

Python AI

機器學習到深度學習



Scan me

周凡剛(Elwing)著

intro

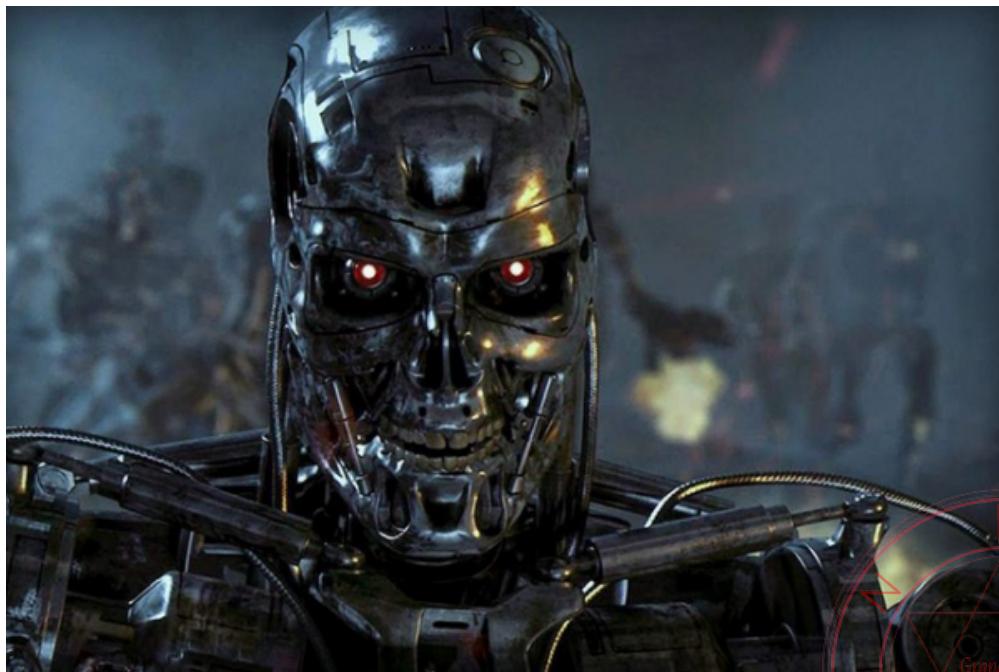
2018 年 6 月 22 日



1 機器學習介紹

說到機器學習，我們就得先定義幾個名詞

1.1 AI (Artificial Intelligence)



機器是否能擁有智慧呢？



如果可以有的話，要怎麼定義擁有智慧這件事呢？

著名的電腦科學家艾倫·圖靈提出一個很有名的測試方法，我們直到現在都還在用它，他叫做圖靈測試。

1.1.1 圖靈測試

描述：如果一個人（代號 C）使用測試對象皆理解的語言去詢問兩個他不能看見的對象任意一串問題。對象為：一個是正常思維的人（代號 B）、一個是機器（代號 A）。如果經過若干詢問以後，C 不能得出實質的區別來分辨 A 與 B 的不同，則此機器 A 通過圖靈測試。

白話文就是：你分出來這台機器跟人類有什麼不同！

從這測試被提出後，我們就開始用圖靈測試當作 AI 的最高守則

但是我們最後發現，一個機器要在所有領域通過圖靈測試，也就是跟真正的人類一樣，以現在技術要達成是做不到的！！

所以我們把通過圖靈測試的機器分成兩種：1. 強人工智慧 2. 弱人工智慧

1.1.2 強人工智慧

如果今天我製造出一個機器，他可以在所有領域都通過圖靈測試，透過學習學會任何判斷，我們就稱為強人工智慧。

但大家普遍的認知，在現今的科技是無法做到的

1.1.3 弱人工智慧

今天有一個機器也可以通過圖靈測試，但只限於某些領域，可能是語言分析領域或者是影像分析領域，那我們就稱它為弱人工智慧。

這也是我們今天在討論的人工智慧！

1.2 AI 發展史

1.2.1 專家系統

有一陣子，大家嘗試著把所有知識化成一條一條的規則輸入到電腦裡，但後來大家發現兩個難處

1. 人類的知識量太大太大了
2. 有些抽象的知識 (ex. 飛的概念) 根本不知道怎麼化成一個描述

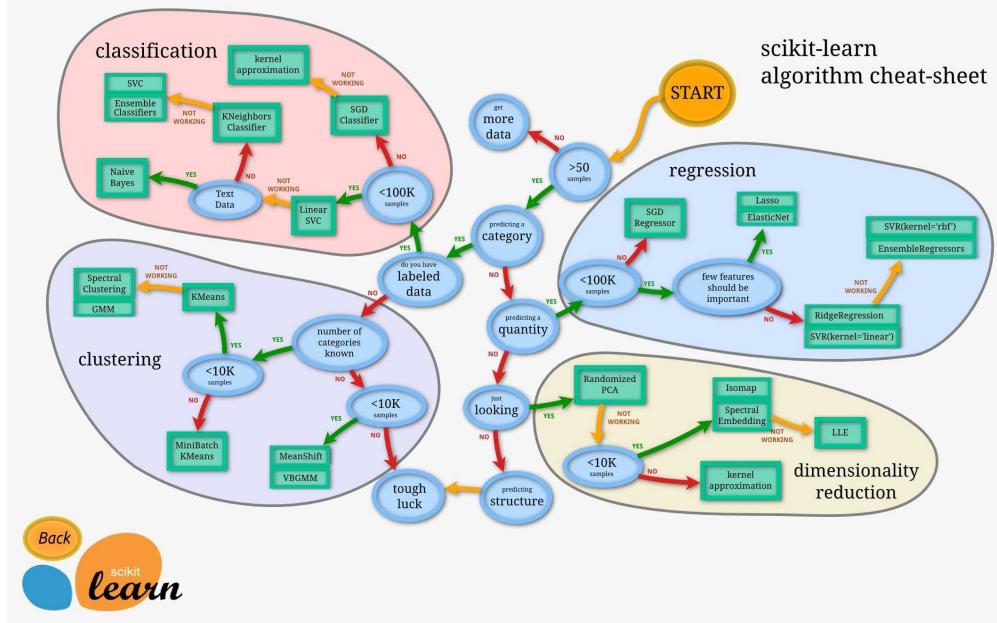
所以專家系統就沈寂了，同時起來的就是透過機率和統計模型來試圖根據得到的資料做出判斷，也就是我們的機器學習



1.2.2 機器學習

簡單來說，就是機率與統計，透過擁有的資料對未知的資料做出判斷，非常多的演算法被提出，每個演算法都有自己擅長的領域以及適合的資料

有哪些演算法呢？給各位一些概念，後續我們會就裡面經典的演算法介紹



1.2.3 深度學習

在機器學習的發展史我們最後遇到一個困境，我們對於預測抽象的概念，譬如什麼是一個椅子，或者什麼是信仰，雖然可以預測

但是預測的結果不盡理想，而且做出來的模型解釋性很差，所以就有人想了一個概念，我們何不模擬人類神經運作的方式呢？

雖然神經運作的方式至今沒人理解，但大家有個共同的共識，就是人類在學習的過程中是由具象（椅子的邊角）學習到抽象（什麼是椅子）

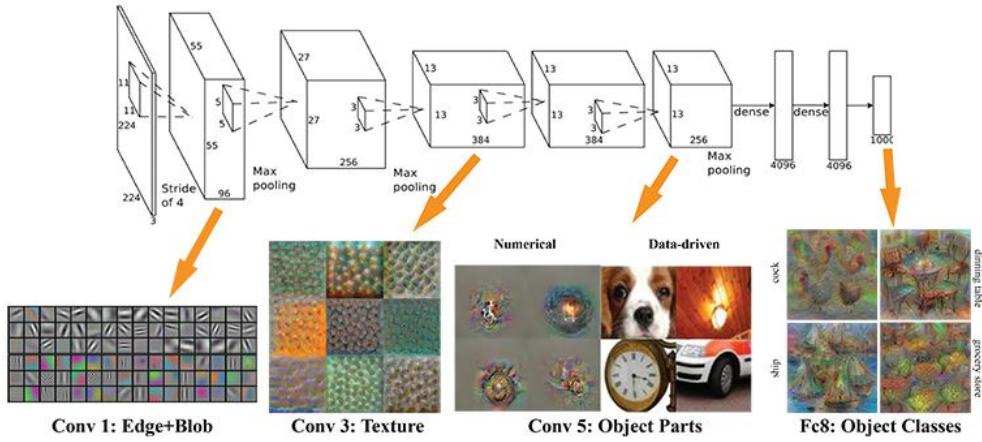
所以機器學習其中的一種演算法（人工神經網路）就此興起，因為這種演算法可以在中間加入很多層的過濾，看起來就好像我們的神經系統

一層一層從具象到抽象！！

所以其實所有的人工神經網路都可以擴展成深度的人工神經網路

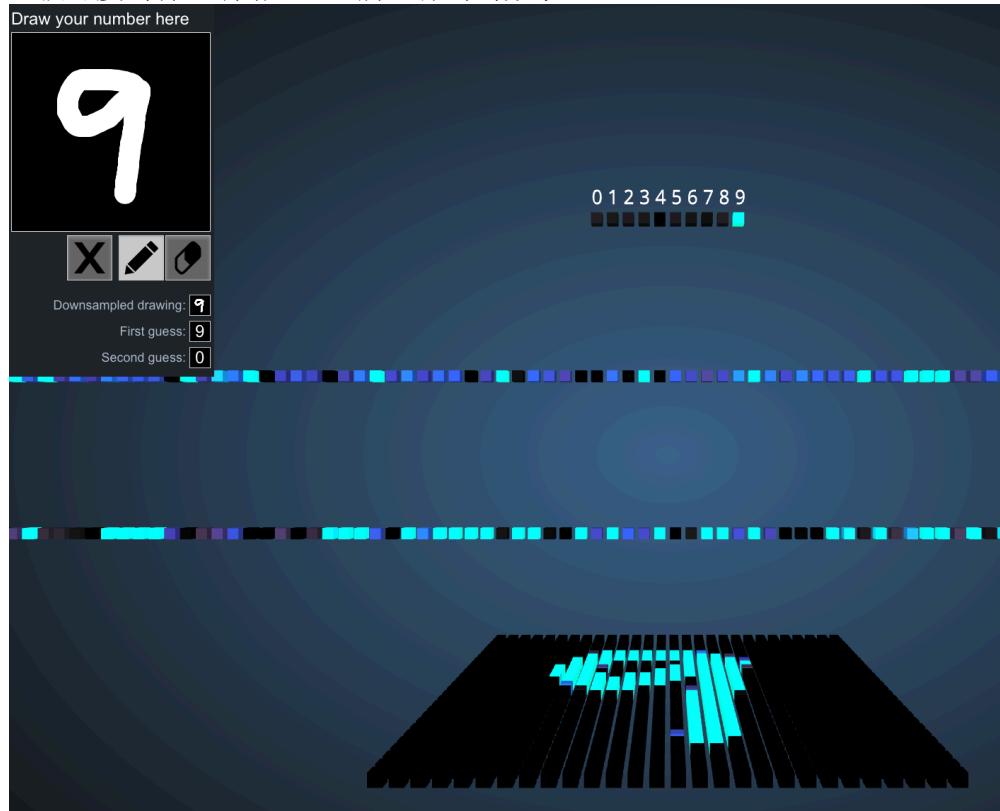
給大家一個例子來看看什麼是具象到抽象，圖擷取自 <http://xilinx.eetrend.com/article/10827>





有興趣的讀者可以玩玩看 <http://scs.ryerson.ca/~aharley/vis/fc/>

一個深度的神經網路，可以辨認你寫的數字



1.3 學習類型

1.3.1 資料類型

根據我們手上的資料類型，可以分成三種處理方式

1. 監督式學習：資料已經被填上了正確答案，例如：每一首詩已經被註明是哪個詩人寫的



2. 非監督式學習：有時候因為資料太多，或者沒有一個可以填入正確答案的手段，我們就只能靠電腦自己去填已有資料的答案。
3. 強化學習：算是介於中間，是有正確答案的，不過這正確答案是從環境產生，例如：教電腦玩遊戲，遊戲中的分數其實就是你每一個資料的答案

前兩種方式是接下來我們討論的重點，當然你已經可以預想到，有答案的資料總是比沒有答案的資料訓練效果要好

就像老師已經給了你十題例題的答案，你做未知的第一十一題就比較容易對!!!

