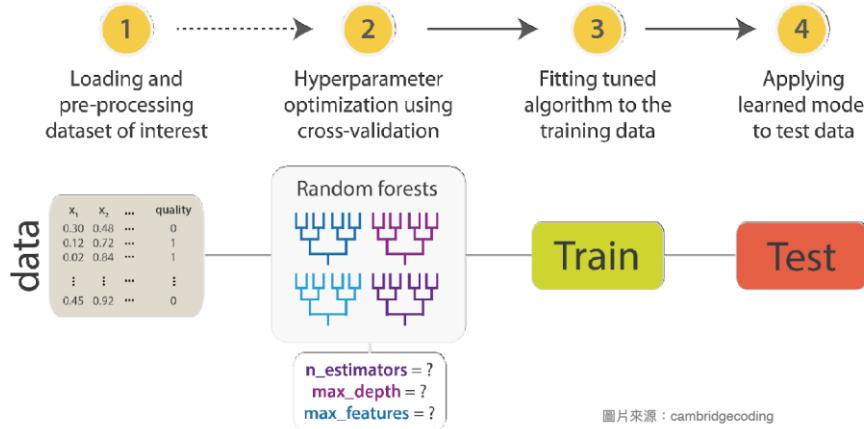
#補充 Day47 <https://medium.com/rants-on-machine-learning/smarter-parameter-sweeps-or-why-grid-search-is-plain-stupid-c17d97a0e881>
超參數調整方法

- 窮舉法 (Grid Search)：直接指定超參數的組合範圍，每一組參數都訓練完成，再根據驗證集 (validation) 的結果選擇最佳參數
- 隨機搜尋 (Random Search)：指定超參數的範圍，用均勻分布進行參數抽樣，用抽到的參數進行訓練，再根據驗證集的結果選擇最佳參數
- 隨機搜尋通常都能獲得更佳的結果，詳見此

3-3 超參數 (Hyper-parameter) 調整與優化

複習講義 Day047_Hyper-parameter 超參數調整與優化，並總複習期中考題目。

機器學習模型訓練步驟



圖片來源：cambridgecoding

機器學習模型中的超參數

- 之前接觸到的所有模型都有超參數需要設置
 - LASSO, Ridge: α 的大小
 - 決策樹：樹的深度、節點最小樣本數
 - 隨機森林：樹的數量
- 這些超參數都會影響模型訓練的結果，建議先使用預設值，再慢慢進行調整
- 超參數會影響結果，但提升的效果有限，資料清理與特徵工程才能最有效的提升準確率，調整參數只是一個加分的工具。

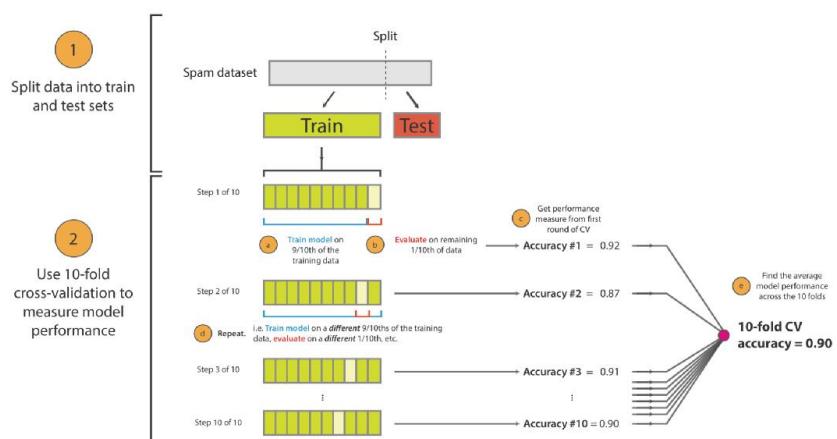
超參數調整方法

- 窮舉法 (Grid Search)：直接指定超參數的組合範圍，每一組參數都訓練完成，再根據驗證集 (validation) 的結果選擇最佳參數
- 隨機搜尋 (Random Search)：指定超參數的範圍，用均勻分布進行參數抽樣，用抽到的參數進行訓練，再根據驗證集的結果選擇最佳參數
- 隨機搜尋通常都能獲得更佳的結果，[詳見此](#)

正確的超參數調整步驟

- 若持續使用同一份驗證集 (validation) 來調參，可能讓模型的參數過於擬合該驗證集，正確的步驟是使用 Cross-validation 確保模型泛化性

- 先將資料切分為訓練/測試集，測試集保留不使用
- 將剛切分好的訓練集，再使用 Cross-validation 切分 K 份訓練/驗證集
- 用 grid/random search 的超參數進行訓練與評估
- 選出最佳的參數，用該參數與全部訓練集建模
- 最後使用測試集評估結果



實戰練習 - Enron Fraud Dataset 安隆公司詐欺案資料集

安隆公司曾是一間能源公司，2001 年破產前是世界上最大的電力、天然氣及電信公司之一。擁有上千億資產的公司於 2002 年竟然在短短幾周內宣告破產，才揭露其財報在多年以來均是造假的醜聞。在本資料集中你將會扮演偵探的角色，透過高層經理人內部的 mail 來往的情報以及薪資、股票等財務特徵，訓練出一個機器學習模型來幫忙你找到可疑的詐欺犯罪者是誰！我們已經先幫你找到幾位犯罪者 (Person-of-Interest, poi) 與清白的員工，請利用這些訓練資料來訓練屬於自己的詐欺犯機器學習模型吧！

<https://www.kaggle.com/c/ml100marathon>

特徵說明

有關財務的特徵: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (單位皆為美元)。更詳細的特徵說明請參考 [enron61702insiderpay.pdf](#) 的最後一頁
[工資, 延期付款, 總付款, 貸款高級, 獎金, 遲延受限股票, 遲延收入, 總股票價值, 費用, 已行使的股票期權, 其他, 長期激勵, 限制性股票, 董事費]

有關 email 的特徵: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (除了 email_address, 其餘皆為次數)
[寄出郵件, 電子郵件地址, 從 poi 紿給此人, 收到郵件, 從此人給 poi, 與 poi 共享的收據]

嫌疑人的標記，也就是我們常用的 **y**。POI label: [‘poi’] (boolean, represented as integer)

Code 6: Day049_EnronFraudDataset+hyper_parameter.ipynb

DataFile: insiderpayData

```
# py-xgboost in anaconda : https://anaconda.org/anaconda/py-xgboost
```

```
!python -m pip install xgboost
```

```
Collecting xgboost
  Downloading
```

```
https://files.pythonhosted.org/packages/5e/49/b95c037b717b4ceadc76b6e164603471225c27052d1611d5a2e832757945/xgboost-0.90-py2.py3-none-win\_amd64.whl (18.3MB)
```

```
Requirement already satisfied: numpy in d:\administrator\anaconda3\lib\site-packages (from xgboost) (1.16.5)
```

```
Requirement already satisfied: scipy in d:\administrator\anaconda3\lib\site-packages (from xgboost) (1.3.1)
```

```
Installing collected packages: xgboost
```

```
Successfully installed xgboost-0.90
```

```
'D:\\Administrator\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
'C:\\Users\\Administrator\\.ipython']
```

```
# 載入套件
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
import xgboost as xgb
import seaborn as sns
```

```
# 汇入資料
```

```
train_data = pd.read_csv("train_data.csv")
test_data = pd.read_csv("test_features.csv")
```

```

print(test_data.shape)
test_data.columns
(33, 21)
Index(['name', 'bonus', 'deferral_payments', 'deferred_income',
       'director_fees', 'email_address', 'exercised_stock_options', 'expenses',
       'from_messages', 'from_poi_to_this_person', 'from_this_person_to_poi',
       'loan_advances', 'long_term_incentive', 'other', 'restricted_stock',
       'restricted_stock_deferred', 'salary', 'shared_receipt_with_poi',
       'to_messages', 'total_payments', 'total_stock_value'],
      dtype='object')

```

```

train_data.head()
print(train_data.shape)
(113, 22)

```

```

# 觀察資料後發現一個看起來不像人名，另一個則是全都是 NAN
train_data = train_data[(train_data['name'] != "THE TRAVEL AGENCY IN THE PARK") &
                      (train_data['name'] != "LOCKHART EUGENE E")]

```

```

train_data.shape
(111, 22)

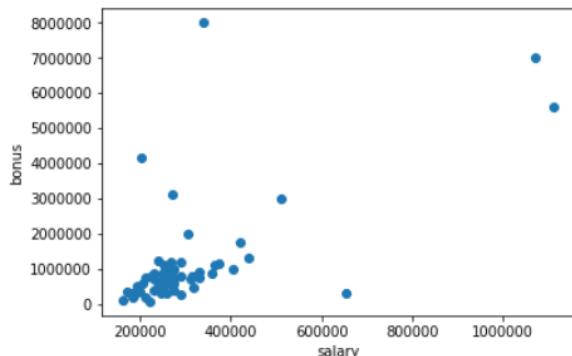
```

Scatter plot for salary & bonus, 看起來左上跟右下有兩個 outlier

```

plt.scatter(train_data['salary'], train_data['bonus'])
plt.xlabel("salary")
plt.ylabel("bonus")
plt.show()

```



檢視 outlier

```

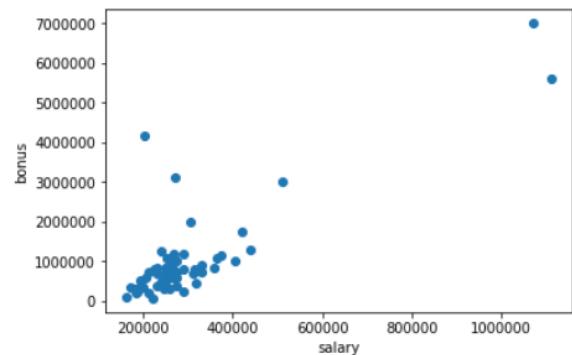
train_data[(train_data['bonus'] == 8000000)]
train_data[(train_data['bonus'] < 1000000) & (train_data['salary'] > 600000)]
# 刪除 outlier
train_data = train_data[(train_data['name'] != "PICKERING MARK R") & (train_data['name'] != "LAVORATO JOHN J")]

```

```

plt.scatter(train_data['salary'], train_data['bonus'])
plt.xlabel("salary")
plt.ylabel("bonus")
plt.show()

```



```

train_data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109 entries, 0 to 112
Data columns (total 22 columns):
name          109 non-null object
bonus          59 non-null float64
deferral_payments 28 non-null float64
deferred_income   34 non-null float64
director_fees    13 non-null float64
email_address    81 non-null object
exercised_stock_options 79 non-null float64
expenses         71 non-null float64
from_messages     63 non-null float64
from_poi_to_this_person 63 non-null float64
from_this_person_to_poi 63 non-null float64
loan_advances     1 non-null float64
long_term_incentive 48 non-null float64
other            67 non-null float64
poi              109 non-null bool
restricted_stock   81 non-null float64
restricted_stock_deferred 10 non-null float64
salary            71 non-null float64
shared_receipt_with_poi 63 non-null float64
to_messages        63 non-null float64
total_payments     93 non-null float64
total_stock_value   96 non-null float64
dtypes: bool(1), float64(19), object(2)
memory usage: 18.8+ KB

#排除 "name", "email_address" 欄位
allFeatures = ['poi','salary','bonus','exercised_stock_options','restricted_stock', 'shared_receipt_with_poi',
               'total_payments', 'expenses', 'total_stock_value', 'deferred_income', 'long_term_incentive',
               'to_messages', 'from_messages', 'from_poi_to_this_person', 'from_this_person_to_poi', 'other']

train_data = train_data[allFeatures]
test_data = test_data[allFeatures[1:]]

correlationDataframe = train_data.corr()

# Drawing a heatmap with the numeric values in each cell
fig1, ax = plt.subplots(figsize=(14,10))
fig1.subplots_adjust(top=.945)
plt.suptitle('Features correlation from the Enron POI dataset', fontsize=14, fontweight='bold')

cbar_kws = {'orientation':'vertical', 'pad':0.025, 'aspect':70}
sns.heatmap(correlationDataframe, annot=True, fmt='.2f', linewidths=.9, ax=ax, cbar_kws=cbar_kws)

<matplotlib.axes._subplots.AxesSubplot at 0x14703708>

```

Features correlation from the Enron POI dataset

	poi	salary	bonus	exercised_stock_options	restricted_stock	shared_receipt_with_poi	total_payments	expenses	total_stock_value	deferred_income	long_term_incentive	to_messages	from_messages	from_poi_to_this_person	from_this_person_to_poi	other
poi	1.00	0.40	0.34	0.59	0.26	0.14	0.27	0.13	0.44	-0.22	0.24	-0.01	-0.12	0.20	-0.09	0.22
salary	0.40	1.00	0.83	0.68	0.58	0.32	0.64	0.14	0.68	-0.15	0.51	0.24	0.02	0.17	0.05	0.50
bonus	0.34	0.83	1.00	0.72	0.54	0.37	0.70	0.03	0.71	-0.33	0.62	0.24	-0.02	0.30	0.05	0.59
exercised_stock_options	0.59	0.68	0.72	1.00	0.70	0.17	0.60	0.03	0.96	-0.31	0.51	0.07	-0.08	0.17	-0.05	0.57
restricted_stock	0.26	0.58	0.54	0.70	1.00	0.25	0.63	0.07	0.86	-0.07	0.69	0.19	-0.01	0.13	0.06	0.70
shared_receipt_with_poi	0.14	0.32	0.37	0.17	0.25	1.00	0.18	0.33	0.18	-0.14	0.07	0.85	0.24	0.63	0.52	0.05
total_payments	0.27	0.64	0.70	0.60	0.63	0.18	1.00	0.11	0.67	0.00	0.52	0.12	-0.04	0.13	-0.03	0.92
expenses	0.13	0.14	0.03	0.03	0.07	0.33	0.11	1.00	0.07	0.12	-0.13	0.23	0.12	-0.09	-0.01	0.05
total_stock_value	0.44	0.68	0.71	0.96	0.86	0.18	0.67	0.07	1.00	-0.26	0.50	0.11	-0.06	0.13	-0.02	0.67
deferred_income	-0.22	-0.15	-0.33	-0.31	-0.07	-0.14	0.00	0.12	-0.26	1.00	-0.16	-0.18	0.49	0.05	-0.24	0.06
long_term_incentive	0.24	0.51	0.62	0.51	0.69	0.07	0.52	-0.13	0.50	-0.16	1.00	0.04	-0.13	0.03	-0.14	0.60
to_messages	-0.01	0.24	0.24	0.07	0.19	0.85	0.12	0.23	0.11	-0.18	0.04	1.00	0.48	0.48	0.69	0.02
from_messages	-0.12	0.02	-0.02	-0.08	-0.01	0.24	-0.04	0.12	-0.06	-0.49	-0.13	0.48	1.00	0.17	0.69	-0.09
from_poi_to_this_person	0.20	0.17	0.30	0.17	0.13	0.63	0.13	-0.09	0.13	0.05	0.03	0.48	0.17	1.00	0.40	0.03
from_this_person_to_poi	-0.09	0.05	0.05	-0.05	0.06	0.52	-0.03	-0.01	-0.02	-0.24	-0.14	0.69	0.69	0.40	1.00	-0.10
other	0.22	0.50	0.59	0.57	0.70	0.05	0.92	0.05	0.67	0.06	0.60	0.02	-0.09	0.03	-0.10	1.00

新增傳送/收到 poi 信件的比例 features

```
train_data['portion_from_poi'] = train_data['from_poi_to_this_person'] / train_data['from_messages']
train_data['portion_to_poi'] = train_data['from_this_person_to_poi'] / train_data['to_messages']
```

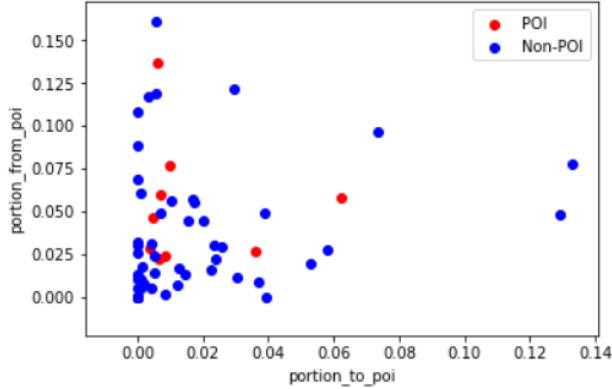
```
train_data['portion_from_poi'] = train_data['portion_from_poi'].fillna(0)
train_data['portion_to_poi'] = train_data['portion_to_poi'].fillna(0)
```

```
test_data['portion_from_poi'] = test_data['from_poi_to_this_person'] / test_data['from_messages']
test_data['portion_to_poi'] = test_data['from_this_person_to_poi'] / test_data['to_messages']
```

```
test_data['portion_from_poi'] = test_data['portion_from_poi'].fillna(0)
test_data['portion_to_poi'] = test_data['portion_to_poi'].fillna(0)
```

```
train_data['ratio_exercised_restricted'] = train_data['exercised_stock_options'] / train_data['restricted_stock']
test_data['ratio_exercised_restricted'] = test_data['exercised_stock_options'] / test_data['restricted_stock']
```

```
is_poi = (True, False)
colors = ('r', "b")
groups = ("POI", "Non-POI")
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
for poi, color, group in zip(is_poi, colors, groups):
    data_plot = train_data[train_data['poi'] == poi]
    ax.scatter(data_plot['portion_to_poi'], data_plot['portion_from_poi'], c=color, label=group)
plt.xlabel("portion_to_poi")
plt.ylabel("portion_from_poi")
plt.legend(loc=1)
plt.show()
```



建立 X_train, y_train 資料

```
#x_train = train_data.drop(["name", 'poi', "email_address"], axis=1)
x_train = train_data.drop(['poi'], axis=1)
y_train = train_data['poi']
x_train = x_train.fillna(0) #用0填補遺失值
```

訓練模型

```
##未調整最佳參數
# 建立模型
clf = xgb.XGBClassifier()
clf.fit(x_train, y_train)

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bytree=1,
              colsample_bynode=1, colsample_bylevel=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
```

```
nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=None, subsample=1, verbosity=1)
```

"未" 調整最佳參數 的結果

```
y_pred_prob = clf.predict_proba(x_train)[:,1]
y_pred = clf.predict(x_train)

print("TRAINING AUC: ", roc_auc_score(y_train, y_pred_prob))
print("TRAINING F1-Score: ", metrics.f1_score(y_train, y_pred)) # 使用 F1-Score 評估
print("TRAINING Precision: ", metrics.precision_score(y_train, y_pred))# 使用 Precision 評估
print("TRAINING Recall: ", metrics.recall_score(y_train, y_pred))# 使用 recall 評估
```

```
TRAINING AUC: 1.0
```

```
TRAINING F1-Score: 1.0
```

```
TRAINING Precision: 1.0
```

```
TRAINING Recall: 1.0
```

調整最佳參數

```
n_estimators = [600, 800, 1000]
max_depth = [2, 4, 6]
scale_pos_weight = [0.4, 1, 1.5]
param_grid = dict(n_estimators=n_estimators, max_depth=max_depth, scale_pos_weight=scale_pos_weight)
```

```
## 建立搜尋物件，放入模型及參數組合字典 (n_jobs=-1 會使用全部 cpu 平行運算)
```

```
grid_search = GridSearchCV(clf, param_grid, n_jobs=-1, verbose=1 )
```

開始搜尋最佳參數

```
grid_result = grid_search.fit(x_train, y_train)
```

```
# 預設會跑 3-fold cross-validation，總共 27 種參數組合，總共要 train 81 次模型
```

```
D:\Administrator\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
    warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done 81 out of 81 | elapsed:    3.6s finished
D:\Administrator\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814:
DeprecationWarning: The default of the `iid` parameter will change from True to False in version
0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
    DeprecationWarning)
```

```
grid_result.best_params_
```

```
{'max_depth': 2, 'n_estimators': 600, 'scale_pos_weight': 0.4}
```

使用最佳參數重新建立模型

```
clf_bestparam = xgb.XGBClassifier(max_depth=grid_result.best_params_['max_depth'],
                                    n_estimators=grid_result.best_params_['n_estimators'],
                                    scale_pos_weight=grid_result.best_params_['scale_pos_weight'])
```

訓練模型

```
clf_bestparam.fit(x_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=2,
              min_child_weight=1, missing=None, n_estimators=600, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=0.4, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```

##已調整最佳參數的結果
y_pred_prob = clf_bestparam.predict_proba(x_train)[:,1]
y_pred = clf_bestparam.predict(x_train)

print("TRAINING AUC: ", roc_auc_score(y_train, y_pred_prob))
print("TRAINING F1-Score: ", metrics.f1_score(y_train, y_pred)) # 使用F1-Score評估
print("TRAINING Precision: ", metrics.precision_score(y_train, y_pred))# 使用Precision評估
print("TRAINING Recall: ", metrics.recall_score(y_train, y_pred))# 使用recall評估
```

```

TRAINING AUC: 1.0
TRAINING F1-Score: 0.9600000000000001 ↓
TRAINING Precision: 1.0
TRAINING Recall: 0.9230769230769231 ↓
```

檢察 test 答案

```

#匯入正確答案對解答
y_test = pd.read_csv(data_path + "test_label.csv")['poi']
```

```

#選取drop變項
#x_test = test_data.drop(['name','email_address'], axis=1)
x_test = test_data.fillna(0)
print(x_test.shape)
```

(33, 18)

```

##未調整最佳參數的結果
y_pred_prob = clf.predict_proba(x_test)[:,1]
y_pred = clf.predict(x_test)

print("TESTING AUC: ", roc_auc_score(y_test, y_pred_prob))
print("TESTING F1-Score: ", metrics.f1_score(y_test, y_pred)) # 使用F1-Score評估
print("TESTING Precision: ", metrics.precision_score(y_test, y_pred))# 使用Precision評估
print("TESTING Recall: ", metrics.recall_score(y_test, y_pred))# 使用recall評估
```

```

TESTING AUC: 0.8785714285714286
TESTING F1-Score: 0.5714285714285714
TESTING Precision: 0.4444444444444444
TESTING Recall: 0.8
```

##已調整最佳參數的結果

```

y_pred_prob = clf_bestparam.predict_proba(x_test)[:,1]
y_pred = clf_bestparam.predict(x_test)

print("TESTING AUC: ", roc_auc_score(y_test, y_pred_prob))
print("TESTING F1-Score: ", metrics.f1_score(y_test, y_pred)) # 使用F1-Score評估
print("TESTING Precision: ", metrics.precision_score(y_test, y_pred))# 使用Precision評估
print("TESTING Recall: ", metrics.recall_score(y_test, y_pred))# 使用recall評估
```

```

TESTING AUC: 0.8714285714285713
TESTING F1-Score: 0.6666666666666666 ↑
TESTING Precision: 0.5714285714285714 ↑
TESTING Recall: 0.8
```

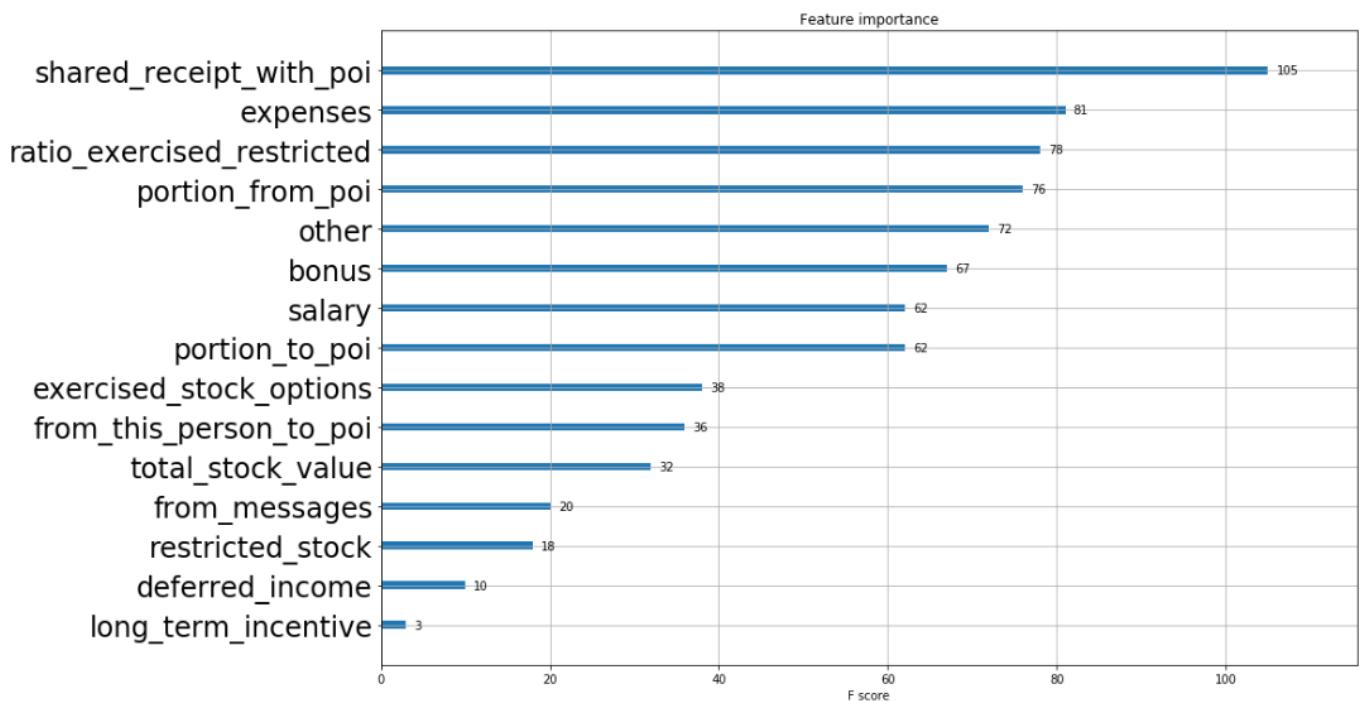
```

#儲存分析結果，上傳Kaggle，求得真正的預測能力
#myDataFrame = pd.DataFrame(y_pred_prob,columns=['poi'],dtype='float64')
#index_column = np.arange(1,33)
#myDataFrame['name'] = test['name']
#myDataFrame.to_csv('Submission_.csv', index=False)
```

```

fig, ax = plt.subplots(figsize=(15, 10))
xgb.plot_importance(clf_bestparam, ax=ax, ylabel=None)
plt.yticks(size=24)
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 <a list of 16 Text yticklabel objects>)

```



[補充] 認識模塊組

<https://ithelp.ithome.com.tw/articles/10185280>

什麼是模組？分二種，一個是 **標準庫**，就是不需要透過額外的安裝就有的模組，另一個叫 **第三方庫**，需要另外安裝才能使用的模組。

本次練習所使用的 `xgboost`，是需要另外安裝的模組。這些模組在哪裡呢？我們可以使用 `sys` 模組查找，這是 Python 標準庫中自帶的一個模組。

利用 `sys.path` 可以查詢模組所在目錄的目錄名列表

```
import sys
sys.path
[....]
'D:\\Administrator\\Anaconda3\\python37.zip',
'D:\\Administrator\\Anaconda3\\DLLs',
'D:\\Administrator\\Anaconda3\\lib',
'D:\\Administrator\\Anaconda3',
 '',
'D:\\Administrator\\Anaconda3\\lib\\site-packages',
'D:\\Administrator\\Anaconda3\\lib\\site-packages\\win32',
'D:\\Administrator\\Anaconda3\\lib\\site-packages\\win32\\lib',
'D:\\Administrator\\Anaconda3\\lib\\site-packages\\Pythonwin',
'D:\\Administrator\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
'C:\\Users\\Administrator\\.ipython']
```

其中書出應該會有類似 `/usr/local/lib/python3.5/site-packages`，這個是當我們自己有編寫模組時，可以放在這路徑底下，若找不到 `site-packages`，就改找 `dist-packages`，也可以把自己寫的模塊放在此路徑下，讓全局可調用。這每個指令的關係雲圖很複雜，中文圖文說明可以參考《撥開 Python, pip, site-packages 的藍色蜘蛛網》<https://medium.com/@will.wang/%E6%92%A5%E9%96%8B-python-pip-site-packages-%E7%9A%84%E8%97%8D%E8%89%B2%E8%9C%98%E8%9B%9B%E7%B6%B2-90e398bb3785>

好惹，這麼多實在看不懂，簡單總結，為確保正在運行 `pip` 與當前 Python 內核相關聯的版本，以便可以在當前筆記本中使用已安裝的軟件包那就使用

```
!python -m pip install <package>
```

而不是

```
!pip install <package>
```

[補充] XGBoost 參數調優

<https://www.itread01.com/articles/1476146171.html>

本次練習所使用的 `xgboost`, **XGBoost (eXtreme Gradient Boosting)** 是 Gradient Boosting 算法的一個優化的版本。詳細的可以參考網址文章。這裡最後提供了 Feature Importance，python 的 XGBoost 模組有一個 `sklearn` 包，`XGBClassifier`，`xgboost` 的 `sklearn` 包沒有 “`feature_importance`” 這個量度，但是可以另外使用 `get_fscore()` 函數，可以有相同的功能。

“`feature_importance`” 是一個簡單地總結在決策樹中出現頻率的計數，就是每個功能被分割多少次的指標。它類似於 R 版本中的 Frequency metric。<https://cran.r-project.org/web/packages/xgboost/xgboost.pdf>

#3.3.1. <認識模塊><https://ithelp.ithome.com.tw/articles/10185280>

#3.3.2. < Installing Python Packages from a Jupyter Notebook > <https://jakevdp.github.io/blog/2017/12/05/installing-python-packages-from-jupyter/>

#3.3.3. < XGBoost 參數調優完全指南> <https://www.itread01.com/articles/1476146171.html>

[補充] 評估指標

Code 7: Day036_evaluation_metricsReview.ipynb

評估指標

設定各項指標來評估模型預測的準確性，最常見的為準確率

Accuracy = 正確分類樣本數/總樣本數

不同評估指標有不同的評估準則與面向，衡量的重點有所不同

■ 回歸問題可以透過 R-square 很快了解預測的準確程度；

- 觀察「預測值」(Prediction) 與「實際值」(Ground truth) 的差距
 - MAE, Mean Absolute Error, 範圍: $[-\infty, \infty]$
 - MSE, Mean Square Error, 範圍: $[-\infty, \infty]$
 - R-square, 範圍: $[0, 1]$

■ 分類問題若為二分類 (binary classification)，通常使用 AUC 評估。

- 但如果希望哪一類別不要分錯，則可使用 F1-Score，觀察 Recall 值或是 Precision 值。

- 觀察「預測值」(prediction) 與「實際值」(Ground truth) 的正確程度
 - ◆ AUC, Area Under Curve, 範圍: $[0, 1]$
 - ◆ F1 - Score (Precision, Recall), 範圍: $[0, 1]$

分類問題中，我們有時會對某一類別的準確率特別有興趣。

例如瑕疵/正常樣本分類，我們希望任何瑕疵樣本都不能被漏掉。

Precision · Recall 則是針對某類別進行評估

- Precision: 模型判定瑕疵，樣本確實為瑕疵的比例
- Recall: 模型判定的瑕疵，佔樣本所有瑕疵的比例 (以瑕疵檢測為例，若為 recall=1 則代表所有瑕疵都被找到)

F1-Score 則是 Precision, Recall 的調和平均數

■ 若是多分類問題，則可使用 top-k accuracy，k 代表模型預測前 k 個類別有包含正確類別即為正確

(ImageNet 競賽通常都是比 Top-5Accuracy)

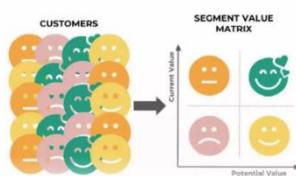
3-4 非監督式學習及 Sklearn

非監督學習允許我們在對結果無法預知時接近問題。非監督學習演算法只基於輸入資料找出模式。當我們無法確定尋找內容，或無標記(y)資料時，通常會用這個演算法，幫助我們了解資料模式。在不清楚資料特性、問題定義、沒有標記的情況下，非監督式學習技術可以幫助我們理清資料脈絡。特徵數太龐大的情況下，非監督式學習可以幫助概念抽象化，用更簡潔的特徵描述資料。非監督式學習以聚類算法及降低維度算法為主，本課程也以這兩門技術進行探究。

- 聚類分析：尋找資料的隱藏模式
 - 降低維度：特徵數太大且特徵間相關性高，以此方式縮減特徵維度
 - 其他：關聯聯法則（購物籃分析）、異常偵測、探索性資料分析等

應用案例（一）：客戶分群

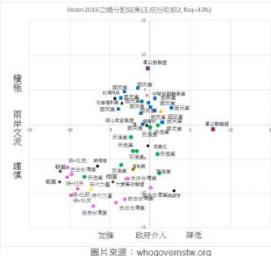
在資料沒有任何標記，或是問題還沒定義清楚前，可用分群的方式幫助理清資料特性。



圖片來源：blog.csdn

應用案例（二）：特徵抽象化

特徵數太多難於理解及呈現的情況下，藉由抽象化的技術幫助降低資料維度，同時不失去原有的資訊，組合成新的特徵。



應用案例（三）：購物籃分析

資料探勘的經典案例，適用於線下或線上零售的商品組合推薦。



周易索隱 · 二三子問

應用案例（四）：非結構化資料分析

非結構化資料如文字、影像等，可以藉由一些非監督式學習的技術，幫助呈現及描述資料。

- 主題模型 (topic model)

圖片來源：stackoverflow

同前幾章所述，資料結構和分析架構其實是最重要的環節，在認識相關的程式之前，不妨先參考看看網路上整理的一些範例 <<利用集群分析掌握消費者輪廓，Python 實作>> 以商業數據作為討論 [#3.31](#)。

3-4-1 聚類算法 Clustering: K-means

聚類算法用於把族群或資料點分隔成一系列的組合，使得相同 cluster 中的資料點比其他的組更相似。

#3.3.1. <利用集群分析掌握消費者輪廓，Python 實作>

利用集群分析掌握消費者輪廓，
Python 實作（一）

Dennis Dsh Follow Jul 24 · 23 min read



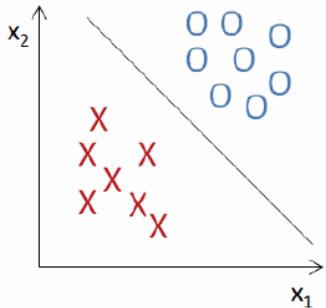
利用機器學習來決定我該將產品賣給誰

論譜

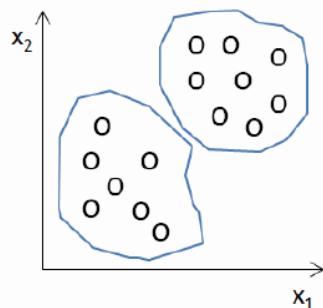
在真實消費場景中，我們通常需要了解TA(Target audience)來制定銷售策略，畢竟企業的資源是有限

Supervised learning vs. clustering

監督式學習目標在於找出決策邊界
(decision boundary)



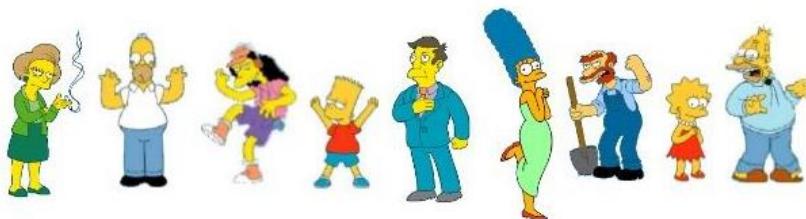
Clustering 目標在於找出資料結構



Why clustering?

在資料還沒有標記、問題還沒定義清楚時，聚類算法可以幫助我們理解資料特性，評估機器學習問題方向等，也是一種呈現資料的方式。

What is a natural grouping among these objects?



Clustering is subjective



Simpson's Family



School Employees



Females



Males

複習 Day055-057 問題不清楚或是資料未有標註的情況下，可以嘗試用分群算法幫助瞭解資料結構，而其中一個方法是運用 K-means 聚類算法幫助分群資料。範例也以輪輪廓分析 (Silhouette analysis)來觀察 K-mean 分群時不同 K 值的比較。資料集與前一小節一樣 Iris，是否能分成三組呢？

Code 8: Day055_K-means Review.ipynb

Code 9: Day056_kmean Review DATA_iris.ipynb

載入套件

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
# Though the following import is not directly being used, it is required
# for 3D projection to work
from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans
from sklearn import datasets

np.random.seed(5)

%matplotlib inline
```

載入 iris 資料集

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

設定 模型 估計參數

```
estimators = [('k_means_iris_8', KMeans(n_clusters=8)),
              ('k_means_iris_3', KMeans(n_clusters=3)),
              ('k_means_iris_bad_init', KMeans(n_clusters=3, n_init=1,
                                              init='random'))]
```

資料建模 並 視覺化 結果

```
fignum = 1
titles = ['8 clusters', '3 clusters', '3 clusters, bad initialization']
for name, est in estimators:
    fig = plt.figure(fignum, figsize=(4, 3))
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

    ## fit data
    est.fit(X)

    labels = est.labels_

    ax.scatter(X[:, 3], X[:, 0], X[:, 2],
               c=labels.astype(np.float), edgecolor='k')

    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel('Petal width')
    ax.set_ylabel('Sepal length')
    ax.set_zlabel('Petal length')
    ax.set_title(titles[fignum - 1])
    ax.dist = 12
    fignum = fignum + 1

# Plot the ground truth
```

```

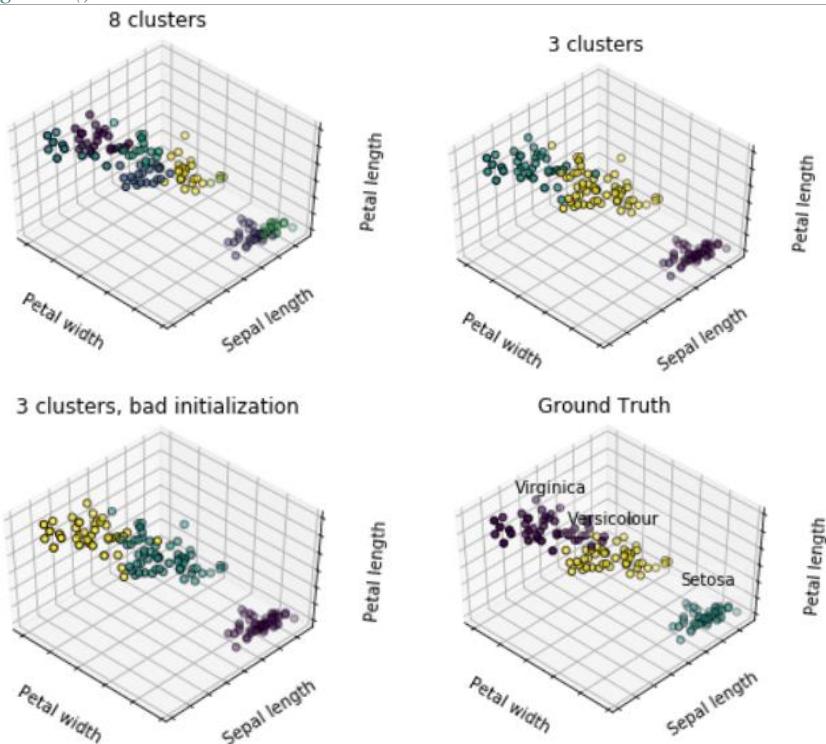
fig = plt.figure(figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

for name, label in [('Setosa', 0),
                    ('Versicolour', 1),
                    ('Virginica', 2)]:
    ax.text3D(X[y == label, 3].mean(),
              X[y == label, 0].mean(),
              X[y == label, 2].mean() + 2, name,
              horizontalalignment='center',
              bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=y, edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('Petal width')
ax.set_ylabel('Sepal length')
ax.set_zlabel('Petal length')
ax.set_title('Ground Truth')
ax.dist = 12

```

```
#fig.show()
```



K-means with Silhouette analysis

我們試著以輪廓分析 (Silhouette analysis) 來觀察 K-mean 分群時不同 K 值的比較

```

# 載入套件
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.metrics import silhouette_samples, silhouette_score

```

```
np.random.seed(5)
```

```
%matplotlib inline
```

```
# 載入 iris 資料集
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
# 設定需要計算的 K 值集合
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
```

```
# 計算並繪製輪廓分析的結果
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_yticks([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters),
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                         0, ith_cluster_silhouette_values,
                         facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
```

```

# Compute the new y_lower for next plot
y_lower = y_upper + 10 # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([]) # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
            c=colors, edgecolor='k')

# Labeling the clusters
centers = clusterer.cluster_centers_
# Draw white circles at cluster centers
ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
            c="white", alpha=1, s=200, edgecolor='k')

for i, c in enumerate(centers):
    ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                s=50, edgecolor='k')

ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Feature space for the 1st feature")
ax2.set_ylabel("Feature space for the 2nd feature")

plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
             "with n_clusters = %d" % n_clusters),
             fontsize=14, fontweight='bold')

plt.show()

```

分析結果解說

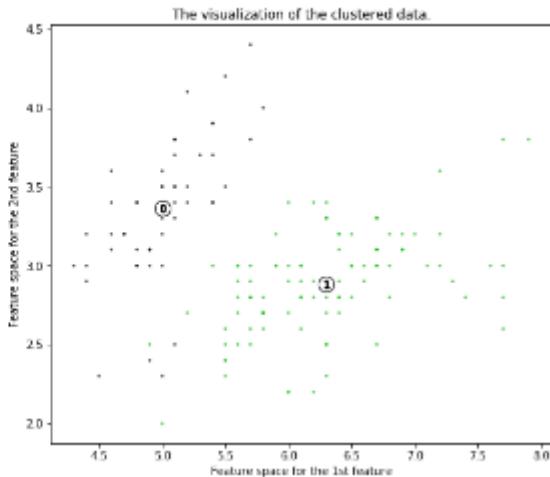
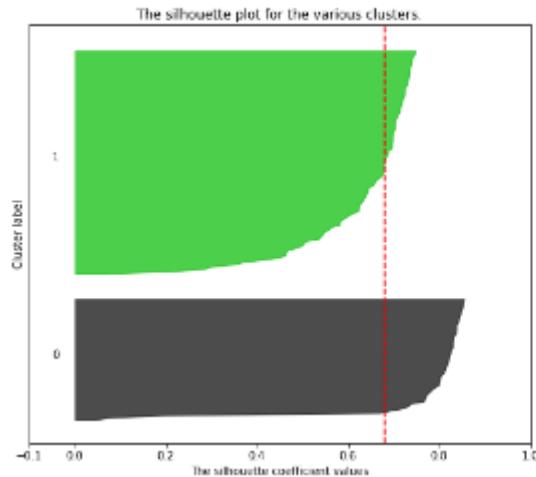
- 觀察輸出值:silhouette_score, 如果是一個適合的分群值，應該要比下一個分群值的分數大很多
- 由結果可以看出：2, 3, 5 都是不錯的分群值(因為比 3, 4, 6 的分數都高很多)，相形之下，4, 6, 7 作為分群的效果就不明顯
-

```

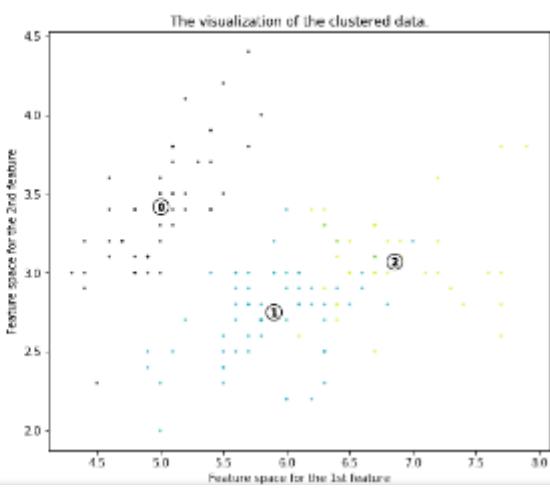
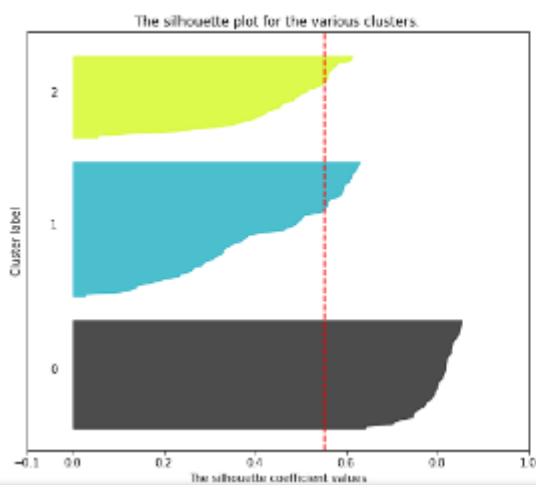
For n_clusters = 2 The average silhouette_score is : 0.6808136202787941
For n_clusters = 3 The average silhouette_score is : 0.5525919445309032
For n_clusters = 4 The average silhouette_score is : 0.49782569008681465
For n_clusters = 5 The average silhouette_score is : 0.4885175508654505
For n_clusters = 6 The average silhouette_score is : 0.3712180503529101
For n_clusters = 7 The average silhouette_score is : 0.3600597995398555
For n_clusters = 8 The average silhouette_score is : 0.36037497069478047

```

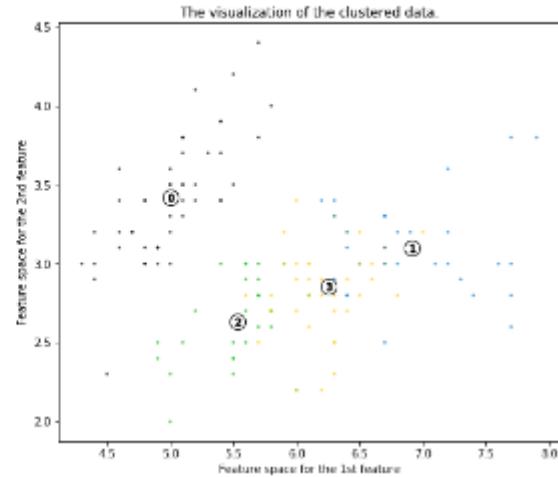
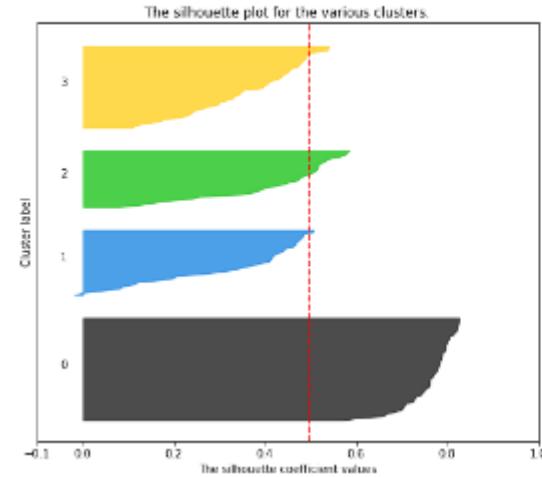
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2



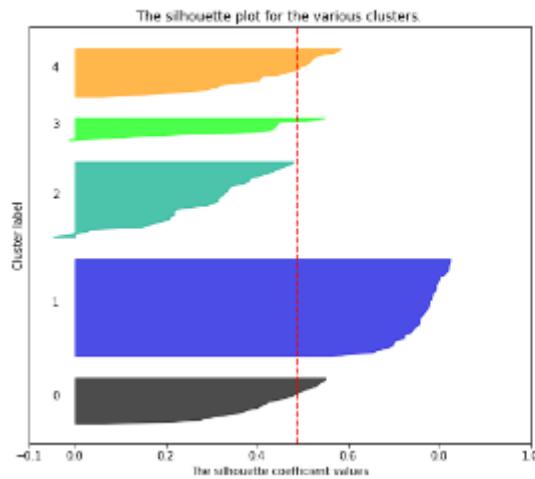
Silhouette analysis for KMeans clustering on sample data with n_clusters = 3



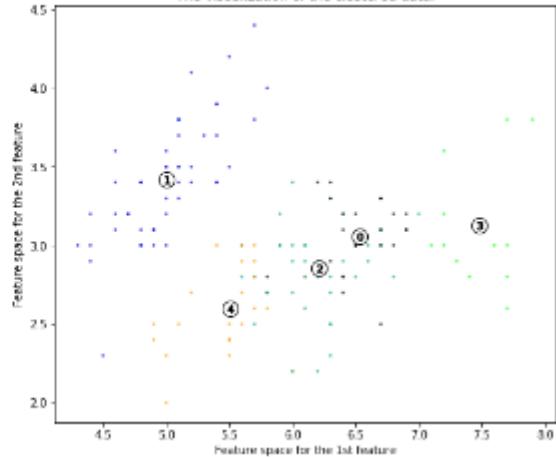
Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



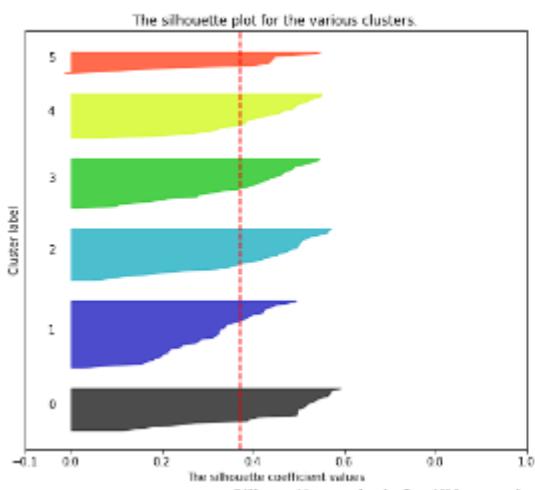
Silhouette analysis for KMeans clustering on sample data with n_clusters = 5



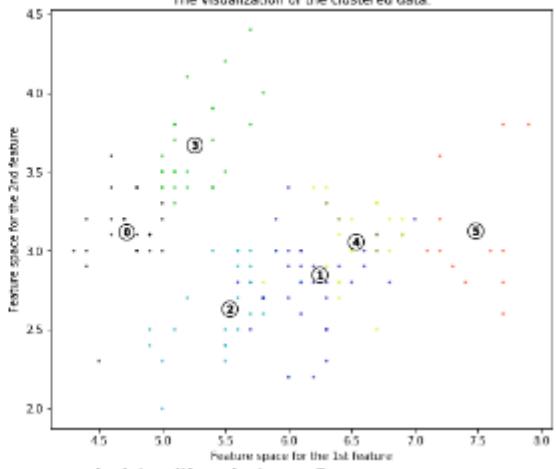
The visualization of the clustered data.



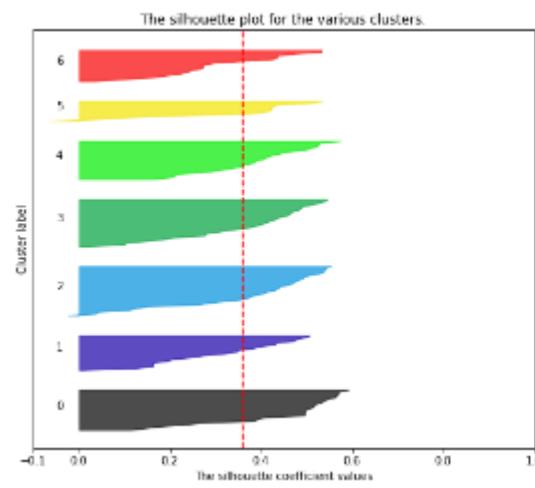
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



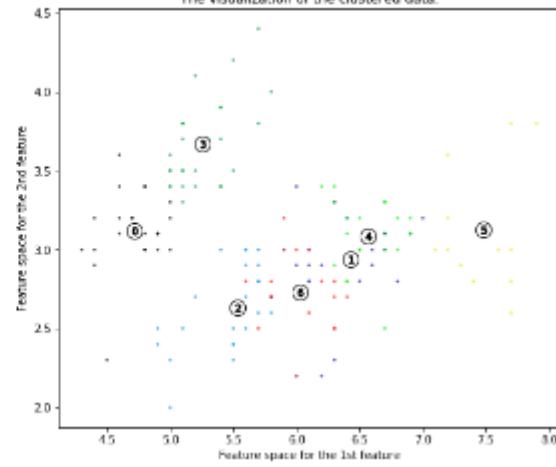
The visualization of the clustered data.



Silhouette analysis for KMeans clustering on sample data with n_clusters = 7



The visualization of the clustered data.



補充：make_blobs 聚類數據生成器

scikit 中的 make_blobs 方法常被用來生成聚類演算法的測試資料，直觀地說，make_blobs 會根據使用者指定的特徵數量、中心點數量、範圍等來生成幾類資料，這些資料可用於測試聚類演算法的效果。

make_blobs 方法的 API 如下：

```
sklearn.datasets.make_blobs(n_samples=100, n_features=2, centers=3, cluster_std=1.0,  
center_box=(-10.0, 10.0), shuffle=True, random_state=None)[source]
```

其中：

n_samples 是待生成的樣本的總數。

n_features 是每個樣本的特徵數。

centers 表示類別數。

cluster_std 表示每個類別的方差，例如我們希望生成 2 類資料，其中一類比另一類具有更大的方差，可以將 cluster_std 設置為[1.0,3.0]。

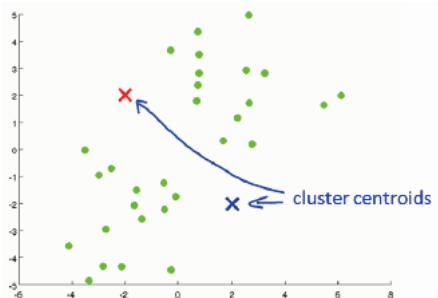
複習

K-means 聚類算法

- 把所有資料點分成 k 個 cluster，使得相同 cluster 中的所有資料點彼此儘量相似，而不同 cluster 的資料點儘量不同。
- 距離測量 (e.g. 歐氏距離) 用於計算資料點的相似度和相異度。每個 cluster 有一個中心點。中心點可理解為最能代表 cluster 的點。

K-means 算法流程

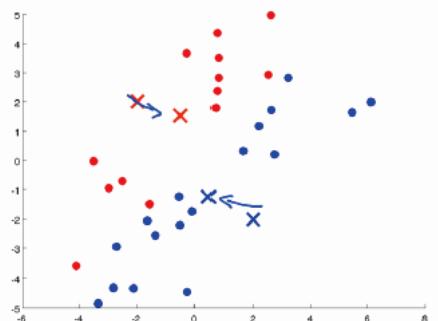
假設下圖是我們的 training set，我們目標是將資料分成 2 群



隨機選取 2 個點，稱為 cluster centroid.

對每一個 training example 根據它距離哪一個 cluster centroid 較近，標記為其中之一 (cluster assignment)

- 然後把 centroid 移到同一群 training examples 的中心點 (update centroid)
- 反覆進行 cluster assignment 及 update centroid, 直到 cluster assignment 不再導致 training example 被 assign 為不同的標記 (算法收斂)

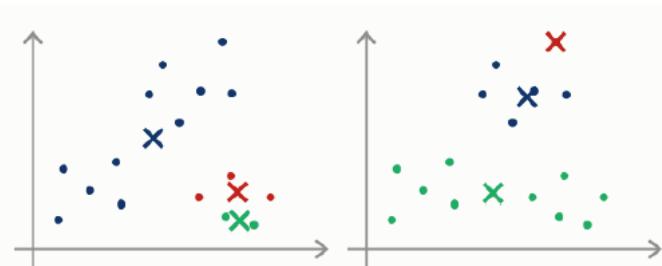


K-means 目標是使總體群內平方誤差最小

$$\sum_{i=0}^n \min_{\mu_j \in C} (\|x_i - \mu_j\|^2)$$

K-means 注意事項

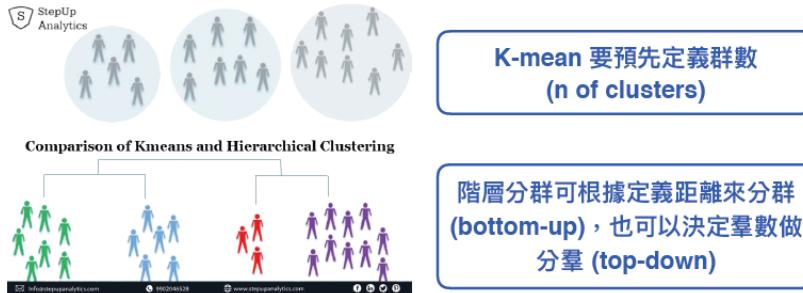
Random initialization: initial 設定的不同，會導致得到不同 clustering 的結果，可能導致 local optima，而非 global optima。



因為沒有預先的標記，對於 cluster 數量多少才是最佳解，沒有標準答案，得靠手動測試觀察。

3-4.2 類算法 Clustering: 階層式分群 Hierarchical Clustering

K-means vs. 階層分群



複習 Day057-58_hierarchical_clustering 階層式分群在無需定義群數的情況下做資料的分群，而後可以用不同的距離定義決定資料群組。分群距離計算方式有 single-link, complete-link, average-link。概念簡單且容易呈現，但不適合用在大資料。範例中另外也將階層式分群套用在 2D 樣板資料上來觀察幾種不同參數的結果有何不同。作業中嘗試改變 Random_state，效果會不一樣。Random_state 一樣時，重複執行，效果也會不一樣。

👉 **Code 10:** Day057_hierarchical_clustering_Review.ipynb

👉 **Code 11:** Day058_hierarchical_clustering_Ans_noisy_moons.ipynb

載入套件

```
import numpy as np
import matplotlib.pyplot as plt
# Though the following import is not directly being used, it is required
# for 3D projection to work
from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import AgglomerativeClustering
from sklearn import datasets

np.random.seed(5)

%matplotlib inline
```

載入 toy example 資料集

```
toy = datasets.make_blobs(centers=3, n_features=4)
X = toy[0]
y = toy[1]
```

設定 模型 估計參數

```
estimators = [('hc_ward', AgglomerativeClustering(n_clusters=3, linkage="ward")),
              ('hc_complete', AgglomerativeClustering(n_clusters=3, linkage="complete")),
              ('hc_average', AgglomerativeClustering(n_clusters=3, linkage="average"))]
```

資料建模 並 視覺化 結果

```
fignum = 1
titles = ['ward', 'complate', 'average']
for name, est in estimators:
    fig = plt.figure(fignum, figsize=(4, 3))
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
```

```

## fit data
est.fit(X)

labels = est.labels_

ax.scatter(X[:, 3], X[:, 0], X[:, 2],
           c=labels.astype(np.float), edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_title(titles[fignum - 1])
ax.dist = 12
fignum = fignum + 1

# Plot the ground truth
fig = plt.figure(fignum, figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

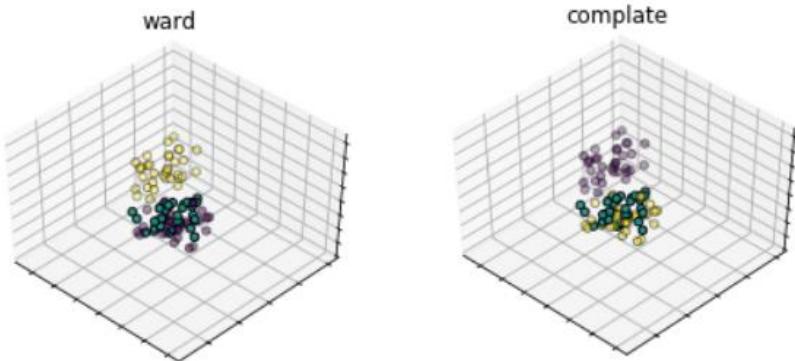
for name, label in [('cls0', 0),
                     ('cls1', 1),
                     ('cls2', 2)]:
    ax.text3D(X[y == label, 3].mean(),
              X[y == label, 0].mean(),
              X[y == label, 2].mean() + 2, name,
              horizontalalignment='center',
              bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))

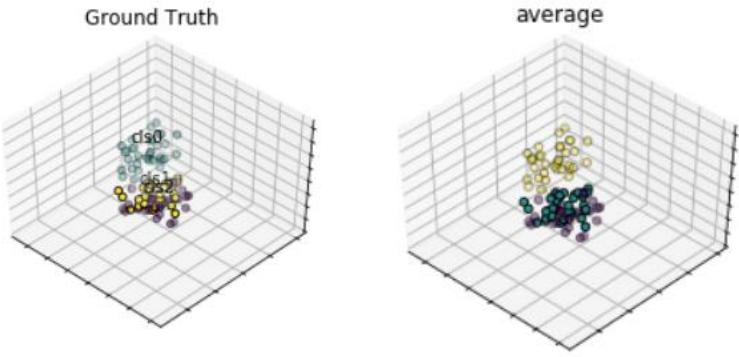
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=y, edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_title('Ground Truth')
ax.dist = 12

#fig.show()

```





Hierarchical clustering with 2D toy datasets

將階層式聚類套用在 2D 樣板資料上，來觀察幾種不同參數的結果有何不同

```
# 載入套件
import time
import warnings

import numpy as np
import matplotlib.pyplot as plt

from sklearn import cluster, datasets
from sklearn.preprocessing import StandardScaler
from itertools import cycle, islice

# 設定 2D 樣板資料
n_samples = 1500
random_state = 100

noisy_circles = datasets.make_circles(n_samples=n_samples, factor=.5, noise=.05)
# noisy_moons = datasets.make_moons(n_samples=n_samples, noise=.05)

# Anisotropically distributed data
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X, transformation)
aniso = (X_aniso, y)

# blobs with varied variances
varied = datasets.make_blobs(n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=random_state)

# 設定群集與繪圖參數
plt.figure(figsize=(10, 8))
plot_num = 1
default_base = {'n_neighbors': 10, 'n_clusters': 3}

datasets = [(noisy_circles, {'n_clusters': 2}), (varied, {'n_neighbors': 2}), (aniso, {'n_neighbors': 2})]
<Figure size 720x576 with 0 Axes>

# 執行各種樣板資料的繪圖迴圈
for i_dataset, (dataset, algo_params) in enumerate(datasets):
    # 複製參數與標準化
    params = default_base.copy()
    params.update(algo_params)
    X, y = dataset
    X = StandardScaler().fit_transform(X)

    # 設定三種不同參數的 Hierarchical clustering
    ward = cluster.AgglomerativeClustering(n_clusters=params['n_clusters'], linkage="ward")
    complete = cluster.AgglomerativeClustering(n_clusters=params['n_clusters'], linkage="complete")
```

```

average = cluster.AgglomerativeClustering(n_clusters=params['n_clusters'], linkage="average")
clustering_algorithms = (
    ('Average Linkage', average),
    ('Complete Linkage', complete),
    ('Ward Linkage', ward))

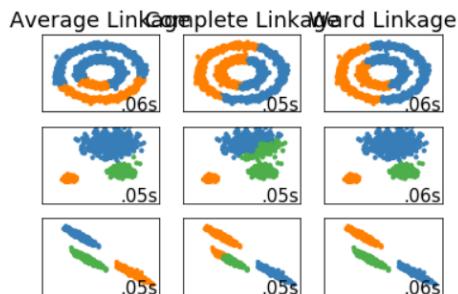
# 繪製三種圖形
for name, algorithm in clustering_algorithms:
    # t0 / t1 : 紀錄時間差
    t0 = time.time()
    algorithm.fit(X)
    t1 = time.time()
    if hasattr(algorithm, 'labels_'):
        y_pred = algorithm.labels_.astype(np.int)
    else:
        y_pred = algorithm.predict(X)

    plt.subplot(len(datasets), len(clustering_algorithms), plot_num)
    if i_dataset == 0:
        plt.title(name, size=18)

    colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',
                                         '#f781bf', '#a65628', '#984ea3',
                                         '#999999', '#e41a1c', '#dede00']), int(max(y_pred) + 1))))
    plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_pred])

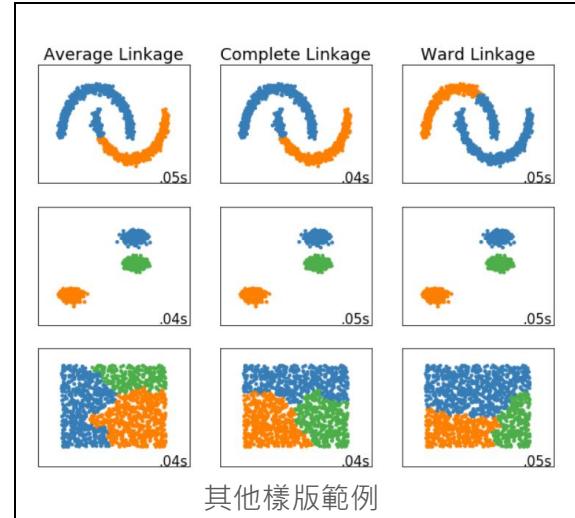
    plt.xlim(-2.5, 2.5)
    plt.ylim(-2.5, 2.5)
    plt.xticks(())
    plt.yticks(())
    plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'), transform=plt.gca().transAxes, size=15,
            horizontalalignment='right')
    plot_num += 1
plt.show()

```



觀察結果

- 同心圓：三者效果中 Complete 的效果似乎是最差
- 三群：Average 與 Ward 效果都還不錯
- 斜三群：三者效果也是以 Complete 較差

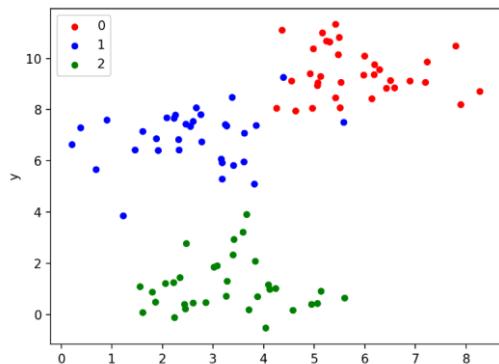


補充：分類和回歸演算法生成測試問題的例子：

分類測試問題: 分類是將標籤分配給資料的問題。在本節中，我們將看三個分類問題：blobs、moons 和 circles。

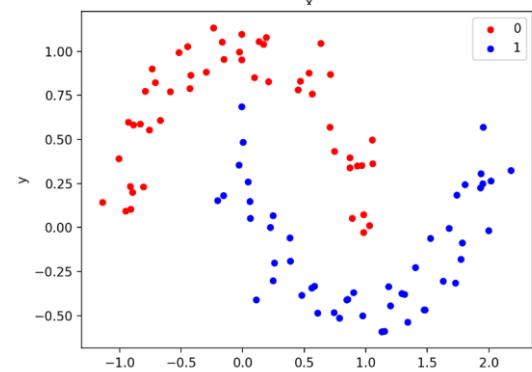
1. Blobs 分類問題

`make_blobs()` 函數可被用於生成具有高斯分佈的 blobs 點。你可以控制生成 blobs 的數量，生成樣本的數量以及一系列其他屬性。考慮到 blobs 的線性可分性質，該問題也適用於線性分類問題。右例是一個多類分類預測問題，它生成了一個具有三個 blobs 的 2D 樣本資料集。每個資料有兩個輸入和 0、1 或 2 個類的值。



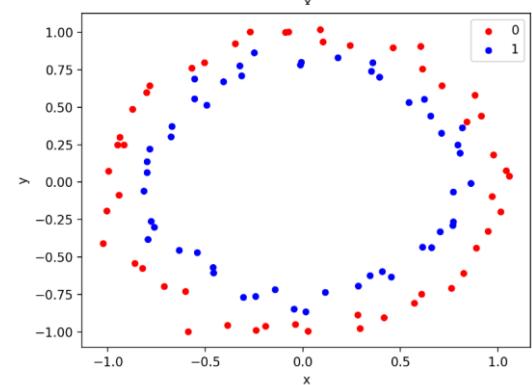
2. Moons 分類問題

`make_moons()` 函數用於二進位分類並且將生成一個漩渦模式，或者兩個 moons。你可以控制 moon 形狀中的雜訊量，以及要生產的樣本數量。這個測試問題適用於能夠學習非線性類邊界的演算法。下面的例子生成了一個中等噪音的 moon 資料集。



3. Circles 分類問題

`make_circles()` 函數生成一個資料集落入同心圓的二進位分類問題。再一次地，與 moons 測試問題一樣，你可以控制形狀中的雜訊量。該測試問題適用於可以學習複雜的非線性流行的演算法。下面的例子中生成了一個具有一定噪音的 circles 資料集。

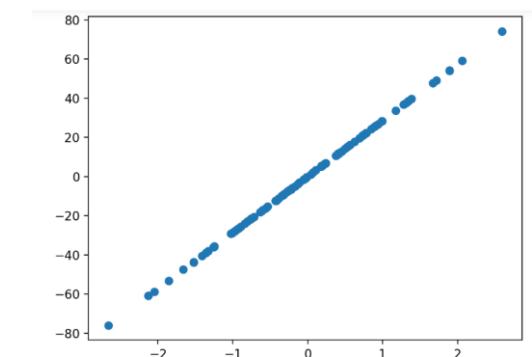


回歸測試問題

回歸是根據觀察資料預測數量的問題。`make_regression()` 函數將創建一個輸入和輸出具有線性關係的資料集。你可以配置樣本數量，輸入特徵數量，雜訊級別等等。該資料集適用於可以學習線性回歸函數的演算法。

From: 如何在 Python 中用 scikit-learn 生成测试数据集

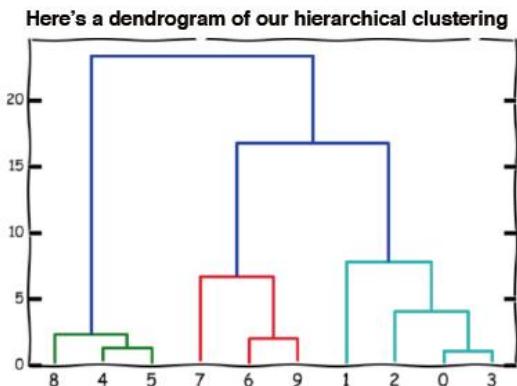
<https://www.jiqizhixin.com/articles/2018-02-05-2>



階層式分析

一種構建 cluster 的層次結構的算法。該算法從分配給自己 cluster 的所有資料點開始。然後，兩個距離最近的 cluster 合併為同一個 cluster。最後，當只剩下一個 cluster 時，該算法結束。

可定義 4, 5 是一群，或 8, 4, 5 是一群，看距離怎麼衡量 (y 軸要切在哪兒)



階層分群演算法流程

不指定分群的數量

- 每筆資料為一個 cluster
- 計算每兩群之間的距離
- 將最近的兩群合併成一群
- 重複步驟 2、3，直到所有資料合併成同一 cluster

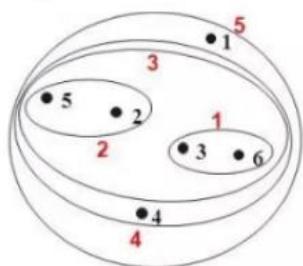
階層分群距離計算方式：

群聚與群聚間的距離可以定義為不同群聚中最接近兩點間的距離。

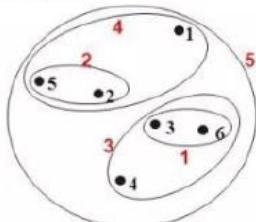
群聚間的距離定義為不同群聚中最遠兩點間的距離，這樣可以保證這兩個集合合併後，任何一對的距離不會大於 d 。

群聚間的距離定義為不同群聚間各點與各點間距離總和的平均。

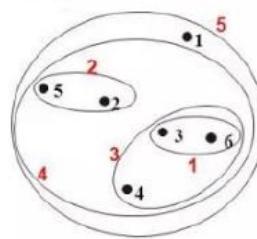
Single-link



Complete-link



Average-link



優點

1. 概念簡單，易於呈現
2. 不需指定群數



缺點

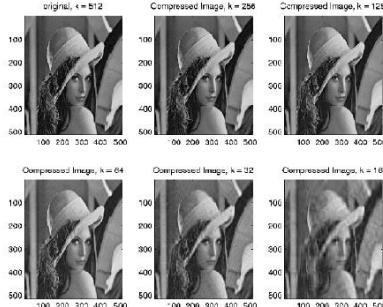
- 只適用於少量資料，大量資料會很難處理

3-4.3 降維方法 - 主成份分析 (PCA)

為什麼需要降低維度？

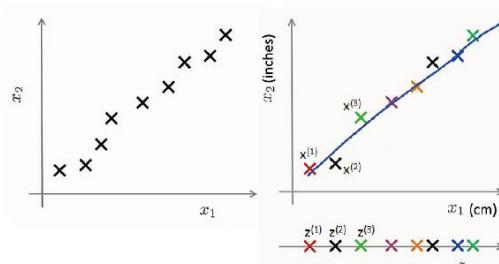
壓縮資料

- 有助於使用較少的 RAM 或 disk space，也有助於加速 learning algorithms
- 影像壓縮
 - 原始影像維度為 512，在降低維度到 16 的情況下，圖片雖然有些許模糊，但依然保有明顯的輪廓和特徵



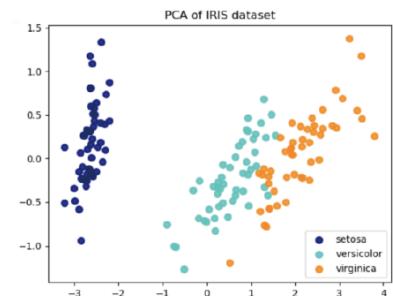
特徵組合及抽象化

- 壓縮資料可進而組合出新的、抽象化的特徵，減少冗餘的資訊。
- 左下圖的 x_1 和 x_2 高度相關，因此可以合併成 1 個特徵 (右下圖)。
 - 把 $x(i)$ 投影到藍色線，從 2 維降低為 1 維。



資料視覺化

- 特徵太多時，很難 visualize data，不容易觀察資料。
- 把資料維度 (特徵) 降到 2 到 3 個，則能夠用一般的 2D 或 3D 圖表表現資料。



複習 Day059-060 降低維度可以幫助我們壓縮及丟棄無用資訊、抽象化及組合新特徵、呈現高維數據。常用的算法為主成分分析。在維度太大發生 overfitting 的情況下，可以嘗試用 PCA 組成的特徵來做監督式學習，但不建議一開始就做。

👉 **Code 12:** Day059_PCA_Review.ipynb

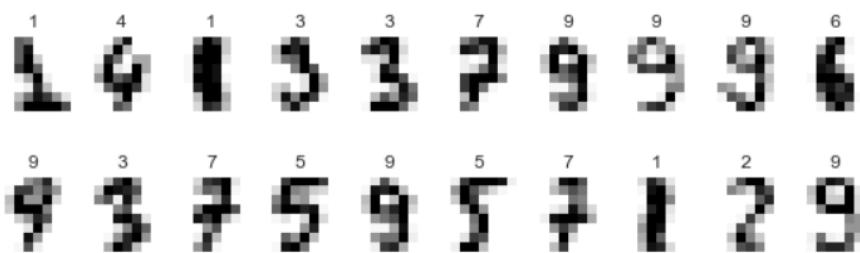
👉 **Code 13:** Day060_PCA 原檔.ipynb

這次練習改用手寫辨識資料集 digit:

```
# Day46_HW_submit
import numpy as np
import pandas as pd
from sklearn import datasets, metrics
import matplotlib.pyplot as plt
plt.style.use('seaborn-white') #圖為"白底黑字"
fig, axes = plt.subplots(2, 10, figsize=(10, 3)) #兩列十欄 圖長寬比 10:3

digits = datasets.load_digits()

for ax in axes.ravel():
    i = np.random.randint(0, digits.data.shape[0], 1)[0]
    ax.imshow(digits.data[i].reshape(8, 8))
    ax.axis('off')
    ax.set_title(digits.target[i])
```



PCA 範例 Day59

載入套件

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from sklearn import decomposition
from sklearn import datasets

np.random.seed(5)
%matplotlib inline
```

載入 digits 資料集

```
digits = datasets.load_digits(n_class=3)
X = digits.data
y = digits.target
```

設定 模型 估計參數

```
centers = [[1, 1], [-1, -1], [1, -1]]
pca = decomposition.PCA(n_components=3)
```

資料建模 並 視覺化 結果

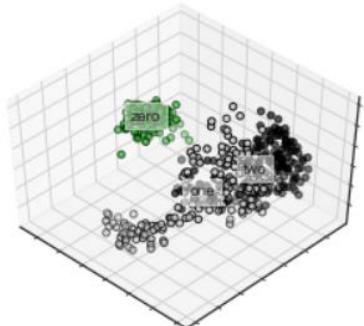
```
pca.fit(X)
X = pca.transform(X)

fig = plt.figure(1, figsize=(4, 3))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
plt.cla()
```

```
for name, label in [('zero', 0),
                     ('one', 1),
                     ('two', 2)]:
    ax.text3D(X[y == label, 0].mean(),
              X[y == label, 1].mean() + 1.5,
              X[y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.nipy_spectral,
           edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])

plt.show()
```



PCA 範例 Day60

- 以 PCA + 邏輯斯迴歸判斷手寫辨識資料集，觀察不同 component 下正確率的變化

```
# 載入套件
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

penalty='l2' 觀察結果

- explained variance ratio：解釋變異性隨著 components 穩定下降
- 分類預測準確率：似乎也隨著 components 提升，但是在 20 以後提升不多

作業

- 將參數 penalty 改為 L1, max_iter 改為 100，觀察 PCA 與羅吉斯迴歸做手寫辨識的效果
執行後可以發現，雖然 explained variance ratio 仍然隨著 components 穩定下降
但是分類預測準確率上升不穩定，可見 penalty 參數調整為 'l1' 不太適合

```
# 定義 PCA 與隨後的羅吉斯迴歸函數
```

```
# penalty 改為 L1, max_iter 改為 100
```

```
logistic = SGDClassifier(loss='log', penalty='l1', max_iter=100, tol=1e-5, random_state=0)
pca = PCA()
pipe = Pipeline(steps=[('pca', pca), ('logistic', logistic)])
```

```
# 載入手寫數字辨識集
```

```
digits = datasets.load_digits()
```

```
X_digits = digits.data
```

```
y_digits = digits.target
```

```
# 執行 GridSearchCV 跑出最佳參數
```

```
param_grid = {
    'pca_n_components': [4, 10, 20, 30, 40, 50, 64],
    'logistic_alpha': np.logspace(-4, 4, 5),
}
search = GridSearchCV(pipe, param_grid, iid=False, cv=5, return_train_score=False)
search.fit(X_digits, y_digits)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

```
Best parameter (CV score=0.912):
```

```
{'logistic_alpha': 0.01, 'pca_n_components': 30}
```

```
# 繪製不同 components 的 PCA explained variance ratio
```

```
pca.fit(X_digits)
```