1.3.2 預測答案類型

根據我們要預測的答案，又可以分成兩種預測

1. 分類：預測的答案不是連續性的數字，而只是單純類型預測 (ex. 預測寫詩的詩人)
2. 迴歸：預測的答案是連續的數字 (ex. 預測房價跟附近環境的關係)

1.3.3 應用

接下來我們看看資料類型 + 預測答案類型對應哪些我們現實生活的應用！



scikit-datasets

2018 年 6 月 22 日



1 scikit-learn 內建資料集

1.1 介紹

scikit 內建一些供我們可以練習基礎的資料集，你可以先藉由這些內建資料集來確定你的模型沒問題以後，再應用到現實世界的資料集機器學習的兩大類問題，我們可以分別先使用裡面的兩個資料集來試試看 1. 分類問題：目標不是連續的數值，而是標籤類型的，我們要預判這些標籤 -> 燕尾花數據集 2. 迴歸問題：目標是連續的數值，我們要預測這些連續的數值 -> 波士頓房產數據集

1.2 使用函式庫

請使用命令列或者 pycharm 安裝以下四個函式庫

1. pandas: 在處理表格和矩陣的時候，我還是習慣統一使用 pandas 做為我們的處理工具
2. numpy: 提供許多數學運算，包括許多的矩陣運算
3. scipy: 提供許多工程運算，包括線性代數，微積分等等
4. scikit-learn: 實作了許多機器學習基本方法的一個函式庫，也提供我們很多處理資料的方法以及內建資料集，基於 scipy 和 numpy

1.3 官方網站和例子

1. 官方網站: <http://scikit-learn.org/stable/index.html>
2. 內建資料集: <http://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets>

1.4 燕尾花資料集 (分類問題)

共有三種不同的燕尾花 (圖片引用自 wiki 百科)，我們要用花瓣 (petal) 的長寬和花萼 (sepal) 的長寬來做出分類



1. 山鳶尾 (Setosa) - 資料集裡以 0 做表示
2. 變色鳶尾 (Versicolor) - 資料集裡以 1 做表示
3. 維吉尼亞鳶尾 (Virginica) - 資料集裡以 2 做表示

```
In [1]: import pandas as pd
```

為了顯示的漂亮，我刻意的把印出來的 row 只顯示 10 個和 column 只顯示十個

大家練習的時候可以去掉下面兩行

```
pd.set_option('display.max_rows', 10)
```

```
pd.set_option('display.max_columns', 10)
```

```
In [2]: from sklearn.datasets import load_iris
```

```
In [3]: iris = load_iris()
```

讀者可以自行把下面行的註解拿掉 印出 iris

```
# iris
```

```
In [4]: # 先用 data 和他的 column 創出 df
```

```
df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
```

```
df["target"] = iris["target"]
```

```
df
```

```
Out[4]:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) \

```

0	5.1	3.5	1.4	0.2
---	-----	-----	-----	-----

1	4.9	3.0	1.4	0.2
---	-----	-----	-----	-----

2	4.7	3.2	1.3	0.2
---	-----	-----	-----	-----

3	4.6	3.1	1.5	0.2
---	-----	-----	-----	-----

4	5.0	3.6	1.4	0.2
---	-----	-----	-----	-----

..
----	-----	-----	-----	-----

145	6.7	3.0	5.2	2.3
-----	-----	-----	-----	-----

146	6.3	2.5	5.0	1.9
-----	-----	-----	-----	-----

147	6.5	3.0	5.2	2.0
-----	-----	-----	-----	-----

148	6.2	3.4	5.4	2.3
-----	-----	-----	-----	-----

149	5.9	3.0	5.1	1.8
-----	-----	-----	-----	-----

	target
0	0
1	0
2	0



```

3          0
4          0
...
145        2
146        2
147        2
148        2
149        2

[150 rows x 5 columns]

```

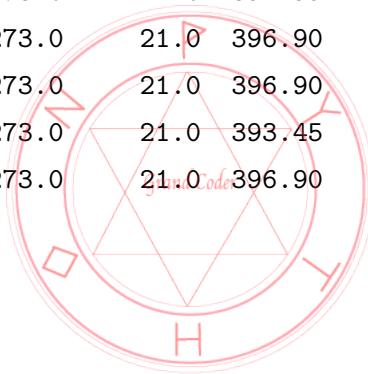
1.5 波士頓地產資料集 (迴歸問題)

把波士頓地區的房價和當地區的特徵列出來，由於房價是一個連續的值，所以我們通常會拿來學習回歸（以下翻譯引用自 http://sklearn.apachecn.org/cn/0.19.0/sklearn/datasets/descr/boston_house_prices.html）

```
In [5]: from sklearn.datasets import load_boston
        boston = load_boston()
        # 讀者可以自行把下面行的註解拿掉 印出 boston
        # boston
```

```
In [6]: df = pd.DataFrame(data = boston['data'], columns = boston['feature_names'])
        df['target'] = boston['target']
        df
```

	CRIM	ZN	INDUS	CHAS	NOX	...	TAX	PTRATIO	B	LSTAT	\
0	0.00632	18.0	2.31	0.0	0.538	...	296.0	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0.0	0.469	...	242.0	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0.0	0.469	...	242.0	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0.0	0.458	...	222.0	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0.0	0.458	...	222.0	18.7	396.90	5.33	
...	
501	0.06263	0.0	11.93	0.0	0.573	...	273.0	21.0	391.99	9.67	
502	0.04527	0.0	11.93	0.0	0.573	...	273.0	21.0	396.90	9.08	
503	0.06076	0.0	11.93	0.0	0.573	...	273.0	21.0	396.90	5.64	
504	0.10959	0.0	11.93	0.0	0.573	...	273.0	21.0	393.45	6.48	
505	0.04741	0.0	11.93	0.0	0.573	...	273.0	21.0	396.90	7.88	



	target
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
..	...
501	22.4
502	20.6
503	23.9
504	22.0
505	11.9

[506 rows x 14 columns]



ClassificationTree

2018 年 6 月 22 日



1 決策樹 (分類)

1.1 介紹

讓我們從決策樹開始進入機器學習的入門，決策樹的概念非常的簡單，就是每次選擇一個特徵，將你的資料分成多個部分

1.2 例子

如果一個女生的擇偶條件，第一個條件是年齡在 x 歲之下，再接下來的第二個條件是要會彈吉他，這個判定的條件就很有決策樹的感覺，所以決策樹其實就是一層層條件的篩選！

1.3 優點

- 建立起來的模型非常符合人類的直覺 (先選一個特徵來做出分類)，也非常容易解釋

1.4 缺點

- 容易發生過擬合 (Overfitting) 的現象，也就是做出來的樹對於離群點 (特立獨行的點) 也想辦法納入考慮了，那這樣正常的資料就有機會無法被正常的分類

1.5 分割理論

既然現在我們想要透過特徵來做出分割，問題就變成了到底要用哪個特徵來分割比較適合！



1.5.1 情境假設

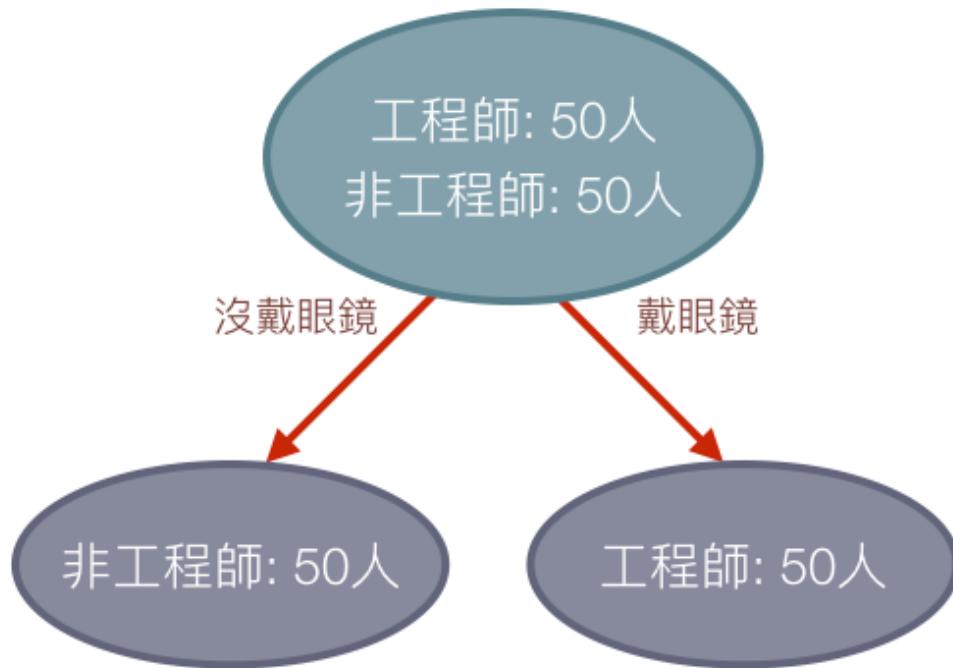
現在我們假設我們想預測一個人是不是軟體設計師，我們有兩個特徵，特徵一：戴眼鏡與否，特徵二：手指有繭與否

寫的更清楚一點：

1. 輸入 1: 戴眼鏡與否 (只有是否兩個值)
2. 輸入 2: 手指有繭與否 (只有是否兩個值)
3. 輸出: 工程師與否 (只有是否兩個值)

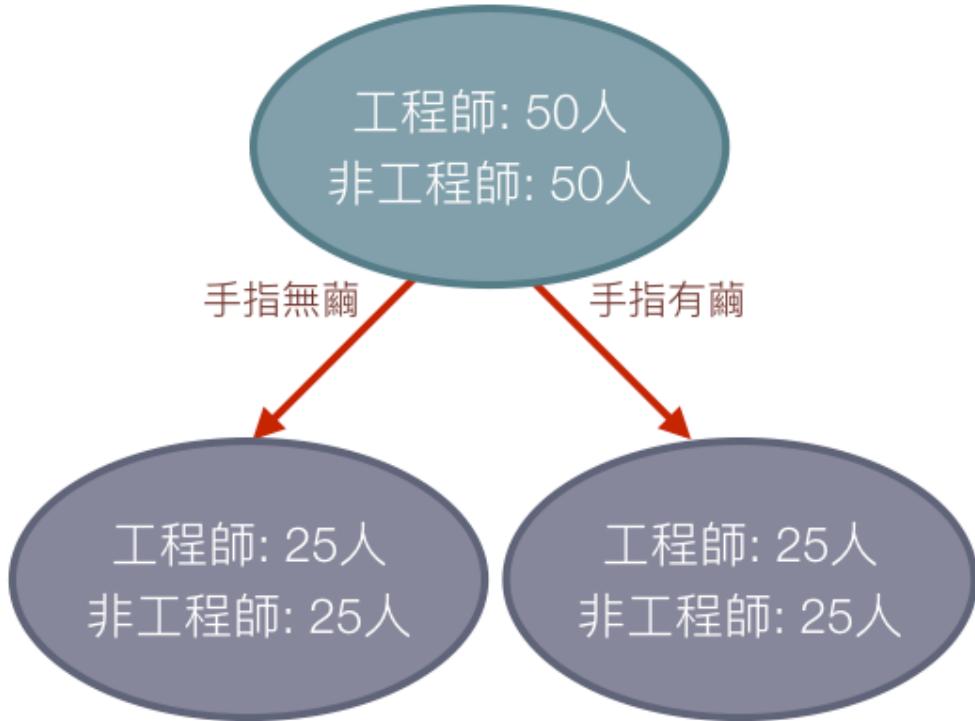
那接下來想請教各位一個問題

情境 1: 假設我們用戴眼鏡與否分出來以後長成這樣



情境 2: 假設我們用手指長繭與否分出來以後長成這樣





究竟該選擇哪一個作為我們的分類特徵呢？聰明如你應該也發現了，情境 2 有分跟沒分是一樣的啊！！工程師與非工程師的分布還是一半一半

那接下來我們用比較清楚的表示來描述一下上面這件事

1.5.2 特徵選擇以及亂度

亂度：不確定性

根據人類物理學研究，發現如果一個系統沒有外界的介入，會傾向於整體能量低（穩定），內部不確定性高（自由）的方向發展，我們不用物理學的方向說，我們以歷史的角度說，我們歷史上的朝代在沒有外界的介入下，發展出了多樣化的職業（自由度高，不確定性高），這些多樣化的職業構成了穩定的時代（穩定度高）。

回到我們的預測，不確定性高代表裡面可能性太多，對於我們的預測來說，因為可能性太多，就只能純粹瞎猜。所以我們在選擇特徵的時候，要反過來，盡量在切割後讓整個系統的不確定性低，才不是單純在瞎猜。

熵 (Entropy): 衡量亂度的一個方式

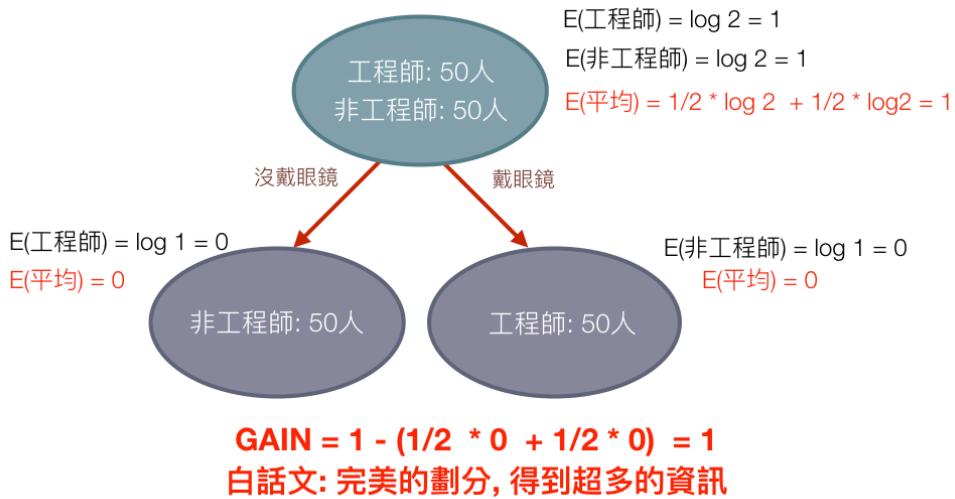
$$E(\text{某種可能性}) = \log(\text{某種可能性的機率}) = \log(\text{某種可能性的數量} / \text{全部數量})$$

$$G(\text{分類後得到的系統增益}) = E(\text{原本亂度}) - E(\text{後來的亂度})$$

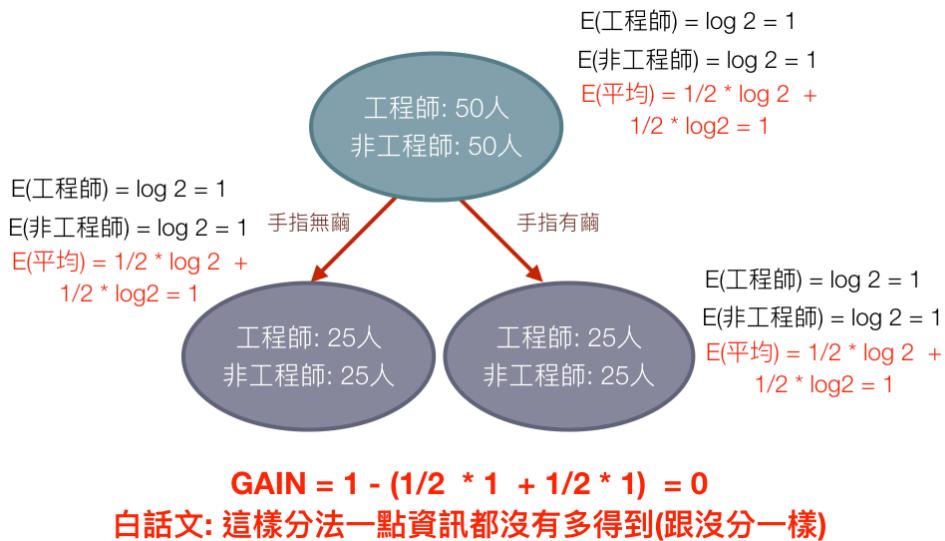
接下來我們針對上面的兩個圖來算一下總體的熵

情境 1: 用戴眼鏡與否分得到的增益





情境 2: 用手指長繭與否分得到的增益



*** 毫無疑問，我選擇增益多的情境 1(戴眼鏡與否) 當我們的分類特徵 ***

*** 只要不斷的重複這步驟，我們就可以把我們樹創建出來 ***

1.6 決策樹的歷史

1.6.1 ID3 決策樹:

根據我們上面的理論，每次就選一個特徵，把特徵所有可能的值分出不同的群出來
但有個很大的問題是，ID3 決策樹會傾向於選擇特徵值可能性最多的來當劃分
舉例來說，假設你有個特徵是學號，那 ID3 幾乎百分百會選擇學號來當劃分的基礎
因為值最多相當於畫出來最多個子樹，亂度當然會比較小