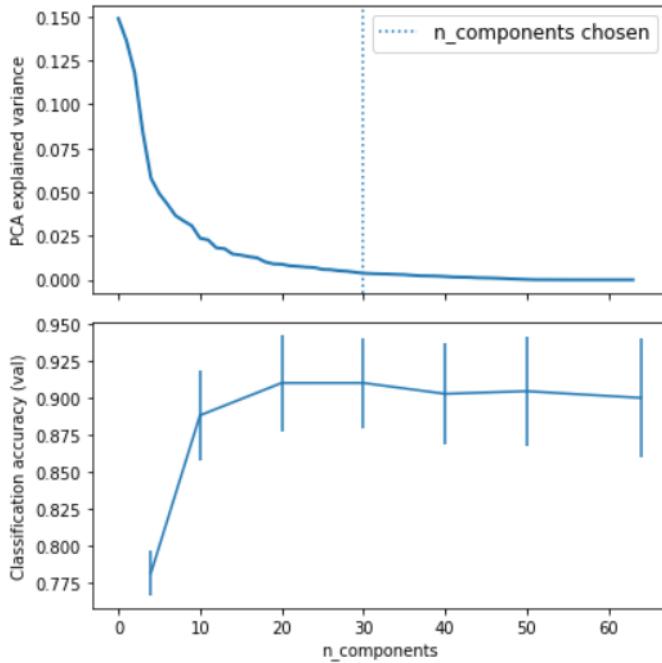
```
fig, (ax0, ax1) = plt.subplots(nrows=2, sharex=True, figsize=(6, 6))
ax0.plot(pca.explained_variance_ratio_, linewidth=2)
ax0.set_ylabel('PCA explained variance')

ax0.axvline(search.best_estimator_.named_steps['pca'].n_components, linestyle=':', label='n_components chosen')
ax0.legend(prop=dict(size=12))

<matplotlib.legend.Legend at 0xb660ac8>
```

```
# 繪製 7 個不同採樣點的 accuracy
results = pd.DataFrame(search.cv_results_)
components_col = 'param_pca__n_components'
best_clfs = results.groupby(components_col).apply(lambda g: g.nlargest(1, 'mean_test_score'))

best_clfs.plot(x=components_col, y='mean_test_score', yerr='std_test_score', legend=False, ax=ax1)
ax1.set_ylabel('Classification accuracy (val)')
ax1.set_xlabel('n_components')
plt.tight_layout()
plt.show()
```



複習

主成份分析 (PCA)

- 實務上我們經常遇到資料有非常多的 features, 有些 features 可能高度相關，有什麼方法能夠把高度相關的 features 去除？
- PCA 透過計算 eigen value, eigen vector, 可以將原本的 features 降維至特定的維度
 - 原本資料有 100 個 features，透過 PCA，可以將這 100 個 features 降成 2 個 features
 - 新 features 為舊 features 的線性組合
- 新 features 彼此不相關

$$Z_1 = a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n$$

$$Z_2 = a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \quad \text{Uncorrelated}$$

⋮

$$Z_n = a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n$$

- 應用：加速監督式學習
 - 組合出來的這些新的 features 可以進而用來做 supervised learning 預測模型
 - 以判斷人臉為例，最重要的特徵是**眼睛、鼻子、嘴巴**，**膚色和頭髮**等都可以捨棄，將這些不必要的資訊捨棄除了可以加速 learning，也可以避免一點 overfitting。

PCA 的問題

- 求共變異數矩陣進行奇異值分解，因此會被資料的差異性影響，無法很好的表現相似性及分佈。
- PCA 是一種線性降維方式，因此若特徵間是非線性關係，會有 underfitting 的問題。

3-4.4 降維方法 - t-SNE

特徵間為非線性關係時 (e.g. 文文字、影像資料)，PCA 很容易 underfitting。t-SNE 對於特徵非線性資料有更好的降維呈現能力。

▶ **Code 14:** Day061_tsne_Review.ipynb

▶ **Code 15:** Day062_tsne_原檔-S 曲線.ipynb

▶ **Code 16:** Day062_tsne_雙同心圓.ipynb

載入套件

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import offsetbox
from sklearn import manifold, datasets

%matplotlib inline
```

載入 digits (4 個數字) 資料集

```
digits = datasets.load_digits(n_class=4)
X = digits.data
y = digits.target
```

設定 模型 估計參數

```
n_samples, n_features = X.shape
n_neighbors = 30
tsne = manifold.TSNE(n_components=2, random_state=0, init='pca', learning_rate=200., early_exaggeration=12.)
```

資料建模 並 視覺化 結果

```
# Scale and visualize the embedding vectors
def plot_embedding(X, title=None):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)

    plt.figure()
    ax = plt.subplot(111)
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(y[i]),
                 color=plt.cm.Set1(y[i] / 10.),
                 fontdict={'weight': 'bold', 'size': 9})

    if hasattr(offsetbox, 'AnnotationBbox'):
        # only print thumbnails with matplotlib > 1.0
        shown_images = np.array([[1., 1.]]) # just something big
        for i in range(X.shape[0]):
            dist = np.sum((X[i] - shown_images) ** 2, 1)
            if np.min(dist) < 4e-3:
                # don't show points that are too close
                continue
            shown_images = np.r_[shown_images, [X[i]]]
            imagebox = offsetbox.AnnotationBbox(
                offsetbox.OffsetImage(digits.images[i], cmap=plt.cm.gray_r),
                X[i])
            ax.add_artist(imagebox)
    plt.xticks([]), plt.yticks([])
    if title is not None:
        plt.title(title)
```

```

# t-SNE embedding of the digits dataset
print("Computing t-SNE embedding")

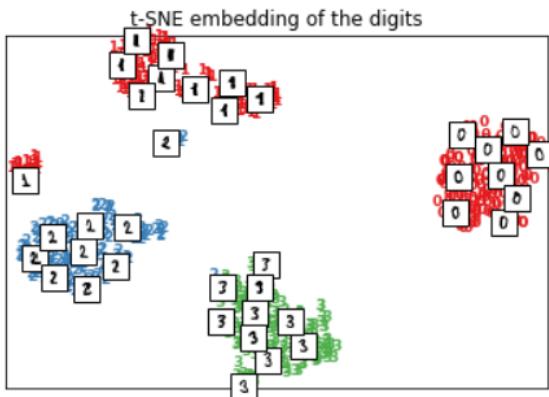
X_tsne = tsne.fit_transform(X)

plot_embedding(X_tsne,
    "t-SNE embedding of the digits")

plt.show()

```

Computing t-SNE embedding



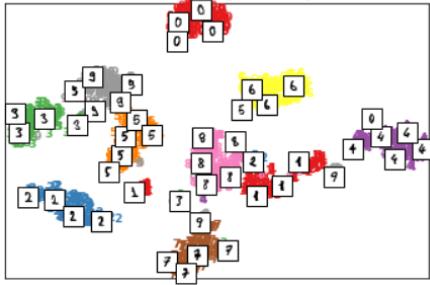
#作業：載入 digits 完整 資料集

```

digits = datasets.load_digits(0)
X = digits.data
y = digits.target

```

t-SNE embedding of the digits



#Day62 範例:

- 觀察雙同心圓使用 t-SNE 不同 perplexity 的分群效果

載入套件

```

import numpy as np
import matplotlib.pyplot as plt

from matplotlib.ticker import NullFormatter
from sklearn import manifold, datasets
from time import time

```

設定模型與繪圖參數

```

n_samples = 300
n_components = 2
(fig, subplots) = plt.subplots(2, 5, figsize=(15, 6))
perplexities = [4, 6, 9, 14, 21, 30, 45, 66, 100]

```

設定 S 曲線資料點

```
X, color = datasets.samples_generator.make_s_curve(n_samples, random_state=0)
```

繪製資料原圖

```

ax = subplots[0][0]
ax.set_title("Original")
ax.scatter(X[red, 0], X[red, 1], c="r")
ax.scatter(X[green, 0], X[green, 1], c="g")
ax.xaxis.set_major_formatter(NullFormatter())
ax.yaxis.set_major_formatter(NullFormatter())
plt.axis('tight')
(-0.05500000000000001,
 0.05500000000000001,
 -0.05500000000000001,
 0.05500000000000001)

```

繪製不同 perplexity 下的 t-SNE 分群圖

```
for i, perplexity in enumerate(perplexities):
```

```

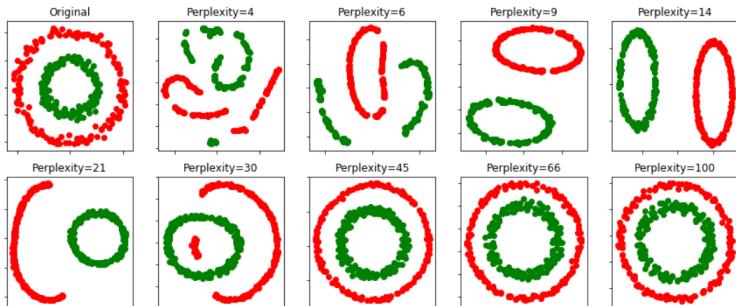
if i<4:
    ax = subplots[0][i+1]
else:
    ax = subplots[1][i-4]

t0 = time()
tsne = manifold.TSNE(n_components=n_components, init='random',
                      random_state=0, perplexity=perplexity)
Y = tsne.fit_transform(X)
t1 = time()
print("circles, perplexity=%d in %.2g sec" % (perplexity, t1 - t0))
ax.set_title("Perplexity=%d" % perplexity)
ax.scatter(Y[red, 0], Y[red, 1], c="r")
ax.scatter(Y[green, 0], Y[green, 1], c="g")
ax.xaxis.set_major_formatter(NullFormatter())
ax.yaxis.set_major_formatter(NullFormatter())
ax.axis('tight')

plt.show()

```

circles, perplexity=4 in 1.8 sec
 circles, perplexity=6 in 1.9 sec
 circles, perplexity=9 in 2.2 sec
 circles, perplexity=14 in 2.8 sec
 circles, perplexity=21 in 3.3 sec
 circles, perplexity=30 in 4.1 sec
 circles, perplexity=45 in 5.4 sec
 circles, perplexity=66 in 5.8 sec
 circles, perplexity=100 in 6.8 sec

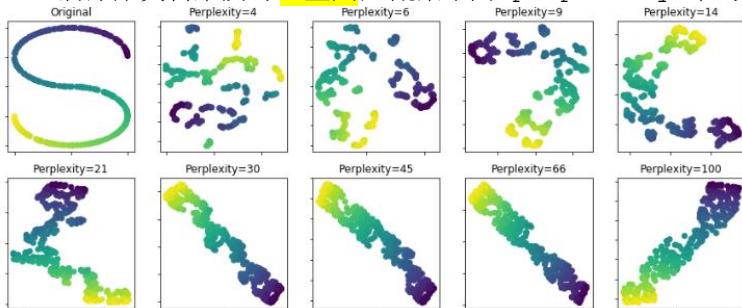


觀察結果

- perplexity 越低時，圖形越零散，越高時不僅分群好，也讓圖形更接近原圖
- 但 perplexity 越高，執行時間也越久

作業

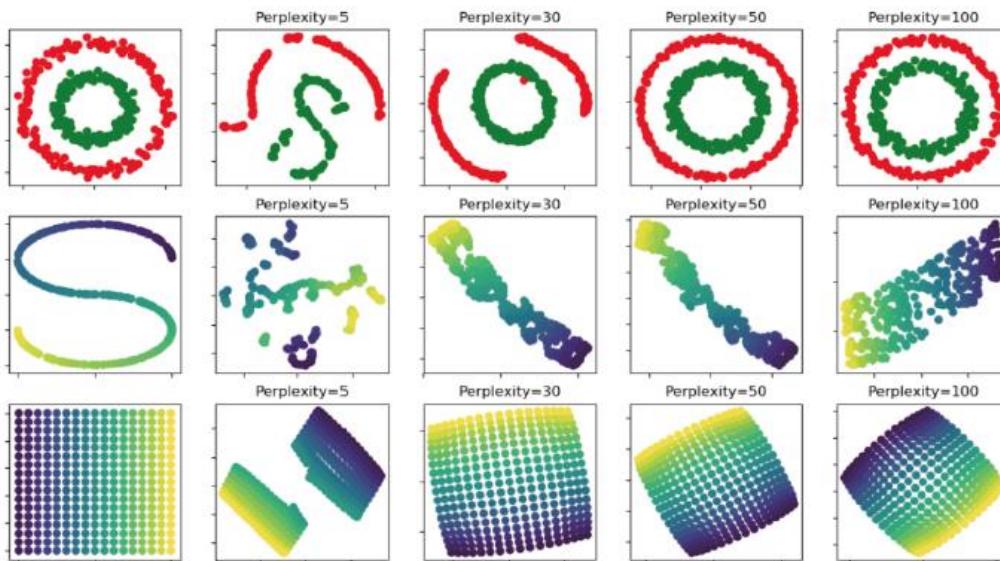
- 將原始資料集換為 S 型圖，觀察不同 perplexity 下的流形還原結果



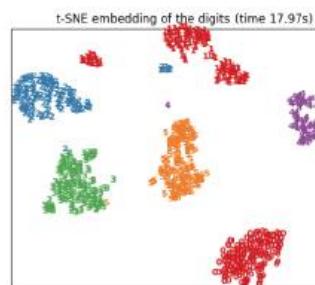
t-SNE

- t-SNE 也是一種降維方式，但它用了更複雜的公式來表達高維和低維之間的關係。
- 主要是將高維的資料用 gaussian distribution 的機率密度函數近似，而低維資料的部分用 t 分佈來近似，在用 KL divergence 計算相似度，再以梯度下降 (gradient descent) 求最佳解。

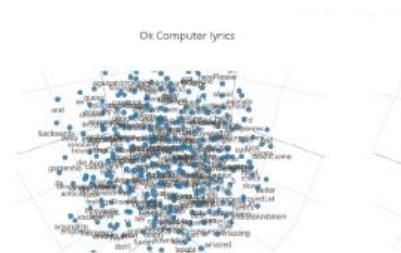
- 視覺化範例



影像資料 (MNIST) 視覺化呈現



文字資料 (MNIST) 視覺化呈現



優點

當特徵數量過多時，使用 PCA 可能會造成
降維後的 underfitting，這時可以考慮使用
t-SNE 來降維

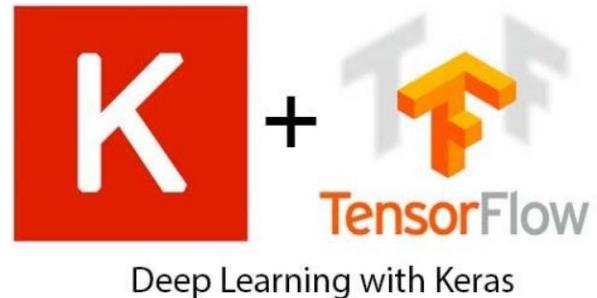


缺點

t-SNE 的需要比較多的時間執行

第四章 圖片資料

【初探深度學習使用 Keras】



Keras 是 Python 深度學習的一種上層框架(模型層 model level)，只負責機器學習模型之建立、訓練與預測，其底層的矩陣運算是依賴後端引擎 (backend engine) 來支援。目前 Keras 提供兩種後端引擎 : Theano 與 TensorFlow。TensorFlow 是 Google Brain 團隊所開發的低階深度學習框架，可在各平台上進行高效能之矩陣運算，原先僅使用於 Gmail 信件過濾、Google 語音辨識、Google 翻譯等 Google 自家產品上，為了透過開源社群之力量使其成為市場標準，Google 於 2015 年 11 月將其原始碼開放。在這裡直接安裝 Anaconda，在 jupyter notebook 安裝 TensorFlow 和 Keras，而其它的安裝方法可以參考本筆記前言的參考資料。

4-1 讀取檔案: txt 檔 & 圖片檔

讀取圖片回顧到機器學習百日馬拉松課程 Day_003-3 內容，不同的資料有不同讀取方式，撇開其他形式如 npy / pickle 等可以儲存經過處理的資料，一般文字格式通常可以用 with open() 讀取；而圖像格式可以使用 skimage 、PIL 、OpenCV，一般皆會使用 PIL 顯示影像，套件匯入如下：

Code 17: Day003-3_ImportImage_Review.ipynb

```
import matplotlib.pyplot as plt  
import numpy as np  
%matplotlib inline
```

讀圖檔的三種方法

skimage、PIL、OpenCV 差異如下，都常被使用：

skimage

```
import skimage.io as skio  
img1 =  
skio.imread('data/example.jpg')  
print(img1)  
plt.imshow(img1)  
plt.show()
```

[[[204 194 184]

[204 194 184]
[204 194 184]

...
[224 210 201]

[224 210 201]

[224 210 201]]

[[204 194 184]

[204 194 184]

[204 194 184]

...
[224 210 201]

[224 210 201]

[224 210 201]]

...
[[112 106 106]

[157 151 151]

[162 156 156]

...
[3 2 0]

[3 2 0]

[3 2 0]]]

[[95 86 87]

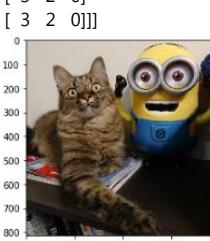
[155 146 147]

[180 171 172]

...
[3 2 0]

[3 2 0]

[3 2 0]]]



PIL

```
from PIL import Image  
img2 =  
Image.open('data/example.jpg') # 這時候還是 PIL object  
print(img2)  
img2 = np.array(img2)  
print(img2)  
plt.imshow(img2)  
plt.show()
```

<PIL.JpegImagePlugin.JpegImageFile image
mode=RGB size=771x813 at 0xA85CC88>

[[[204 194 184]

[204 194 184]

...
[224 210 201]

[224 210 201]

[224 210 201]]

[[204 194 184]

[204 194 184]

...
[224 210 201]

[224 210 201]

[224 210 201]]

[[204 194 184]

[204 194 184]

[204 194 184]

...
[224 210 201]

[224 210 201]

[224 210 201]]

[[112 106 106]

[157 151 151]

[162 156 156]

...
[3 2 0]

[3 2 0]

[3 2 0]]]

[[95 86 87]

[155 146 147]

[180 171 172]

...
[3 2 0]

[3 2 0]

[3 2 0]]]

[[95 86 87]

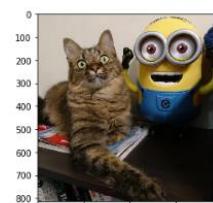
[155 146 147]

[180 171 172]

...
[3 2 0]

[3 2 0]

[3 2 0]]]



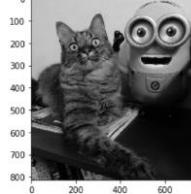
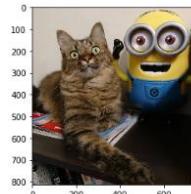
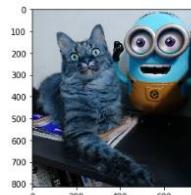
OpenCV

```
import cv2  
img3 = cv2.imread('data/example.jpg')  
plt.imshow(img3)  
plt.show()
```

img3 = cv2.cvtColor(img3,
cv2.COLOR_BGR2RGB)

cv2.cvtColor is an OpenCV function
to convert images to different color
spaces
plt.imshow(img3)
plt.show()

img4 = cv2.cvtColor(img3,
cv2.COLOR_BGR2GRAY)
plt.imshow(img4, cmap='gray')
plt.show()

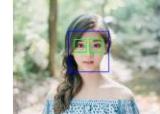


詳細也可復習 D64 講義

(Day003-5_Day064_OPENCV.ipynb)

利用 OpenCV 抓取相片中的臉部數據

detectMultiScale (image, scaleFactor, minNeighbors)：這是一個檢測對象的通用函數，在這種情況下，它會檢測面，因為我們調用了 face cascade。如果找到一個面，它將以“Rect (x, y, w, h)”的形式返回所述面的位置列表，如果沒有，則返回“None”。image：第一個輸入是灰度圖像。因此，請確保圖像為灰度。



Code 18: Day003-5_Day064_OPENCV.ipynb

<https://hardliver.blogspot.com/2017/07/opencv-opencv.html>

比較三種開圖方式的時間 - 比較讀取 1000 次

```
N_times = 1000
```

```
%%timeit
```

```
im = np.array([skio.imread('data/example.jpg') for _ in range(N_times)])
```

14.3 s ± 205 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
%%timeit
```

```
im = np.array([np.array(Image.open('data/example.jpg')) for _ in range(N_times)])
```

13.9 s ± 254 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
%%timeit
```

```
im = np.array([cv2.cvtColor(cv2.imread('data/example.jpg'), cv2.COLOR_BGR2RGB) for _ in range(N_time s)])
```

12 ± 184 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

- 比較結果：OpenCV 會比較快一些。

- 小結：圖像格式可以使用 PIL, Skimage 或是 CV2，但要注意 CV2 的速度較快，但須讀入的格式為 BGR

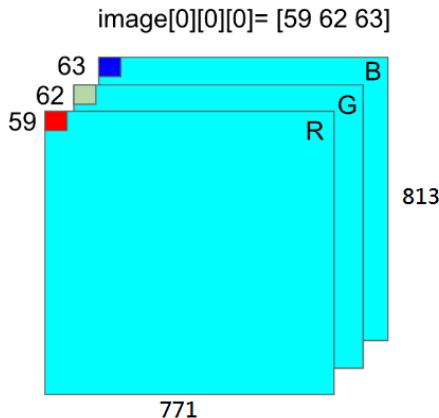
圖檔的資料結構：矩陣中的數字代表的意義

```
im = np.array([cv2.cvtColor(cv2.imread('data/example.jpg'), cv2.COLOR_BGR2RGB)])  
print('image shape:', im.shape) #顯示訓練集圖片之 shape
```

image shape: (1, 813, 771, 3)

```
im[0]
```

```
array([[[204, 194, 184],           #第 1 列畫素開始 (0)  
       [204, 194, 184],  
       [204, 194, 184],  
       ...,  
       [224, 210, 201],           #第 1 列畫素結束 (812)  
       [224, 210, 201],  
       [224, 210, 201]],           #第 2 列畫素開始 (0)  
       [[204, 194, 184],  
       [204, 194, 184],  
       [204, 194, 184],  
       ...,  
       [224, 210, 201],           #第 2 列畫素結束 (812)  
       [224, 210, 201],  
       [224, 210, 201]],           #第 771 列畫素開始 (0)  
       ...  
       [[ 95,  86,  87],  
        [155, 146, 147],  
        [180, 171, 172],  
        ...  
        [ 3,   2,   0],           #第 771 列畫素結束 (812)  
        [ 3,   2,   0],  
        [ 3,   2,   0]]], dtype=uint8)
```



shape: (筆數, 圖片高, 圖片寬, 彩色=3/黑白=1), 這邊例子筆數為 1 筆之 813*771 的 RGB 彩色圖片，向量中的 3 個數值分別為 RGB 顏色之色碼 (0~255)，一張圖片由三個色版相疊而成，左為 數字圖片結構之示意圖範例。

情境練習：讀取 txt 檔案中的圖片連結

Code 19: Day003-4_ImportImage2_Review.ipynb

STEP1: 讀取 txt 檔

方法 1 一次讀取所有資料，再一行一行處理

回顧到機器學習百日馬拉松課程 Day_003-2 內容，使用內建功能讀取 txt 檔，建議利用 with 指令開啟檔案，優點是檔案會自動關閉，不需要撰寫 .close() 指令，說明可參考 <http://yltang.net/tutorial/python/13/>。Python 讀檔原本就是將每行資料存到串列中的元素，即同語法<filename>.readlines()，一次讀取所有資料，之後再一行一行處理。

讀取 [text file](#) (事先點擊下載 `imagenet_urls_examples.txt`)

```
import chardet #由於不同字元在不同系統會影響讀取  
#因此使用chardet可以找到檔案的編碼  
with open('data/imagenet_urls_examples.txt', 'rb') as f:  
    result = chardet.detect(f.read())  
    result  
{'encoding': 'utf-8', 'confidence': 0.99, 'language': ''}
```

讀取連結 txt 檔: https://raw.githubusercontent.com/vashineyu/slides_and_others/master/tutorial/examples/imagenet_urls_examples.txt

```
##假如我們不想把資料載到自己的電腦裡?  
import requests  
target_url =  
"https://raw.githubusercontent.com/vashineyu/slides_and_others/master/tutorial/examples/imagenet_urls_examples.txt"
```

```
data = []  
with open("data/imagenet_urls_examples.txt", 'r',  
encoding=result['encoding']) as f:  
    for line in f: #一次讀取所有資料，再一行一行處理  
        line = line.replace("\n", "").split(',')  
        #將每句最後的\n取代成空值後，再以逗號斷句  
        data.append(line)  
data[0:1] #預視前兩筆資料  
[['n00015388_157\\http://farm1.static.flickr.com/145/430300483_21e993670c.jpg'],  
['n00015388_238\\http://farm2.static.flickr.com/1005/3352960681_37b9c1d27b.jpg']]
```

```
response = requests.get(target_url)  
data = response.text  
  
#用request傳送回來的資料不會認得斷行符號  
print(len(data))  
data[0:100]  
784594  
'n00015388_157\\http://farm1.static.flickr.com/145/430300483_21e993670c.jpg\\nn00015388_238\\http://farm2'
```

```
[n00015388_238\\http://farm2.static.flickr.com/1005/3352960681_37b9c1d27b.jpg]
```

```
#再一行一行處理斷行  
data = data.split("\n")  
print(len(data))  
data[0]  
9996  
'n00015388_157\\http://farm1.static.flickr.com/145/430300483_21e993670c.jpg'
```

```
#將txt轉成pandas dataframe  
import pandas as pd  
  
arrange_data = []
```

```

for d in data:
    line = d.split("\t")
    arrange_data.append(line)

df = pd.DataFrame(arrange_data)
df.head()

0      0          1
0  n00015388_157  http://farm1.static.flickr.com/145/430300483_2...
1  n00015388_238  http://farm2.static.flickr.com/1005/3352960681...
2  n00015388_304  http://farm1.static.flickr.com/27/51009336_a96...
3  n00015388_327  http://farm4.static.flickr.com/3025/2444687979...
4  n00015388_355  http://img100.imageshack.us/img100/3253/forres...

```

方法 2 懶人解法：直接用 pandas 讀取網路連結

```

## 注意：一行中的分隔符號有時候為 "," (預設), 但也常常是 tab (\t)
## 如果非 "," 的話，記得要加上 sep 參數告訴 pd.read_csv

import requests
target_url =
"https://raw.githubusercontent.com/vashineyu/slides_and_others/master/tutorial/examples/imagenet_urls_examples.txt"
df_lazy = pd.read_csv(target_url, sep="\t", header=None)
df_lazy.head()

0      0          1
0  n00015388_157  http://farm1.static.flickr.com/145/430300483_2...
1  n00015388_238  http://farm2.static.flickr.com/1005/3352960681...
2  n00015388_304  http://farm1.static.flickr.com/27/51009336_a96...
3  n00015388_327  http://farm4.static.flickr.com/3025/2444687979...
4  n00015388_355  http://img100.imageshack.us/img100/3253/forres...

```

STEP2: 讀取圖片，請讀取上面 data frame 中的前 5 張圖片

```

from PIL import Image
from io import BytesIO
import numpy as np
import matplotlib.pyplot as plt

response = requests.get(df.loc[0, 1]) # 先讀取第一筆資料的圖片
img = Image.open(BytesIO(response.content))
img = np.array(img)
print(img.shape)
plt.imshow(img)
plt.show()

```

(436, 500, 3)



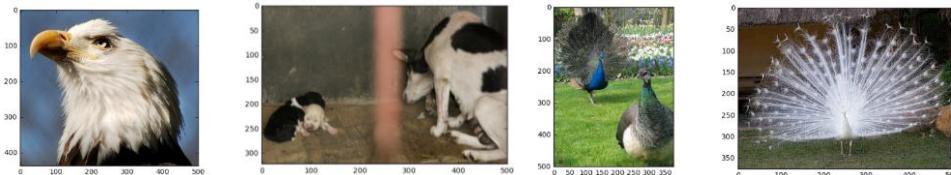
補充說明: resize

未來使用 tensorflow 訓練自己的模型的時候發現，tensorflow 官網上所給的例子，都是用處理好數據格式的 mnist 數據或者其他格式的數據，所以在訓練自己的模型的第一步就卡住了。Resize 將會是一個解決方案.....

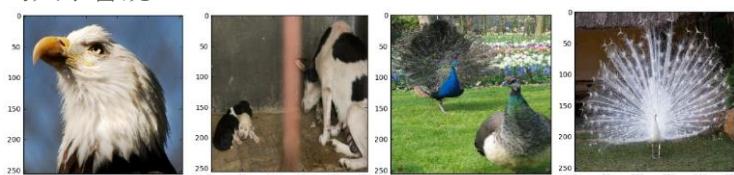
```
# resize 範例說明，未來使用 tensorflow 訓練自己的模型的時候發現，tensorflow 官網上所給的例子，都是用處理好數據格式的 mnist 數據或者其他格式的數據，所以在訓練自己的模型的第一步就卡住了。Resize 將會是一個解決方案.....  
  
def img2arr_fromURLs(url_list, resize = False):  
    img_list = []  
    for url in url_list:  
        response = requests.get(url)  
        try:  
            img = Image.open(BytesIO(response.content))  
            if resize:  
                img = img.resize((256,256)) # 假如 resize, 就先統一到 256 x 256  
            img = np.array(img)  
            img_list.append(img)  
        except:  
            # 只有在 response.status_code 為 200 時，才可以取得圖片。  
            # 若有 404 或其他 status code, 會碰到 Error, 所以我們用 Try 語法避開取不到的狀況  
            pass # print("Error")  
  
    return img_list
```

```
result = img2arr_fromURLs(df[0:5][1].values)  
print("Total images that we got: %i" % len(result)) # 如果不等於 5-6, 代表有些連結失效囉  
  
for im_get in result:  
    plt.imshow(im_get)  
    plt.show()
```

Total images that we got: 4



如果 resize=True，則把上一段地一句話加上，result = img2arr_fromURLs(df[0:5][1].values, resize=True)，全部的大小會統一 256 x 256。



4-2 Keras 資料庫

Keras 是什麼？

易學易懂的深度學習套件

- Keras 設計出發點在於容易上手，因此隱藏了很多實作細節，雖然自由度稍嫌不夠，但很適合教學
- Keras 實作並**優化了各式經典組件**，因此即使是同時熟悉TensorFlow 與 Keras 的老手，開發時也會兩者並用互補

Keras包含的組件有哪些？

- Keras 的組件很貼近直覺，因此我們可以用 TensorFlow PlayGround 體驗所學到的概念，分為兩大類來理解（非一一對應）
- 模型形狀類
 - 直覺概念：神經元數 / 隱藏層數 / 啟動函數
 - Keras組件：Sequential Model / Functional Model / Layers
- 配置參數類
 - 直覺概念：學習速率 / 批次大小 / 正規化
 - Keras組件：Optimizer / Regularizers / Callbacks

深度學習寫法封裝

TensorFlow 將深度學習中的 GPU/CPU 指令封裝起來，減少語法差異，Keras 則是將前者更進一步封裝成單一套件，用少量的程式便能實現經典模型

Keras的後端

圖中所示：Keras 的實現，實際上完全依賴 TensorFlow 的語法完成，這種情形我們稱 TensorFlow 是 Keras 的一種後端（Backend）

Keras/TensorFlow的比較

	Keras	Tensorflow
學習難度	低	高
模型彈性	中	高
主要差異	處理神經層	處理資料流
代表組件	Layers/Model	Tensor / Session / Placeholder

註：由於目前整併兩者的 TensorFlow 2.0
仍在測試階段，本教材仍以 Keras 為主

Keras 更新或安裝回顧到機器學習百日馬拉松課程 Day66_Introduction_of_Keras (新講義) 內容。

WARNING: You are using pip version 19.3.1; however, version 20.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

!python -m pip install --upgrade pip

Requirement already up-to-date: pip in c:\users\administrator\anaconda3\lib\site-packages (19.0.3)

!pip install tensorflow;
!pip install keras

(略)

```
# 載入套件
import keras
print(keras.__version__)# 檢查Keras 版本
```

Using TensorFlow backend.

2.2.4

在 Anaconda 成功安裝 tensorflow 和 keras

<https://reurl.cc/k5DNo3>

Keras 安裝流程

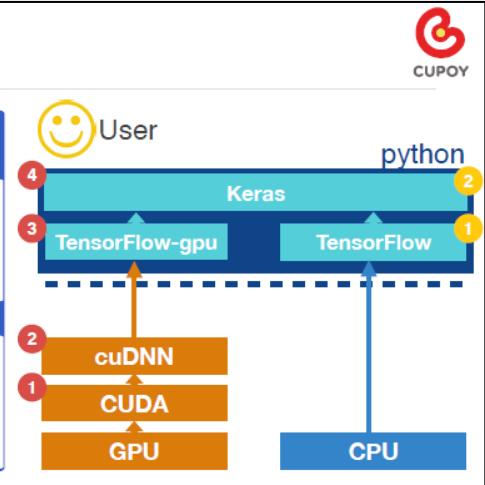
安裝分歧點

是否有GPU

如圖所示，因為有GPU則需要先裝GPU的指令集，所以有GPU則需要 4 個步驟，沒有就只需要 2 步驟

作業系統

因為不同作業系統間，GPU的安裝步驟會因介面或指令有所不同，所以我們會分 Windows / Linux (以Ubuntu為例) / Mac 分別介紹流程



Keras 自帶數據集與模型

- Keras自帶的數據集
 - CIFAR10小圖像分類
 - CIFAR100小圖像分類
 - IMDB電影評論情緒分類
 - 路透社newswire話題分類
 - 手寫數字的MNIST數據庫
 - 時尚文章的時尚MNIST數據庫
 - 波士頓房屋價格回歸數據集

如何使用Keras 自帶數據集做目標學習

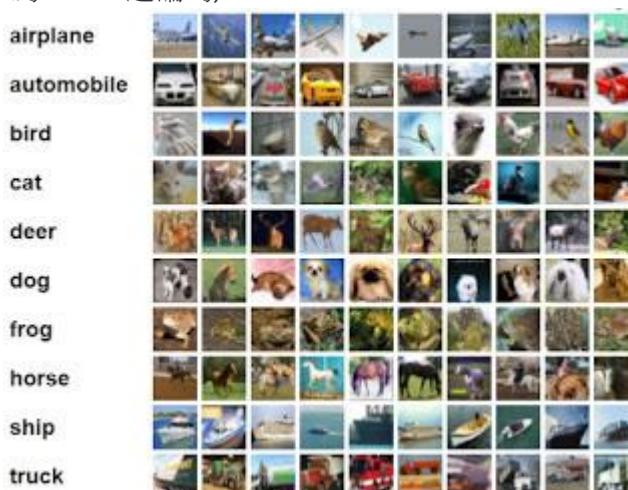
- 適用於文本分析與情緒分類
 - IMDB 電影評論情緒分類
 - 路透社新聞專題主題分類
- 適用於影像分類與識別學習
 - CIFAR10/CIFAR100
 - MNIST/ Fashion-MNIST
- 適用於 Data/Numerical 學習
 - Boston housing price regression dataset

Keras 資料庫簡介: Cifar-10 (說明可參考 <https://yhuang1966.blogspot.com/2018/04/keras-cifar-10.html>)

Cifar-10 資料集是由深度學習大師 Geoffrey Hinton 教授與其在加拿大多倫多大學的學生 Alex Krizhevsky 與 Vinod Nair 所整理之影像資料集, 包含 6 萬筆 32*32 低解析度之彩色圖片, 其中 5 萬筆為訓練集; 1 萬筆為測試集, 是機器學習中常用的圖片辨識資料集：

Cifar-10 的所有圖片被分為 10 個類別 (以 0~9 數字作為 Label 之編碼) :

- 0 : airplane (飛機)
- 1 : automobile (汽車)
- 2 : bird (鳥)
- 3 : cat (貓)
- 4 : deer (鹿)
- 5 : dog (狗)
- 6 : frog (青蛙)
- 7 : horse (馬)
- 8 : ship (船)
- 9 : truck (卡車)



參考：

```
# https://en.wikipedia.org/wiki/CIFAR-10
# https://www.cs.toronto.edu/~kriz/cifar.html
(可下載 Cifar-10 資料集)
```

```
在程式中需要下載對應的資料集時，首先導入對應的模組，然後調用.load_data()函數
#從Keras導入相應的模組
from keras.datasets import cifar10
#從網路即時下載
(x_train, y_train), (x_validate, y_validate) = cifar10.load_data()
```

Cifar-10 名稱來自加拿大高等研究院 (Canadian Institute For Advanced Research), 10 表示其包含 10 種類別圖片. Cifar-10 事實上是一個包含 8000 萬個已標記 (Labeled) 圖庫的子集合, 它還有一個更大的姊妹 Cifar-100 資料集, 同樣包含 60000 個圖片, 但有 100 種類別。

Keras 資料庫簡介回顧到機器學習百日馬拉松課程 Day67_Keras_Dataset 及 D97_使用 CNN 完成 CIFAR-10 資料集內容。

4-3 資料前處理

Code 20: Day067_Keras_Dataset_Review.ipynb

STEP1. 載入套件

```
import numpy
from keras.datasets import cifar10
import numpy as np
np.random.seed(10)
Using TensorFlow backend.
```

STEP2.1 資料預視

```
# 讀取資料並檢視
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

x_train shape: (50000, 32, 32, 3)

50000 train samples

10000 test samples

Train Data

x_img_train.shape

(50000, 32, 32, 3)

x_img_train

```
array([[[[ 59, 62, 63],
       [ 43, 46, 45],
       [ 50, 48, 43],
       ...,
       [158, 132, 108],
       [152, 125, 102],
       [148, 124, 103]],
```

....

```
[[122, 119, 114],
 [118, 116, 110],
 [120, 116, 111],
```

....

```
[179, 177, 173],
 [164, 164, 162],
 [163, 163, 161]]], dtype=uint8)
```

y_label_train.shape

(50000, 1)

y_label_train

```
array([[6],
       [9],
       [9],
       ...,
       [9],
       [1],
       [1]], dtype=uint8)
```

Test Data

x_img_test.shape

(10000, 32, 32, 3)

x_img_test[0]

```
array([[[[158, 112, 49],
       [159, 111, 47],
       [165, 116, 51],
       ...,
       [137, 95, 36],
       [126, 91, 36],
       [116, 85, 33]],
```

....

```
[[ 54, 107, 160],
 [ 56, 105, 149],
 [ 45,  89, 132],
 ...,
 [ 24,  77, 124],
 [ 34,  84, 129],
 [ 21,  67, 110]], dtype=uint8)
```

y_label_test.shape

(10000, 1)

y_label_test

```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]])
```

資料標籤

```
label_dict={0:"airplane",1:"automobile",2:"bird",3:"cat",4:"deer",
            5:"dog",6:"frog",7:"horse",8:"ship",9:"truck"}
```

```
#看資料(圖片)
import matplotlib.pyplot as plt
def plot_images_labels_prediction(images,labels,prediction,idx,num=10):
    fig = plt.gcf()
    fig.set_size_inches(12, 14)
    if num>25: num=25
    for i in range(0, num):
        ax=plt.subplot(5,5, 1+i)
        ax.imshow(images[idx],cmap='binary')

        title=str(i)+';'+label_dict[labels[i][0]]
        if len(prediction)>0:      #第一次 prediction IS NULL, 第二次以後累加資訊
            title+= '>' +label_dict[prediction[i]]

        ax.set_title(title,fontsize=10)
        ax.set_xticks([]);ax.set_yticks([])
        idx+=1
    plt.show()
```

```
plot_images_labels_prediction(x_img_train,y_label_train,0)
#plot_images_labels_prediction( 圖素資料, 標籤, 預測值, 指定第一筆資料, 預設 SHOW 出 10 筆資料)
```



從第 0 筆抓出 10 筆資料預視圖片內容。

STEP2.2 資料預處理之概念

Image normalize

```
print("original x_img_train[0][0][0]: ", x_img_train[0][0][0])
# Image normalize
x_img_train_normalize = x_img_train.astype('float32') / 255.0
x_img_test_normalize = x_img_test.astype('float32') / 255.0
print("normalize x_img_train[0][0][0]: ", x_img_train_normalize[0][0][0])
original x_img_train[0][0][0]: [59 62 63]
normalize x_img_train[0][0][0]: [0.23137255 0.24313726 0.24705882]
```

轉換 label 為 OneHot Encoding

```
y_label_train.shape
(50000, 1)
y_label_train[:5]
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
from keras.utils import np_utils
y_label_train_OneHot = np_utils.to_categorical(y_label_train)
y_label_test_OneHot = np_utils.to_categorical(y_label_test)
```

```

y_label_train_OneHot.shape
(50000, 10)
y_label_train_OneHot[:5]
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.]])

```

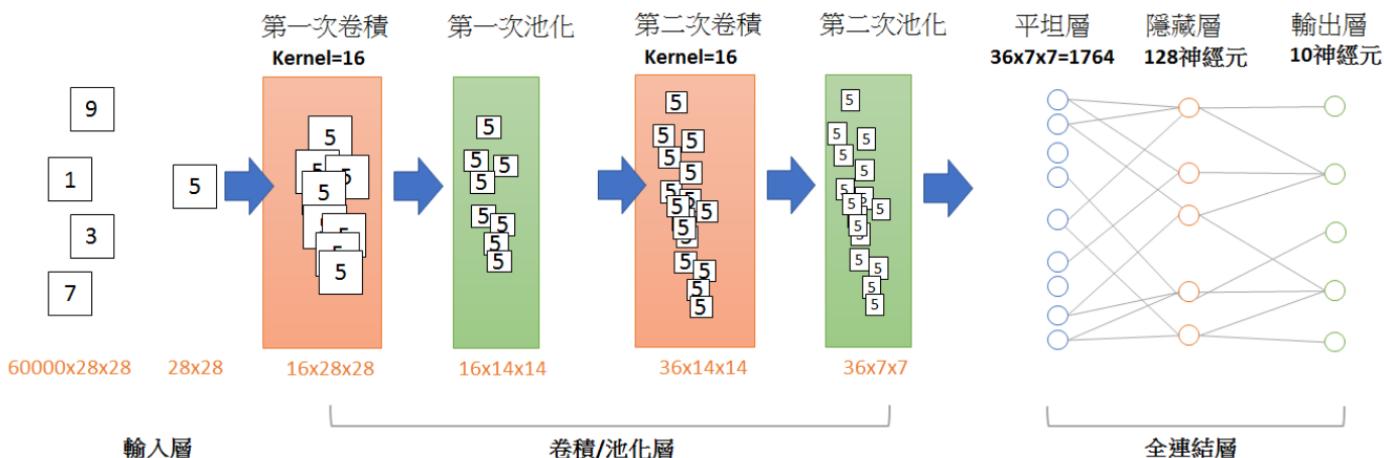
STEP3 設定模型與運算 (DNN vs. CNN / plus ImageDataGenerator)

回顧 Day097_Keras_CNN_vs_DNN_Review 的課程程式，這邊先簡單帶過機器學習的發展，可以參考 <機器學習的衰頹興盛：從類神經網路到淺層學習>^{#4.3.1} 等系列文章。機器學習模型的第一波浪潮「類神經網路 (Neural Network, NN)」，是由生物大腦的神經元運作方式所啟發而建立出來的數學模型。「深度神經網絡 (Deep Neural Networks, DNN)」是一種具備至少一個隱藏層的神經網路^{#4.3.2}。DNN 與 NN 主要的區別在於把 sigmoid 函數替換成了 ReLU，maxout，克服了梯度消失的問題^{#4.3.3}，細節設定在之後章節也會談到。

卷積深度神經網路 (Convolutional Neural Networks, CNN) 並不是所有上下層神經元都能直接相連，而是通過「卷積核」作為仲介^{#4.3.3}。CNN 由一個或多個卷積層和頂端的全連通層（對應經典的神經網路）組成，同時也包括關聯權重和池化層 (pooling layer)。這一結構使得卷積神經網路能夠利用輸入資料的二維結構^{#4.3.2}。以 Keras 建立卷積神經網路作範例，如圖^{#4.3.4} <https://ppt.cc/f8wb1x>，圖中使用 MNIST 手寫資料集來預定建立具有兩層的卷積與池化、以及一個 128 個神經元全連結層的 CNN 模型。

- 文中利用 `model = Sequential()` 建立一個線性堆疊模型，之後只要將建立的神經網路層依次加入即可。
- `model.add()` 建立第一個卷積層加入 `model`，該層有 **16 個 filters**(也稱 kernel)，kernel 大小為 5×5 ，本層會輸出 16 個 28×28 的影像。
- 接下來是池化層，只要定義 **pool size 為 $(2,2)$** ，即尺寸縮小為一半，因此上一層卷積層的 16 個 28×28 影像會輸出為 16 個 14×14 的影像。
- 之後再建立第二個卷積層，這次有 **36 個 filters** 之 14×14 的影像。
- 再加入第二個池化層，參數與第一個池化層完全相同，本層會接收 36 個 14×14 影像後，輸出 36 個 7×7 的影像。
- 在各兩層的卷積與池化後，加入 **Dropout 層**，功用是減少過度擬合。

*在深度學習的訓練過程中，`Dropout` 會讓每次 batch run 都依據機率丟棄一定比例的神經元不予計算，使得每一次都好像在訓練不同的神經網路一樣。



- 接著我們需要一個**平坦層**，將特徵值轉為一維資料以供後續的全連結層使用。本層接收第二個池化層傳來的 $36 \times 7 \times 7 (=1764)$ 資料之後，轉為 **1 維的 1764**。建立全連結層中的**隱藏層**(即傳統神經網路中的全連結層)，指定其神經元數目為 **128 個**，
- 一樣再加入一層 **Dropout** 來防止過度擬合。(此層亦可省略，我們可視執行結果來決定)
- 最後，本模型的最後一層是**輸出層**，也就是要輸出十種 0~9 的 **10 種分類值**。

文中測試如果多加一層捲積/池化以及全連結的隱藏層，效果比起直接增加捲積的 filter 數目來得更好，辨識率提高到 99.31%。

複習 Day099_data_augmentation_Review 課程內容，這邊會針對資料量不足 另外提到資料預處理和資料擴張的部分。為了充分利用一些訓練例子，我們將透過一系列的隨機變換“增強”它們，從而讓模型絕不會兩次看到完全相同的圖像。這有助於防止過度擬合，並說明該模型更好的泛化能力。在 Keras 中，這可以用 `keras.preprocessing.image.ImageDataGenerator` 類來執行^{#4.3.5}。<https://ppt.cc/fKHsZx>

#4.3.1 <機器學習的衰頹興盛：從類神經網路到淺層學習> <https://ppt.cc/fpHzJx>

#4.3.2 <Wiki 深度學習> <https://zh.wikipedia.org/wiki/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0>

#4.3.3 <CNN (卷積神經網絡)、RNN (循環神經網絡)、DNN (深度神經網絡) 的內部網絡結構的區別> <https://kknews.cc/news/6n84mm3.html>

#4.3.4 <學習使用 Keras 建立卷積神經網路> <https://ppt.cc/f8wb1x>

#4.3.5 <使用非常少的数据构建强大的图像分类模型> <https://ppt.cc/fKHsZx>

比較範例一 from Day097_Keras_CNN_vs_DNN_Review

Code 21: Day097_Keras_CNN_vs_DNN_Review.ipynb

DNN (MLP)

由於 DNN 只能輸入一維的資料，我們要先將影像進行攤平，若 $(50000, 32, 32, 3)$ 的影像，攤平後會變成 $(50000, 32323) = (50000, 3072)$

```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.optimizers import RMSprop, Adam

batch_size = 128 #batch 的大小，如果出現 OOM error 請降低這個值
num_classes = 10 # 類別的數量，Cifar 10 共有 10 個類別
epochs = 10 # 訓練的 epochs 數量
```

```
# 讀取資料並檢視
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')

# 對 label 進行 one-hot encoding (y_train 原本是純數字)
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# 將資料攤平成一維資料
x_train = x_train.reshape(50000, 3072)
x_test = x_test.reshape(10000, 3072)

# 將資料變為 float32 並標準化
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

```
model = Sequential()
model.add(Dense(512, activation='relu',
               input_shape=(3072,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
```

CNN

```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import RMSprop, Adam
```

```
batch_size = 128 #batch 的大小，如果出現 OOM error 請降低這個值
num_classes = 10 # 類別的數量，Cifar 10 共有 10 個類別
epochs = 10 # 訓練的 epochs 數量
```

```
# 讀取資料並檢視
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                input_shape=x_train.shape[1:])) ← 卷積層
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu')) ← 池化層
model.add(MaxPooling2D(pool_size=(2, 2))) ←
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

```
model.compile(loss='categorical_crossentropy',
```

```

metrics=['accuracy'])

history = model.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1,
                     validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	1573376
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
Total params: 1,841,162		
Trainable params: 1,841,162		
Non-trainable params: 0		

Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [=====] - 5s 93us/step -
loss: 2.1639 - acc: 0.2466 - val_loss: 1.8197 - val_acc: 0.3513
Epoch 2/10
50000/50000 [=====] - 2s 43us/step -
loss: 1.8590 - acc: 0.3265 - val_loss: 1.8009 - val_acc: 0.3321
... ...
Epoch 9/10
50000/50000 [=====] - 2s 43us/step -
loss: 1.6143 - acc: 0.4241 - val_loss: 1.5522 - val_acc: 0.4561
Epoch 10/10
50000/50000 [=====] - 2s 42us/step -
loss: 1.6003 - acc: 0.4278 - val_loss: 1.5236 - val_acc: 0.4728
Test loss: 1.5236255004882813
Test accuracy: 0.4728

```

optimizer=RMSProp(),
metrics=['accuracy'])

history = model.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1,
                     validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
activation_1 (Activation)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
activation_2 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_3 (Dropout)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 15, 15, 64)	18496
activation_3 (Activation)	(None, 15, 15, 64)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	36928
activation_4 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_4 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_4 (Dense)	(None, 512)	1180160
activation_5 (Activation)	(None, 512)	0
dropout_5 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 10)	5130
activation_6 (Activation)	(None, 10)	0
Total params: 1,250,858		
Trainable params: 1,250,858		
Non-trainable params: 0		

Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [=====] - 6s 127us/step -
loss: 1.7666 - acc: 0.3592 - val_loss: 1.3987 - val_acc: 0.5000
... ...
Epoch 10/10
50000/50000 [=====] - 4s 79us/step -
loss: 0.6736 - acc: 0.7655 - val_loss: 0.7365 - val_acc: 0.7479
Test loss: 0.7365292016983033
Test accuracy: 0.7479

同樣運算 10 個 epochs，但 CNN 在 test data 的準確率顯著優於 DNN!

實務情境上補充說明 – 資料量過大或過小的處理技巧

資料量過大

Cifar-10 資料集相對於常用到的影像來說是非常小，所以可以先把資料集全部讀進記憶體裡面，要使用時直接從記憶體中存取，速度會相當快。複習 Day098_Python_generator 課程內容，但是如果我們要處理理的資料集超過電腦記憶體的容量量呢？桌上電腦的記憶體多為 32, 64, 128 GB，當處理理超大圖片、3D 影像或影片時，就可能遇到 Out of Memory error。

因此，如同訓練神經網路時，Batch (批次) 的概念一樣。我們可以將資料一批一批的讀進記憶體，當從 GPU/CPU 訓練完後，將這批資料從記憶體釋出，在讀取下一批資料

如何用 Python 撰寫批次讀取資料的程式碼

- 使用 Python 的 generator 來幫你完成這個任務！
- Generator 可以使用 next(your_generator) 來執行下一次循環
- 假設有一個 list，其中有 5 個數字，我們可以撰寫一個 generator，用 next(generator) 會自動吐出 list 的第一個數字，再用第二次 next 則會吐出第二個數字，以此類推

Generator 可以使用 next 來進行循環中的一步

文字上有點難解釋，直接來看範例就能了解什麼是 Generator!

撰寫一個 Generator，一次吐出 list 中的一個值

```
def output_from_list_generator(your_list):
    for i in your_list:
        yield i

my_list = [1, 2, 3, 4, 5]

gen = output_from_list_generator(my_list)

print(next(gen))
1
print(next(gen))
2
print(next(gen))
3
print(next(gen))
4
print(next(gen))
5
print(next(gen))

StopIteration
```

從上面的範例程式碼我們可以看到，當使用一次 next(generator) 就會跑 for_loop 一次，因此得到 list 中的第一個值，當再使用一次後，for_loop 記得上次的循環，所以吐出第二個值，最後一次，因為 for loop 已經執行結束了，所以再使用 next 就會看到 StopIteration，無法在得到值

我們可以撰寫一個無限循環的 Generator，只要使用 While True 即可

```
def inf_loop_generator(your_list):
    while True:
        for i in your_list:
            yield i

gen = inf_loop_generator(my_list)

print(next(gen))
1
print(next(gen))
2
print(next(gen))
3
print(next(gen))
4
print(next(gen))
5
print(next(gen))
1
print(next(gen))
2
```

上面的程式碼因為我們使用了 While True，所以 for loop 不會結束，只要 call next 就一定會跑一次循環，並返回值

雖然 Cifar-10 的資料可以全部讀進記憶體，但讓我們試著用 Generator，批次的把 Cifar 10 的資料取出來，一次取 32 張出來！

Code 22: Day098_Python_generator_Ans.ipynb

```
def img_combine(img, ncols=8, size=1, path=False):
    from math import ceil
    import matplotlib.pyplot as plt
    import numpy as np
    nimg = len(img)
    nrows = int(ceil(nimg / ncols))
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, sharex=True, sharey=True, figsize=(ncols * size, nrows * size))
    if nrows == 0:
        return
    elif ncols == 1:
        for r, ax in zip(np.arange(nrows), axes):
            nth=r
            if nth < nimg:
                ax.imshow(img[nth], cmap='rainbow', vmin=0, vmax=1)

            ax.set_axis_off()
    elif nrows == 1:
        for c, ax in zip(np.arange(ncols), axes):
            nth=c
```

```

if nth < nimg:
    ax.imshow(img[nth], cmap='rainbow', vmin=0, vmax=1)
    ax.set_axis_off()
else:
    for r, row in zip(np.arange(nrows), axes):
        for c, ax in zip(np.arange(ncols), row):
            nth=r*ncols+c
            if nth < nimg:
                ax.imshow(img[nth], cmap='rainbow', vmin=0, vmax=1)
                ax.set_axis_off()
plt.show()

```

```

from keras.datasets import cifar10
(x_train, x_test), (y_train, y_test) = cifar10.load_data()

```

```

def cifar_generator(image_array, batch_size=32):
    while True:
        for indexes in range(0, len(image_array), batch_size):
            images = x_train[indexes: indexes+batch_size]
            labels = x_test[indexes: indexes+batch_size]
            yield images, labels

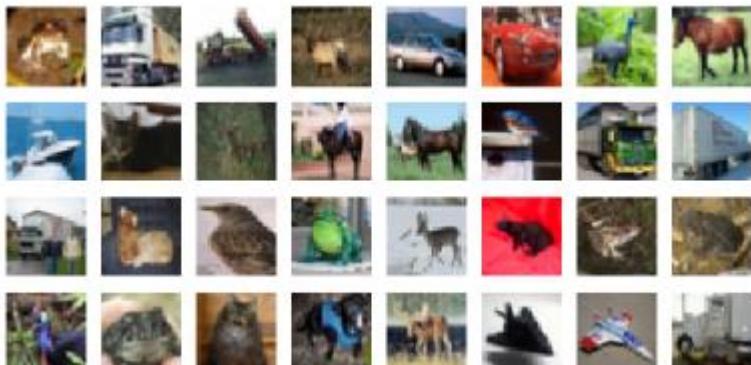
```

```
cifar_gen = cifar_generator(x_train)
```

```

images, labels = next(cifar_gen)
img_combine(images)

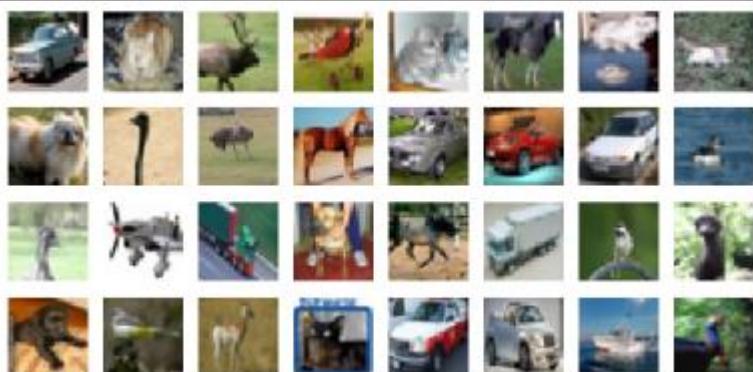
```



```

images, labels = next(cifar_gen)
img_combine(images)

```



可以看到兩次的圖片並不一樣，
這樣就可以開始訓練囉！

練習作業：承比較範例一 CNN 的結果，我們將訓練資料讀取方式改寫成 Generator，並將原本的 model.fit 改為 model.fit_generator 來進行訓練。參考 Keras 官方文件中 fit_generator 的說明

- 將原本 Python function 中的 return 改為 yield，這樣 Python 就知道這是一個 Generator 囉

```
from sklearn.utils import shuffle
def my_generator(x, y, batch_size):
    while True:
        for idx in range(0, len(x), batch_size): # 讓 idx 從 0 開始，一次增加 batch size。假設 batch_size=32, idx = 0, 32, 64, 96, ...
            batch_x, batch_y = x[idx:idx+batch_size], y[idx:idx+batch_size]
            yield batch_x, batch_y
        x, y = shuffle(x, y) # loop 結束後，將資料順序打亂再重新循環
```

```
train_generator = my_generator(x_train, y_train, batch_size) # 建立好我們寫好的 generator
```

```
history = model.fit_generator(train_generator,
                               steps_per_epoch=int(len(x_train)/batch_size), # 一個 epochs 要執行幾次 update，通常是資料量除以 batch size
                               epochs=epochs,
                               verbose=1,
                               validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
Epoch 1/10
390/390 [=====] - 9s 23ms/step - loss: 1.7365 - acc: 0.3744 - val_loss: 1.4217 - val_acc: 0.4895
Epoch 2/10
390/390 [=====] - 5s 12ms/step - loss: 1.2902 - acc: 0.5443 - val_loss: 1.0764 - val_acc: 0.6251
Epoch 3/10
390/390 [=====] - 4s 11ms/step - loss: 1.0756 - acc: 0.6242 - val_loss: 0.9013 - val_acc: 0.6901
Epoch 4/10
390/390 [=====] - 5s 12ms/step - loss: 0.9430 - acc: 0.6687 - val_loss: 0.8646 - val_acc: 0.7029
Epoch 5/10
390/390 [=====] - 5s 12ms/step - loss: 0.8599 - acc: 0.7014 - val_loss: 0.7568 - val_acc: 0.7347
Epoch 6/10
390/390 [=====] - 5s 12ms/step - loss: 0.7947 - acc: 0.7237 - val_loss: 0.8461 - val_acc: 0.7167
Epoch 7/10
390/390 [=====] - 5s 12ms/step - loss: 0.7403 - acc: 0.7425 - val_loss: 0.7498 - val_acc: 0.7458
Epoch 8/10
390/390 [=====] - 5s 12ms/step - loss: 0.6996 - acc: 0.7601 - val_loss: 0.7422 - val_acc: 0.7505
Epoch 9/10
390/390 [=====] - 4s 11ms/step - loss: 0.6782 - acc: 0.7674 - val_loss: 0.6794 - val_acc: 0.7682
Epoch 10/10
390/390 [=====] - 4s 11ms/step - loss: 0.6565 - acc: 0.7741 - val_loss: 0.6455 - val_acc: 0.7810
Test loss: 0.6454871427536011
Test accuracy: 0.781
```

資料量不足 1

承比較範例一 CNN 的結果，複習 Day099_data_augmentation_Review 課程內容，實務上進行各種機器學習專案時，我們也經常會遇到資料量不足的情形，常見原因：資料搜集困難或是成本極高、資料標註不易、資料品質不佳，除了繼續搜集資料以外，資料增強 (Data augmentation) 是很常見的方法之一。

對影像進行一些隨機的處理如翻轉、平移、旋轉、改變亮度等各樣的影像操作，藉此將一張影像增加到多張



注意資料增強並非萬靈丹！適度的資料增強通常都可以提升準確率。選用的增強方法則須視資料集而定，例如人臉辨識就不太適合用上下翻轉，因為實際使用時不會有上下顛倒的臉部，另外需特別注意要先對資料做 train/test split 後再做資料增強！否則其實都是同樣的影像，誤以為模型訓練得非常好。

Data augmentation in Keras

Keras 提供許多常用的資料增強函數方便使用

```
from keras.preprocessing.image import ImageDataGenerator  
  
Aug_generator =  
ImageDataGenerator(rotation_range=30,  
width_shift_range=0.1, height_shift_range=0.1)  
  
以上的 Generator 會對圖片做隨機的旋轉正負 30 度，垂直&水平 平移 10% pixels
```

如同名稱顯示的，這是一個 Generator，要使用 next(generator) 來取出做完資料增強的影像

```
Augmented_image = next(Aug_generator)
```

常見問題

Q: 跑資料增強時程式碼好像都會出錯？

A: 要特別注意，資料增強應該在圖像標準化之前完成 (e.g. 除以 255、減去平均值)！因為多數資料增強的函數都以圖像為一般的 0-255 RGB 影像為主，若已經先經過標準化，有可能造成程式碼錯誤

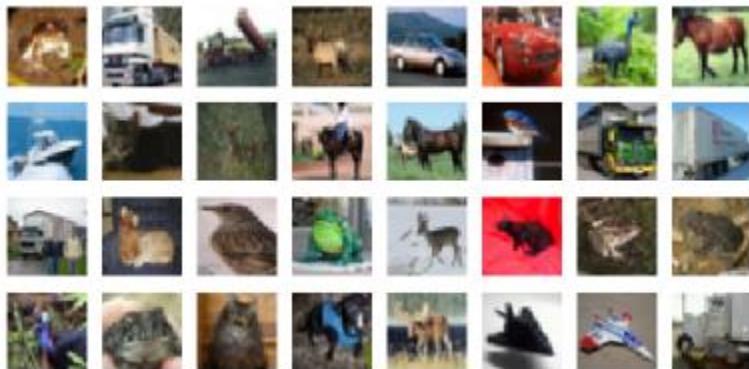
Code 23: Day099_data_augmentation_Review.ipynb

```
# 載入套件  
from keras.datasets import cifar10  
from keras.preprocessing.image import ImageDataGenerator  
%matplotlib inline  
  
Using TensorFlow backend.  
# 此函數會幫我們把多張影像畫成一張多宮格圖  
def img_combine(img, ncols=8, size=1, path=False):  
    from math import ceil  
    import matplotlib.pyplot as plt  
    import numpy as np  
    nimg = len(img)  
    nrows = int(ceil(nimg/ncols))  
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, sharex=True, sharey=True, figsize=(ncols*size,nrows*size))  
    if nrows == 0:  
        return  
    elif ncols == 1:  
        for r, ax in zip(np.arange(nrows), axes):  
            nth=r  
            if nth < nimg:  
                ax.imshow(img[nth], cmap='rainbow', vmin=0, vmax=1)  
  
            ax.set_axis_off()  
    elif nrows == 1:  
        for c, ax in zip(np.arange(ncols), axes):  
            nth=c  
            if nth < nimg:  
                ax.imshow(img[nth], cmap='rainbow', vmin=0, vmax=1)  
            ax.set_axis_off()  
    else:  
        for r, row in zip(np.arange(nrows), axes):  
            for c, ax in zip(np.arange(ncols), row):  
                nth=r*ncols+c  
                if nth < nimg:
```

```
ax.imshow(img[nth], cmap='rainbow', vmin=0, vmax=1)
ax.set_axis_off()
plt.show()
```

```
# 讀取 Cifar-10 資料集
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
# 取前 32 張圖片做視覺化
images = x_train[:32]
img_combine(images)
```



```
# 建立 ImageDataGenerator，並指定我們要做資料增強的數值範圍
```

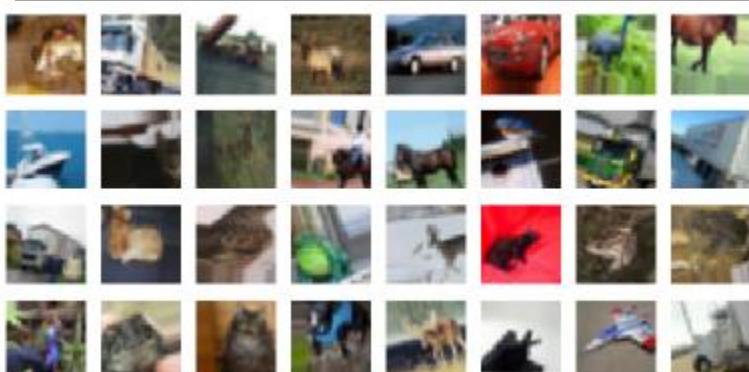
```
data_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
```

注意！！ImageDataGenerator 是一個 Generator (生成器)！對 Generator 不熟悉的同學請回到 Day098 做複習。

使用 flow 後，就會對我們的影像進行增強，再 call next 取出 generator 的圖像。(shuffle=False 因為我們希望圖像的順序不要改變，方便觀察。實際訓練時預設是 shuffle=True)

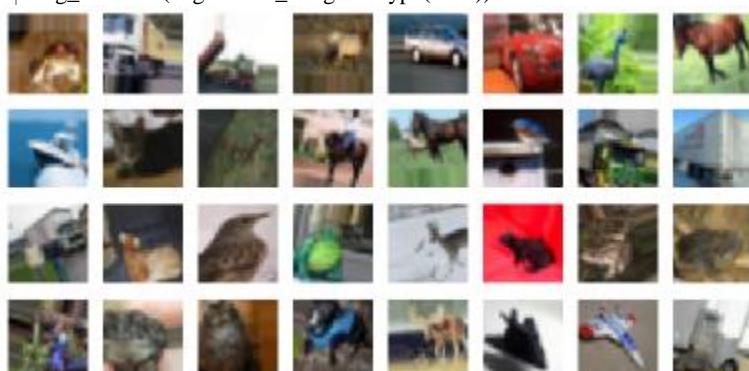
```
augmented_images = next(data_generator.flow(images, shuffle=False))
```

```
img_combine(augmented_images.astype("int")) # 注意在訓練時神經網路時，圖像資料必須要是 float32，但在做視覺化時要轉為 int 才能順利畫圖。所以為了畫圖才把資料轉為 int
```



```
# 因為隨機性的關係，所以一樣的圖像再經過一次 generator 後的結果不一定相同
```

```
augmented_images = next(data_generator.flow(images, shuffle=False))
img_combine(augmented_images.astype("int"))
```



練習作業：

```
# 使用 ImageDataGenerator 來進行 Cifar-10 資料集的訓練，是否比上述的 CNN 結果還佳？
```

```
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import RMSprop, Adam
```

```
batch_size = 128 # batch 的大小，如果出現 OOM error，請降低這個值
```

```
num_classes = 10 # 類別的數量，Cifar 10 共有 10 個類別
```

```
epochs = 10 # 訓練的 epochs 數量
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
# Convert class vectors to binary class matrices.
```

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

```
augment_generator = ImageDataGenerator(rotation_range=10, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
```

```
history = model.fit_generator(augment_generator.flow(x_train, y_train, batch_size=batch_size),
                               steps_per_epoch=int(len(x_train)/batch_size), # 一個 epochs 要執行幾次 update，通常是資料量除以 batch size
                               epochs=epochs,
                               verbose=1,
                               validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

(略)

```
Test loss: 0.7168990790367127
Test accuracy: 0.7493
```

因為在資料中增加了隨機性，因此通常會需要更多的訓練時間來達到更好的結果，可以看到 Training loss 比起原本沒有使用資料增強的 Loss 來得高一些，讓模型盡可能地增加泛化性，所以在 test data 上面通常可以獲得更好的結果。

補充資料

- 延伸閱讀連結：[Keras ImageDataGenerator 範例與介紹](#)
- 若你覺得 Keras 官方的資料增強函數太少，也可以使用這個非常紅的套件：[imgaug](#)，有非常多實作好的影像增強函數，使用方法也與 Keras 一樣，十分方便。

imgaug

This python library helps you with augmenting images for your machine learning projects. It converts a set of input images into a new, much larger set of slightly altered images.

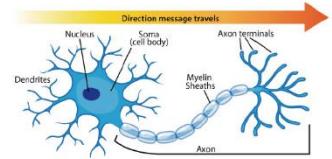
[build passing](#) [codecov 96%](#) [code quality A](#)

	Image	Heatmaps	Seg. Maps	Keypoints	Bounding Boxes, Polygons
Original Input					
Gauss. Noise + Contrast + Sharpen					
Affine					
Crop + Pad					
Flplr + Perspective					

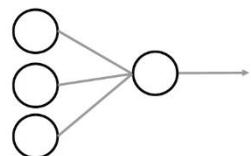
4-4 深度神經網路及多層感知(Multi-layer Perception, MLP)

類神經網路 (Neural Network)

- 在1956年的達特茅斯會議中誕生，以數學模擬神經傳導輸出預測，在初期人工智慧領域中就是重要分支
- 因層數一多計算量就大幅增加等問題，過去無法解決，雖不斷有學者試圖改善，在歷史中仍不免大起大落
- 直到近幾年在**算法、硬體能力與巨量資料**的改善下，多層的類神經網路才重新成為當前人工智慧的應用主流



圖片來源：mropengate.blogspot



類神經網路與深度學習的比較

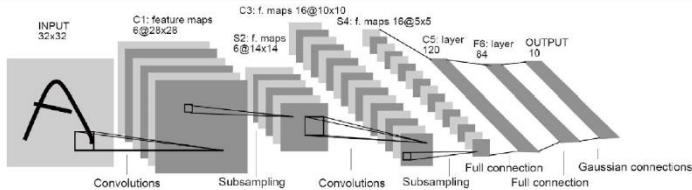
- 就基礎要素而言，深度學習是**比較多層**的類神經網路
- 但就實務應用的層次上，因著設計思路與連結架構的不同，兩者有了很大的差異性

類神經網路 (Neural Network)	深度學習 (Deep Learning)
隱藏層數量	1~2層
活躍年代	1956~1974
代表結構	感知器 (Perceptron) 啟動函數 (Activation Function)
解決問題	基礎迴歸問題
	影像、自然語言處理等多樣問題

深度學習應用爆發的三大關鍵

- 類神經的應用曾沉寂二三十年，直到2012年AlexNet在ImageNet圖像分類競賽獲得驚艷表現後，才重回主流舞台
- 深度學習相比於過去，到底有哪些關鍵優勢呢？
 - 算法改良**
 - 網路結構：CNN與RNN等結構在神經連結上做有意義的精省，使得計算力得以用在刀口上
 - 細節改良：DropOut(隨機移除)同時有節省連結與集成的效果，BatchNormalization(批次正規化)讓神經層間有更好的傳導力
 - 計算機硬體能力提升**
 - 圖形處理器(GPU)的誕生，持續了晶片摩爾定律，讓計算成為可行
 - 巨量資料**
 - 個人行動裝置的普及及網路速度的持續提升，帶來巨量的資料量，使得深度學習有了可以學習的素材

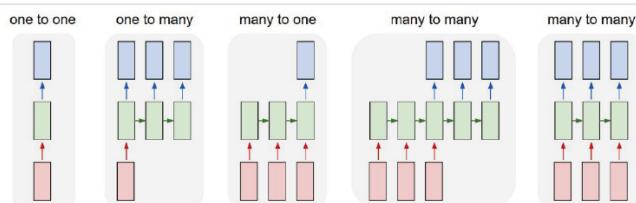
卷積神經網路 (CNN, Convolutional Neural Network)



圖片來源：adeshpande3

- **設計目標：**影像處理
- **結構改進：**CNN 參考**像素遠近省略**神經元，並且用影像特徵的**平移不變性**來共用權重，大幅減少了影像計算的負擔
- **衍伸應用：**只要符合上述兩種特性的應用，都可以使用 CNN 來計算，例如 AlphaGo 的 v18 版的兩個主網路都是 CNN

遞歸神經網路 (RNN, Recurrent Neural Network)



圖片來源：karpathy

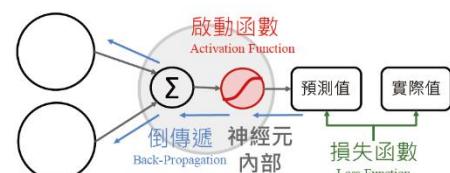
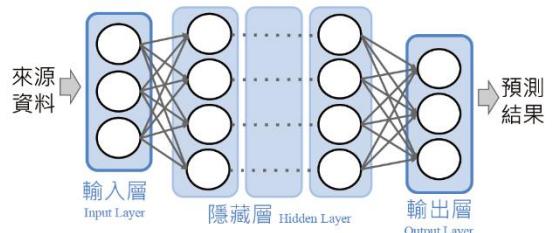
- **設計目標：**時序資料處理
- **結構改進：**RNN 雖然看似在 NN 外增加了時序間的橫向傳遞，但實際上還是依照**時間遠近省略**了部分連結
- **衍伸應用：**只要資料是**有順序性**的應用，都可以使用 RNN 來計算，近年在**自然語言處理 (NLP)** 上的應用反而成為大宗

深度學習 - 巨觀結構

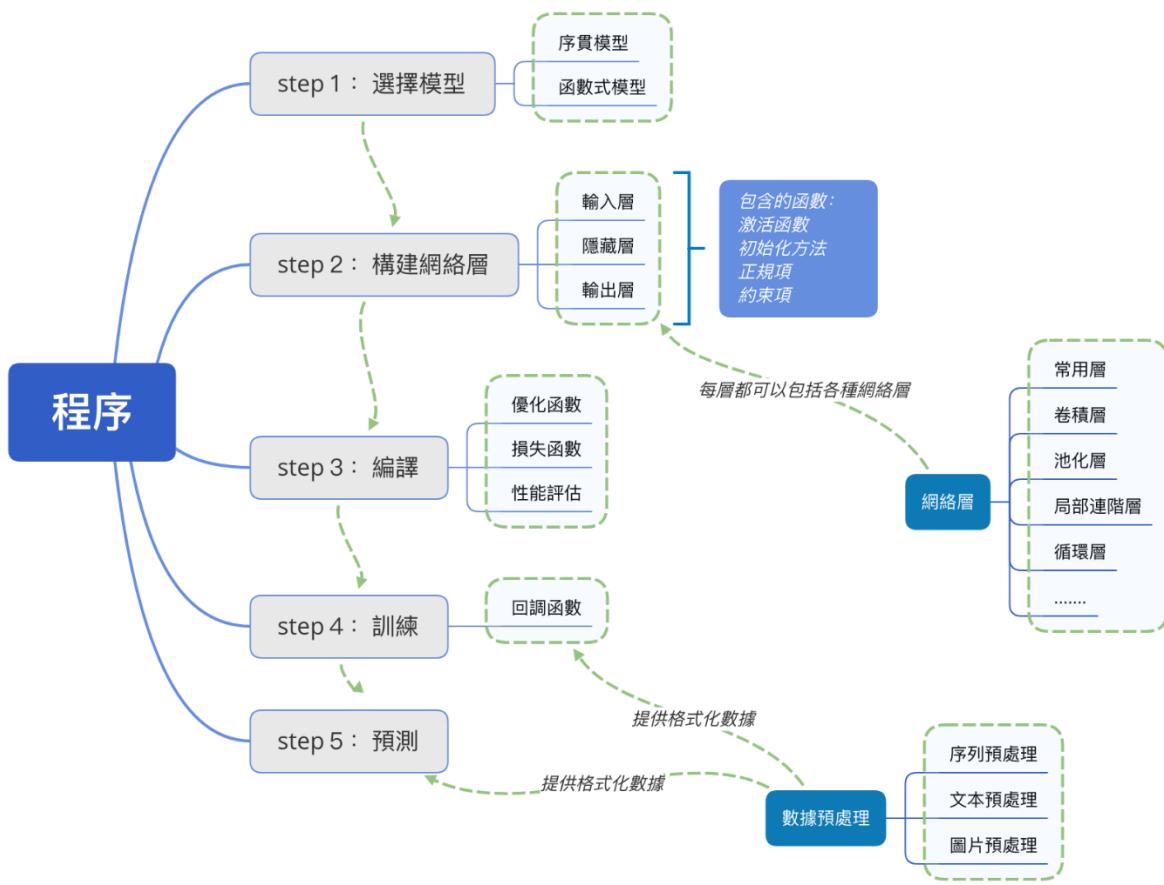
- **輸入層：**輸入資料進入的位置
- **輸出層：**輸出預測值的最後一層
- **隱藏層：**除了上述兩層外，其他層都稱為隱藏層

深度學習 - 微觀結構

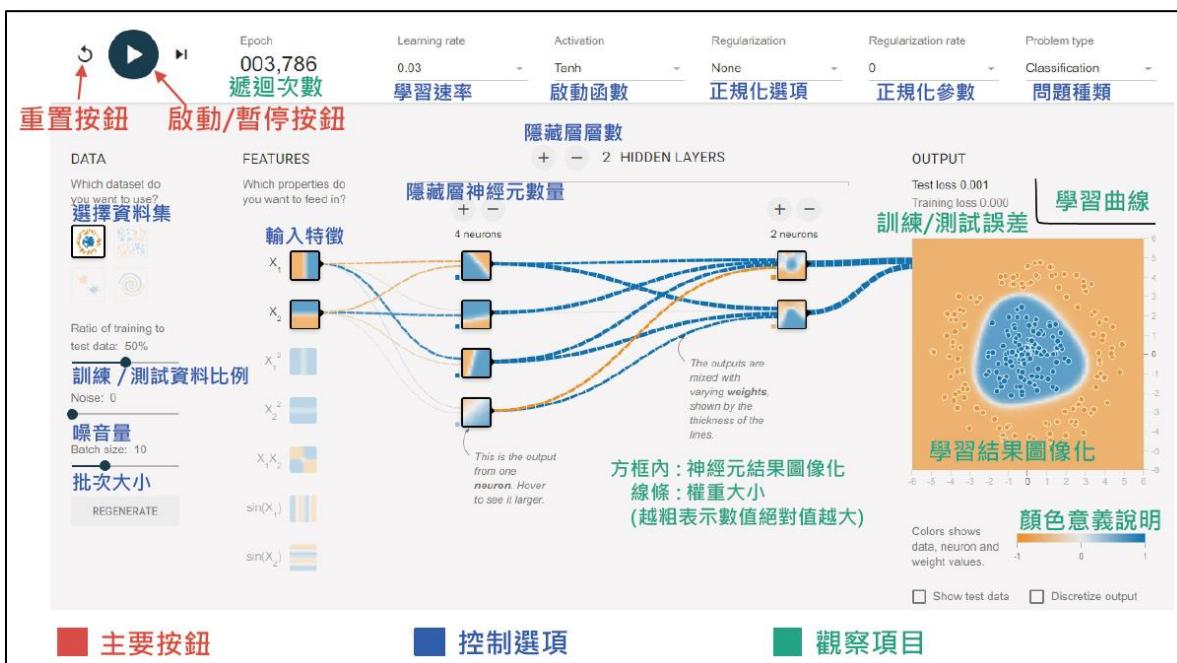
- **啟動函數(Activation Function)：**位於神經元內部，將上一層神經元的輸入總和，轉換成這一個神經元輸出值的函數
- **損失函數(Loss Function)：**定義預測值與實際值的誤差大小
- **倒傳遞Back-Propagation)：**將損失值，轉換成類神經權重更新的方法



執行程序藍圖與體驗



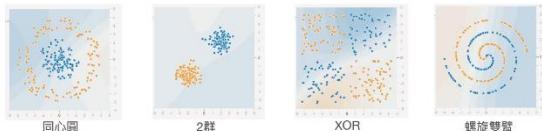
Day64 更新教材，透過 **TensorFlowPlayGround** 平台(<https://playground.tensorflow.org>)的操作，了解深度學習效果。TensorFlow PlayGround 是 Google 精心開發的體驗網頁，提供學習者在接觸語言之前，就可以對深度學習能概略了解。以下練習將逐步體驗重要概念。



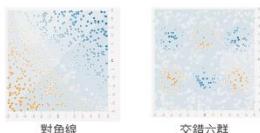
平台上的資料集

- 平台上目前有 4 個分類問題與 2 個迴歸問題，要先切換右上問題種類後，再選擇左上的資料集

- 分類



- 迴歸



練習 2：增減隱藏層數

- 練習操作

- 資料集切換：分類資料集(左下)-2 群，**調整層數**後啟動學習
- 資料集切換：分類資料集(左上)-同心圓，**調整層數**後啟動學習
- 資料集切換：迴歸資料集(左)-對角線，**調整層數**後啟動學習



- 實驗結果

- 2 群與對角線：因資料集結構簡單，即使沒有隱藏層也會收斂
- 同心圓：資料及稍微複雜 (無法線性分割)，因此最少要一層隱藏層才會收斂

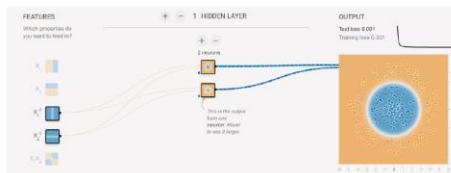
練習 4：切換不同特徵

- 練習操作

- 資料集切換：分類資料集(左上)-同心圓，隱藏層 1 層，隱藏神經元 2 個
- 切換 **任選不同的 2 個特徵** 後啟動，看看學習效果有何不同？

- 實驗結果

- 當特徵選到兩個特徵的平方時，即使中間只有 2 個神經元也會收斂



練習 5：切換批次大小

- 練習操作

- 資料集切換：分類資料集(右下)-螺旋雙臂，特徵全選，隱藏層 1 層 /8 神經元
- 調整 **不同的批次大小** 後執行 500 次遞迴，看看學習效果有何不同？

- 實驗結果

- 批次大小很小時，雖然收斂過程非常不穩定，但平均而言會收斂到較好的結果



** 註：實務上，批次大小如果極小，效果確實比較好，但計算時間會相當久，因此通常會依照時間需要而折衷

練習 1：按下啟動，觀察指標變化

- 全部使用預設值，按下啟動按鈕，看看發生了什麼變化？
 - 遞迴次數 (Epoch, 左上)：逐漸增加
 - 神經元 (中央)：方框圖案逐漸明顯，權重逐漸加粗，滑鼠移至上方會顯示權重
 - 訓練/測試誤差：開始時明顯下降，幅度逐漸趨緩
 - 學習曲線：訓練/測試誤差
 - 結果圖像化：圖像逐漸穩定
- 後續討論觀察，如果沒有特別註明，均以**訓練/測試誤差是否趨近 0**為主，這種情況我們常稱為**收斂**



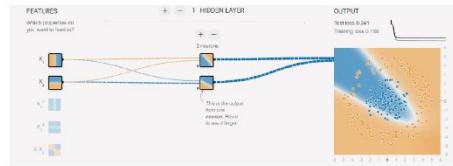
練習 3：增減神經元數

- 練習操作

- 資料集切換：分類資料集(左上)-同心圓，隱藏層設為 1 後啟動學習
- 切換 **不同隱藏層神經元數量** 後，看看學習效果有何不同？

- 實驗結果

- 當神經元少於等於兩個以下時，將無法收斂(如下圖)



複習 練習 1 - 練習 4

- 雖然圖像化更直覺，但是並非量化指標且可視化不容易，故深度學習的觀察指標仍**以損失函數/誤差為主**
- 對於不同資料類型，適合加深與加寬的問題都有，但加深適合的問題類型**較多**
- 輸入特徵的選擇影響結果甚鉅，因此深度學習也需要考慮**特徵工程**

練習 6：切換學習速率

練習操作

- 資料集切換：分類資料集(右下)-螺旋雙臂，特徵全選，隱藏層 1 層 /8 神經元，批次大小固定 10
- 調整 **不同的學習速率** 後執行 500 次遞迴，看看學習效果有何不同？

實驗結果

- 小於 0.3 時 學習速率較大時，收斂過程會越不穩定，但會收斂到較好的結果
- 大於 1 時 因為過度不穩定而導致無法收斂

選1時無法收斂



崩壞的學習曲線

練習 7：切換啟動函數

練習操作

- 資料集切換：分類資料集(右下)-螺旋雙臂，特徵全選，隱藏層 1 層 /8 神經元，學習速率固定 1
- 調整 **不同的啟動函數** 後執行 500 次遞迴，看看學習效果有何不同？

實驗結果

- 在這種極端的情形下，Tanh 會無法收斂，Relu 很快就穩定在很糟糕的分類狀態，惟有 Sigmoid 還可以收斂到不錯的結果
- 但實際上，Sigmoid 需要大量計算時間，而 Relu 則相對快得很多，這也是需要取捨的，在本例中因位只有一層，所以狀況不太明顯

最後的希望：Sigmoid



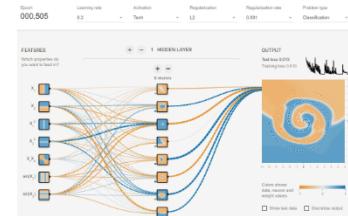
練習 8：切換正規化選項與參數

練習操作

- 資料集切換：分類資料集(右下)-螺旋雙臂，特徵全選，隱藏層 1 層 /8 神經元，批次大小固定 10，學習速率固定 0.3，啟動函數設為 Tanh
- 調整 不同的正規化選項與參數 後執行 500 次遞迴，看看學習效果有何不同？

實驗結果

- 我們已經知道上述設定本來就會收斂，只是在較小的 L1 / L2 正規化參數下收斂比較穩定一點
- 但正規化參數只要略大，反而會讓本來能收斂的設定變得無法收斂，這點 L1 比 L2 情況略嚴重，因此本例中最適合的正規化參數是 L2 + 參數 0.001
- 實際上：L1 / L2 較常使用在非深度學習上，深度學習上效果有限



複習 練習 5 - 練習 8

- 批次大小越小：學習曲線越**不穩定**、但**收斂越快**
- 學習速率越大：學習曲線越**不穩定**、但**收斂越快**，但是與批次大小不同的是 - 學習速率大於一定以上時，有可能不穩定到**無法收斂**
- 當類神經網路層數不多時，啟動函數 Sigmoid / Tanh 的效果比 Relu 更好
- L1 / L2 正規化在非深度學習上效果較明顯，而**正規化參數較小才有效果**

4-4.1 步驟一：選擇模型

4-4.1.1 序列模型搭建網路(Keras Sequential API)

Sequential 序貫模型序為最簡單的線性、從頭到尾的結構順序，一路到底。Sequential 模型的基本元件一般需要：

- Model 宣告 [Step2]
- model.add · 添加層 [Step2]
- model.compile · 模型訓練 [Step3]
- model.fit · 模型訓練參數設置 + 訓練 [Step4]
- 模型評估 [Step4]
- 模型預測 [Step5]

複習講義 D068_Keras Sequential API · sequential 是一系列模型的簡單線性疊加，可以在構造函數中傳入一些列的網路層：

語法--

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([Dense(32, _input_shape=(784,)), Activation("relu")])

也可以透過 .add

model = Sequential()
model.add(Dense(32, _input_dim=784))
model.add(Activation("relu"))
```