1.6.2 C4.5 決策樹:

為了解決上面 ID3 的缺點，我們引入了一個懲罰項，把廣度和均勻度納入考量，不要讓樹分成太多個子節點，但每個節點只有少數幾個資料

1.6.3 CART(Classification And Regression Tree) 樹:

CART 樹是我們最後發展出來的樹，也是 scikit-learn 所使用的樹演算法，他最重要的特點是每次劃分一定只劃分出兩個子節點

這樣劃分的話，上面的問題就不存在了，因為每一次的劃分都是分出左右兩個，而不會分出超級多個節點

而且，最重要的是

這樣的劃分，也讓 CART 樹可在很好的利用在迴歸問題上，我們下一節就來看看迴歸的問題

1.7 開始撰寫程式

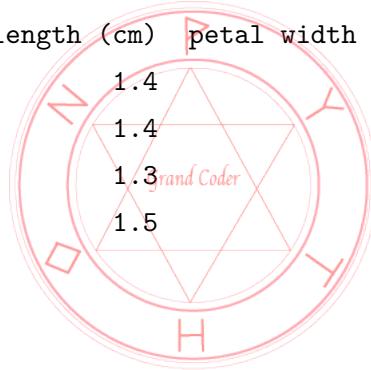
1.7.1 Step 0. 讀入我們的鳶尾花數據集作為練習

```
In [1]: from sklearn.datasets import load_iris
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)

# 使用 scikit-learn 提供的鳶尾花資料庫
iris = load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df["target"] = iris["target"]
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	



4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
..
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

	target
0	0
1	0
2	0
3	0
4	0
5	0
6	0
..	...
143	2
144	2
145	2
146	2
147	2
148	2
149	2

[150 rows x 5 columns]

1.7.2 Step 1. 先畫個圖

相關係數: 兩個東西的相關性

完全正相關 (兩個東西總是一起上升): 1

完全正相關 (一個東西上升的時候, 另一個總是下降): -1

我們先使用 heatmap(根據不同的數值給你不同的顏色), 並且把我們的鳶尾花數據做出一個相

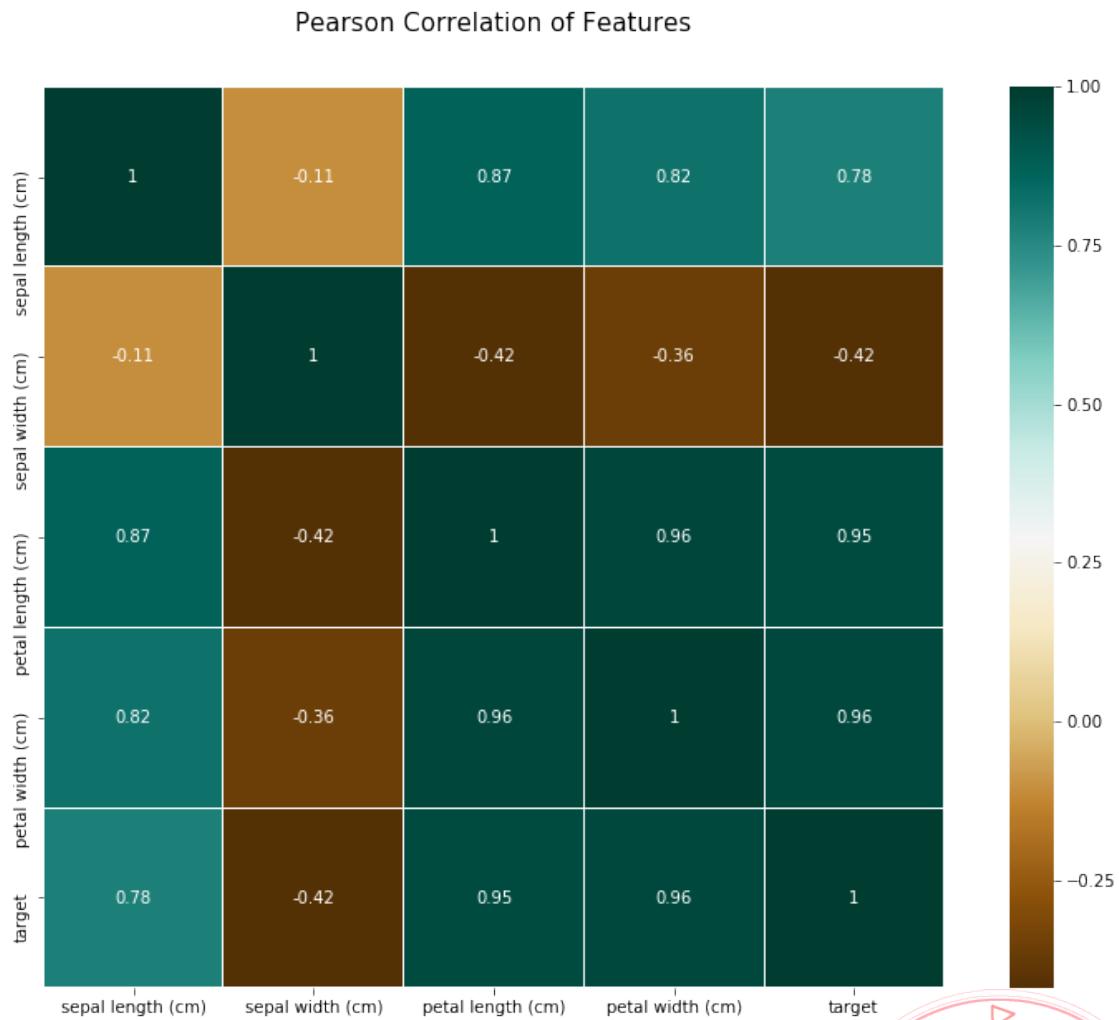


關係數矩陣，丟入我們的 heatmap

補充：你可以使用 https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html 選擇一個你喜歡的調色盤放入 cmap 參數

```
In [2]: plt.figure(figsize=(14,10))
        plt.title('Pearson Correlation of Features', y=1.05, size=15)
        sns.heatmap(df.astype(float).corr(), cmap = "BrBG",
                    linewidths=0.1, square=True, linecolor='white',
                    annot=True)
```

Out [2]: <matplotlib.axes._subplots.AxesSubplot at 0x10422d550>



在這裡我們可以先有了一個小小的想法，對於我們資料集，我們先看一下最下面的一列，我們



發現，對於我們的目標來說，花瓣是一個非常重要的資訊，不管是長度和寬度對於我們的目標分類都是幾乎完全的正相關，所以理論上，之後我們的樹應該會選擇花瓣作為分類基準

In [3]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練

```
from sklearn.model_selection import train_test_split
# 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
data_train, data_test, target_train, target_test = train_test_split(iris['data'],
                                                               iris['target'],
                                                               test_size=0.1)
```

1.7.3 Step 2. 訓練模型

我們使用 DecisionTreeClassifier 來訓練

在使用 sklearn 的時候我們都會用下面兩步驟來訓練

1. 創好一個 Classifier
2. 使用 fit 將你要訓練的數據餵進來

In [4]: from sklearn.tree import DecisionTreeClassifier

```
# 我們先試試看讓樹的最大深度是 3
clf = DecisionTreeClassifier(max_depth = 3)
clf = clf.fit(data_train, target_train)
```

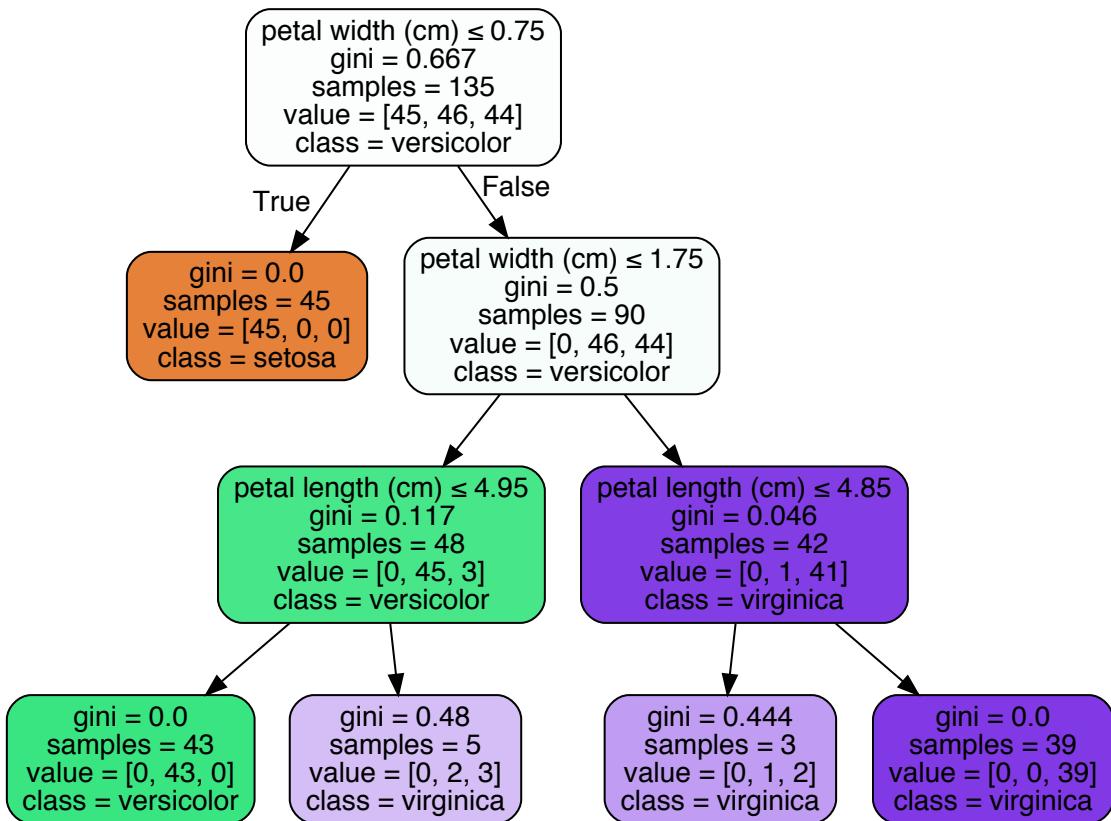
訓練完畫個圖吧，我們使用第三方軟體 graphviz 以及 python 函式庫來操作 graphviz

請完成下面的安裝步驟 1. 請來到 <http://www.graphviz.org/download/> 安裝 graphviz 到你的電腦 2. 請使用 pip 或者是 pycharm 安裝好 graphviz 函式庫

In [5]: from sklearn.tree import export_graphviz
import graphviz
dot_data = export_graphviz(clf, out_file=None,
 feature_names=iris.feature_names,
 class_names=iris.target_names,
 filled=True, rounded=True,
 special_characters=True)
graph = graphviz.Source(dot_data)
這行可以輸出一個 pdf，讀者可以自行把註解拿掉試試看
graph.render("iris2")
graph



Out [5] :



你發現，在選擇標籤的時候，的確以花瓣的長寬做為我們的主要選擇就可以達到幾乎完美的分類程度。

1.7.4 Step 3. 開始預測

使用你剛剛的 classifier 進行 predict，predict 完成以後使用 sklearn 內建的 accuracy_score 來算出正確機率

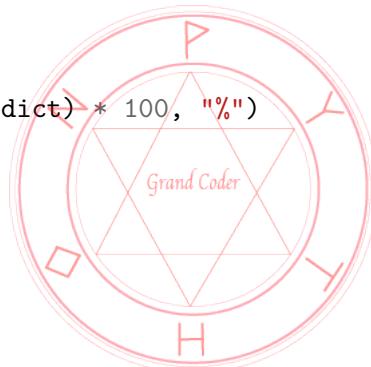
In [6]: `from sklearn.metrics import accuracy_score`

```

predict = clf.predict(data_test)
print("預測:", predict)
print("正確標籤:", target_test)
print("正確率: ", accuracy_score(target_test, predict) * 100, "%")
  
```

預測: [0 0 2 1 1 2 2 0 0 2 2 1 1 0 1]

正確標籤: [0 0 2 1 2 2 2 0 0 2 2 1 1 0 1]



正確率： 93.3333333333 %

1.7.5 Step 4. 確認一下分類錯誤

你可以使用混淆矩陣 (confusion matrix) 來判斷正確的分類以及不正確的分類

像下面的圖，列是你的正確標籤，行則是你的預測，當你的列數字不等於行數字的時候就是分類錯誤的情況

```
In [7]: from sklearn.metrics import confusion_matrix
cm = confusion_matrix(target_test, predict)
pd.DataFrame(cm)
```

```
Out[7]:   0  1  2
0  5  0  0
1  0  4  0
2  0  1  5
```

最後的最後，你可以檢查一下我們做出來的分類器，對於每個特徵的看重程度

你會發現，跟我們一開始由圖預測的一樣，花瓣 (後兩個) 的重要程度比花萼 (前兩個) 來的重要太多了

```
In [8]: clf.feature_importances_
```

```
Out[8]: array([ 0.          ,  0.          ,  0.04456771,  0.95543229])
```

1.8 剪枝

現在，我們回去調整一下你的 Classifier 的 depth，你會發現，深度的提升並不一定是件好事

深度的提升代表你用了更多個特徵，甚至是一些根本無用或者是比較特異的特徵 (記得，我們預測是針對大群體) 來做出分類

這件事情我們叫做過擬合，換成現實的話，就是你考慮了太多意外的要素或根本不重要的要素
我們有兩種方法來解決這困境

1. 深度的設置，也就是剪枝，有兩種的設置方法，前剪枝或後剪枝
2. 森林 (隨機森林) 的設置，利用多個只比亂猜好一點點的樹 (稍稍大於 50% 即可)，讓他們投票，選出大家都同意的結果，我們會在後續提到這方法

1.8.1 前剪枝

其實就是我們剛剛所做的，預先設置最大深度，不讓你的決策樹超過最大深度

但你也發現了，這根本沒有一個公式可以來算出深度的設置怎樣才合理，只能使用經驗法則，或是一層層試過，試出最佳深度



1.8.2 後剪枝 (scikit-learn 沒有提供方法)

後剪枝的概念是無論如何能做多深就多深，最後我們再來一個一個節點把它試試刪掉
如果刪掉一個節點對我們的預測精確度無影響的話，我們就真的把它刪除掉
但 scikit-learn 並沒有提供後剪枝的方法，我們可以期待在以後會有提供後剪枝的 API



RegressionTree

2018 年 6 月 22 日



1 決策樹 (迴歸)

1.1 介紹

在我們的上一節，提到 CART 決策樹不僅可以用於分類，還可以用在迴歸問題，我們就一起來試試看

1.2 資料集

scikit-learn 內建的 Boston 地產資料集

1.3 開始撰寫程式

1.3.1 Step 0. 讀入我們的波士頓地產數據集作為練習

```
In [1]: from sklearn.datasets import load_boston  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個  
# 大家練習的時候可以去掉下面兩行
```



```

pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)

boston = load_boston()
df = pd.DataFrame(boston['data'], columns = boston['feature_names'])
df['target'] = boston['target']
# 如果你想要的話，你可以把他輸出成 csv 觀察看看
# df.to_csv('boston.csv')
df

```

Out[1]:

	CRIM	ZN	INDUS	CHAS	NOX	...	TAX	PTRATIO	B	LSTAT	\
0	0.00632	18.0	2.31	0.0	0.538	...	296.0	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0.0	0.469	...	242.0	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0.0	0.469	...	242.0	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0.0	0.458	...	222.0	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0.0	0.458	...	222.0	18.7	396.90	5.33	
5	0.02985	0.0	2.18	0.0	0.458	...	222.0	18.7	394.12	5.21	
6	0.08829	12.5	7.87	0.0	0.524	...	311.0	15.2	395.60	12.43	
..	
499	0.17783	0.0	9.69	0.0	0.585	...	391.0	19.2	395.77	15.10	
500	0.22438	0.0	9.69	0.0	0.585	...	391.0	19.2	396.90	14.33	
501	0.06263	0.0	11.93	0.0	0.573	...	273.0	21.0	391.99	9.67	
502	0.04527	0.0	11.93	0.0	0.573	...	273.0	21.0	396.90	9.08	
503	0.06076	0.0	11.93	0.0	0.573	...	273.0	21.0	396.90	5.64	
504	0.10959	0.0	11.93	0.0	0.573	...	273.0	21.0	393.45	6.48	
505	0.04741	0.0	11.93	0.0	0.573	...	273.0	21.0	396.90	7.88	

	target
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
5	28.7
6	22.9
..	...
499	17.5



```
500    16.8
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
```

[506 rows x 14 columns]

1.3.2 Step 1. 先畫個圖

相關係數: 兩個東西的相關性

完全正相關 (兩個東西總是一起上升): 1

完全正相關 (一個東西上升的時候, 另一個總是下降): -1

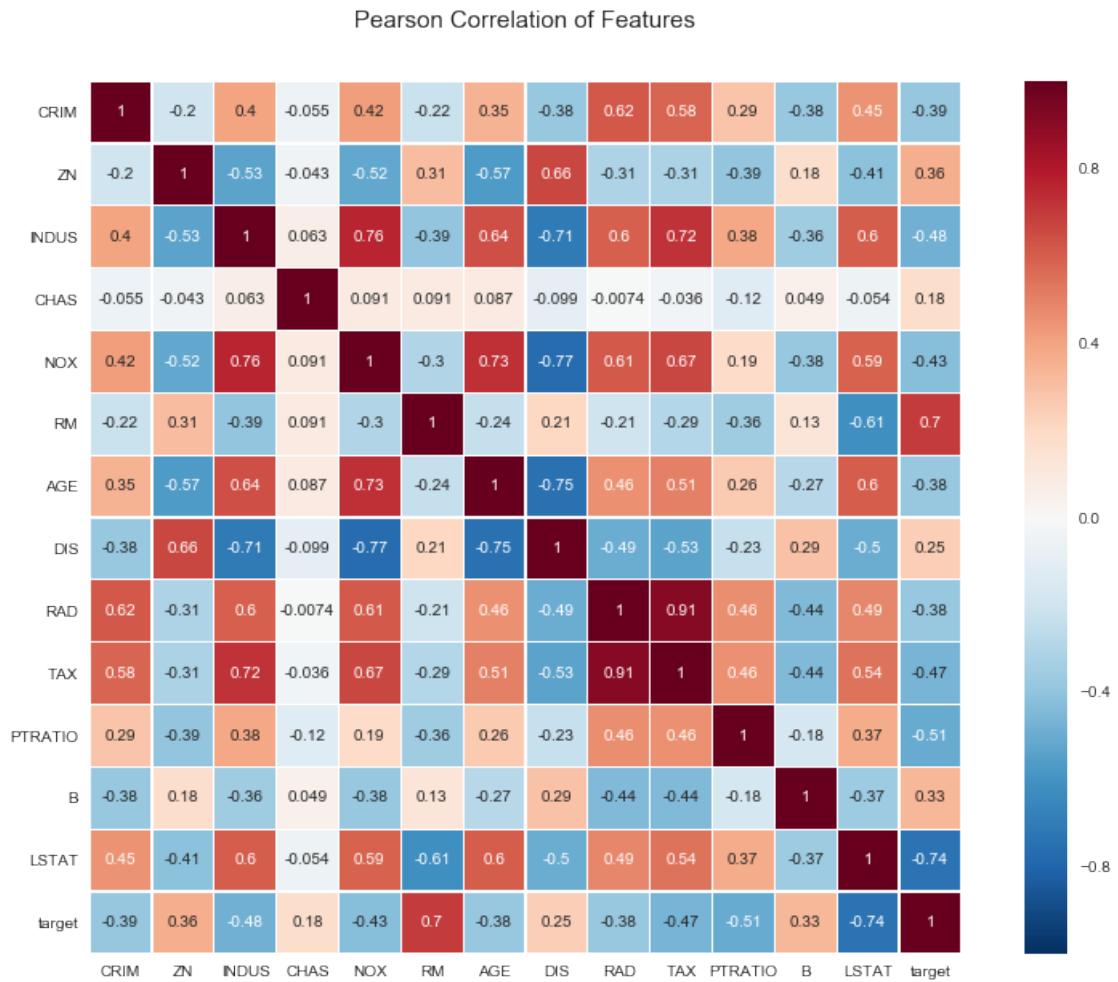
一樣先畫個圖試試看

補充: 你可以使用 https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html
選擇一個你喜歡的調色盤放入 cmap 參數

```
In [2]: plt.figure(figsize=(14,10))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(df.astype(float).corr(), linewidths=0.1,
            square=True, linecolor='white', annot=True)
```

Out [2]: <matplotlib.axes._subplots.AxesSubplot at 0x10de5dd30>





一樣你發現了，像 CHAS, DIS 這些特徵影響的幅度應該都是比較小的 RM 和 LSTAT 的幅度應該是比較大的

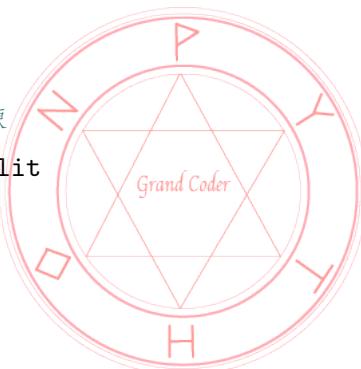
1.3.3 Step 2. 訓練模型

我們使用 DecisionTreeRegressor 來訓練
一樣的步驟

- 創好一個 Regressor
- 使用 fit 將你要訓練的數據餵進來

In [3]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練

```
from sklearn.model_selection import train_test_split
# 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
```



```
data_train, data_test, target_train, target_test = train_test_split(boston['data'],
                                                                boston['target'],
                                                                test_size=0.1)
```

In [4]: `from sklearn.tree import DecisionTreeRegressor
regr = DecisionTreeRegressor(max_depth = 3)
regr.fit(data_train, target_train)`

Out [4]: `DecisionTreeRegressor(criterion='mse', max_depth=3, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')`

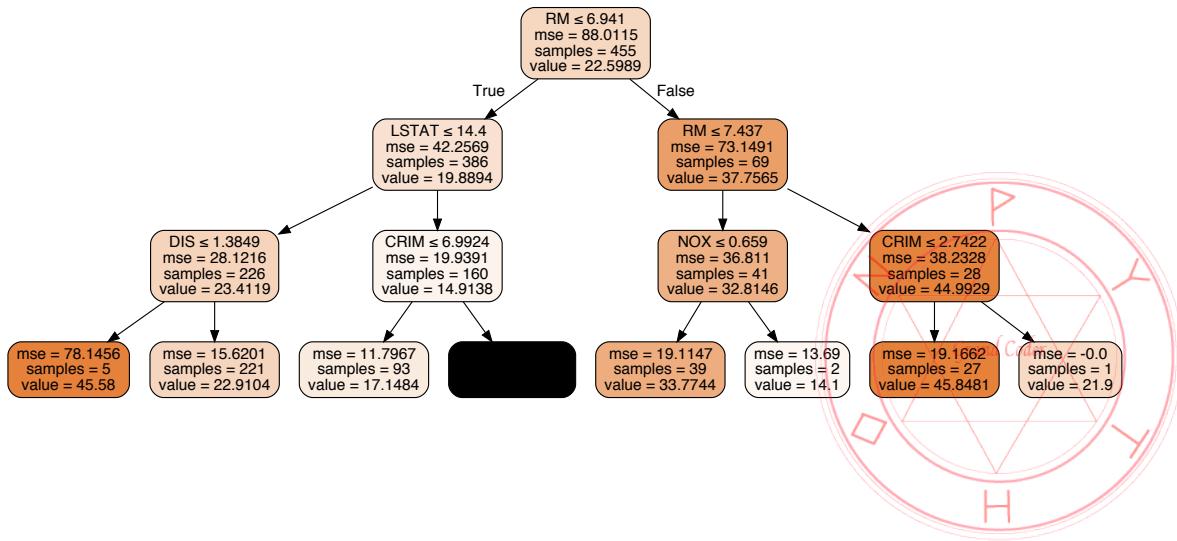
訓練完畫個圖吧，我們使用第三方軟體 graphviz 以及 python 函式庫來操作 graphviz

請完成下面的安裝步驟 1. 請來到 <http://www.graphviz.org/download/> 安裝 graphviz 到你的電腦 2. 請使用 pip 或者是 pycharm 安裝好 graphviz 函式庫

In [5]: `from sklearn.tree import export_graphviz
import graphviz`