Sequential 的第一層(只有第一層，後面的層會自動匹配)需要知道輸入的 shape

- 在第一層加入一個 input_shape 參數，input_shape 應該是一個 shape 的 tuple 資料類型。
- input_shape 是一系列整數的 tuple，某些位置可為 None
- input_shape 中不用指明 batch_size 的數目。
- 2D 的網路層，如 Dense，允許在層的構造函數的 input_dim 中指定輸入的維度。
- 對於某些 3D 時間層，可以在構造函數中指定 input_dim 和 input_length 來實現。
- 對於某些 RNN，可以指定 batch_size。這樣後面的輸入必須是(batch_size,input_shape)的輸入

常用參數說明

名稱	作用	原型參數
Dense	實現全連接層	Dense(units,activation,use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros')
Activation	對上層輸出應用激活函數	Activation(activation)
Dropout	對上層輸出應用 dropout 以防 止過擬合	Dropout(ratio)
Flatten	對上層輸出一維化	Flatten()
Reshape	對上層輸出reshape	Reshape(target_shape)

簡單舉例程式可以參考 [4-3 資料前處理](#) 的範例。

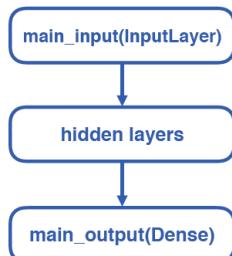
4-4.1.2 函數式模型搭建網路(Functional API (or Model))

函數式模型是最廣泛的一類模型，上一節所述的序貫模型(Sequential)只是它的一種特殊情況。如果你的模型需要多於一個的輸出，那麼你總應該選擇函數式模型。Keras函數式模型接口是用戶定義多輸出模型、非循環有向模型或具有共享層的模型等複雜模型的途徑。

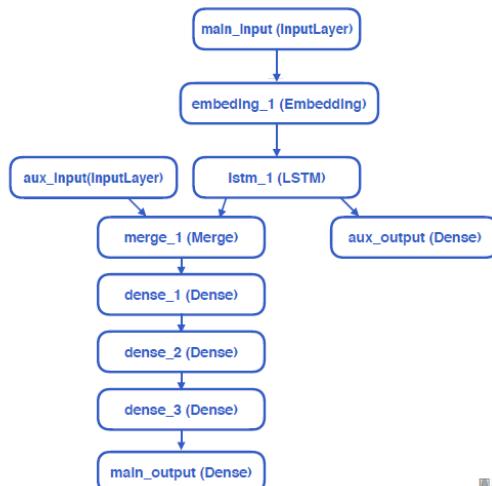
複習講義 D069_Keras Module API，使用函數式模型的一個典型場景是搭建多輸入、多輸出的模型

語法--

```
from keras.layers import Input  
from keras.models import Model  
  
main_input = Input(shape=(100,), dtype='int32', name='main_input')
```



- 層對象接受張量為參數，返回一個張量。
- 輸入是張量，輸出也是張量的一個框架就是一個模型，通過 Model 定義。
- 這樣的模型可以被像 Keras 的 Sequential 一樣被訓練



舉例說明

- 利用函數式 API，可以輕易地重用訓練好的模型：可以將任何模型看作是一個層，然後通過傳遞一個張量來調用它。注意，在調用模型時，您不僅重用模型的結構，還重用了它的權重。
- 具有多個輸入和輸出的模型。函數式 API 使處理大量交織的數據流變得容易。
- 試圖預測 Twitter 上的一條新聞標題有多少轉發和點贊數

圖片來源：keras.io

```
from keras.layers import Input, Embedding, LSTM, Dense  
from keras.models import Model  
  
# 主要輸入接收新聞標題本身，即一個整數序列（每個整數編碼一個詞）。  
# 這些整數在 1 到 10,000 之間（10,000 個詞的詞彙表），且序列長度為 100 個詞  
# 宣告一個 NAME 去定義 Input  
main_input = Input(shape=(100,), dtype='int32', name='main_input')
```

```
# Embedding 層將輸入序列編碼為一個稠密向量的序列，  
# 每個向量維度為 512。  
x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)
```

```
# LSTM 層把向量序列轉換成單個向量，  
# 它包含整個序列的上下文信息  
lstm_out = LSTM(32)(x)
```

Using TensorFlow backend.

```
WARNING:tensorflow:From C:\Users\Administrator\Anaconda3\lib\site-  
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from  
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.  
Instructions for updating:  
Colocations handled automatically by placer.
```

```
# 插入輔助損失，使得即使在模型主損失很高的情況下，LSTM 層和 Embedding 層都能被平穩地訓練  
news_output = Dense(1, activation='sigmoid', name='news_out')(lstm_out)
```

```
# 宣告 MODEL API, 分別採用自行定義的 Input/Output Layer  
model = Model(inputs=[main_input, news_input], outputs=[main_output, news_output])
```

```
model.compile(optimizer='rmsprop',  
              loss={'main_output': 'binary_crossentropy', 'news_out': 'binary_crossentropy'},  
              loss_weights={'main_output': 1., 'news_out': 0.2})
```

```
model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
main_input (InputLayer)	(None, 100)	0	
embedding_1 (Embedding)	(None, 100, 512)	5120000	main_input[0] [0]
lstm_1 (LSTM)	(None, 32)	69760	embedding_1[0] [0]
aux_input (InputLayer)	(None, 5)	0	
concatenate_1 (Concatenate)	(None, 37)	0	lstm_1[0] [0] aux_input[0] [0]
dense_1 (Dense)	(None, 64)	2432	concatenate_1[0] [0]
dense_2 (Dense)	(None, 64)	4160	dense_1[0] [0]
main_output (Dense)	(None, 1)	65	dense_2[0] [0]
aux_output (Dense)	(None, 1)	33	lstm_1[0] [0]
=====			
Total params: 5,196,450			
Trainable params: 5,196,450			
Non-trainable params: 0			

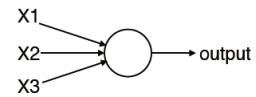
講義作業可以修改 Name 中，自定義的 Layer 名稱，再增加一層全連階層，宣告 MODEL API，分別採用自行定義的 Input/Output Layer、model.summary 查看 Layers stack

與 4-3 資料前處理的範例一樣，使用 cifar10 資料庫，可以參考 4-4.4 的程式碼。

4-4.1.3 多層感知(Multi-layer Perception, MLP)

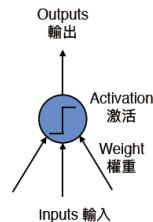
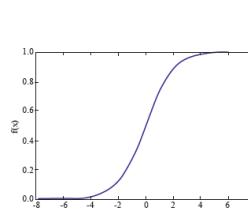
複習講義 D063 神經網路介紹，何謂「Neural Network 基本元素: 感知器」？如圖表示：

感知器可以被理解為需要多個輸入並產生一個輸出的任何東西。



上述結構需要三個輸入並產生一個輸出，那下一個邏輯問題是輸入和輸出之間的關係是什麼？**神經元模型(NN)！**

神經網路的構建模塊是人工神經元。它們都是一些簡單的具有加權輸入信號的計算單元，並且使用激活函數產生輸出信號。



重要知識點複習 : 人工神經網路定義

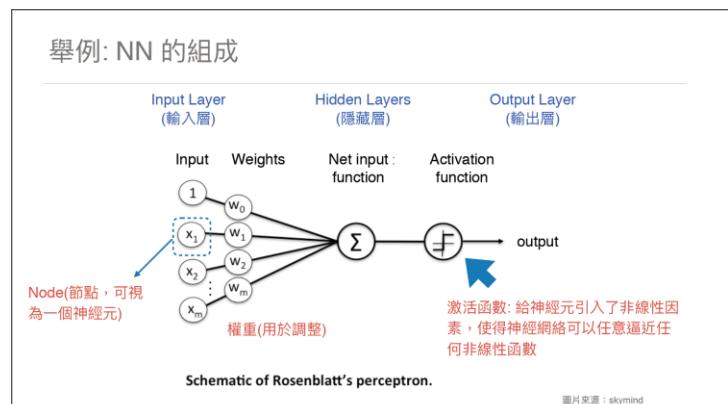
- 人工神經網路與生物神經網路的相似之處在於，它可以集體地、並列地計算函式的各個部分，而不需要描述每一個單元的特定任務
- 具有下列特點的統計模型可以被稱作是「神經化」的：
 - 具有一組可以被調節的權重（被學習演算法調節的數值參數），可調節的權重可以被看做神經元之間的連接強度
 - 可以估計輸入資料的非線性函式關係

重要知識點複習 : 人工神經網路具有以下三個部分

- **結構 (Architecture)** 結構指定了網路中的變數和它們的拓撲關係。
- **激勵函式 (Activity Rule)** 大部分神經網路模型具有一個短時間尺度的動力學規則，來定義神經元如何根據其他神經元的活動來改變自己的激勵值。
- **學習規則 (Learning Rule)** 學習規則指定了網路中的權重如何隨著時間推進而調整。這一般被看做是一種長時間尺度的動力學規則。

複習講義 D070_Multi-layer Perception 多層感知 · Multi-layer Perceptron (MLP)為一種監督式學習的演算法。此算法將可以使用非線性近似將資料分類或進行迴歸運算。

- 多層感知機其實就是可以用多層和多個 perception 來達到最後目的
- 在機器學習領域像是我們稱為 multiple classification system 或是 ensemble learning
- 深度神經網路(deep neural network, DNN) · 神似人工神經網路的 MLP
- 若每個神經元的激活函數都是線性函數，那麼，任意層數的 MLP 都可被約簡成一個等價的單層感知器

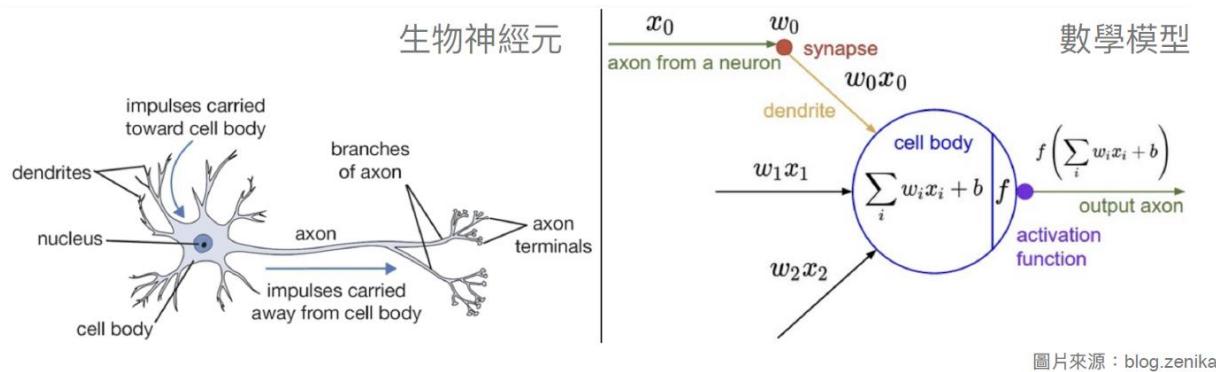


圖片來源: skymind

- MLP 優點：
 - 有能力建立非線性的模型
 - 可以使用\$partial_fit\$建立 real-time 模型
- MLP 缺點：
 - 擁有大於一個區域最小值，使用不同的初始權重，會讓驗證時的準確率浮動
 - MLP 模型需要調整每層神經元數、層數、疊代次數
 - 對於特徵的預先處理很敏感

4-4.2 步驟二：建構網路層

4-4.2.1 激活函數(Activation Function)



圖片來源：blog.zenika

複習講義 D072_Activation_function 激活函數，激活函數定義了每個節點（神經元）的輸出和輸入關係的函數為神經元提供規模化非線性化能力，讓神經網絡具備強大的擬合能力。

- 當激活函數輸出值是有限的時候，基於梯度的優化方法會更加穩定，因為特徵的表示受有限權值的影響更顯著
- 當激活函數的輸出是無限的時候，模型的訓練會更加高效

激活函數的作用

深度學習的基本原理是基於人工神經網絡，信號從一個神經元進入，經過非線性的 activation function

- 如此循環往復，直到輸出層。正是由於這些非線性函數的反覆疊加，才使得神經網絡有足夠的 capacity 來抓取複雜的 pattern

激活函數的最大作用就是非線性化

- 如果不用激活函數的話，無論神經網絡有多少層，輸出都是輸入的線性組合

激活函數的另一個重要特徵是：它應該是可以區分

- 以便在網絡中向後推進以計算相對於權重的誤差（丟失）梯度時執行反向優化策略，然後相應地使用梯度下降或任何其他優化技術優化權重以減少誤差

常用的激活函數介紹

Code 24: Day072-Activation_function.ipynb

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

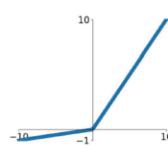
如果是 DNN 用於分類，則一般在輸出層使用 softmax 激活函數

tanh

$$\tanh(x)$$

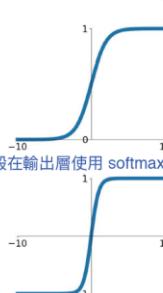
Leaky ReLU

$$\max(0.1x, x)$$



ReLU

$$\max(0, x)$$

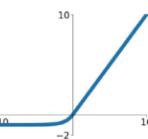


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

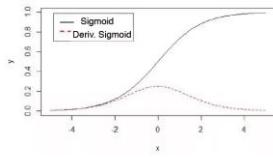


常用激活函數介紹：Sigmoid

- 特點是會把輸出限定在 0~1 之間，在 $x < 0$ ，輸出就是 0，在 $x > 0$ ，輸出就是 1，這樣使得數據在傳遞過程中不容易發散
- 兩個主要缺點
 - 一是 Sigmoid 容易過飽和，丟失梯度。這樣在反向傳播時，很容易出現梯度消失的情況，導致訓練無法完整
 - 二是 Sigmoid 的輸出均值不是 0

原函數及導數圖如下：

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

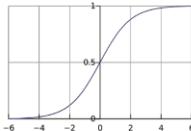


常用激活函數介紹：Softmax

- Softmax 把一個 k 維的 real value 向量 $(a_1, a_2, a_3, a_4, \dots)$ 映射成一個 $(b_1, b_2, b_3, b_4, \dots)$ 其中 b_i 是一個 $0 \sim 1$ 的常數，輸出神經元之和為 1.0，所以可以拿來做多分類的機率預測
- 為什麼要取指數
 - 第一個原因是模擬 max 的行為，所以要讓大的更大。
 - 第二個原因是需要一個可導的函數

原函數及導數圖如下：

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$



圖片來源：www.ouura.com

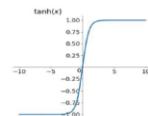
常用激活函數介紹：Tanh

- tanh 讀作 Hyperbolic Tangent
- tanh 也稱為雙切正切函數，取值範圍為 $[-1, 1]$ 。
- tanh 在特徵相差明顯時的效果會很好，在循環過程中會不斷擴大特徵效果

原函數及導數圖如下：

$$\tanh(x) = 2\sigma(2x) - 1$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



常用激活函數介紹：Sigmoid vs Softmax

- Sigmoid 將一個 real value 映射到 $(0, 1)$ 的區間，用來做二分類
- Softmax 把一個 k 維的 real value 向量 $(a_1, a_2, a_3, a_4, \dots)$ 映射成一個 $(b_1, b_2, b_3, b_4, \dots)$ 其中 b_i 是一個 $0 \sim 1$ 的常數，輸出神經元之和為 1.0，所以可以拿來做多分類的機率預測
- 二分類問題時 sigmoid 和 softmax 是一樣的，求的都是 cross entropy loss

常用激活函數介紹：Sigmoid vs Tanh

- tanh 函數將輸入值壓縮到 $-1 \sim 1$ 的範圍，因此它是 0 均值的，解決了 Sigmoid 函數的非 zero-centered 問題，但是它也存在梯度消失和累運算的問題。
- 其實 $\tanh(x) = 2\text{sigmoid}(2x) - 1$



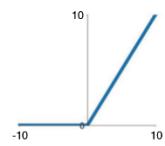
左邊是 Sigmoid 非線性函數，將實數壓縮到 $[0, 1]$ 之間。右邊是 Tanh 函數，將實數壓縮到 $[-1, 1]$ 。

常用激活函數介紹：ReLU

- 修正線性單元 (Rectified linear unit, ReLU)

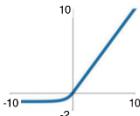
ReLU 函數公式和曲線如下：

$$f(x) = \max(0, x)$$



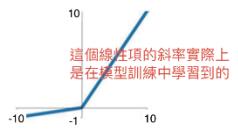
ELU 函數公式和曲線如下：

$$f(x) = \begin{cases} x, & x > 0 \\ a(e^x - 1), & x \leq 0 \end{cases}$$



PRelu 函數公式和曲線如下：

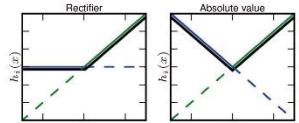
$$f(x) = \max(ax, x)$$



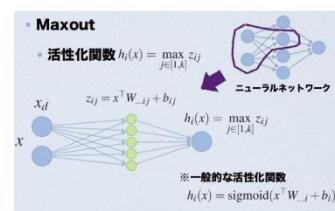
常用激活函數介紹：Maxout

- Maxout 是深度學習網絡中的一層網絡，就像池化層、卷積層一樣，可以看成是網絡的激活函數層
- Maxout 神經元的激活函數是取得所有這些「函數層」中的最大值
- Maxout 的擬合能力是非常強的，優點是計算簡單，不會過飽和，同時又沒有 ReLU 的缺點
- Maxout 的缺點是過程參數相當於多了一倍

$$f(x) = \max(wT1x+b1, wT2x+b2)$$



$$f(x) = \max(wT1x+b1, wT2x+b2)$$



圖片來源：blog.csdn.net

如何選擇正確的激活函數

根據各個函數的優缺點來配置

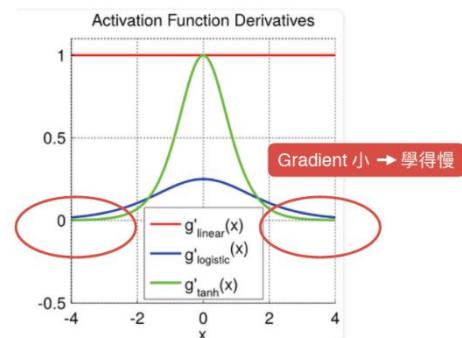
- 如果使用 ReLU，要小心設置 learning rate，注意不要讓網絡出現很多「dead」神經元，如果不好解決，可以試試 Leaky ReLU、PReLU 或者 Maxout

根據問題的性質

- 用類器時，Sigmoid 函數及其組合通常效果更好
- 由於梯度消失問題，有時要避免使用 sigmoid 和 tanh 函數。ReLU 函數是一個通用的激活函數，目前在大多數情況下使用
- 如果神經網絡中出現死神經元，那麼 PReLU 函數就是最好的選擇
- ReLU 函數建議只能在隱藏層中使用

考慮 DNN 損失函數和激活函數

- 如果使用 sigmoid 激活函數，則交叉熵損失函數一般肯定比均方差損失函數好；
- 如果是 DNN 用於分類，則一般在輸出層使用 softmax 激活函數
- ReLU 激活函數對梯度消失問題有一定程度的解決，尤其是在 CNN 模型中。
- 梯度消失 Vanishing gradient problem
 - 原因：前面的層比後面的層梯度變化更小，故變化更慢
 - 結果：Output 變化慢 → Gradient 小 → 學得慢
 - Sigmoid、Tanh 都有這樣特性
 - 不適合用在 Layers 多的 DNN 架構



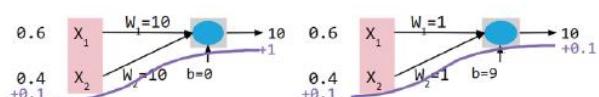
圖片來源：[the clever machine](#)

4-4.2.2 正規化(Regularization)

Regularizers

複習講義 D081_Regularization 正規化，Regularizer 的效果能讓模型參數的數值較小，使得 Inputs 的改變不會讓 Outputs 有大幅的改變。

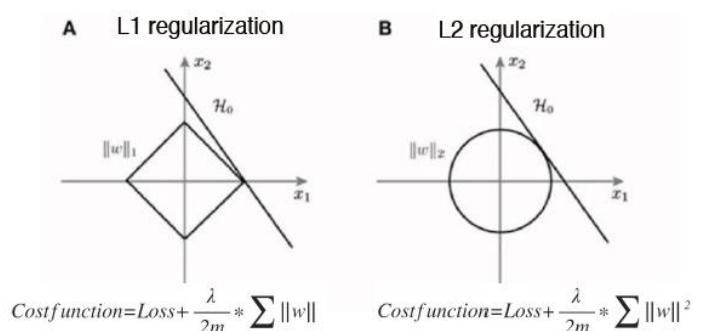
- Cost function = Loss + Regularization
- 透過 regularization，可以使的模型的 weights 變得比較小
- w_i 較小 → Δx_i 對 \hat{y} 造成的影響($\Delta \hat{y}$)較小
 - 對 input 變化比較不敏感
 - better generalization



程式設定：(以 L1 為例)

```
from keras.regularizers import l1

input_layer = keras.layers.Input(...)
keras.layers.Dense(units=n_units,
                    activation="relu",
                    kernel_regularizer=l1(0.001))(input_layer)
```



隨機缺失 (Dropout)

複習講義 D082_Dropout 隨機缺失，Dropout 在訓練時隨機將某些參數暫時設為 0 (刻意讓訓練難度提升)，強迫模型的每個參數有更強的泛化能力，也讓網路能在更多參數組合的狀態下習得表徵。

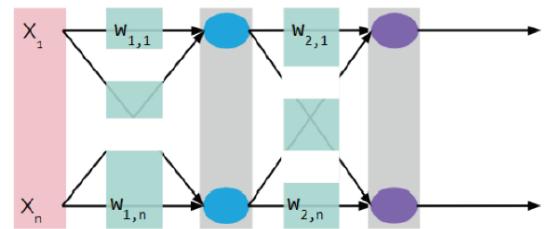
在訓練過程中，在原本全連結的前後兩層 layers，隨機拿掉一些連結(weights 設為 0)？

解釋 1：增加訓練的難度 – 當你知道你的同伴中有豬隊友時，

你會變得要更努力學習

解釋 2：被視為一種 model 自身的 ensemble 方法，

因為 model 可以有 2^n 種 weights combination



程式設定：

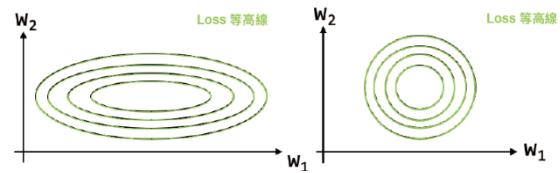
```
from keras.layers import Dropout

x = keras.layers.Dense(units=n_units,
                      activation="relu")(x)
x = Dropout(0.2)(x) # 隨機在一次 update 中，忽略 20% 的 neurons 間之 connection
```

批次正規化 (Batch Normalization)

複習講義 D083_Batch Normalization 批次正規化，除了在 Inputs 做正規化以外，Batch normalization 批次正規層讓我們能夠將每一層的輸入/輸出做正規化。各層的正規化使得 Gradient 消失 (gradient vanish) 或爆炸 (explode) 的狀況得以減輕 (但最近有 paper 對於這項論點有些爭議)

- 每個 input feature 獨立做 normalization
對於 Input 的數值，前面提到建議要 re-scale
 - Weights 修正的路徑比較會在同心圓山谷中往下滑只加在輸入層 re-scale 不夠，你可以每一層都 re-scale !!
- 利用 batch statistics 做 normalization 而非整份資料
- 同一筆資料在不同的 batch 中會有些微不同
- BN：將輸入經過 t 轉換後輸出
 - 訓練時：使用 Batch 的平均值
 - 推論時：使用 Moving Average
- 可以解決 Gradient vanishing 的問題
- 可以用比較大的 learning rate
- 加速訓練
- 取代 dropout & regularizes
- 目前大多數的 Deep neural network 都會加



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

```
 $\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$  // mini-batch mean
 $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$  // mini-batch variance
 $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$  // normalize
 $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$  // scale and shift
```

程式設定：

```
from keras.layers import BatchNormalization
x = keras.layers.Dense(units=n_units,
                      activation="relu")(x)
x = BatchNormalization()(x)
```

4-4.3 步驟三：編譯

4-4.3.1 損失函數(Loss Function)

複習講義 D071_Loss Function 損失函數，機器學習中所有的算法都需要最大化或最小化一個函數，這個函數被稱為「目標函數」。其中，我們一般把最小化的一類函數，稱為「損失函數」。它能根據預測結果，衡量出模型預測能力的好壞。損失函數大致可分為：分類問題的損失函數和回歸問題的損失函數。

損失函數為什麼是最小化呢？就是希望達到模型預測出來的東西可以跟實際的值一樣。但是，事實上，預測出來的東西基本上跟實際值都會有落差，因此，損失函數中的損失就是「實際值和預測值的落差」，表示如下：

- 在回歸問題稱為「殘差(residual)」
$$\text{loss/residual} = y - \hat{y}$$

$$\text{error rate} = \frac{\sum_{i=1}^n \text{sign}(y_i \neq \hat{y}_i)}{n}, \text{sign}(y_i \neq \hat{y}_i) = \begin{cases} 1, & y_i \neq \hat{y}_i \\ 0, & y_i = \hat{y}_i \end{cases}$$

- 在分類問題稱為「錯誤率(errorrate)」

y 表示實際值， \hat{y} 表示預測值

常用函數

損失函數的分類介紹 - mean_squared_error

- 均方誤差(mean_squared_error)：就是最小平方法(Least Square)的目標函數 -- 預測值與實際值的差距之平均值。還有其他變形的函數，如 mean_absolute_error、mean_absolute_percentage_error、mean_squared_logarithmic_error。

$$\sum (\hat{y}^2 - y^2)/N$$

- 使用時機：
 - n 個樣本的預測值 (y) 與 ($y_$) 的差距
 - Numerical 相關
- Keras 上的調用方式：
 - from keras import losses
 - model.compile(loss='mean_squared_error', optimizer='sgd')
 - 其中，包含 y_{true} ， y_{pred} 的傳遞，函數是表達如下：
 - keras.losses.mean_squared_error(y_{true} , y_{pred})

損失函數的分類介紹 - Cross Entropy

- 當預測值與實際值愈相近，損失函數就愈小，反之差距很大，就會更影響損失函數的值
- 要用 Cross Entropy 取代 MSE，因為，在梯度下時，Cross Entropy 計算速度較快，
- 使用時機：
 - 整數目標：Sparse categorical_crossentropy
 - 分類目標：categorical_crossentropy
 - 二分類目標：binary_crossentropy。
- Keras 上的調用方式：
 - from keras import losses
 - model.compile(loss='categorical_crossentropy', optimizer='sgd')
 - 其中，包含 y_{true} ， y_{pred} 的傳遞，函數是表達如下：
 - keras.losses.categorical_crossentropy(y_{true} , y_{pred})

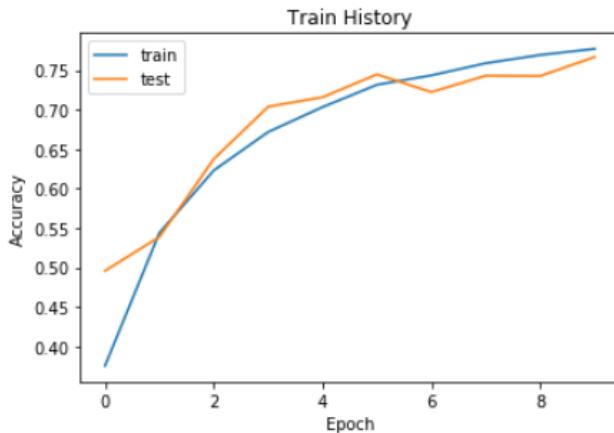
延續 4-3 資料前處理的範例，講義作業要我們使用兩種不同的函數比較如下。

Code 25: Day067_Keras_Dataset_Review.ipynb+Day71-LossFunction 比較

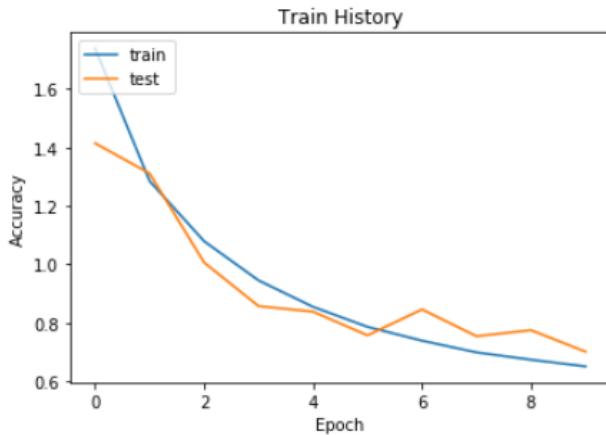
```
model.compile(loss='categorical_crossentropy',.....
```

```
Test loss: 0.7007924050331116  
Test accuracy: 0.7668
```

```
show_train_history('acc','val_acc')
```



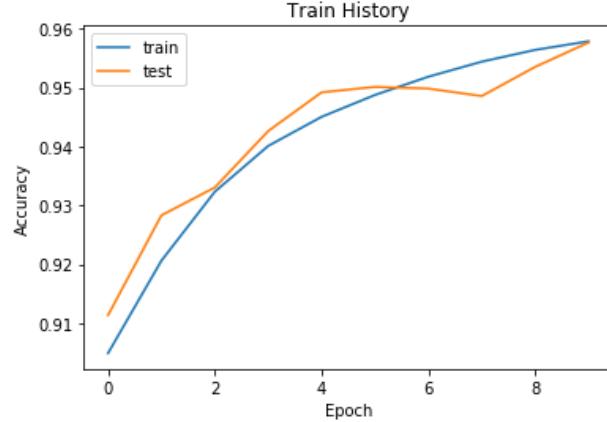
```
show_train_history('loss','val_loss')
```



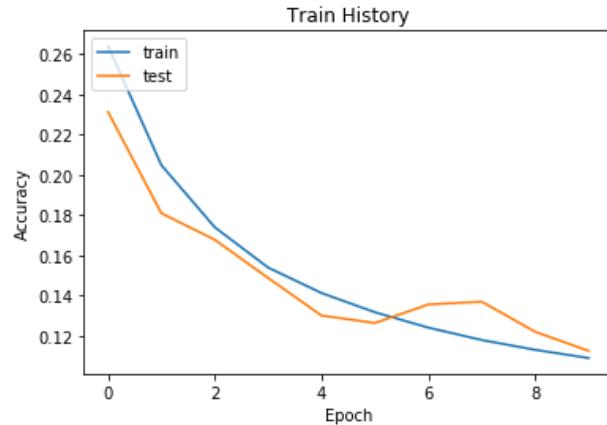
```
model.compile(loss='binary_crossentropy',.....
```

```
Test loss: 0.11257997715473175  
Test accuracy: 0.9576400003433228
```

```
show_train_history('acc','val_acc')
```



```
show_train_history('loss','val_loss')
```



討論結果：

由簡介文章介紹我們知道，損失函數追求最小化。所以在這裡局部片面的解釋，會誤以為 loss 值愈小代表模型好。但事實上，驗證模型的好壞，主要還是看 Training/Validation 的結果，如果結果有問題，此時會需要從 loss function 看，再選擇 optimizers 來做 fine tune。

以本例來說，binary_crossentropy 與 categorical_crossentropy 的差異，一個是當成 multi-label，另一個則是 multi-class，這兩個已經是完全不同的 task。在 cifar10 這樣的資料集（應為多元分類），使用 categorical_crossentropy 的計算會更嚴謹，但結果卻是 binary_crossentropy 比 categorical_crossentropy 的結果好？其實再往下分析，您會發現用 categorical_crossentropy 的計算，再修改幾個 layers(為了執行速度加快)，從 training/ Validation 在收斂之後，兩者幾乎沒有差異；反觀，binary_crossentropy 中，training/ Validation 沒有交集，從驗證結果來看，使用 categorical_crossentropy 反而比較好。

這裡影響 Training loss 最有可能的是，優化器獲得了很高的動量，並且從某個時刻開始繼續向錯誤的方向發展，碰到這個現象，我們會回到基本的 SGD 來避免動量。因此 Loss function 是一個檢查機制，不是一個評量的結果評斷。

其它函數或自己撰寫

損失函數的分類介紹: Hinge Error (hinge)

- 是一種單邊誤差，不考慮負值同樣也有多種變形，`squared_hinge`、`categorical_hinge`

$$\ell(y) = \max(0, 1 - t \cdot y)$$

- 使用時機：
 - 適用於『支援向量機』(SVM)的最大間隔分類法(maximum-margin classification)
- Keras 上的調用方式：
 - `from keras import losses`
 - `model.compile(loss='hinge', optimizer='sgd')`
 - 其中，包含`y_true`，`y_pred`的傳遞，函數是表達如下：
 - `keras.losses.hinge(y_true, y_pred)`

Custom loss function in Keras

- 在 Keras 中，除了使用官方提供的 Loss function 外，亦可以自行定義/修改 loss function
- 所定義的函數
 - 最內層函式的參數輸入須根據 output tensor 而定，舉例來說，在分類模型中需要有 `y_true`, `y_pred`
 - 需要使用 tensor operations – 即在 tensor 上運算而非在 numpy array 上進行運算
 - 回傳的結果是一個 tensor

特殊情境語法可以複習講義 D089_自己撰寫 Loss Function，舉例：

特別的案例：自定義損失函數

- 根據問題的實際情況，定制合理的損失函數
- 舉例：預測果汁日銷量問題，如果預測銷量大於實際銷量則會損失成本；如果預測銷量小於實際銷量則會損失利潤。
 - 考慮重點：製造一盒果汁的成本和銷售一盒果汁的利潤不是等價的
 - 需要使用符合該問題的自定義損失函數自定義損失函數為：

$$loss = \sum n f(y - y_{\text{true}})$$

- 損失函數表示若預測結果 y 小於標準答案 y_{true} ，損失函數為利潤乘以預測結果 y 與標準答案之差
- 若預測結果 y 大於標準答案 y_{true} ，損失函數為成本乘以預測結果 y 與標準答案之差
- Tensorflow 函數表示為：

```
loss = tf.reduce_sum(tf.where(tf.greater(y, y_), COST*(y-y_), PROFIT*(y_-y)))
```

Code 26: Day089.1_CustomizedLoss.ipynb + Day089.9_Ans.ipynb

在 Keras 中，我們可以自行定義函式來進行損失的運算。一個損失函數必須

- 有 `y_true` 與 `y_pred` 兩個輸入
- 必須可以微分
- 必須使用 tensor operation，也就是在 tensor 的狀態下，進行運算。如 `K.sum` ...