```
dot_data = export_graphviz(regr, out_file=None,  
                           feature_names=df.columns,  
                           filled=True, rounded=True,  
                           special_characters=True)  
  
graph = graphviz.Source(dot_data)  
# 你可以把註解解除，輸出一個 pdf  
# graph.render("boston")  
graph
```

Out [5] :



In [6]: # 我們可以直接使用 numpy 來實現兩個 list 的直接相減

```
import numpy as np
predict = regr.predict(data_test)
print("實際的價錢:", target_test)
print("預測的價錢:", predict)
interval = np.subtract(predict, target_test)
print("差異:", interval)
```

實際的價錢: [20.3 19.3 28.2 24.5 29.4 29. 19.9 21. 21. 21.6 43.1 19.2 12.7 27.5
14.3 22.4 10.2 32.2 13.4 19.3 22.6 9.7 27.5 22.3 19.4 20.7 24.8 14.4
18.2 21.9 31.7 50. 22. 15. 18.8 22. 23.6 26.5 20.4 16.7 23.9 15.
26.4 14.8 13.1 16.1 26.4 24.8 21.2 20.6 20.1]

預測的價錢: [22.91040724 22.91040724 22.91040724 22.91040724 22.91040724 33.77435897
22.91040724 22.91040724 22.91040724 22.91040724 45.84814815 22.91040724
17.1483871 22.91040724 17.1483871 17.1483871 11.8119403 33.77435897
11.8119403 22.91040724 22.91040724 11.8119403 22.91040724 22.91040724
17.1483871 22.91040724 22.91040724 17.1483871 22.91040724 22.91040724
33.77435897 45.84814815 22.91040724 17.1483871 17.1483871 22.91040724
22.91040724 22.91040724 22.91040724 11.8119403 33.77435897 14.1
22.91040724 17.1483871 11.8119403 17.1483871 22.91040724 22.91040724
22.91040724 22.91040724 11.8119403]

差異: [2.61040724 3.61040724 -5.28959276 -1.58959276 -6.48959276 4.77435897
3.01040724 1.91040724 1.91040724 1.31040724 2.74814815 3.71040724
4.4483871 -4.58959276 2.8483871 -5.2516129 1.6119403 1.57435897
-1.5880597 3.61040724 0.31040724 2.1119403 -4.58959276 0.61040724
-2.2516129 2.21040724 -1.88959276 2.7483871 4.71040724 1.01040724
2.07435897 -4.15185185 0.91040724 2.1483871 -1.6516129 0.91040724
-0.68959276 -3.58959276 2.51040724 -4.8880597 9.87435897 -0.9
-3.48959276 2.3483871 -1.2880597 1.0483871 -3.48959276 -1.88959276
1.71040724 2.31040724 -8.2880597]

1.3.4 Step 4. 確認一下預測差異

這裡我們不使用混淆矩陣了，而是使用 r2 score，r2 score 的公式如下



(擷取自 wiki <https://zh.wikipedia.org/wiki/%E5%86%B3%E5%AE%9A%E7%B3%BB%E6%95%B0>)

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i.$$

於是**可以得到總平方和**

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

回歸平方和

$$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2,$$

殘差平方和

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

由此，決定係數可定義為

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

1 代表最佳，0 代表普通

我們也不用太數學的方式解釋

我們只看兩個特殊值

1. $r2 = 1$, 代表 $res = 0$, 也就是你所有猜的數字都跟真正的數字一樣!
2. $r2 = 0$, 代表 $res = 1$, 也就是你所有猜的數字都是平均值! 跟我們之前分類的時候 50% 50% 是一模一樣的意思

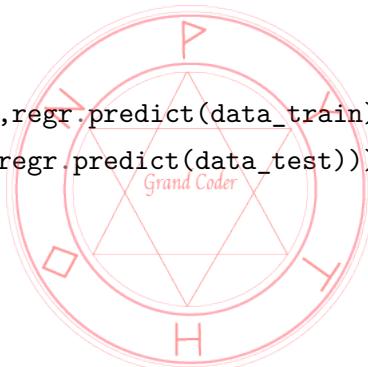
當然跟我們之前分類的一樣，訓練的時候如果你得到 $r2 = 1$ ，未必是最佳的，也就是分類所說的過擬合 (Overfitting)

我們寧願放棄一點點，讓 $r2$ 大於 0.7 其實就是可以接受的值了

在預測的時候， $r2$ 大於 0.5 其實就是可以接受的值了

當然， $r2$ 的選擇依不同領域，不同情況可能會需要更高一點

```
In [7]: from sklearn.metrics import r2_score
print("訓練資料 r2 score:", r2_score(target_train,regr.predict(data_train)))
print("測試資料 r2 score:", r2_score(target_test,regr.predict(data_test)))
```



訓練資料 r2 score: 0.8198453341022476

測試資料 r2 score: 0.7707977640804566

觀察一下哪個特徵最重要，看起來跟我們一開始畫的圖大致符合，RM 的重要性最高 (0.6)，再接下來是 LSTAT(0.2)

In [13]: `regr.feature_importances_`

Out[13]: `array([0.0506248 , 0.64434195, 0.0, 0.07653469, 0.0, 0.02242999, 0.20606857])`



kNN

2018 年 7 月 5 日

Machine Learning

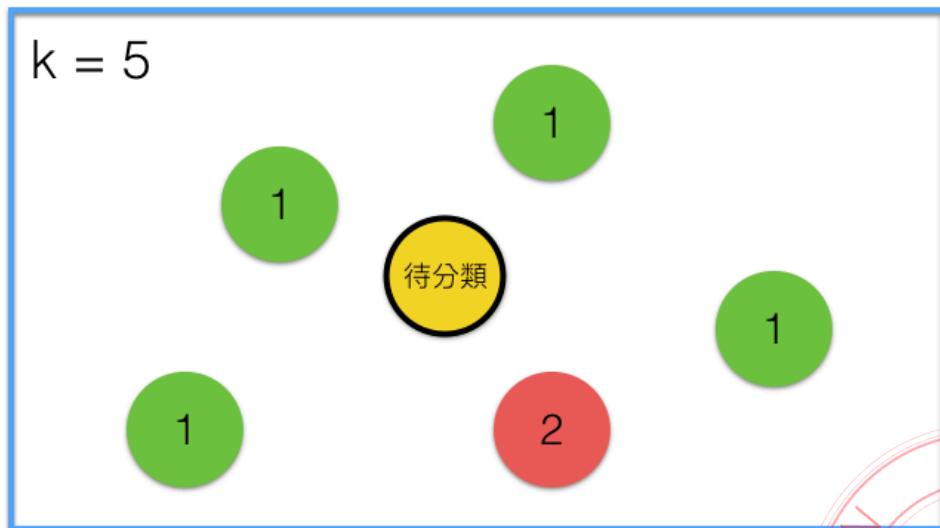


1 kNN 演算法

1.1 介紹

kNN(k-nearest neighbors) 演算法是一個非常簡單的演算法，它採用一個很簡單的概念，『近朱者赤，近墨者黑。』

你的鄰居越多某種分類，就把以當成某種分類



我們這裡選擇 $k = 5$ ，也就是找五個最近的鄰居，那我們看到四個是分類 1，~~1~~ 個是分類 2，那毫無疑問的，未分類的就猜測為分類 1

1.2 資料集

scikit-learn 內建的鳶尾花資料集，由於 kNN 比較直覺，我們就先練習一下 kNN 的概念就好利用之前已經用過的鳶尾花數據集

1.3 開始撰寫程式

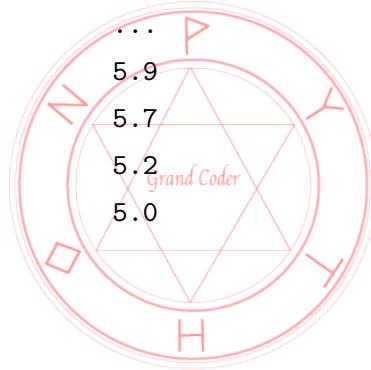
1.3.1 Step 0. 讀入我們的鳶尾花數據集作為練習

```
In [1]: from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)

# 使用 scikit-learn 提供的鳶尾花資料庫
iris = load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df["target"] = iris["target"]
df
```

```
Out[1]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0              5.1             3.5            1.4            0.2
1              4.9             3.0            1.4            0.2
2              4.7             3.2            1.3            0.2
3              4.6             3.1            1.5            0.2
4              5.0             3.6            1.4            0.2
5              5.4             3.9            1.7            0.4
6              4.6             3.4            1.4            0.3
...
143             6.8             3.2            5.9            2.3
144             6.7             3.3            5.7            2.5
145             6.7             3.0            5.2            2.3
146             6.3             2.5            5.0            1.9
```



147		6.5		3.0	5.2	2.0
148		6.2		3.4	5.4	2.3
149		5.9		3.0	5.1	1.8

	target
0	0
1	0
2	0
3	0
4	0
5	0
6	0
..	...
143	2
144	2
145	2
146	2
147	2
148	2
149	2

[150 rows x 5 columns]

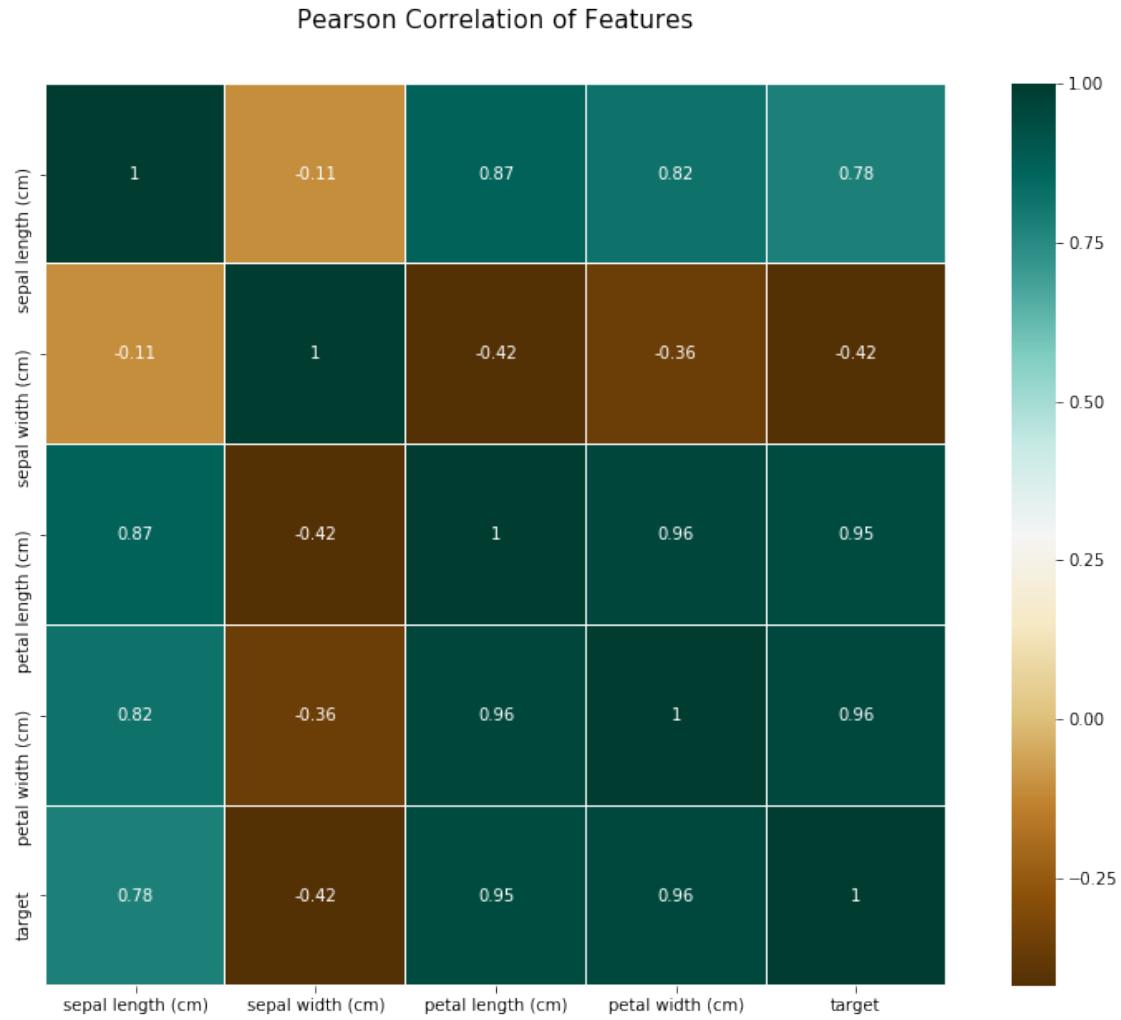
1.3.2 Step 1. 先畫個圖

一樣畫個 heatmap 來觀察一下

```
In [2]: plt.figure(figsize=(14,10))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(df.astype(float).corr(), cmap = "BrBG",
            linewidths=0.1, square=True, linecolor='white',
            annot=True)
```

Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x1049f16d8>





In [3]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練

```
from sklearn.model_selection import train_test_split
# 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
data_train, data_test, target_train, target_test = train_test_split(iris['data'],
                                                               iris['target'],
                                                               test_size=0.1)
```

1.3.3 Step 2. 分類器

通常我們在選 k 值的時候，是用經驗法則在選擇

但一個通常的規則，不要選太少（小於 3），因為你選不夠多人，就無法達到多數決的效果
也不要選太多（大於 20），因為這樣選出來的範圍太大，沒有找鄰居的效果！



```
In [4]: from sklearn.neighbors import KNeighborsClassifier
# 我喜歡先從 8 個鄰居開始試試看
clf = KNeighborsClassifier(n_neighbors=8)
clf = clf.fit(data_train, target_train)
```

1.3.4 Step 3. 開始預測

使用你剛剛的 classifier 進行 predict, predict 完成以後使用 sklearn 內建的 accuracy_score 來算出正確機率

```
In [5]: from sklearn.metrics import accuracy_score

predict = clf.predict(data_test)
print("預測:", predict)
print("正確標籤:", target_test)
print("正確率: ", accuracy_score(target_test, predict) * 100, "%")
```

預測: [2 0 2 1 2 1 1 2 0 0 0 2 0 0 1]

正確標籤: [2 0 2 1 2 1 1 2 0 0 0 2 0 0 1]

正確率: 100.0 %

1.3.5 Step 4. 確認一下分類錯誤

一樣確認一下混淆矩陣

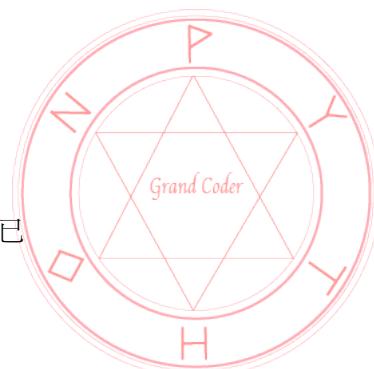
```
In [6]: from sklearn.metrics import confusion_matrix
cm = confusion_matrix(target_test, predict)
pd.DataFrame(cm)
```

```
Out[6]:   0   1   2
      0   6   0   0
      1   0   4   0
      2   0   0   5
```

1.4 優缺點

1.4.1 優點

1. 非常直覺
2. 計算量不管你的資料多大，始終都是看你找幾個鄰居這麼大而已



1.4.2 缺點

1. 解釋性非常差，大概就跟你說 “xxx 都這樣，所以我也要這樣” 一樣差！
2. 如果你的標籤數量本來就不平均，譬如 A 類 100 個，B 類 5 個，你很難正確的分到 B 類
3. 並沒有考慮整體資料，只考慮的附近的資料



kMeans

2018 年 7 月 5 日



1 kMeans(分群)

1.1 介紹

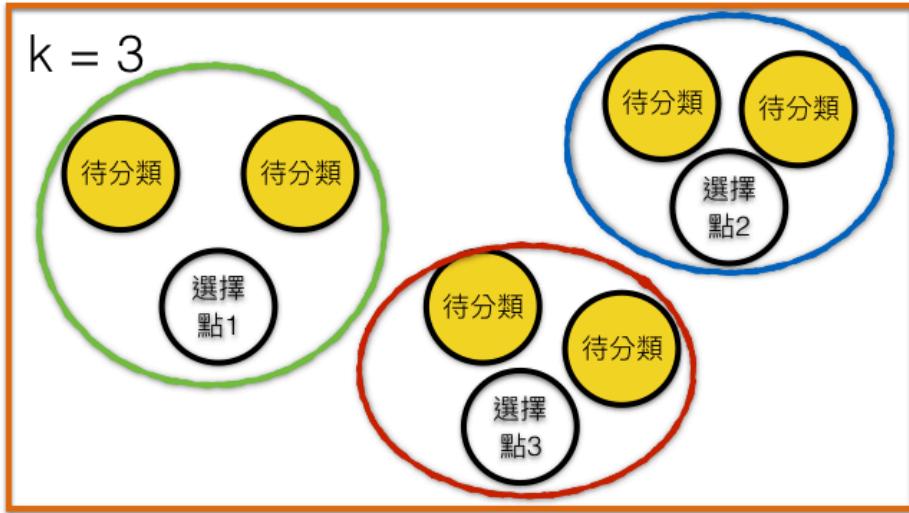
kMeans 跟之前介紹的決策樹和 kNN 有一個決定性的不同，就是他是一個非監督式演算法

有時候我們的數據太大量了，又沒人可以幫忙填寫正確的標籤，我們就希望電腦可以自動的幫我把標籤算出來。

你也可以想像成：非監督式學習就是讓電腦去判斷標籤的過程

注意：雖然他跟 kNN 都有個 k，但實際的內容卻是完全不一樣的





1.2 理論基礎

1.2.1 KMeans

k 意味著在所有的資料裡選出 k 個質量中心，也就是以這 k 個質量中心分成 k 類，詳細步驟如下

1. 隨機選擇 k 個點當中心
2. 對於剩餘的點歸類到 k 類
3. 對於分類好的資料再次選擇一次 k 個質量中心 (讓質量中心更接近理想)
4. 重複步驟 2 和 3 直到穩定

1.2.2 KMeans++

你也發現了，我們的第一步驟，有可能選到非常爛的點當初始中心，那你就會花很多的步驟達到最後的穩定。

所以後來的人做了個改進，就是在選初始 k 點的時候盡量選遠一點的！這就是 KMeans++

1.3 k 值的選擇

k 值的選擇總共有兩種方法，我們先介紹第一種方法，後續再用第二種方法比較

第一種方法非常簡單，你想像成你已經知道數據有 n 類了 (ex. 鳶尾花有三類)，只是沒有人幫你標注這些類別

那毫無疑問的就是直接將 k 設定成你知道的 n ！



1.4 開始撰寫程式

1.4.1 Step 0. 讀入我們的鳶尾花數據集作為練習

這裡我們用鳶尾花數據集來做實驗，但在訓練模型的時候我當作完全沒有 target 這件事
target 只用來在最後我的分群完成以後偷偷來看一下分的好不好

```
In [1]: from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個

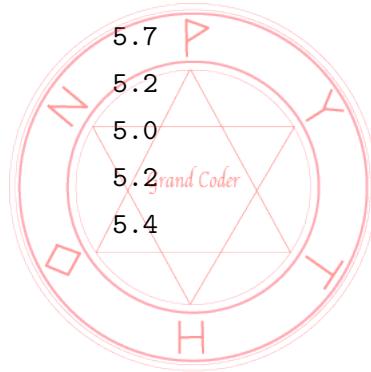
大家練習的時候可以去掉下面兩行

```
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)
```

使用 scikit-learn 提供的鳶尾花資料庫

```
iris = load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df["target"] = iris["target"]
df
```

```
Out[1]:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) \
0           5.1            3.5            1.4            0.2
1           4.9            3.0            1.4            0.2
2           4.7            3.2            1.3            0.2
3           4.6            3.1            1.5            0.2
4           5.0            3.6            1.4            0.2
5           5.4            3.9            1.7            0.4
6           4.6            3.4            1.4            0.3
...
143          6.8            3.2            5.9            2.3
144          6.7            3.3            5.7            2.5
145          6.7            3.0            5.2            2.3
146          6.3            2.5            5.0            1.9
147          6.5            3.0            5.2            2.0
148          6.2            3.4            5.4            2.3
```



	target
0	0
1	0
2	0
3	0
4	0
5	0
6	0
..	...
143	2
144	2
145	2
146	2
147	2
148	2
149	2

[150 rows x 5 columns]

1.4.2 Step 1. (略過) 畫圖

因為我們已經畫過很多次了，這次就先把 heatmap 略過，讀者可以自行練習一下！

In [2]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練

```
from sklearn.model_selection import train_test_split
# 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
data_train, data_test, target_train, target_test = train_test_split(iris['data'],
                                                               iris['target'],
                                                               test_size=0.1)
```

1.4.3 Step 2. 訓練模型

我們使用 kMeans 來訓練

1. 創好一個 Cluster
2. 使用 fit 將你要訓練的數據餵進來



```
In [3]: from sklearn.cluster import KMeans
# 我事先已經有三類了，只是別人沒有幫我標註
# 所以這裡要注意!! 我完全沒有帶入 target 喔
clu = KMeans(n_clusters = 3)
clu.fit(data_train)
```

```
Out[3]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
random_state=None, tol=0.0001, verbose=0)
```