```
import keras.backend as K

def dice_coef(y_true, y_pred, smooth):
    y_pred = y_pred >= 0.5
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)

    return (2. * intersection + smooth) / \
        (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def dice_loss(smooth, thresh):
    def dice(y_true, y_pred):
        return -dice_coef(y_true, y_pred, smooth, thresh)
    return dice

dice = dice_loss(smooth=1., thresh=0.5)
```

輸出為 Tensor

4-4.3.2 優化(Optimizer)

複習講義 Day076_Optimizer 優化，機器學習算法當中，大部分算法的本質就是建立優化模型，通過最優化方法。對目標函數進行優化從而訓練出最好的模型。優化算法的功能，是通過改善訓練方式，來最小化(或最大化)損失函數 $E(x)$ 。優化策略和算法，是用來更新和計算影響模型訓練和模型輸出的網絡參數，使其逼近或達到最優值。最常用的優化算法是梯度下降(Gradient Descent)，細節會再下一節介紹。在這裡先介紹常用的優化器。

SGD-隨機梯度下降法(stochastic gradient decent)

找出參數的梯度(利用微分的方法)，往梯度的方向去更新參數(weight)。

優點：SGD 每次更新時對每個樣本進行行梯度更新，對於很大的數據集來說，可能會有相似的樣本，而 SGD 一次只進行一次更新，就沒有冗餘，而且比較快。

缺點：但是 SGD 因為更新比較頻繁，會造成 cost function 有嚴重的震盪。

程式設定：

- keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
 - lr : <float> 學習率。
 - Momentum 動量 : <float> 參數，用於加速 SGD 在相關方向上前進，並抑制震盪
 - Decay(衰變) : <float> 每次參數更新後學習率衰減值。
 - nesterov : 布爾值。是否使用 Nesterov 動量。

```
from keras import optimizers
```

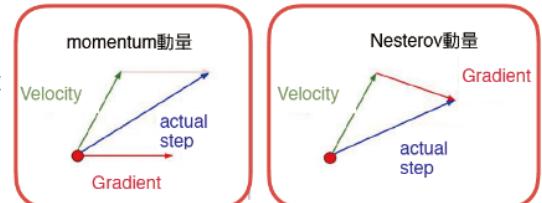
```
model = Sequential()  
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))  
model.add(Activation('softmax'))
```

```
# 實例化一個優化器對象，然後將它傳入 model.compile()，可以修改參數  
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)  
model.compile(loss='mean_squared_error', optimizer=sgd)
```

```
# 通過名稱來調用優化器，將使用優化器的默認參數。  
model.compile(loss='mean_squared_error', optimizer='sgd')
```

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

w 為權重(weight)參數，L 為損失函數(loss function)， η 是學習率(learning rate)， $\frac{\partial L}{\partial w}$ 是損失函數對參數的梯度(微分)



SGD, mini-batch gradient descent

batch-gradient，其實就是普通的梯度下降算法但是採用批量處理。

- 當數據集很大 (比如有 100000 個左右時)，每次 iteration 都要將 1000000 個數據跑一遍，機器帶不動。於是有了 mini-batch-gradient —— 將 1000000 個樣本分成 1000 份，每份 1000 個，都看成一組獨立的數據集，進行 forward_propagation 和 backward_propagation。

在整個算法的流程中，cost function 是局部的，但是 W 和 b 是全局的。

- 批量梯度下降對訓練集上每一個數據都計算誤差，但只在所有訓練數據計算完成後才更新模型。
- 對訓練集上的一次訓練過程稱為一代 (epoch)。因此，批量梯度下降是在每一個訓練 epoch 之後更新模型。

程式設定：

- epoch、iteration、batchsize，mini-batch
- batchsize：批量大小，即每次訓練在訓練集中取batchsize個樣本訓練；
 - batchsize=1；
 - batchsize = mini-batch；
 - batchsize = whole training set
- iteration：1個iteration等於使用batchsize個樣本訓練一次；
- epoch：1個epoch等於使用訓練集中的全部樣本訓練一次；

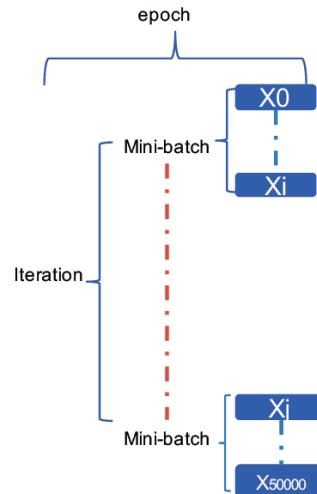
Example:

features is (50000, 400)

labels is (50000, 10)

batch_size is 128

Iteration = $50000/128+1 = 391$



怎麼配置 mini-batch 梯度下降

- Mini-batch sizes，簡稱為「batch sizes」，是算法設計中需要調節的參數。
- 較小的值讓學習過程收斂更快，但是產生更多噪聲。
- 較大的值讓學習過程收斂較慢，但是準確的估計誤差梯度。
- batch size 的默認值最好是 32 盡量選擇 2 的冪次方，有利於 GPU 的加速。
- 調節 batch size 時，最好觀察模型在不同 batch size 下的訓練時間和驗證誤差的學習曲線。
- 調整其他所有超參數之後再調整 batch size 和學習率。

Adagrad

對於常見的數據給予比較小的學習率去調整參數，對於不常見的數據給予比較大的學習率調整參數。

- 每個參數都有不同的 learning rate。
- 根據之前所有 gradient 的 root mean square 修改。

$\text{第 } t \text{ 次更新} \\ g^t = \frac{\partial L}{\partial \theta} _{\theta=\theta^t}$	$\text{Gradient descent} \\ \theta^{t+1} = \theta^t - \eta g^t$	Adagrad $\theta^{t+1} = \theta^t - \frac{\eta}{\sigma^t} g^t$ $\sigma^t = \sqrt{\frac{(g^0)^2 + \dots + (g^t)^2}{t+1}}$ <p>Root mean square(RMS) of all Gradient</p>
---	---	--

優點：Adagrad 的是減少了學習率的手動調節。

缺點：它的缺點是分母會不斷積累，這樣學習率就會收縮並最終會變得非常小。

程式設定：

- 超參數設定值:

```
keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
```

- lr : float >= 0. 學習率.一般 η 就取 0.01
- epsilon : float >= 0. 若為 None，默認為 K.epsilon().
- decay : float >= 0. 每次參數更新後學習率衰減值

```
from keras import optimizers
```

```
model = Sequential()  
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))  
model.add(Activation('softmax'))
```

#實例化一個優化器對象，然後將它傳入model.compile()，可以修改參數

```
opt = optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)  
model.compile(loss='mean_squared_error', optimizer=opt)
```

RMSprop

RMSProp 算法也旨在抑制梯度的鋸齒下降，但與動量相比，RMSProp 不需要手動配置學習率超參數，由算法自動完成。

更重要的是，RMSProp 可以為每個參數選擇不同的學習率。

RMSprop 是為了解決 Adagrad 學習率急劇下降問題的，所以比對梯度更新規則：

Adagrad	RMSprop
$\theta^{t+1} = \theta^t - \frac{\eta}{\sigma^t} g^t$	$\theta^{t+1} = \theta^t - \frac{\eta}{\sqrt{r^t}} g^t$
$\sigma^t = \sqrt{\frac{(g^0)^2 + \dots + (g^t)^2}{t + 1}}$	$r^t = (1-p)(g^t)^2 + pr^{t-1}$

Root mean square (RMS) of all Gradient

分母換成了過去的梯度平方的衰減平均值

程式設定：

- keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
- This optimizer is usually a good choice for recurrent neural networks.

Arguments

- lr : float >= 0. Learning rate.
- rho : float >= 0.
- epsilon : float >= 0. Fuzz factor. If None, defaults to K.epsilon().
- decay : float >= 0. Learning rate decay over each update.

```
from keras import optimizers
```

```
model = Sequential()  
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))  
model.add(Activation('softmax'))
```

#實例化一個優化器對象，然後將它傳入model.compile()，可以修改參數

```
opt = optimizers.RMSprop(lr=0.001, epsilon=None, decay=0.0)  
model.compile(loss='mean_squared_error', optimizer=opt)
```

Adam

除了像 RMSprop 一樣存儲了過去梯度的平方方 vt 的指數衰減平均值，也像 momentum 一樣保持了過去梯度 mt 的指數衰減平均值，「 t 」：

$$\begin{aligned} \text{the first moment (the mean)} \quad m_t &= \beta_1 m_t + (1 - \beta_1) g_t \\ \text{the second moment (the uncentered variance)} \quad v_t &= \beta_2 v_t + (1 - \beta_2) g_t^2 \end{aligned}$$

計算梯度的指數移動平均數， m_0 初始化為 0。綜合考慮之前時間步的梯度動量。

β_1 係數為指數衰減率，控制權重分配（動量與當前梯度），通常取接近於 1 的值。默認為 0.9

其次，計算梯度平方的指數移動平均數， v_0 初始化為 0。 β_2 係數為指數衰減率，控制之前的梯度平方方的影響情況。類似於 RMSProp 算法，對梯度平方進行加權均值。默認為 0.999

由於 m_0 初始化為 0，會導致 mt 偏向於 0，尤其在訓練初期階段。所以，此處需要對梯度均值 mt 進行偏差糾正，降低偏差對訓練初期的影響。

與 m_0 類似，因為 v_0 初始化為 0 導致訓練初始階段 vt 偏向 0，對其進行糾正。

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad \text{the decay rates are small (i.e. } \beta_1 \text{ and } \beta_2 \text{ are close to 1)}$$

更新參數，初始的學習率 lr 乘以梯度均值與梯度方差的平方根之比。其中默認學習率 $lr = 0.001$, ϵ (epsilon) $= 10^{-8}$ ，避免除數變為 0。

對更新的步長計算，能夠從梯度均值及梯度平方兩個角度進行自適應地調節，而不是直接由當前梯度決定

程式設定：

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

- **lr** : float ≥ 0 . 學習率。
- **beta_1** : float, $0 < \beta_1 < 1$. 通常接近於 1。
- **beta_2** : float, $0 < \beta_2 < 1$. 通常接近於 1。
- **epsilon** : float ≥ 0 . 模糊因數，若為 None，默認為 $K.epsilon()$ 。
- **amsgrad** : boolean. 是否應用此演算法的 AMSGrad 變種，來自論文「On the Convergence of Adam and Beyond」
- **decay** : float ≥ 0 . 每次參數更新後學習率衰減值。

```
from keras import optimizers
```

```
model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))
```

實例化一個優化器對象，然後將它傳入model.compile()，可以修改參數

```
opt = optimizers.Adam(lr=0.001, epsilon=None, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt)
```

如何選擇優化器

- 隨機梯度下降 SGD (Stochastic Gradient Descent) 指的是 mini batch gradient descent，隨機梯度下降法這種方法是將數據分成一小批一小批的進行訓練。但是速度比較慢。且對所有參數使用相同的學習率。
- AdaGrad 是採用改變學習率的方式
- RMSProp 這種方法是將 Momentum 與 AdaGrad 部分相結合，自適應調節學習率。對學習率進行了約束，適合處理非平穩目標和 RNN。
- Adam 容易收斂到局部最優，結合 AdaGrad 和 RMSProp 兩種優化算法的優點。對梯度的一階矩估計 (First Moment Estimation，即梯度的均值) 和二階矩估計 (Second Moment Estimation，即梯度的未中心化的方差) 進行綜合考慮，計算出更新步長。

優點：

1. 經過偏置校正後，每一次迭代都有確定的範圍，使得參數比較平穩。善於處理稀疏梯度和非平穩目標。
2. 對內存需求小。
3. 對不同內存計算不同的學習率。

整個詳細梯度調整訓練得其他參數設定，在下一節會說明，並於 4-4.5 作程式範例。

4-4.3.3 梯度調整(Gradient Descent)：優化器的參數調整

本小節接續上節的優化器，要如何調整優化器的參數呢？

複習講義 Day073-074_Gradient Descent 梯度調整，機器學習算法當中，優化算法的功能，是通過改善訓練方式，來最小化(或最大化)損失函數，最常用的優化算法是梯度下降：

- 何謂 Gradient 梯度？

➤ 在微積分裡面，對多元函數的參數求 ∂ 偏導數，把求得的各個參數的偏導數以向量的形式寫出來，就是梯度。

比如函數 $f(x)$, 對 x 求偏導數，求得的梯度向量量就是 $(\partial f / \partial x)$ ，簡稱 $\text{grad } f(x)$ 或者 $\nabla f(x)$

$$\text{Function}(x) = y_{\text{data}} = b + w^* x_{\text{data}}$$

➤ 目的：沿著目標函數梯度下降的方向搜索極小值（也可以沿著梯度上升的方向搜索極大值）

➤ 要計算 Gradient Descent，考慮

- $\text{Loss} = \text{實際 } y_{\text{data}} - \text{預測 } y_{\text{data}}$

$$= w^* \text{ 實際 } x_{\text{data}} - w^* \text{ 預測 } x_{\text{data}} (\text{bias 為 init value，被消除})$$

- $\text{Gradient} = \nabla f(\theta)$ ($\text{Gradient} = \partial L / \partial w$)

- 調整後的權重 = 原權重 - η (Learning rate) * Gradient

So,

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$



(更新，每走一步更新一次)

- 參數 η 是學習率。這個參數既可以設置為固定值，也可以用一維優化方法沿著訓練的方向逐步更新計算

- 學習率定義了每次疊代中應該更改的參數量。換句話說，它控制我們應該收斂到最低的速度。

- Learning rate 選擇，實際上取值取決於數據樣本，如果損失函數在變小，說明取值有效，否則要增大 Learning rate。
- 小學習率可以使疊代收斂，大學習率可能超過最小值

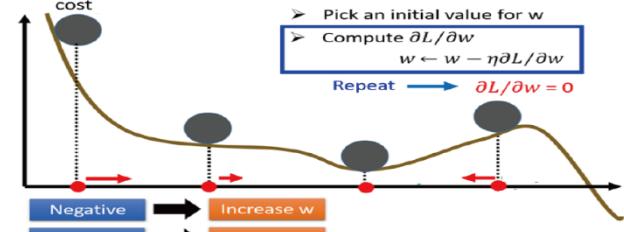
Compute $\partial L/\partial w$
 $w \leftarrow w - \eta \partial L/\partial w$

Learning rate
學習率



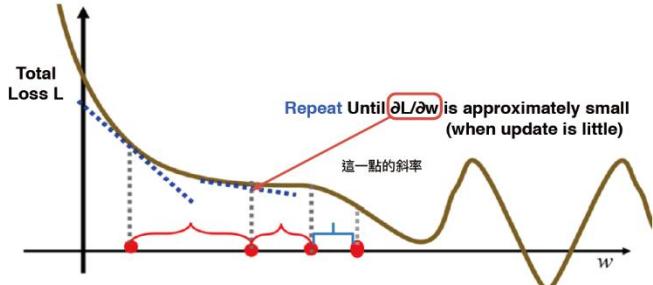
- 參數的更新分為兩步：第一步計算梯度下降的方向，第二步計算合適的學習

- 首先需要設定一個**初始參數值**，通常情況下將初值設為零($w=0$)，接下來需要計算成本函數 cost
- 然後計算函數的導數-某個點處的斜率值，並設定**學習效率參數(η)**的值。
- 重複執行上述過程，直到參數值收斂，這樣我們就能獲得函數的最優解



- 怎麼確定到極值點了呢？

- η 又稱**學習率**，是一個挪動步長的基數。 $df(x)/dx$ 是導函數，當離得遠的時候導數大，移動的就快，當接近極值時，導數非常小，移動的就非常小，防止跨過極值點。



- Gradient descent never guarantee global minima
- Different initial point will be caused reach different minima, so different results

avoid local minima



Popular & Simple Idea: Reduce the learning rate by some factor every few epochs

在訓練神經網絡的時候，通常在訓練剛開始的時候使用較大的

learning rate，隨著訓練的進行，我們會慢慢的減小learning rate

參數	意義
decayed_learning_rate	衰減後的學習率
learning_rate	初始學習率
decay_rate	衰減率
global_step	當前的 step
decay_steps	衰減週期



具體就是每次迭代的時候減少學習率的大小，更新公式：

$$\text{decayed_learning_rate} = \text{learning_rate} * \text{decay_rate}^{\text{global_step}/\text{decay_steps}}$$

- 自動更新 Learning rate - 衰減因子 decay

- 算法參數的初始值選擇。初始值不同，獲得的最小值也有可能不同，因此梯度下降求得的只是局部最小值；當然如果損失函數是凸函數則一定是最優解。

學習率衰減公式

- $lr_i = lr_{start} * 1.0 / (1.0 + decay * i)$
- 其中 lr_i 為第一迭代 i 時的學習率。 lr_{start} 為初始值。 $decay$ 為一個介於 $[0.0, 1.0]$ 的小數。從公式上可看出：
 - $decay$ 越小，學習率衰減地越慢，當 $decay = 0$ 時，學習率保持不變
 - $decay$ 越大，學習率衰減地越快，當 $decay = 1$ 時，學習率衰減最快

➤ 使用 momentum 是梯度下降法中一種常用的加速技術。程式介紹可以參考前一節優化器的簡介。

- 「一顆球從山上滾下來，在下坡的時候速度越來越快，遇到上坡，方向改變，速度下降」

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial w}$$

如果上一次的梯度跟這次同方向的話， $|V_t|$ (速度)會越來越大(代表梯度增強)， w 參數的更新梯度便會越來越快，
如果方向不同， $|V_t|$ 便會比上次更小(梯度減弱)， w 參數的更新梯度便會變小

$$w \leftarrow w + V_t$$

- 加入的這一項，可以使得梯度方向不變的維度上速度變快，梯度方向有所改變的維度上的更新速度變慢，這樣就可以加快收斂並減小震盪

- Gradient Descent 的實現: SGD, 對於一般的 SGD，其表達式為

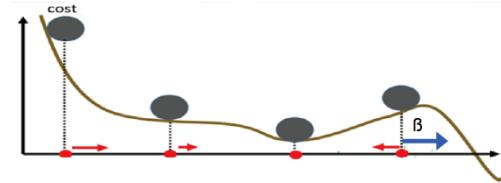
$$x \leftarrow x - \alpha * dx \quad (x \text{ 沿負梯度方向下降})$$

- 而帶 momentum 項的 SGD 則寫成如下形式：

$$v = \beta * v - a * d x$$

$$x \leftarrow x + v$$

- 其中 β 即 momentum 系數，通俗的理解上面式子就是，如果上一次的 momentum (即 β) 與這一次的負梯度方向是相同的，那這次下降的幅度就會加大，所以這樣做能夠達到加速收斂的過程



• 何謂反向傳播？

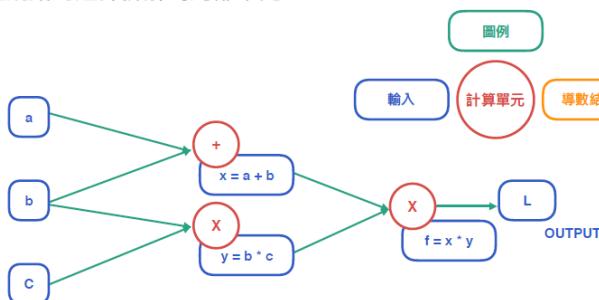
複習 Day75-Back_Propagation 反向傳播 (BP : Backpropagation) 是「誤差反向傳播」的簡稱，是一種與最優化方法 (如梯度下降法) 結合使用的方法，此對網路中所有權重計算損失函數的梯度。這個梯度會反饋給最優化方法，用來更新權值以最小化損失函數。

反向傳播要求有對每個輸入值想得到的已知輸出，來計算損失函數梯度。因此，它通常被認為是一種監督式學習方法，可以對每層疊代計算梯度。反向傳播要求人工神經元 (或「節點」) 的激勵函數可微。

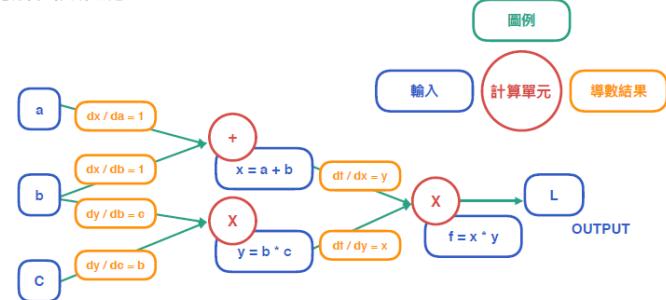
推導流程



將神經網路的運算拆解為局部單元



如何解函數微分



Code 27: Day075-Back_Propagation_PPTexample.ipynb

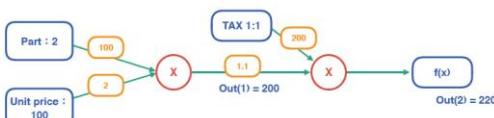
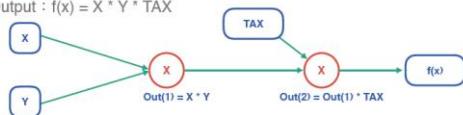
以預測水果銷售為例

- 水果銷售所應給付的價格決定因子

- 數量(顆數或是單位重量)
- 單價
- 稅金

- 建立運算單元：

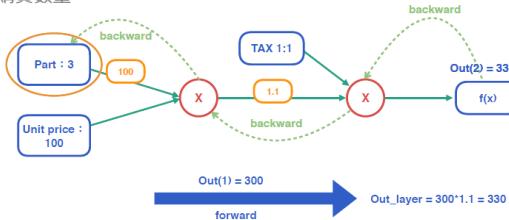
- 稅金是恆定的，可以當成是 Bias，給定 TAX
- Input-1：數量，給定 X
- Input-2：單價，給定 Y
- Output : $f(x) = X * Y * \text{TAX}$



- 要驗證網路模型是否正確？
- 更改 Init Data：
 - 更改購買數量
 - TAX的增加

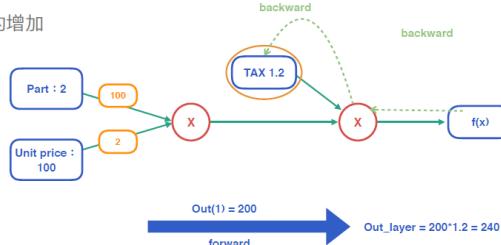
- 更改 Init Data

更改購買數量



所以，結帳金額 $f(x)$ 被影響的是 $(3-2) \times 100 \times 1.1 = 110$

TAX的增加

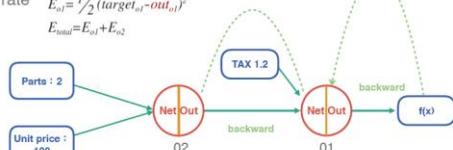


所以，結帳金額 $f(x)$ 被影響的是 $2 \times 100 \times (1.2 - 1.1) = 20$

進一步說明

更改 init data，輸出會有變動，模型的執行結果跟預期有落差也是變動，這個落差就是 error rate

- Error rate = (Target 輸出)-(實際輸出)
- 導入 activation function，以 MSE loss function 為例
- Error rate $E_{\text{err}} = \frac{1}{2}(\text{target}_{\text{err}} - \text{out}_{\text{err}})^2$
 $E_{\text{total}} = E_{\text{err}} + E_{\text{out}}$



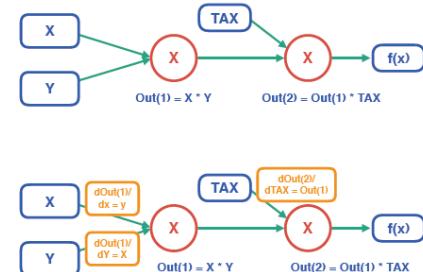
Init Network data

```
# Init Data
n_X = 2
price_Y = 100
b_TAX = 1.1
```

```
# Build _Network
mul_fruit_layer = mul_layer()
Mul_tax_layer = mul_layer()
```

建立 Forward & Backward

```
class mul_layer():
    def __init__(self):
        self.x = None
        self.y = None
    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x*y
        return out
    def backward(self, dout):
        dx = dout * self.y
        dy = dout * self.x
        return dx, dy
```



```
#forward
fruit_price = mul_fruit_layer.forward(price_Y, n_X)
total_price = mul_tax_layer.forward(fruit_price, b_TAX)
```

```
#backward
dtotal_price = 1 #this is linear function, which y=x, dy/dx=1
d_fruit_price, d_b_TAX = mul_tax_layer.backward(dtotal_price)
d_price_Y, d_n_X = mul_fruit_layer.backward(d_fruit_price)
```

```
#result
print("fruit price: %i" % fruit_price)
print("針對所有水果價格微分，得到 TAX: %2f" % d_fruit_price)
```

```
fruit price: 200
針對所有水果價格微分，得到 TAX: 1.100000
```

在課程的範例程式：

- BP Neural Network
- 實現 forward network，解函數微分求 Loss rate
 - Linear : Error rate = (target_out – real_out)
 - Weights refresh per iteration
 - Training and update
 - 得出 Loss rate

課程作業補充：BP 演算法訓練的神經網路

Code 28: Day075-Back_Propagation.ipynb

BP 神經網絡是一種按照逆向傳播算法訓練的多層前饋神經網絡，其優缺點列點如下：

優點：具有任意複雜的模式分類能力和優良的多維函數映射能力，解決了簡單感知器不能解決的問題。

- 從結構上講，BP 神經網絡具有輸入層、隱含層和輸出層；
- 從本質上講，BP 算法就是以網絡誤差平方目標函數、採用梯度下降法來計算目標函數的最小值。

缺點：

- ①學習速度慢，即使是一個簡單的過程，也需要幾百次甚至上千次的學習才能收斂。
- ②容易陷入局部極小值。
- ③網絡層數、神經元個數的選擇沒有相應的理論指導。
- ④網絡推廣能力有限。

應用：①函數逼近。②模式識別。③分類。④數據壓縮

步驟分解說明：

第 1 階段：解函數微分

- 每次疊代中的傳播環節包含兩步：
 - (前向傳播階段) 將訓練輸入送入網絡以獲得激勵響應；
 - (反向傳播階段) 將激勵響應同訓練輸入對應的目標輸出求差，從而獲得輸出層和隱藏層的響應誤差。

第 2 階段：權重更新

- Follow Gradient Descent
- 第 1 和第 2 階段可以反覆循環疊代，直到網絡對輸入的響應達到滿意的預定的目標範圍為止。

練習與總結

Gradient descent 是一個一階最佳化算法，通常也稱為最速下降法。要使用梯度下降法找到一個函數的局部極小值，必須向函數上當前點對應梯度（或者是近似梯度）的[反方向](#)的規定步長距離點進行疊代搜索。

➤ 梯度下降法的缺點包括：

- 靠近極小值時速度減慢。
- 直線搜索可能會產生一些問題。
- 可能會「之字型」地下降

➤ avoid local minima

- Item-1：在訓練神經網絡的時候，通常在訓練剛開始的時候使用較大的 learning rate，隨著訓練的進行，我們會慢慢的減小 learning rate
 - ✓ 學習率較小時，收斂到極值的速度較慢。
 - ✓ 學習率較大時，容易易在搜索過程中發生震盪。

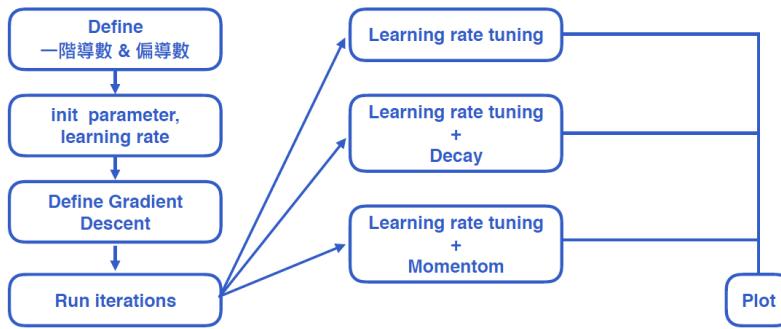
• Item-2：隨著 iteration 改變 Learning

- ✓ 衰減越大，學習率衰減地越快。衰減確實能夠對震盪起到減緩的作用。

Item-3 : momentum

- ✓ 如果上一次的 momentum 與這一次的負梯度方向是相同的，那這次下降的幅度就會加大，所以這樣做能夠達到加速收斂的過程
- ✓ 如果上一次的 momentum 與這一次的負梯度方向是相反的，那這次下降的幅度就會縮減，所以這樣做能夠達到減速收斂的過程

➤ python 程式 對照



Code 29: Day074-Gradient_Descent_Mathexplian.ipynb

梯度 Gradient

在微積分裡面，對多元函數的參數求 ∂ 偏導數，把求得的各個參數的偏導數以向量的形式寫出來，就是梯度。比如函數 $f(x)$ ，對 x 求偏導數，求得的梯度向量就是 $(\partial f / \partial x)$ ，簡稱 $\text{grad } f(x)$ 或者 $\nabla f(x)$

梯度下降法

給定起始點與目標函數的一階導函數(偏導數)，求在 epochs 次迭代中 x 的更新值

$y = w^*x$ (為了計算方便，假設 $w=x$)，所以 $y=(x)^2$,

$\text{func}(x) = y=(x)^2$

一階導函數: $dy/dw=2*x$, $df = dy/dw$

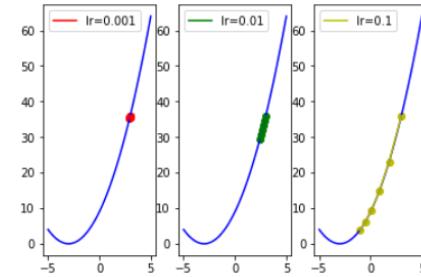
v 表示 w 要改變的幅度 $v = -(\text{dL}/\text{dw}) * lr$

$w \leftarrow w + v$ ($w \leftarrow w - lr * df$)

考慮 bias: $y = b + w^*x$ (set $b=0$)

學習率對梯度下降法的影響

學習率較小時，收斂到正確結果的速度較慢。學習率較大時，容易在搜索過程中發生震盪。



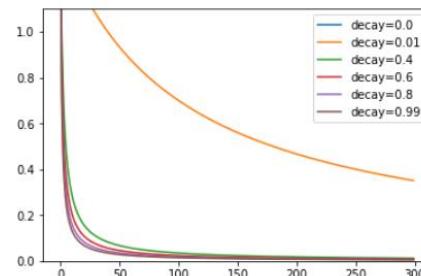
學習率衰減公式

$$lr_i = lr_{\text{start}} * 1.0 / (1.0 + decay * i)$$

其中 lr_i 為第一迭代 i 時的學習率， lr_{start} 為原始學習率， $decay$ 為一個介於 $[0.0, 1.0]$ 的小數。從公式上可看出：

$decay$ 越小，學習率衰減地越慢，當 $decay = 0$ 時，學習率保持不變。

$decay$ 越大，學習率衰減地越快，當 $decay = 1$ 時，學習率衰減最快



Momentum (動量)

如何用“動量”來解決：

(1) 學習率較小時，收斂到極值的速度較慢。

(2) 學習率較大時，容易在搜索過程中發生震盪。

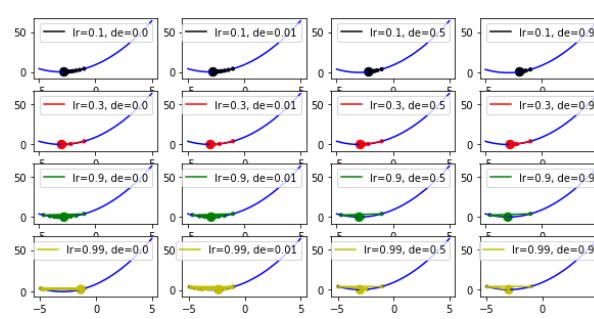
當使用動量時，則把每次 w 的更新量 v 考慮為本次的梯度下降量 $(-\text{dx} * lr)$ ，與上次 w 的更新量 v 乘上一個介於 $[0, 1]$ 的因子 momentum 的和

$w \leftarrow x - \alpha * dw$ (x 沿負梯度方向下降)

$v = \beta * v - \alpha * d w$

$w \leftarrow w + v$

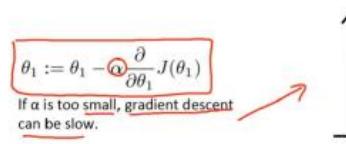
(β 即 momentum 係數，通俗的理解上面式子就是，如果上一次的 momentum (即 β) 與這一次的負梯度方向是相同的，那這次下降的幅度就會加大，所以這樣做能夠達到加速收斂的過程；如果上一次的 momentum (即 β) 與這一次的負梯度方向是相反的，那這次下降的幅度就會縮減，所以這樣做能夠達到減速收斂的過程)



4-4.4 訓練、預測模型範例

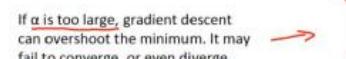
4-4.4.1 Optimizers & Learning rates (D080 練習)

藉由 D079 練習，了解了 Learning rate 的影響：如果 Learning rate (LR, alpha) 太大，將會導致每步更新時，無法在陡峭的損失山谷中，順利的往下滑動；但若太小，則要滑到谷底的時間過於冗長，且若遇到平原區則無法找到正確的方向。

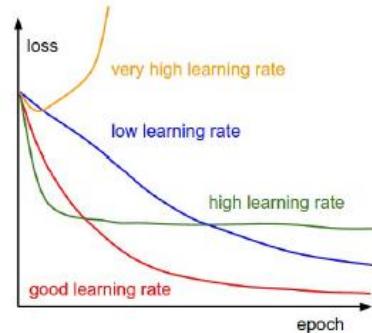


$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$

If α is too small, gradient descent can be slow.



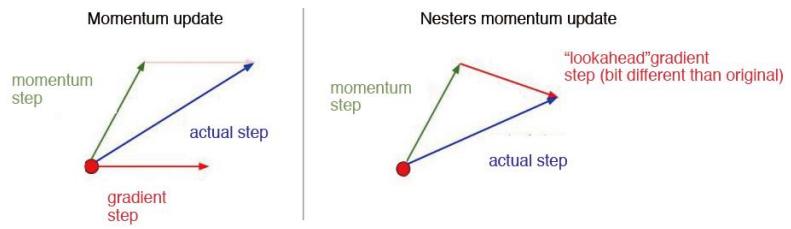
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



圖片來源：cs231n.github

加入 SGD optimizer，其動量方法調整，在損失方向上，加上一定比率的動量協助擺脫平原或是小山谷。

- ✓ Momentum：動量在更新方向以外，加上一個固定向量，使得真實移動方向會介於算出來的 gradient step 與 momentum 間。
 - Actual step = momentum step + gradient step
- ✓ Nesterov Momentum：拔草測風向
 - 將 momentum 納入 gradient 的計算
 - Gradient step computation is based on x + momentum



圖片來源：cs231n.github

這一節複習 D080 練習，繼續使用 cifar10 資料庫，結合前面的知識與程式碼，比較不同的 optimizer 與 learning rate 組合對訓練的結果與影響。常見的 optimizer 包含：SGD、RMSprop、AdaGrad、Adam。

- ✓ 這裡選擇使用 Functional API 建構網路層，因為執行速度較快，所以假設使用 DNN 模型執行(固定 EPOCHS = 10、BATCH_SIZE = 128)。可以初步看出，Learning rate 為 0.001 及 optimizer 為 Adam，有最小 LOSS 及最大 Accuracy。
- ✓ Optimizer 除了調整不同的 learning rate，當然還可以調整 nesterov、momentum、decay 預設值 `keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)`，在這裡範例沒有特別調整。

Code 30: Day080_AnsReview.ipynb

本節分析資料匯入

```
import os
import keras
import tensorflow
# Disable GPU
os.environ["CUDA_VISIBLE_DEVICES"] = ""
```

Using TensorFlow backend.

```
train, test = keras.datasets.cifar10.load_data()
```

```
## 資料前處理
def preproc_x(x, flatten=True):
    x = x / 255.
    if flatten:
        x = x.reshape((len(x), -1))
    return x

def preproc_y(y, num_classes=10):
    if y.shape[-1] == 1:
        y = keras.utils.to_categorical(y, num_classes)
    return y
```

```
Using TensorFlow backend.
```

```
x_train, y_train = train
x_test, y_test = test
```

```
# Preproc the inputs
x_train = preproc_x(x_train)
x_test = preproc_x(x_test)
```

```
# Preproc the outputs
y_train = preproc_y(y_train)
y_test = preproc_y(y_test)
```

比較不同的 **optimizer** 與 **learning rate** 組合對訓練的結果與影響。

```
def build_mlp(input_shape, output_units=10, num_neurons=[512, 256, 128]):
    input_layer = keras.layers.Input(input_shape)

    for i, n_units in enumerate(num_neurons):
        if i == 0:
            x = keras.layers.Dense(units=n_units, activation="relu", name="hidden_layer"+str(i+1))(input_layer)
        else:
            x = keras.layers.Dense(units=n_units, activation="relu", name="hidden_layer"+str(i+1))(x)

    out = keras.layers.Dense(units=output_units, activation="softmax", name="output")(x)

    model = keras.models.Model(inputs=[input_layer], outputs=[out])
    return model
```

```
## 超參數設定
```

```
LEARNING_RATE = [1e-1, 1e-2, 1e-3]
EPOCHS = 10
BATCH_SIZE = 128
OPTIMIZER = [keras.optimizers.SGD, keras.optimizers.RMSprop, keras.optimizers.Adagrad, keras.optimizers.Adam]
```

```
results = {}
for lr, opti in itertools.product(LEARNING_RATE, OPTIMIZER):
    keras.backend.clear_session() # 把舊的 Graph 清掉
    print("Experiment with LR = %.6f, Optimizer = %s" % (lr, str(opti)))
    model = build_mlp(input_shape=x_train.shape[1:])
    model.summary()
    #optimizer = keras.optimizers.opti(lr=lr, nesterov=True)
    optimizer = opti(lr=lr)
    model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer=optimizer)

    model.fit(x_train, y_train,
              epochs=EPOCHS,
```

```

batch_size=BATCH_SIZE,
validation_data=(x_test, y_test),
shuffle=True)

# Collect results
train_loss = model.history.history["loss"]
valid_loss = model.history.history["val_loss"]
train_acc = model.history.history["acc"]
valid_acc = model.history.history["val_acc"]

exp_name_tag = "exp-lr-%s-optimizer-%s" % (str(lr), str(opti))
results[exp_name_tag] = {'train-loss': train_loss,
                       'valid-loss': valid_loss,
                       'train-acc': train_acc,
                       'valid-acc': valid_acc}

```

(略)

```

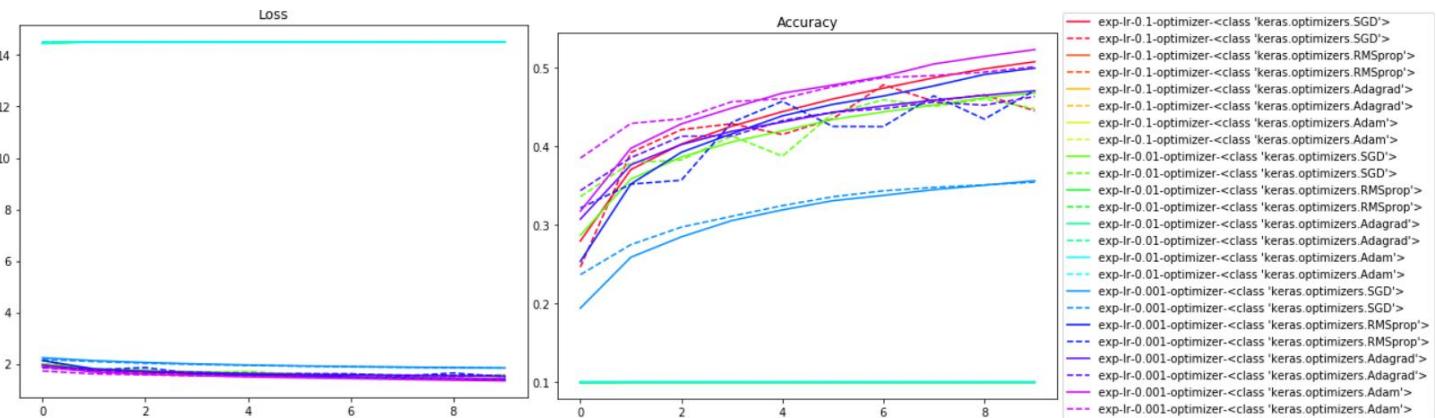
import matplotlib.pyplot as plt
%matplotlib inline

NUM_COLORS = len(results.keys())
cm = plt.get_cmap('gist_rainbow')
color_bar = [cm(1.*i/NUM_COLORS) for i in range(NUM_COLORS)]

plt.figure(figsize=(8,6))
for i, cond in enumerate(results.keys()):
    plt.plot(range(len(results[cond]['train-loss'])),results[cond]['train-loss'], ' ', label=cond, color=color_bar[i])
    plt.plot(range(len(results[cond]['valid-loss'])),results[cond]['valid-loss'], ' ', label=cond, color=color_bar[i])
plt.title("Loss")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()

plt.figure(figsize=(8,6))
for i, cond in enumerate(results.keys()):
    plt.plot(range(len(results[cond]['train-acc'])),results[cond]['train-acc'], ' ', label=cond, color=color_bar[i])
    plt.plot(range(len(results[cond]['valid-acc'])),results[cond]['valid-acc'], ' ', label=cond, color=color_bar[i])
plt.title("Accuracy")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()

```



4-4.4.2 正規化(D084 練習)

Code 31: Day084_AnsReview.ipynb

正規化可以調整的參數組合很多，複習 D081-D083 的練習題，列點如下：

1. **Regularizers**：隱藏層內的 **L1 / L2** 正規化，比較使用 **l1, l2, l1_l2** 及不同比例下的訓練結果。
2. **Dropout**：隨機省略神經元輸出的正規化，比較使用不同層數以及不同 Dropout rate 對訓練的效果。
3. **Batch-normalization**：傳遞時神經元橫向平衡的正規化，比較有 BN 在不同 Batch_size 下的差異。BN 放在 Activation 之前，或是 BN 放在 Input Layer 後的結果。

或加上不同 optimizer 的組合。這邊結合前面的知識與程式碼，比較不同的 **regularization** 的組合對訓練的結果與影響：如 **dropout, regularizers, batch-normalization** 等，範例：

```
from keras.layers import BatchNormalization, Activation, Dropout, regularizers
```

```
def build_mlp(input_shape,
               output_units=10,
               num_neurons=[512, 256, 128],
               use_bn=True,
               drp_ratio=0.,
               l2_ratio=0.):
    input_layer = keras.layers.Input(input_shape)

    for i, n_units in enumerate(num_neurons):
        if i == 0:
            x = keras.layers.Dense(units=n_units,
                                   kernel_regularizer=regularizers.l2(l2_ratio),
                                   name="hidden_layer"+str(i+1))(input_layer)

            if use_bn:
                x = BatchNormalization()(x)
            x = Activation("relu")(x)
            x = Dropout(drp_ratio)(x)

        else:
            x = keras.layers.Dense(units=n_units,
                                   kernel_regularizer=regularizers.l2(l2_ratio),
                                   name="hidden_layer"+str(i+1))(x)

            if use_bn:
                x = BatchNormalization()(x)
            x = Activation("relu")(x)
            x = Dropout(drp_ratio)(x)

    out = keras.layers.Dense(units=output_units, activation="softmax", name="output")(x)
    model = keras.models.Model(inputs=[input_layer], outputs=[out])
    return model
```

```
i## 超參數設定
LEARNING_RATE = 1e-3
EPOCHS = 10
BATCH_SIZE = 128
USE_BN = [True, False]
DRP_RATIO = [0., 0.4]
L2_RATIO = [0., 1e-6]
```

```
import keras.backend as K
results = []
for i, (use_bn, drp_ratio, l2_ratio) in enumerate(itertools.product(USE_BN, DRP_RATIO, L2_RATIO)):
    K.clear_session()
    print("Numbers of exp: %i, with bn: %s, drp_ratio: %.2f, l2_ratio: %.2f" % (i, use_bn, drp_ratio, l2_ratio))
    model = build_mlp(input_shape=x_train.shape[1:], use_bn=use_bn, drp_ratio=drp_ratio, l2_ratio=l2_ratio)
```

```

model.summary()
optimizer = keras.optimizers.Adam(lr=LEARNING_RATE)
model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer=optimizer)

model.fit(x_train, y_train,
          epochs=EPOCHS,
          batch_size=BATCH_SIZE,
          validation_data=(x_test, y_test),
          verbose=1,
          shuffle=True)

# Collect results
exp_name_tag = ("exp-%s" % (i))
results[exp_name_tag] = {'train-loss': model.history.history["loss"],
                        'valid-loss': model.history.history["val_loss"],
                        'train-acc': model.history.history["acc"],
                        'valid-acc': model.history.history["val_acc"]}

```

(略)

```

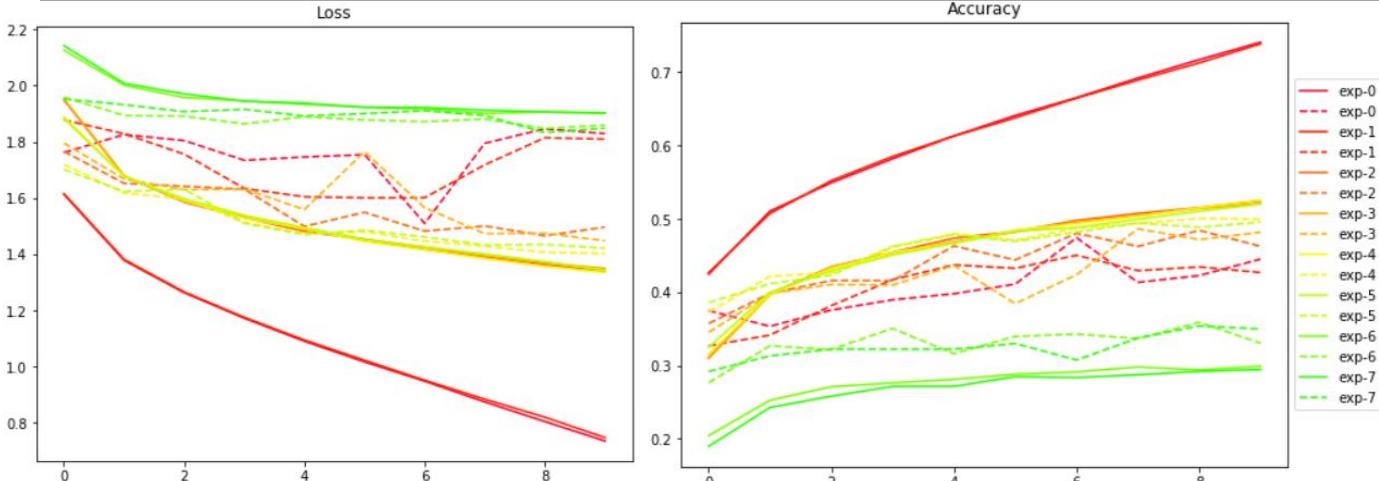
import matplotlib.pyplot as plt
import matplotlib.cm as mplcm
import matplotlib.colors as colors
%matplotlib inline
NUM_COLORS = 20

cm = plt.get_cmap('gist_rainbow')
cNorm = colors.Normalize(vmin=0, vmax=NUM_COLORS-1)
scalarMap = mplcm.ScalarMappable(norm=cNorm, cmap=cm)
color_bar = [scalarMap.to_rgba(i) for i in range(NUM_COLORS)]

plt.figure(figsize=(8,6))
for i, cond in enumerate(results.keys()):
    plt.plot(range(len(results[cond]['train-loss'])), results[cond]['train-loss'], '--', label=cond, color=color_bar[i])
    plt.plot(range(len(results[cond]['valid-loss'])), results[cond]['valid-loss'], '--', label=cond, color=color_bar[i])
plt.title("Loss")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()

plt.figure(figsize=(8,6))
for i, cond in enumerate(results.keys()):
    plt.plot(range(len(results[cond]['train-acc'])), results[cond]['train-acc'], '--', label=cond, color=color_bar[i])
    plt.plot(range(len(results[cond]['valid-acc'])), results[cond]['valid-acc'], '--', label=cond, color=color_bar[i])
plt.title("Accuracy")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()

```



4-4.4.3 callbacks 函數

Callbacks function：在訓練過程中，我們可以透過一些函式來監控/介入訓練

✓ EarlyStopping

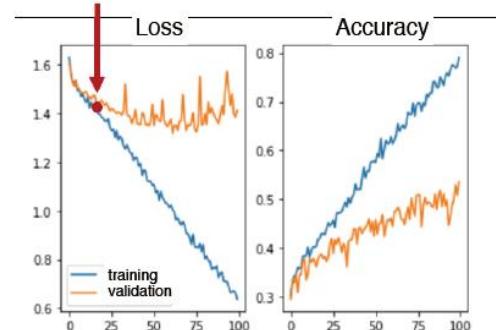
👉 Code 32: Day085_CB_EarlyStop_Ans.ipynb

複習講義 Day085_CB_EarlyStop_，如果一個模型會 overfitting，那 training 與 validation 的表現只會越差越遠，不如在開始變糟之前就先停下。

注意：Earlystop 不會使模型得到更好的結果，僅是避免更糟
程式設定：

```
from keras.callbacks import EarlyStopping

earlystop = EarlyStopping(monitor="val_loss",      # what to monitor
                          patience=5,          # epochs to wait
                          verbose=1            # print information
                        )
```



```
model.fit(x_train, y_train,
          epochs=EPOCHS,
          batch_size=BATCH_SIZE,
          validation_data=(x_test, y_test),
          shuffle=True,
          callbacks=[earlystop]
        )
```

✓ ModelCheckPoint

👉 Code 33: Day086_CB_ModelCheckPoint_Ans.ipynb

複習講義 Day086_CB_ModelCheckPoint_，假如電腦突然斷線、當機該怎麼辦？難道我只能重新開始？

ModelCheckPoint 會自動將目前最佳的模型權重存下。

- 我們可以透過監控 validation loss 來保證所存下來的模型是在 validation set 表現最好的一個。
- 假如要做 Inference：可以保證使用的是對 monitor metric 最佳的權重。

程式設定：

```
from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint('model.h5', # path to save
                            monitor = 'val_loss', # target to monitor
                            verbose = 1,          # print information
                            save_best_only = True, # save best checkpoint
                          )
model.fit(x_train, y_train,
          epochs=EPOCHS,
          batch_size=BATCH_SIZE,
          validation_data=(x_test, y_test),
          shuffle=True,
          callbacks=[checkpoint]
        )
```

✓ Reduce Learning Rate

Code 34: Day087_CB_ReduceLR_Ans.ipynb

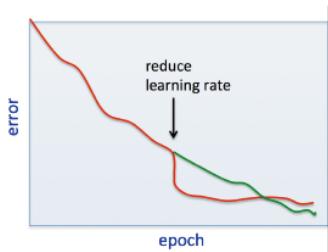
複習講義 Day087_CB_ReduceLR_ · Reduce learning rate on plateau : 模型沒辦法進步的可能是因為學習率太大，導致每次改變量太大而無法落入較低的損失平面，透過適度的降低，就有機會得到更好的結果。因為我們可以透過這樣的監控機制，初始的 Learning rate 可以調得比較高，讓訓練過程與 callback 來做適當的 learning rate 調降。

Reduce Learning Rate: 隨訓練更新次數，將 Learning rate 逐步減小。

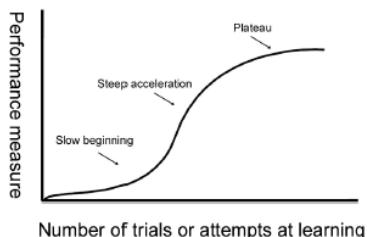
- 因為通常損失函數越接近谷底的位置，開口越小 – 需要較小的 Learning rate 才可以再次下降。

可行的調降方式：

- 每更新 n 次後，將 Learning rate 做一次調降 – schedule decay
- 當經過幾個 epoch 後，發現 performance 沒有進步 – Reduce on plateau



圖片來源：stats.stackexchange.com



圖片來源：the-knowledgesmart-blog

程式設定：

```
from keras.callbacks import ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(factor=0.5,
                              min_lr=1e-12,
                              monitor='val_loss',
                              patience=5,
                              verbose=1)

model.fit(x_train, y_train,
          epochs=EPOCHS,
          batch_size=BATCH_SIZE,
          validation_data=(x_test, y_test),
          shuffle=True,
          callbacks=[reduce_lr]
        )
```

✓ Callback 參數調整

Code 35: Day088_CB_CustomizedCallbacks_Ans.ipynb

複習講義 Day088_CB_CustomizedCallbacks_，上述所提到的 Callbacks，都是在 epoch end 時啟動 (也是最常使用的狀況)，在 Keras 中，僅需要實作你想要啟動的部分即可，舉例來說，假如你想要每個 batch 都記錄 loss 的話：

```
from keras.callbacks import Callback
class My_Callback(Callback):
    def on_train_begin(self, logs={}):
        return

    def on_train_end(self, logs={}):
        return

    def on_epoch_begin(self, logs={}):
        return

    def on_epoch_end(self, epoch, logs={}):
        return

    def on_batch_begin(self, batch, logs={}):
        return

    def on_batch_end(self, batch, logs={}):
        self.losses.append(logs.get('loss'))
        return
```

- Callback 在訓練時的呼叫時機
 - on_train_begin：在訓練最開始時
 - on_train_end：在訓練結束時
 - on_batch_begin：在每個 batch 開始時
 - on_batch_end：在每個 batch 結束時
 - on_epoch_begin：在每個 epoch 開始時
 - on_epoch_end：在每個 epoch 結束時

✓ 各參數比較練習

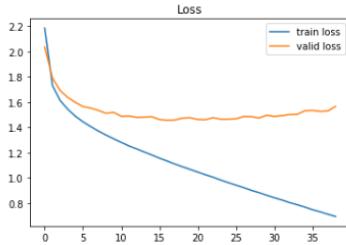
EX1. EarlyStopping: monitor 變更 monitor "Validation Accuracy" 及調整等待次數至 10, 25 比較結果

```
earlystop = EarlyStopping(monitor="val_acc",
                           patience=10,
                           verbose=1
                           )
```

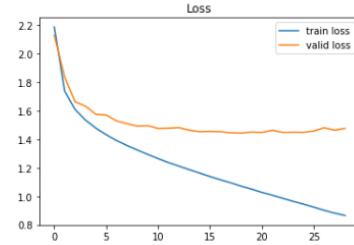
```
earlystop = EarlyStopping(monitor="val_loss",
                           patience=10,
                           verbose=1
                           )
```

```
earlystop = EarlyStopping(monitor="val_loss",
                           patience=25,
                           verbose=1
                           )
```

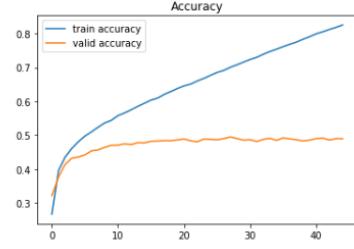
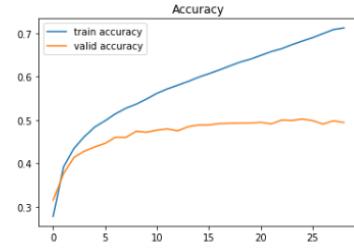
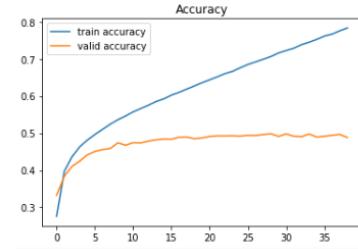
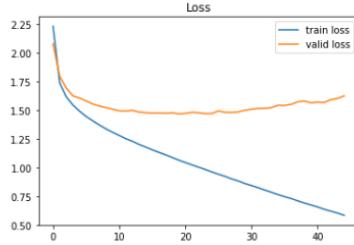
```
Epoch 39/50
50000/50000 [=====] - 4s 70us/step - loss: 0.6929 - acc: 0.7834 - val_loss: 1.5656 - val_acc: 0.488
Epoch 00039: early stopping
```



```
Epoch 29/50
50000/50000 [=====] - 3s 65us/step - loss: 0.8645 - acc: 0.7121 - val_loss: 1.4764 - val_acc: 0.494
Epoch 00029: early stopping
```



```
Epoch 45/50
50000/50000 [=====] - 3s 67us/step - loss: 0.5854 - acc: 0.8255 - val_loss: 1.6244 - val_acc: 0.489
Epoch 00045: early stopping
```



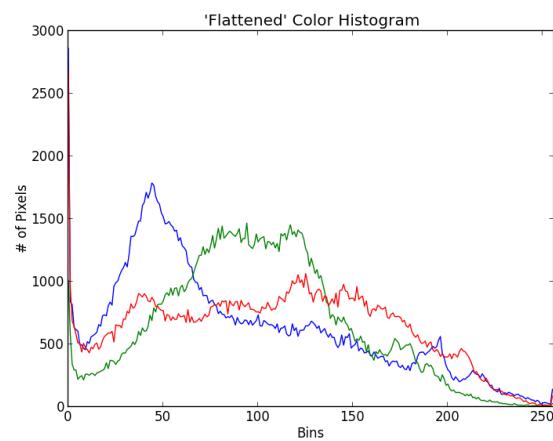
在這裡"Validation Loss"比 "Validation Accuracy"的 test 錯誤率及精準度較好一些，且等待次數由 10 到 25 不會更好，而且也增加了計算的時間。

4-5 傳統電腦視覺提取特徵的方法：color histogram 及 HOG 特徵

傳統電腦視覺提取特徵的方法，為了有更直觀的理解，Day091_介紹一種最簡單提取特徵的方法 1)如何描述顏色？2) 顏色直方圖。顏色直方圖是將顏色信息轉化為特徵一種方法，將顏色值 RGB 轉為直方圖值，來描述色彩和強度的分佈情況。舉例來說，一張彩色圖有 3 個 channel，RGB，顏色值都介於 0-255 之間，最小可以去統計每個像素值出現在圖片的數量，也可以是一個區間如 (0 - 15)、(16 - 31)、...、(240 - 255)。可表示如圖

然而傳統影像視覺描述特徵的方法是一個非常「手工」的過程，可以想像顏色直方圖在要辨認顏色的場景就會非常有用，但可能就不適合用來做邊緣檢測的任務，因為從顏色的分佈沒有考量到空間上的信息。不同的任務，我們就要想辦法針對性地設計特徵來進行後續影像辨識的任務。

原 Day092_課程中嘗試比較用 color histogram 和 HOG 特徵分類 cifar10 準確度各在 training 和 testing data 的差別，我們得知：靠人工設計的特徵在簡單的任務上也許是堪用，但複雜的情況，比如說分類的類別多起來，就能明顯感覺到這些特徵的不足之處，體會這一點能更幫助理解接下來的卷積神經網路的意義。



4-6 卷積神經網路 Convolution Neural Network_CNN

Day092_093 講議，承 4-3 所提到的卷積，卷積神經網路 CNN 目前在多數電腦視覺的任務中，都有優於人類的表現。在 4-4 的範例使用傳統的 DNN（即 Deep neural network），其最大問題在於它會忽略資料的形狀，例如：

- 輸入影像的資料時，該 data 通常包含了水平、垂直、color channel 等三維資訊，但傳統 DNN 的輸入處理必須是平面的、也就是須一維的資料。
- 一些重要的空間資料，只有在三維形狀中才能保留下來。
- RGB 不同的 channel 之間也可能具有某些關連性、而遠近不同的像素彼此也應具有不同的關聯性。

而 Deep learning 中的 CNN 較傳統的 DNN 多了 Convolutional (卷積) 及 Pooling 兩層 layer，用以維持形狀資訊並且避免參數大幅增加。

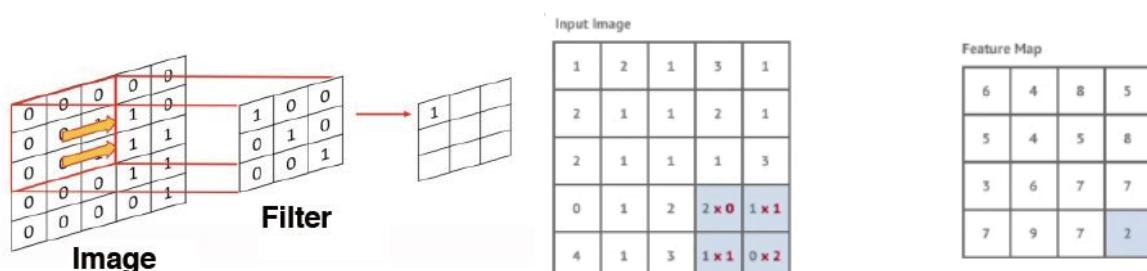
濾波器 (filter)的概念

透過卷積，我們可以找出圖像上與濾波器具有相同特徵的區域。而兩個不同濾波器(filter)得到的特徵圖(Feature map)也不相同。如圖，透過一層又一層的神經網路疊加，可以看到底層的濾波器再找線條與顏色的特徵，中層則是輪輪廓與形狀(輪胎)，高層的則是相對完整的特徵(如車窗、後照鏡等)。

如果你已經了解濾波器是用來找圖像上是否有同樣特徵，那接下來的問題是濾波器 (filter)中的數字是怎麼得來的呢？

在介紹細節之前，這邊簡單說，這些數字其實是透過資料學習而來的！這也就是 CNN 模型中的參數(或叫權重 weights)。CNN 會自動從訓練資料中學習出適合的濾波器來完成你的任務(分類、偵測等)。

卷積其實只是簡單的數學乘法與加法，利用濾波器 (filter) 對圖像做卷積來找尋規則，下圖左的濾波器是一個斜直線，可用來搜尋圖像上具有斜直線的區域。下圖右紅色的數字就是濾波器 (filter)，可以看到是一個 2×2 的矩陣(值為 $[[0, 1], [1, 2]]$)，卷積是將影像與 filter 的值相乘後再進行加總，即可得到特徵圖(Feature Map)。



GIF 動畫

<https://i.imgur.com/XsO8YzY.gifv>

- 卷積是圖像的通用濾鏡效果。
 - 將矩陣應用於圖像和數學運算，由整數組成
 - 卷積是通過相乘來完成的
 - 像素及其相鄰像素矩陣的顏色值
 - 輸出是新修改的過濾圖像
- Kernel 內核 (or 過濾器 filter):
 - 內核 (通常) 很小 用於的數字矩陣 圖像卷積。
 - 「不同大小的內核」包含不同的模式 數字產生不同的結果
 - 在卷積下。「內核的大小是任意的」但經常使用 3×3

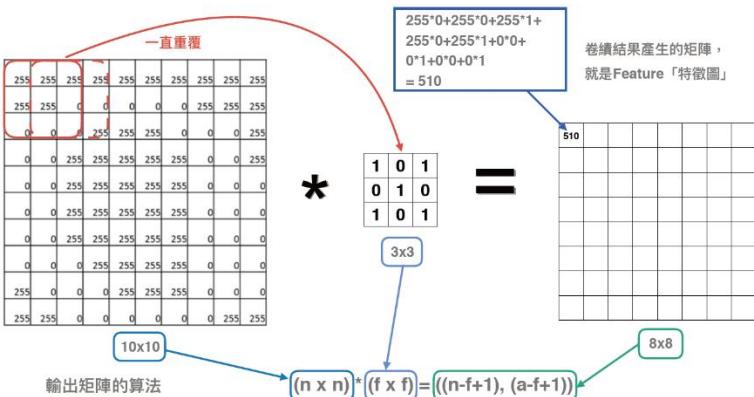
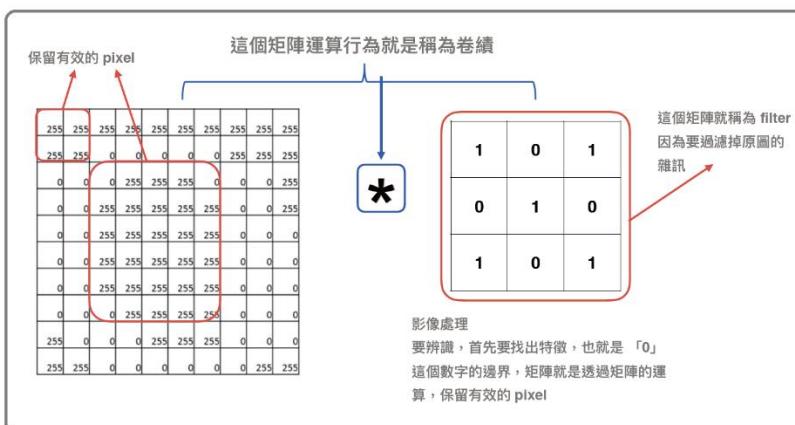
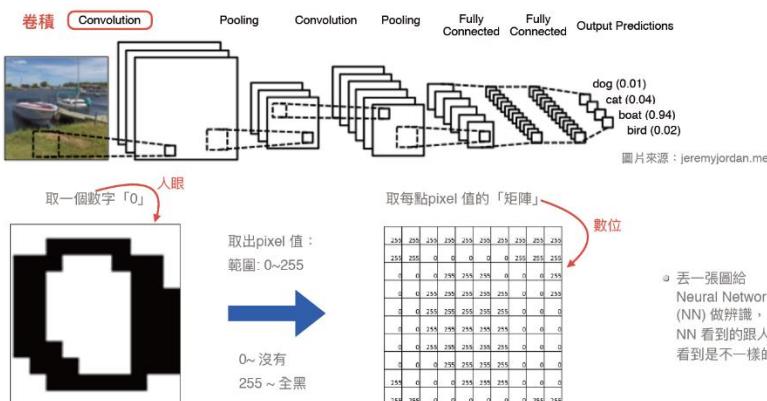
卷積網路的組成

1. Convolution Layer 卷積層
2. Pooling Layer 池化層
3. Flatten Layer 平坦層
4. Fully connection Layer 全連接層

Convolution 原理是透過一個指定尺寸的 window，由上而下依序滑動取得圖像中各局部特徵作為下一層的輸入，這個 sliding window 在 CNN 中稱為 Convolution kernel (卷積內核)。利用此方式來取得圖像中各局部的區域加總計算後，透過 ReLU activation function 輸出為特徵值再提供給下一層使用。

1. Convolution Layer 卷積層

卷積如何做



卷積 (Convolution) 的超參數(Hyper parameter)

- 內核大小 (Kernel size)
- 深度(Depth, Kernel的總數)
- 填充(Padding)
- 選框每次移動的步數(Stride)

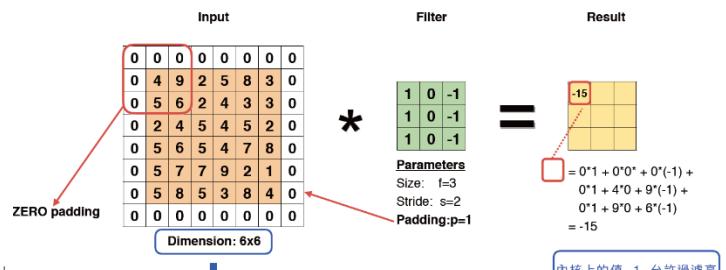
```
model.add(Convolution2D(25, 3, 3, input_shape=(1, 28, 28)))
(卷積引數: filter數量, filter長, fliter寬, 輸入影象的三維 (RGB, 長, 寬))
```

給定了 Filter 的 深度與維度

填充或移動步數(Padding/Stride)的用途

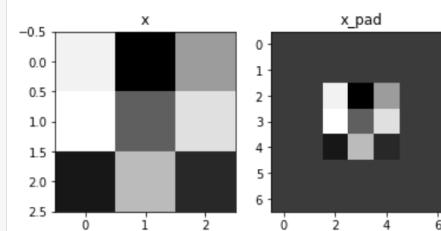
RUN 過 CNN，兩個問題

- 是不是卷積計算後，卷積後的圖是不是就一定只能變小？
 - 可以選擇維持一樣大
 - 卷積計算是不是一次只能移動一格？
- 控制卷積計算的圖大小 - Valid and Same convolutions
 - `padding = 'VALID'` 等於最一开始敘述的卷積計算，圖根據 filter 大小和 stride 大小而變小
 - `new_height = new_width = (W - F + 1) / S`
 - `padding = 'Same'`的意思就是要讓輸入和輸出的大小是一樣的
 - `pad=1`, 表示圖外圈額外加 1 圈 0, 假設 `pad=2`, 圖外圈額外加 2 圈 0, 以此類推



```
# GRADED FUNCTION: zero_pad
def zero_pad(X, pad):
    """
    對image X 做 zero-padding。
    參數定義如下:
    X -- python numpy array, 呈現維度 (m, n_H, n_W, n_C), 代表一批 m 個圖像
        n_H: 圖高, n_W: 圖寬, n_C: color channels 數
    pad -- 數字, 加幾圈的 zero padding。
    Returns:
    X_pad -- image of shape (m, n_H + 2*pad, n_W + 2*pad, n_C) 做完zero-padding 的結果
    """
    ### Code 起始位置
    X_pad = np.pad(X, ((0, 0), (pad, pad), (pad, pad), (0, 0)), 'constant', constant_values=(0, 0))
    return X_pad

x = np.random.randn(4, 3, 3, 2) #產生gray image
x_pad = zero_pad(x, 2) # 加兩層 Pad
```



多個通道(channels)的卷積作法

- 考慮多種顏色-針對 RGB
- 會有3個對應的 kernel



0	0	0	0	0	0	0	...
0	156	155	156	158	158	...	
0	153	154	157	159	159	...	
0	149	151	155	158	159	...	
0	146	146	149	153	158	...	
0	145	143	143	148	158	...	
...	

0	0	0	0	0	0	0	...
0	167	166	167	169	169	...	
0	164	165	168	170	170	...	
0	160	162	166	169	170	...	
0	156	156	159	163	165	...	
0	155	158	153	158	168	...	
...	

0	0	0	0	0	0	0	...
0	163	162	163	165	165	...	
0	160	161	164	166	166	...	
0	156	158	162	165	166	...	
0	155	155	158	162	167	...	
0	154	152	152	157	167	...	
...	

Input Channel #1 (Red)

Input Channel #2 (Green)

Input Channel #3 (Blue)

Kernel Channel #1

Kernel Channel #2

Kernel Channel #3

Output

308

+

-498

+

164 + 1 = -25

Bias = 1

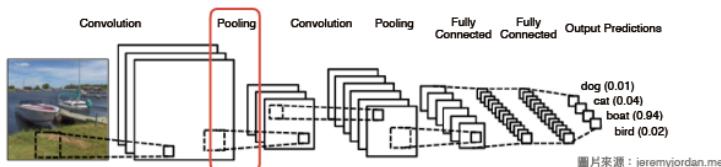
Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Filter W1 (3x3x3)	Output Volume (3x3x2)
x(:,:,1,0)	w0(:,:,1,0)	w1(:,:,1,0)	o(:,:,1,0)
0 0 0 0 0 0 0	-1 0 1 0 0 0 0 0 1	0 1 0 0 0 0 0 0 1	2 3 3 3 7 3
0 0 0 1 0 2 0	0 0 0 1 0 0 0 0 0	0 -1 0 1 0 0 0 0 0	8 10 -3
0 1 0 2 0 1 0	1 -1 1 -1 0 0 0 1 0	1 -1 0 -1 0 0 1 0 0	o(:,:,1,1) -8 -8 -3
0 2 0 0 2 0 0	0 1 0 -1 0 1 0 2 1	1 -1 0 -1 0 1 0 2 1	-3 1 0
0 0 2 0 2 0 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	1 -1 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 -1 -1
x(:,:,1,1,1)	w0(:,:,1,1,1)	w1(:,:,1,1,1)	w1(:,:,1,1,2)
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	-1 1 -1
0 2 1 2 1 0 0	0 1 0 -1 0 1 0 2 1	1 1 0 -1 1 0 0 2 1	0 -1 0
0 0 2 1 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	1 0 0
0 1 2 2 2 0 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 1 2 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0
x(:,:,1,2,1)	w0(:,:,1,2,1)	w1(:,:,1,2,1)	w1(:,:,1,2,2)
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	-1 1 -1
0 2 1 2 1 0 0	0 1 0 -1 0 1 0 2 1	1 1 0 -1 1 0 0 2 1	0 -1 0
0 0 2 1 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	1 0 0
0 1 2 2 2 0 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 1 2 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0
x(:,:,1,2,2)	w0(:,:,1,2,2)	w1(:,:,1,2,2)	w1(:,:,1,2,3)
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	-3 -8 -5
0 2 1 2 1 0 0	0 1 0 -1 0 1 0 2 1	1 1 0 -1 1 0 0 2 1	o(:,:,1,3,1) -3 -8 -5
0 0 2 1 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	1 0 0
0 1 2 2 2 0 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 1 2 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0
x(:,:,1,3,2)	w0(:,:,1,3,2)	w1(:,:,1,3,2)	w1(:,:,1,3,3)
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0
0 2 1 2 1 0 0	0 1 0 -1 0 1 0 2 1	1 1 0 -1 1 0 0 2 1	0 0 0
0 0 2 1 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 1 2 2 2 0 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 1 2 0 1 0	0 0 0 0 1 0 0 2 1	0 0 0 0 1 0 0 2 1	0 0 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0

toggle movement

Code 36: Day094.1-CNN_Convolution.ipynb + Day095.1-CNN_Pooling.Padding.ipynb

2. Pooling Layer 池化層

- Pooling layer 稱為池化層，它的功能很單純，就是將輸入的圖片尺寸縮小（大部份為縮小一半）以減少每張 feature map 維度並保留重要的特徵，其好處有：
 - 特徵降維，減少後續 layer 需要參數。
 - 具有抗干擾的作用：圖像中某些像素在鄰近區域有微小偏移或差異時，對 Pooling layer 的輸出影響不大，結果仍是不變的。
 - 減少過度擬合 over-fitting 的情況。

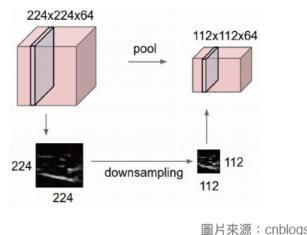


◎ 池化層(Pooling Layer) 如何調用

- 以 Keras 為例
- keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
- pool_size：整數，沿（垂直，水平）方向縮小比例的因數。
 - (2, 2)會把輸入張量的兩個維度都縮小一半。
- strides：整數，2 個整數表示的元組，或者是"None"。表示步長值。
 - 如果是 None，那麼默認值是 pool_size。
- padding："valid"或者"same"（區分大小寫）。
- data_format：channels_last(默認)或 channels_first 之一。表示輸入各維度的順序
- channels_last 代表尺寸是(batch, height, width, channels)的輸入張量，
- channels_first 代表尺寸是(batch, channels, height, width)的輸入張量。

◎ 池化層(Pooling Layer) 超參數

- 前端輸入feature map 維度：W1×H1×D1
- 有兩個hyperparameters：
 - Pooling filter 的維度- F,
 - 移動的步數 S,
- 所以預計生成的輸出是 W2×H2×D2:
 - $W2 = (W1 - F)/S + 1$
 - $H2 = (H1 - F)/S + 1$
 - $D2 = D1$



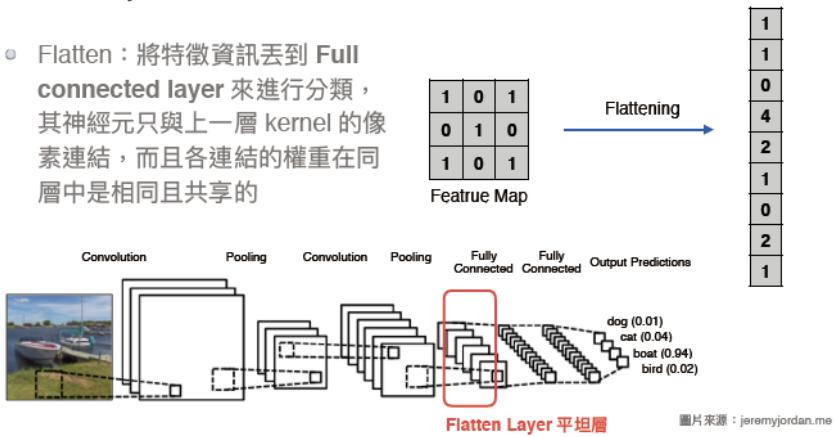
◎ 池化層(Pooling Layer) 常用的類型

- Pooling Layer 常用的類型：
- Max pooling (最大池化)
- Average pooling (平均池化)



3. Flatten Layer 平坦層

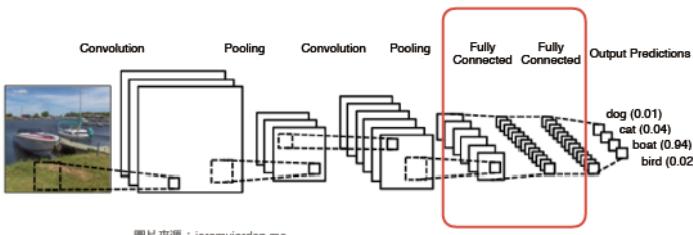
- Flatten：將特徵資訊丟到 Full connected layer 來進行分類，其神經元只與上一層 kernel 的像素連結，而且各連結的權重在同層中是相同且共享的



圖片來源：jeremyjordan.me

4. Fully connection Layer 全連接層

- 卷積和池化層，其最主要的目的分別是提取特徵及減少圖像參數，然後將特徵資訊丟到 Full connected layer 來進行分類，其神經元只與上一層kernel的像素連結，而且各連結的權重在同層中是相同且共享的



圖片來源：jeremyjordan.me

5. 回顧--卷積神經網路(CNN)特性

- 適合用在影像上
 - 因為 fully-connected networking (全連接層) 如果用在影像辨識上，會導致參數過多(因為像素很多)，導致 over-fitting(過度擬合)
 - CNN 對影像辨識的特性，特別設計過，來減少參數
 - Convolution(卷積)：學出 filter 比對原始圖片，產生出 feature map (特徵圖, 也當成 image)
 - Max Pooling (最大池化)：將 feature map 縮小
 - Flatten (平坦層)：將每個像素的 channels (有多少個 filters) 展開成 fully connected feedforward network (全連接的前行網路)
- AlphaGo 也用了 CNN，但是沒有用 Max Pooling (所以不同問題需要不同 model)
- Pooling Layer (池化層) 適用的場景

特徵提取的誤差主要來自兩個方面：

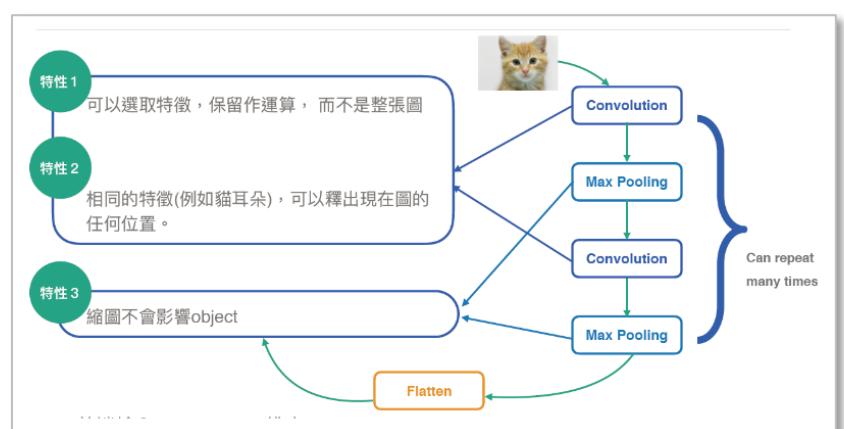
(1) 鄰域大小受限造成的估計值方差增大；

(2) 卷積層超參數與內核造成估計均值的偏移。

一般來說，

- average-pooling 能減小第一種誤差，更多的保留圖像的背景信息。

- max-pooling 能減小第二種誤差，更多的保留紋理信息



程式架構範例

CNN 模型必要的：Convolution, Pooling, Flatten, Fully connection, Output

```
#導入相關模組
import keras
from keras import layers
from keras import models
from keras.models import Sequential
from keras.layers import Conv2D, Activation, MaxPooling2D, Flatten, Dense

#確認 keras 版本
print(keras.__version__)
Using TensorFlow backend.
2.2.4

#建立一個序列模型
model = models.Sequential()
#建立一個卷積層, 32 個內核, 內核大小 3x3,
#輸入影像大小 28x28x1
model.add(layers.Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
```

```
#新增一池化層, 採用 maxpooling
model.add(MaxPooling2D(2,2))
```

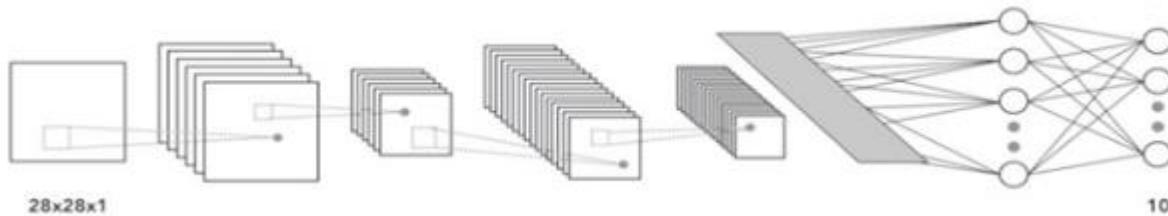
```
#建立第二個卷積層, 池化層,
#請注意, 不需要再輸入 input_shape
model.add(layers.Conv2D(25, (3, 3)))
model.add(MaxPooling2D(2,2))
```

```
#新增平坦層
model.add(Flatten())
```

```
#建立一個全連接層
model.add(Dense(units=100))
model.add(Activation('relu'))
```

```
#建立一個輸出層, 並採用 softmax
model.add(Dense(units=10))
model.add(Activation('softmax'))
```

網路模型



Convolutional over volume

- **input 上的變化**
 - 單色圖片的 input，是 2D，Width x Height
 - 彩色圖片的 input，是 3D，Width x Height x Channels
- **filter 上的變化**
 - 單色圖片的 filter，是 2D，Width x Height
 - 彩色圖片的 filter，是 3D，Width x Height x Channels
但 2 個 filter 的數值是一樣的
- **feature map 上的變化**
 - 單色圖片，一個 filter，是 2D，Width x Height
 - 多個 filters，Width x Height x filter 數量
 - 彩色圖片，也是如此

隨堂小考

from D92_ 請閱讀以下文獻，理解 CNN 原理後回答問題

卷積神經網路原理 - 中文 https://brohrer.mcknote.com/zh-Hant/how_machine_learning_works/how_convolutional_neural_networks_work.html

CNN for beginner's guide <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>

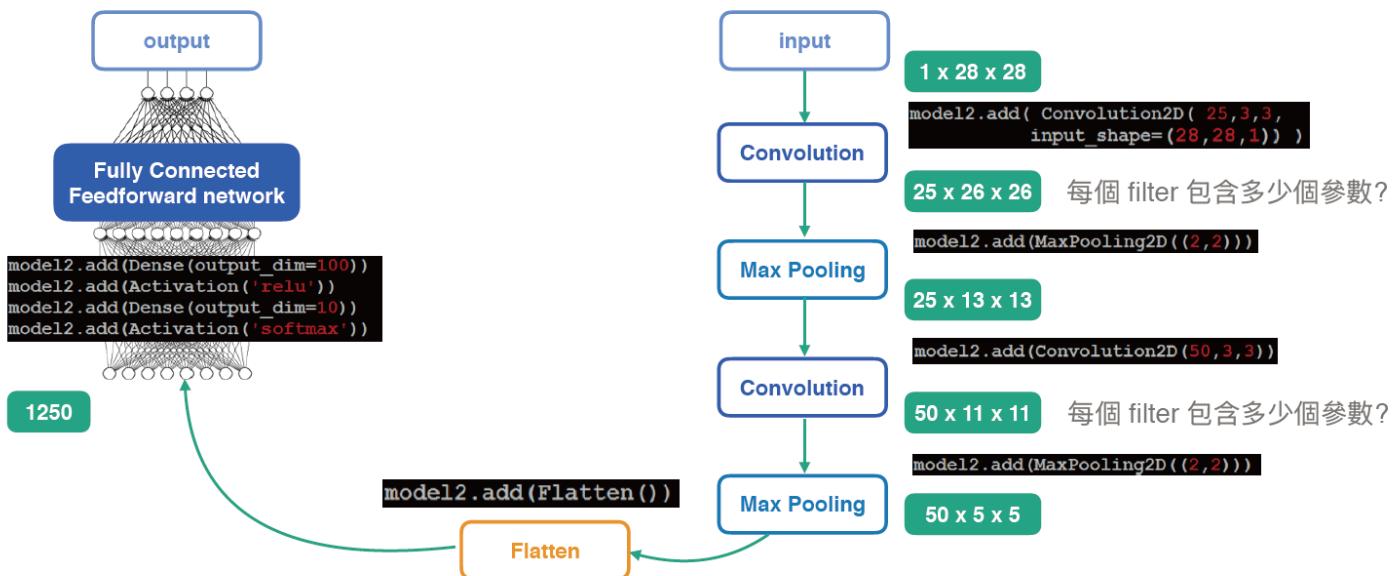
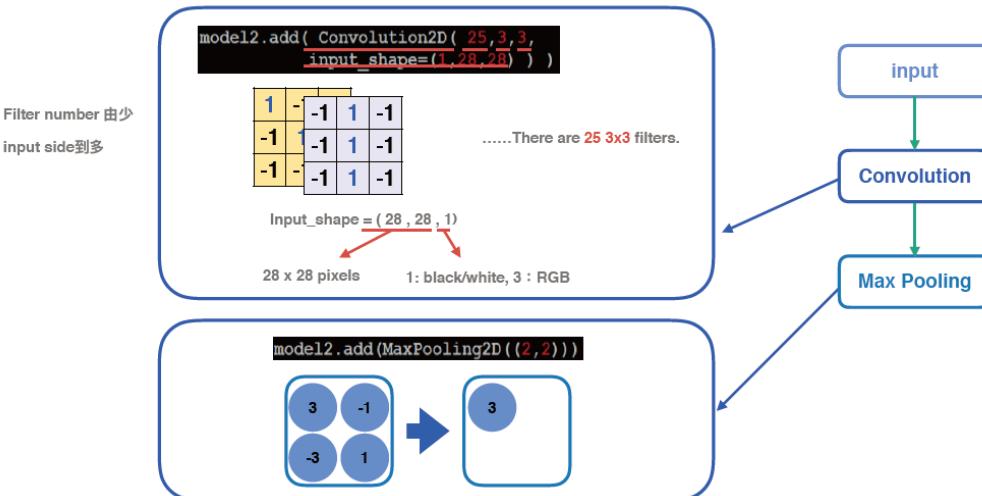
Q1. 若使用總共 128 個濾波器 (filters)，對一張 256×256 的影像做完卷積後，共會得到幾張特徵圖 (feature maps)？

A: 共會有 128 張特徵圖。使用一個 filter 可以得到一張特徵圖，若使用 128 個 filter 則是 128 張。

Q2. 卷積神經網路中，濾波器的大小 (3x3 or 5x5) 與內部數字的值 (value) 是怎麼決定的呢？

A: 濾波器的大小設定是超參數 (hyper-parameter)，必須要先由我們給定，目前主流模型皆是用 3x3 或 5x5；濾波器中的值則是屬於參數 (weights)，是透過訓練資料學習而來

每個 filter 包含多少個參數？



4-7 Keras 中的 CNN layers -- Conv2D

複習上一節，卷積神經網路就是透過疊起一層又一層的卷積層、池化層產生的。影像經過卷積後稱作特徵圖 (feature map)，經過多次卷積層後，特徵圖的尺寸(width, height)會越來越小，但是通道數(Channel)則會越來越大。

import Keras 中的 Conv2D，借下來對 input_image 進行 2D 卷積，即可得到我們的特徵圖 feature maps，下：

```
from keras.layers import Conv2D
```

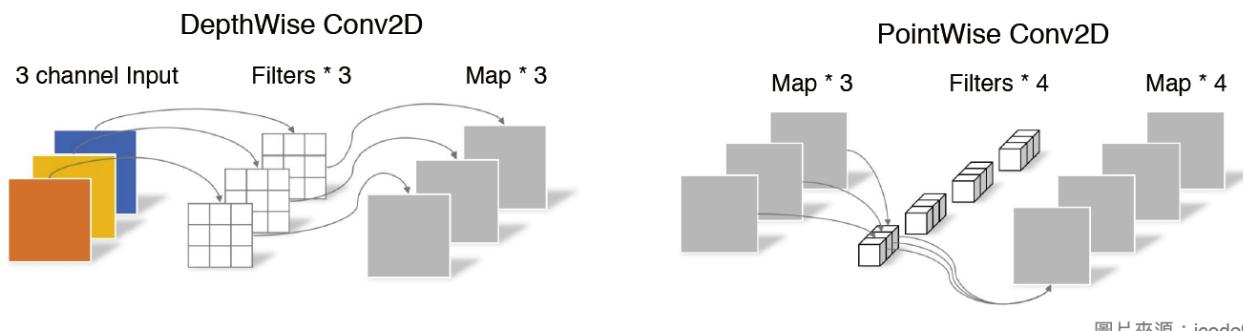
```
feature_maps = Conv2D(filters=128, kernel_size=(3,3), input_shape=input_image.shape)(input_image)
```

參數意義

- filters：濾波器的數量。此數字會等於做完卷積後特徵圖的通道數，通常設定為 2 的 n 次方。
- kernel_size：濾波器的大小。通常都是使用 3x3 或是 5x5。
- input_shape：只有對影像做第一次卷積時要指定，之後 Keras 會自動計算 input_shape。
- strides：做卷積時，濾波器移動的步長。[圖\(連結\)](#) strides 就是 1 (一次移動一格)。
- padding：是否要對輸入影像的邊緣補值。[圖\(連結\)](#) padding='same' (邊緣補一層 0)，稱為 same 的原因是因為做完 padding 再卷積後，輸出的特徵圖尺寸與輸入影像的尺寸不會改變。

Keras 中的 CNN layers- SeparableConv2D

全名稱做 Depthwise Separable Convolution，與常用的 Conv2D 效果類似，但是參數量可以大幅減少，減輕對硬體的需求。其對影像做兩次卷積，第一次稱為 DepthWise Conv，對影像的三個通道獨立做卷積，得到三張特徵圖；第二次稱為 PointWise Conv，使用 1x1 的 filter 尺寸做卷積。兩次卷積結合起來可以跟常用的卷積達到接近的效果，但參數量卻遠少於常見的卷積。更多資訊可參考[連結](#)。



圖片來源 : icode9

- filters, kernel_size, strides, padding 都與 Conv2D 相同
- depth_multiplier：在做 DepthWise Conv 時，輸出的特徵圖 Channel 數量會是 filters * depth_multiplier，預設為 1，上頁頁的簡報即為 1 的 DepthWise Conv

Code 37: Day096.9_Keras_CNN_layers_Ans.ipynb

作業

請閱讀 Keras 官方範例 [mnist_cnn.py](#)

並回答下列問題。僅有 70 行程式碼，請確保每一行的程式碼你都能夠理解目的

- 是否有對資料做標準化 (normalization)？如果有，在哪幾行？
- 使用的優化器 Optimizer 為何？
- 模型總共疊了幾層卷積層？
- 模型的參數量是多少？

4-8 遷移學習(Transfer Learning) -- ResNet

一般神經網路訓練前的初始參數是隨機產生的，不具備任何意義。**遷移學習(Transfer Learning)**是透過在其他龐大資料集上訓練好的模型參數，讓我們使用這個參數當成起始點，改用在自己的資料集上訓練！

為何可以用遷移學習？

後面 CNN 的課程會提到，CNN 淺層的過濾器 (filter) 是用來偵測線條與顏色等簡單的元素。因此不管圖像是什麼類型，基本的組成應該要是一樣的大型資料集訓練好的參數具有完整的顏色、線條 filters，從此參數開始，我們訓練在自己的資料集中，逐步把 filters 修正為適合自己資料集的結果。

同學們對於要疊幾層 CNN，filters 數量要選擇多少？

Stride, Pooling 等參數要設定多少？這些應該都很疑惑。許多學者們研究了許多架構與多次調整超參數，並在大型資料集如 ImageNet 上進行測試得到準確性高並容易泛化的網路架構，我們也可以從這樣的架構開始！

TRANSFER OF LEARNING



The application of skills, knowledge, and/or attitudes that were learned in one situation to another learning situation (Perkins, 1992)

圖片來源：[Transfer of learning](#)

Transfer learning in Keras: ResNet-50

```
from keras.applications.resnet50 import ResNet50  
  
resnet_model = ResNet50(input_shape=(224,224,3),  
weights="imagenet", pooling="avg",  
include_top=False)
```

我們使用了 ResNet50 網路結構，其中可以看到

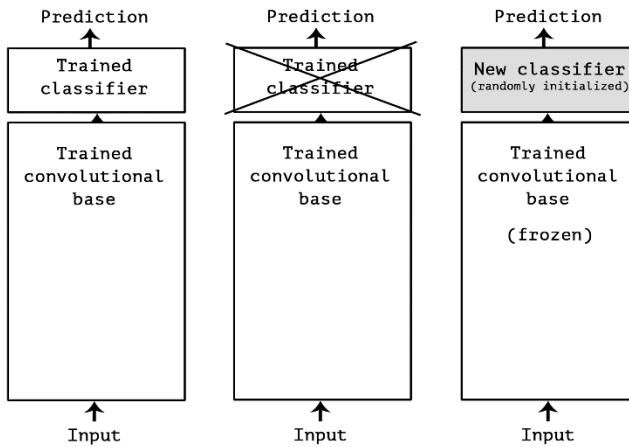
- weights="imagenet"，代表我們使用從 imagenet 訓練好的參數來初始化，並指定輸入的影像大小為 (224,224,3)。
- pooling=avg 代表最後一層使用 Global Average pooling，把 featuremaps 變成一維的向量。
- include_top=False 代表將原本的 Dense layer 拔掉，因為原本這個網路路是用來做 1000 個分類的模型，我們必須替換成自己的 Dense layer 來符合我們自己資料集的類別數量。

```
last_featuremaps = resnet_model.output  
flatten_featuremap = Flatten()(last_featuremaps)  
output = Dense(num_classes)(flatten_featuremap)  
  
New_resnet_model =  
Model(inputs=resnet_model.input, outputs=output)
```

上一步的模型我們已經設定成沒有 Dense layers，且最後一層做 GAP，使用 resnet_model.output 我們就可以取出最後一層的 featuremaps。

將其使用 Flatten 攤平後，再接上我們的 Dense layer，神經元數量與資料集的類別數量一致，重建立模型，就可以得到一個新的 ResNet-50 模型，且參數是根據 ImageNet 大型資料集預訓練好的。

整體流程如下圖，我們保留 Trained convolutional base，並新建 New classifier (Dense 的部分)，最後 convolutional base 是否要 frozen (不訓練) 則是要看資料集與預訓練的 ImageNet 是否相似，如果差異很大則建議訓練時不要 frozen，讓 CNN 的參數可以繼續更新。



From <https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.3-using-a-pretrained-convnet.ipynb>

練習作業：

要注意的是，以上的程式碼是提供給有 GPU 且具有較大影像尺寸資料集的同學參考，若沒有 GPU 的同學可以直接參考程式碼學習。Cifar-10 並不適合直接使用 transfer learning 原因是多數模型都是在 ImageNet 上預訓練好的，而 ImageNet 影像大小為 (224,224,3)，圖像差異極大，硬套用的結果反而不好。

礙於不是所有同學都有 GPU，因此程式碼範例使用的是簡化版本的 **ResNet**，確保所有同學都能夠訓練！可先參考這篇連結(<https://blog.gtwang.org/programming/keras-resnet-50-pre-trained-model-build-dogs-cats-image-classification-system/>)，有非常詳盡的文章，基本上已經涵蓋了所有訓練 CNN 常用的技巧，只要能夠妥善運用這些技巧（資料增強、遷移學習、調整學習率），即使簡單的模型也能帶來很高的基準點，再以此基準點微調參數，達到最佳的結果。

另外這些技巧在 Kaggle 上也會被許多人使用，更有人會開發一些新的技巧，例如使把預訓練在 ImageNet 上的模型當成 feature extractor 後，再拿擷取出的特徵重新訓練新的模型，有興趣的同學也可以參考 <https://www.kaggle.com/insaff/img-feature-extraction-with-pretrained-resnet>

進階挑戰！

有志以成為機器學習工程師為目標的同學們可以參加這個挑戰，這網站記錄了歷年來 Cifar-10 中排名最高的論文，請試著閱讀論文並撰寫出相對應的程式碼，復現出論文的結果。

這樣的能力在機器學習領域中是非常重要的，具備閱讀他人論文並實現的能力，可為自己在履歷上增添不少分數，當然難度也相當高，若是不透徹了解文章內容或是程式能力不夠紮實，可是不能復現別人辛苦的研究成果的喔！就請各位同學好好努力，往自己的機器學習之路邁進吧：）

#範例解答

https://github.com/keras-team/keras/blob/master/examples/cifar10_resnet.py

 **Code 38:** Day100.1_transfer_learning.ipynb

4-9 Kaggle 資料實戰練習 --花朵辨識

<https://www.kaggle.com/c/ml100marathon-final-exam>

The screenshot shows the 'Overview' tab of a Kaggle competition. The title is 'Cupoy ML 100 marathon Final exam' with the subtitle 'Cupoy 機器學習百日馬拉松-期末考'. It displays '32 teams · 2 days to go'. Below the tabs 'Overview' (selected), 'Data', 'Kernels', 'Discussion', 'Leaderboard', and 'Rules', there is a large 'Join Competition' button. The 'Description' section contains the competition's purpose: identifying five types of flowers (daisy, dandelion, rose, sunflower, tulip) using training images and applying learned knowledge to test images. The 'Evaluation' section describes the task as identifying the flower type based on image inputs.

此為是花朵辨識的資料，選定的範圍內共有五種不同種類的花：雛菊(daisy)、蒲公英(dandelion)、玫瑰(rose)、向日葵(sunflower)、鬱金香(tulip)，請以同學使用訓練資料當中的照片，並應用在深度學習階段所學到的內容，來辨識照片中的是哪種花。

特徵說明

圖形辨識的特徵就是圖檔本身，因此訓練特徵就是圖片本身，不另做說明。而作答的 id 就是檔名，可以詳閱 "Data" 分頁的說明以及 `sample_submission.csv` 的內容。

比較不同的是預測的輸出值，請同學特別注意：以數字 0 / 1 / 2 / 3 / 4 輸出你的要提交預測類別，而不是以花朵名稱輸出（建議以 Python Dictionary 轉換，或輸出時直接是類別碼，例如：`flower_mapping = {'daisy':0, 'dandelion':1, 'rose':2, 'sunflower':3, 'tulip':4}`）

檔案描述

- `sample_submission.csv` - 提交預測檔案範例
- `image_data.zip` - 本測驗所有的訓練與預測用圖片檔
 - `test` - 需要預測的所有圖片檔
 - `train` - 提供同學訓練用的圖片檔
 - `daisy` - [分類 0] 訓練用的雛菊圖檔
 - `dandelion` - [分類 1] 訓練用的蒲公英圖檔
 - `rose` - [分類 2] 訓練用的玫瑰圖檔
 - `sunflower` - [分類 3] 訓練用的向日葵圖檔
 - `tulip` - [分類 4] 訓練用的鬱金香圖檔



00c08828f ce04e360c 732cac01e dad9e.jpg	00d366e7 877b6a78 b104b57d 67b60e6...	00e9cb1dc a4078108 56e77b31 309d5ab.j..

Code 39: Day101-2_Final_KerasCNN_Solution_basic.ipynb

範例解答 - 方法一 (ImageDataGenerator)

```
# 先來看看訓練資料下有那些目錄吧
import os
train_path = './input/train/'
test_path = './input/test/'
print(os.listdir(train_path))

['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

STEP1. 載入套件

```
# 載入基礎套件
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 繪圖設定
from matplotlib import style
%matplotlib inline
style.use('fivethirtyeight')

# 設定忽略警告訊息
import warnings
warnings.filterwarnings('ignore')

# 載入 Keras 預處理套件 - ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator

# 載入 Keras 其他深度學習套件
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense, Flatten, Activation, Conv2D, MaxPooling2D
from keras.optimizers import Adam
#from keras.callbacks import ReduceLROnPlateau # 作用：當監控的目標值停止改善時，開始降低學習速率
from keras.utils import to_categorical # Keras 做 One Hot Encoding 的函數

# 其他相關套件
import tensorflow as tf
import random as rn
import cv2
from tqdm import tqdm # 進度條顯示套件
Using TensorFlow backend.
```

STEP 2. 資料預處理

```
# 初始化 Array, 並設定讀取路徑等參數
X_train = []
y_train = []
X_test = []
id_test = []
IMG_SIZE = 150

# 設定花朵名稱對應的類別碼
flower_mapping = {'daisy':0, 'dandelion':1, 'rose':2, 'sunflower':3, 'tulip':4}
flowers = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```

# 定義讀取資料的函數, 將圖檔資料讀取到輸入用的 Array 之中
def make_train_data(flower_type):
    dir_path = f'{train_path}{flower_type}/'
    for img in tqdm(os.listdir(dir_path)):
        y_train.append(str(flower_mapping[flower_type]))
        path = os.path.join(dir_path, img)
        img = cv2.imread(path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        X_train.append(np.array(img))
def make_test_data():
    dir_path = test_path
    for img in tqdm(os.listdir(dir_path)):
        id_test.append(img[-4])
        path = os.path.join(dir_path, img)
        img = cv2.imread(path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        X_test.append(np.array(img))

```

```

# 載入訓練資料
for flower in flowers:
    make_train_data(flower)
    print(f'Loading {flower} pictures, X_train length : {len(X_train)}')

```

Loading daisy pictures, X_train length : 500
 Loading dandelion pictures, X_train length : 1187
 Loading rose pictures, X_train length : 1702
 Loading sunflower pictures, X_train length : 2190
 Loading tulip pictures, X_train length : 2823

```

# 載入測試資料
make_test_data()
print(len(X_test))

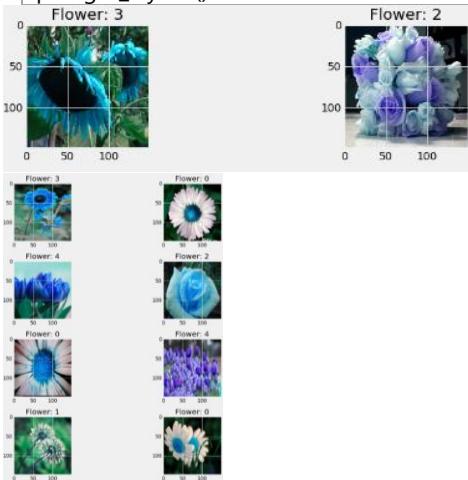
```

Loading tulip pictures, X_train length : 2823

```

# 隨機顯示 10 張花朵圖案, 確認資料無誤且載入無誤
fig, ax = plt.subplots(5, 2)
fig.set_size_inches(15, 15)
for i in range(5):
    for j in range(2):
        l = rn.randint(0, len(y_train))
        ax[i, j].imshow(X_train[l])
        ax[i, j].set_title('Flower: ' + y_train[l])
plt.tight_layout()

```



```
# 將影像特徵最大最小化(除以 255), 並將類別標籤做獨熱編碼(One-Hot Encoding)
X_train = np.array(X_train)
X_train = X_train / 255
y_train = to_categorical(y_train, 5)
X_test = np.array(X_test)
X_test = X_test / 255

# 設定隨機亂數種子
np.random.seed(123)
rn.seed(123)
tf.set_random_seed(123)
```

STEP 3. 設定模型與運算

```
# 使用 Keras Sequential 定義一個 CNN 模型
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'Same', activation = 'relu', input_shape = (150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = 'Same', activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(filters = 96, kernel_size = (3, 3), padding = 'Same', activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(filters = 96, kernel_size = (3, 3), padding = 'Same', activation = 'relu'))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

```
# 設定參數並使用 Keras 的 Callback 函數 - ReduceLROnPlateau
batch_size = 64
epochs = 50
#red_lr = ReduceLROnPlateau(monitor='val_acc', patience=3, verbose=1, factor=0.1)
```

```
# 設定 Image Generator
datagen = ImageDataGenerator(
    featurewise_center=False, # 是否將每個點的平均值歸 0
    samplewise_center=False, # 是否將每張圖的平均值歸 0
    featurewise_std_normalization=False, # 是否除以每個點的標準差
    samplewise_std_normalization=False, # 是否除以每張圖的標準差
    rotation_range=10, # 隨機旋轉角度的上限(角度值: 0 ~ 180)
    zoom_range = 0.1, # 影像倍率放大縮小的上限
    width_shift_range=0.2, # 左右隨機平移上限
    height_shift_range=0.2, # 上下隨機平移上限
    horizontal_flip=True, # 是否左右翻轉
    vertical_flip=False) # 是否上下翻轉
datagen.fit(X_train)
```

```
# 設定模型目標函數, 並顯示模型摘要
model.compile(optimizer = Adam(lr = 0.001), loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 150, 150, 32)	2432
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 75, 75, 32)	0
<hr/>		
(略)		
dense_2 (Dense)	(None, 5)	2565
<hr/>		
Total params: 4,143,749		
Trainable params: 4,143,749		
Non-trainable params: 0		

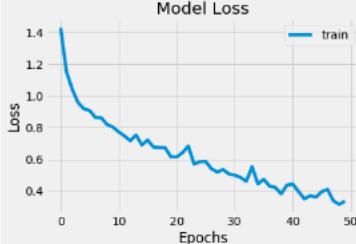
```
# 訓練模型，並記錄訓練集的 loss 與 Accuracy
History = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size), verbose = 1,
                             epochs = epochs, steps_per_epoch = X_train.shape[0] // batch_size)
print(len(X_test))

Epoch 1/50
44/44 [=====] - ETA: 2:30 - loss: 1.6180 - acc: 0.171 - ETA: 1:14 - loss: 1.6334 - acc: 0.171 - ETA: 0s - loss: 1.4272 - acc: 0.352 - ETA: 0s - loss: 1.4219 - acc: 0.354 - 11s 248ms/step - loss: 1.4187 - acc: 0.3555
(略)
Epoch 50/50
44/44 [=====] - ETA: 9s - loss: 0.2735 - acc: 0.906 - ETA: 9s - loss: 0.2796 - acc: 0.890 - ETA: 9s - loss: 0.3488 - acc: 0.869 - ETA: 9s - loss: 0.3208 - acc: 0.882 - ETA: 9s - loss: 0.3161 - acc: 0.881 - ETA: 9s - loss: 0.3004 - acc: 0.885 - ETA: 8s - loss: 0.2870 - acc: 0.895 - ETA: 8s - loss: 0.2955 - acc: 0.894 - ETA: 7s - loss: 0.2831 - acc: 0.891 - ETA: 6s - loss: 0.2846 - acc: 0.890 - ETA: 6s - loss: 0.2721 - acc: 0.897 - ETA: 6s - loss: 0.2857 - acc: 0.894 - ETA: 6s - loss: 0.2870 - acc: 0.894 - ETA: 6s - loss: 0.2887 - acc: 0.895 - ETA: 5s - loss: 0.2843 - acc: 0.897 - ETA: 5s - loss: 0.2861 - acc: 0.898 - ETA: 5s - loss: 0.2811 - acc: 0.900 - ETA: 5s - loss: 0.2922 - acc: 0.897 - ETA: 5s - loss: 0.2930 - acc: 0.898 - ETA: 4s - loss: 0.2955 - acc: 0.897 - ETA: 4s - loss: 0.2933 - acc: 0.898 - ETA: 4s - loss: 0.2921 - acc: 0.896 - ETA: 4s - loss: 0.2973 - acc: 0.894 - ETA: 4s - loss: 0.3029 - acc: 0.894 - ETA: 3s - loss: 0.3075 - acc: 0.892 - ETA: 3s - loss: 0.3079 - acc: 0.892 - ETA: 3s - loss: 0.3142 - acc: 0.889 - ETA: 3s - loss: 0.3121 - acc: 0.889 - ETA: 2s - loss: 0.3112 - acc: 0.890 - ETA: 2s - loss: 0.3241 - acc: 0.888 - ETA: 2s - loss: 0.3269 - acc: 0.886 - ETA: 2s - loss: 0.3314 - acc: 0.885 - ETA: 2s - loss: 0.3315 - acc: 0.885 - ETA: 1s - loss: 0.3304 - acc: 0.885 - ETA: 1s - loss: 0.3342 - acc: 0.883 - ETA: 1s - loss: 0.3341 - acc: 0.884 - ETA: 1s - loss: 0.3336 - acc: 0.883 - ETA: 0s - loss: 0.3340 - acc: 0.882 - ETA: 0s - loss: 0.3325 - acc: 0.883 - ETA: 0s - loss: 0.3311 - acc: 0.883 - ETA: 0s - loss: 0.3298 - acc: 0.884 - 10s 228ms/step - loss: 0.3327 - acc: 0.8814
```

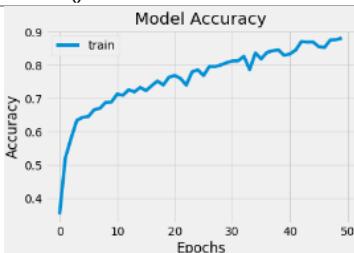
```
# 顯示三個 Array 的大小
print(f'X_tr:{X_train.shape} / y_tr:{y_train.shape} / X_te:{X_test.shape}')
X_tr:(2823, 150, 150, 3) / y_tr:(2823, 5) / X_te:(2000, 150, 150, 3)
```

STEP 4. 訓練結果影像化，並輸出預測

```
# 顯示訓練集的 Loss 結果
plt.plot(History.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train'])
plt.show()
```



```
# 顯示訓練集的 Accuracy 結果
plt.plot(History.history['acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train'])
plt.show()
```



```
# 輸出預測檔
pred = model.predict(X_test)
pred_digits = np.argmax(pred, axis=1)
sub = pd.DataFrame({'id': id_test, 'flower_class': pred_digits})
sub.to_csv('flower_CNN.csv', index=False)
```

Code 40: Day101-3_Final_Keras_PreTrain_SolutionFixed.ipynb

範例解答 - 方法二 (ImageDataGenerator +ResNet50)

```
import tensorflow as tf  
import numpy as np  
import os
```

```
# 先來看看訓練資料下有那些目錄吧  
import os  
train_path = './input/train/'  
test_path = './input/test/'  
print(os.listdir(train_path))  
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
# 載入基礎套件  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
# 繪圖設定  
from matplotlib import style  
%matplotlib inline  
style.use('fivethirtyeight')  
  
# 設定忽略警告訊息  
import warnings  
warnings.filterwarnings('ignore')
```

```
import tensorflow as tf  
import numpy as np  
import os
```

```
# 載入 Keras 預處理套件 - ImageDataGenerator  
from keras.preprocessing import image  
from keras.preprocessing.image import ImageDataGenerator  
  
# 載入 Keras 其他深度學習套件  
import keras  
from keras import backend as K  
from keras.models import Sequential, Model  
from keras.layers import Dense, Flatten, Activation, Conv2D, MaxPooling2D  
from keras.optimizers import Adam  
from keras.utils import to_categorical # Keras 做 One Hot Encoding 的函數  
from keras.callbacks import EarlyStopping, ModelCheckpoint  
from keras.applications.resnet50 import ResNet50, preprocess_input
```

Using TensorFlow backend.

```
# 其他相關套件  
import tensorflow as tf  
import random as rn  
import cv2  
import glob  
from PIL import Image  
from tqdm import tqdm # 進度條顯示套件
```

2. 資料預處理

```
# 設定執行參數
IMAGE_SIZE = (240, 320)
BATCH_SIZE = 16 # 設 32 效果會更好, 但同學要依照 GPU 上的記憶體量力而為, 如果跑不動可以降到 8 或更低

# 設定花朵名稱對應的類別碼
flower_mapping = {'daisy':0, 'dandelion':1, 'rose':2, 'sunflower':3, 'tulip':4}
flowers = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
# 設定 ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=10, # 隨機旋轉角度的上限 (角度值: 0 ~ 180)
    width_shift_range=0.2, # 左右隨機平移上限
    height_shift_range=0.2, # 上下隨機平移上限
    shear_range=0.2, # 逆時針的剪切變形角度
    zoom_range=0.2, # 影像倍率放大縮小的上限
    horizontal_flip=True, # 是否左右翻轉
    fill_mode='nearest', # 填補周圍空缺方法
    validation_split=0.2, # 測試集採樣比
    preprocessing_function=preprocess_input) # 資料前處理
```

```
# 設定輸入的 train / test 資料流
train_generator = datagen.flow_from_directory(train_path, target_size=(240, 320), classes=flowers,
                                              batch_size=BATCH_SIZE, subset='training') # 設為訓練集
valid_generator = datagen.flow_from_directory(train_path, target_size=(240, 320), classes=flowers,
                                              batch_size=BATCH_SIZE, subset='validation') # 設為驗證集
```

Found 2260 images belonging to 5 classes.

Found 563 images belonging to 5 classes.

3. 設定模型與運算

```
# 使用 Keras 定義一個 Resnet 模型
resnet = ResNet50(input_shape=(240, 320, 3), weights='imagenet', pooling='avg', include_top=False)
output = keras.layers.Dense(5, activation="softmax")(resnet.output)
model = Model(inputs=[resnet.input], outputs=[output])
```

```
# 顯示模型形狀
model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 240, 320, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 246, 326, 3)	0	input_1[0][0]
conv1 (Conv2D)	(None, 120, 160, 64)	9472	conv1_pad[0][0]
bn_conv1 (BatchNormalization)	(None, 120, 160, 64)	256	conv1[0][0]
.....(略).....			
bn5c_branch2c (BatchNormalizati	(None, 8, 10, 2048)	8192	res5c_branch2c[0][0]
add_16 (Add)	(None, 8, 10, 2048)	0	bn5c_branch2c[0][0]
activation_46	[0][0]		
activation_49 (Activation)	(None, 8, 10, 2048)	0	add_16[0][0]
global_average_pooling2d_1 (Glo	(None, 2048)	0	activation_49[0][0]
dense_1 (Dense)	(None, 5)	10245	global_average_pooling2d_1[0][0]
Total params:	23,597,957		
Trainable params:	23,544,837		
Non-trainable params:	53,120		

```
# 設定模型收斂方式並編譯模型
model.compile(optimizer=keras.optimizers.Adam(lr=1e-5), loss='categorical_crossentropy', metrics=['acc'])
```

```
# 設定 early stop (若 valid accuracy 沒有更好就提早停下) 與 call back (紀錄最好一次的模型參數)
early_stopping = EarlyStopping(monitor='val_acc', patience=30)
model_checkpoint = ModelCheckpoint("best_model.h5", save_best_only=True)
```

```
# 執行, 並將結果儲存於 history 物件中
history = model.fit_generator(train_generator, steps_per_epoch=train_generator.samples // BATCH_SIZE,
                             validation_data=valid_generator, validation_steps=valid_generator.samples // BATCH_SIZE,
                             use_multiprocessing=False, verbose=2, epochs=100,
                             workers=8, callbacks=[early_stopping, model_checkpoint])
```

Epoch 1/100
- 57s - loss: 0.9264 - acc: 0.6693 - val_loss: 0.4155 - val_acc: 0.8615

(略)

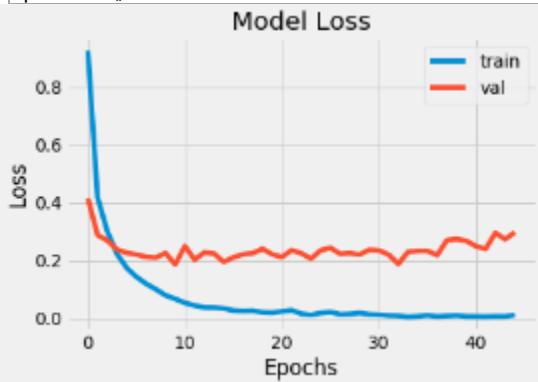
Epoch 45/100
- 44s - loss: 0.0177 - acc: 0.9951 - val_loss: 0.2973 - val_acc: 0.9325

重新載回最好一次的模型參數

```
best_model = keras.models.load_model("best_model.h5")
#model.load_weights("best_model.h5")
```

4. 訓練結果影像化, 並輸出預測

```
# 顯示訓練集的 Loss 結果
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'val'])
plt.show()
```



```
# 顯示訓練集的 Accuracy 結果
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'val'])
plt.show()
```



```
# 設定最後的輸出資料
X_test = list()
id_test = []
```

```
# 使用 Keras 內建的載入寫法
def load_image(img_path):
    img = image.load_img(img_path, target_size=(240, 320))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    return img
def make_test_data():
    global X_test
    dir_path = test_path
    for img in tqdm(os.listdir(dir_path)):
        id_test.append(img[:-4])
        path = os.path.join(dir_path, img)
        img = load_image(path)
        img = np.squeeze(img, axis=0)
        X_test.append(np.array(img))
make_test_data()
X_test = np.array(X_test)
```

100% | 2000/2000 [00:05<00:00, 367.01it/s]

```
# 輸出預測檔
pred = best_model.predict(X_test)
pred_digits = np.argmax(pred, axis=1)
sub = pd.DataFrame({'id': id_test, 'flower_class': pred_digits})
sub.to_csv('flower_Pretrain.csv', index=False)
```

附錄 網路學習資源

[推薦] GitHub 機器學習- 100 天自學專案 <https://buzzorange.com/techorange/2018/08/06/github-100-days-learning/>

README.md

100-Days-Of-ML-Code

100 Days of Machine Learning Coding as proposed by Siraj Raval

Get the datasets from [here](#)

Data PreProcessing | Day 1

Check out the code from [here](#).