In [4]: # 我們大概可以看到資料已經被分成三類了
`clu.labels_`

```
Out[4]: array([2, 1, 2, 1, 1, 0, 1, 1, 2, 1, 0, 0, 1, 0, 0, 1, 0, 2, 2, 1, 0, 2, 1,
2, 1, 1, 0, 1, 2, 2, 1, 0, 1, 2, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0,
1, 0, 1, 1, 0, 1, 1, 2, 0, 2, 0, 0, 0, 1, 2, 1, 1, 0, 2, 0, 0, 1, 1,
2, 1, 2, 1, 1, 0, 2, 0, 1, 1, 1, 2, 0, 1, 1, 1, 2, 1, 1, 1, 0,
0, 0, 1, 0, 1, 0, 1, 2, 0, 1, 1, 0, 0, 2, 0, 2, 2, 1, 0, 0, 2, 0, 0,
1, 1, 0, 0, 0, 0, 1, 1, 0, 2, 2, 1, 0, 2, 1, 0, 1, 0, 0], dtype=int32)
```

In [5]: from sklearn.metrics import accuracy_score

```
predict = clu.predict(data_test)
print("預測標籤:", predict)
print("正確標籤:", target_test)
```

預測標籤: [0 1 0 2 2 1 1 2 0 1 0 2 1 2 0]
 正確標籤: [0 1 0 2 2 1 1 2 0 1 0 2 1 2 0]

你可以看到我們已經正確的預測了，不過這裡有時候要小心，因為我們沒有事先給標籤
 所以預測的 1 並不一定是正確標籤的 1。你要稍微做個轉換再來對照

1.5 不知道 k 的時候

當我們連 k 都不知道的時候 (ex. 分類人的性格，你不知道要分成幾種分類)

我們只能一個一個開始試，不過我們有一個很好的方法可以幫我們測試選的 k 究竟好不好



1.5.1 Silhouette 方法

Silhouette 是檢查一個點是不是分在最佳群的方法

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Which can be also written as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

From the above definition it is clear that

$$-1 \leq s(i) \leq 1$$

a 是這個點離他所在群內的其他點的平均距離， b 是這個點離他最鄰近的群的點的平均距離
這個值會在-1~1之間

算出來的值越大，代表這個 k 的選擇越好，我們只看上面的 $1 - a/b$ ，這東西要是 1 的話， a 必須為 0，也就是這個群根本就完美的聚集在一個點上

所以簡單來說，我們希望每一個點離他所在的群越近，離另外的群越遠，就是最棒的分類

```
In [9]: from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
%matplotlib inline

scores = []
ks = []

for i in range(2, 8):
    clu = KMeans(n_clusters = i)
    clu.fit(iris['data'])
    clu_score = silhouette_score(iris['data'], clu.labels_)
    scores.append(clu_score)
    ks.append(i)

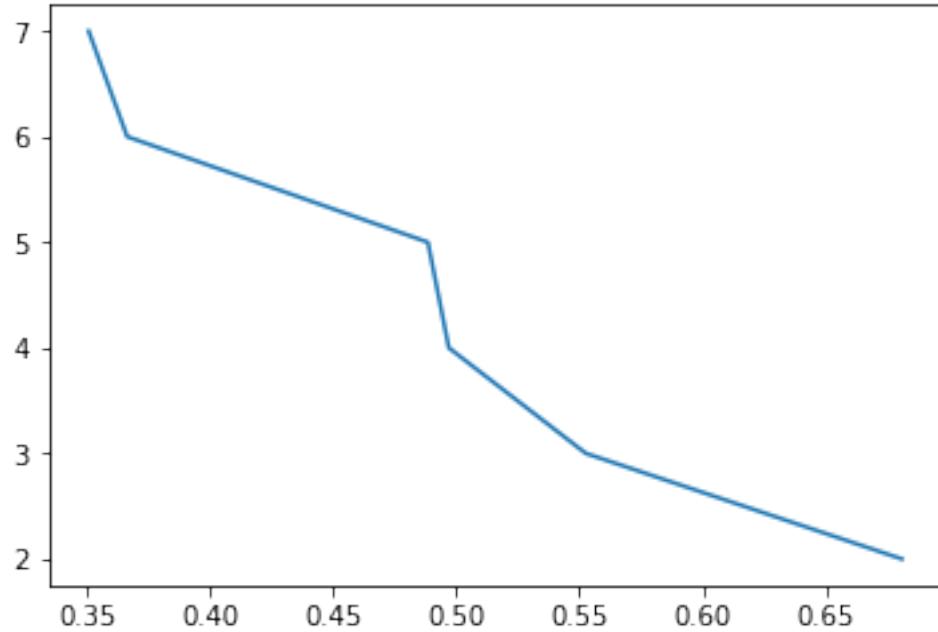
print("分數:", scores)
print("K 值:", ks)
plt.plot(scores, ks)
```

分數: [0.68081362027135084, 0.55259194452136762, 0.49699284994925963, 0.48851755085386322, 0.36



K 值: [2, 3, 4, 5, 6, 7]

Out[9]: [`<matplotlib.lines.Line2D at 0x10bd148d0>`]



你可以看到，大概只有 $k = 2$ 和 $k = 3$ 的時候是一個合理的選擇，符合我們所知道的！

總共有三類的鳶尾花，那 2 為什麼也有很高的 score 呢？我們可以合理的推測其實這三類有兩類是很像的！



Classifier

2018 年 7 月 5 日



1 單純貝氏 (分類)

1.1 介紹

單純貝氏 (Naive Bayes) 是一個非常經典的機率算法，利用機率來推測出某個特徵值下，是某種分類的機率。

這個方法對於文字的分類非常的有效果，而且可以經由少數的數據就達成一個很棒的效果。

1.2 條件機率

在介紹單純貝氏前，我們一定要先介紹條件機率，條件機率其實是一個非常簡單的概念。

條件機率 = 設定一個條件下，某件事情發生的機率

舉個例子，我們就以文章的例子來說好了

$P(\text{這是一篇愛情相關的文章} | \text{文章中出現“伴侶”這詞})$ 這就是一個所謂的條件機率

整個意思是當我已經設定好 (觀察到) 這篇文章中有出現 “伴侶” 這個詞的時候，文章是愛情相關的機率是多少

1.3 貝氏定理

那這個機率跟我們的分類問題有什麼關係呢？

今天我給你一篇文章，你已經觀察到裡面出現了伴侶這個詞，但是你想預測他是什麼分類的。

那該怎麼做呢？很簡單，假設我們有三類文章：宗教，愛情，工作。



你只要計算 $P(\text{宗教} \mid \text{伴侶詞出現})$, $P(\text{愛情} \mid \text{伴侶詞出現})$, $P(\text{工作} \mid \text{伴侶詞出現})$

再比較一下三個機率誰大誰小！大的就是我們的預測分類！

問題是怎麼簡易計算這機率呢？

貝氏定理就可以拿出來了，我們用最簡單的方式描述一次貝氏定理

$$P(B \text{ 發生}) \propto P(A \text{ 發生} \mid B \text{ 發生}) = P(A \text{ 發生}) \propto P(B \text{ 發生} \mid A \text{ 發生}) = P(AB \text{ 同時發生}) = P(A \mid B)$$

套用到我們上面的問題

假設 A: 宗教或愛情或工作 B: 伴侶詞出現

$$P(\text{宗教} \mid \text{伴侶詞出現}) = P(\text{宗教}) \propto P(\text{伴侶詞出現} \mid \text{宗教}) \propto P(\text{伴侶詞出現})$$

$P(\text{宗教})$: 宗教文章數量 / 我們蒐集的文章總數量

$P(\text{伴侶詞出現} \mid \text{宗教})$: 把你蒐集的宗教文章拿出來看看伴侶出現的機率是多少

$P(\text{伴侶詞出現})$: 把所有文章拿出來看看伴侶出現機率，但事實上你不用計算，因為我們只是要比大小！而不需要真正的值。

你會發現，你想得到的答案，可以藉由你事先蒐集的資料比較好統計的部分推算出來!!!

而且只要比較 $P(\text{類別}) \propto P(\text{伴侶詞出現} \mid \text{類別})$ 就可以了!!!!

1.4 單純貝氏

事實上，我們必須把上面的式子寫個更完整一點

$$P(\text{宗教} \mid \text{伴侶詞出現 } n \text{ 次}) = P(\text{宗教}) \propto P(\text{伴侶詞出現 } n \text{ 次} \mid \text{宗教}) \propto P(\text{伴侶詞出現 } n \text{ 次})$$

你會發現比較上面的 $P(\text{宗教}) \propto P(\text{伴侶詞出現 } n \text{ 次} \mid \text{宗教})$ 就可以了！這個值是可以計算出來的(方法類似丟骰子 5 次，出現 3 次 6 機率是多少)

再擴展一下，

$$P(\text{宗教} \mid \text{伴侶詞出現 } n \text{ 次且信仰詞出現 } m \text{ 次}) = P(\text{宗教}) \propto P(\text{伴侶詞出現 } n \text{ 次且信仰詞出現 } m \text{ 次} \mid \text{宗教}) \propto P(\text{伴侶詞出現 } n \text{ 次且信仰詞出現 } m \text{ 次})$$

還是比較 $P(\text{宗教}) \propto P(\text{伴侶詞出現 } n \text{ 次且信仰詞出現 } m \text{ 次} \mid \text{宗教})$ 就可以了，但你發現，這個是很難計算的！

因為你根本不知道這兩件事情(伴侶和信仰這兩個詞)到底有沒有關係！

但我們做了一個很“單純”的假設，就假設這兩個詞的出現完全沒有關係(當然是不符合現實的，不過我們只是要比大小，所以這假設並不會影響結果太多)

假設完以後，我們就可以說：

$$P(\text{伴侶詞出現 } n \text{ 次且信仰詞出現 } m \text{ 次} \mid \text{宗教}) = P(\text{伴侶詞出現 } n \text{ 次} \mid \text{宗教}) \propto P(\text{信仰詞出現 } m \text{ 次} \mid \text{宗教})$$

做完假設以後!! 你發現我們整個東西是可以計算的，就可以開始來比大小，決定我們的分類了！



1.5 文字資料的整理

1.5.1 詞向量

文字最基本的整理方法就是把分詞後的結果，做成一個向量(有方向的量)，你可以把向量想像成往某個方向走出幾步，再往某個方向走幾步的感覺。

例子：我們現在有兩篇文章

文章 1：我喜歡吃牛排，也喜歡吃雞排
文章 2：你喜歡閱讀和旅遊

切割 1：我/喜歡/吃/牛排/，/也/喜歡/吃/雞排
切割 2：你/喜歡/閱讀/和/旅遊

這裡要注意，由於特徵是針對所有的文章計算，所以你要把所有你要訓練的文章先統計好有幾種不同的詞，這個種數就是你要做出的維度

這上面的例子總共有我/喜歡/吃/牛排/也/雞排/你/閱讀/和/旅遊 10 個維度

特徵	我	喜歡	吃	牛排	也	雞排	你	閱讀	和	旅遊
向量 1	1	2	1	1	1	1	0	0	0	0
向量 2	0	1	0	0	0	0	1	1	1	1

1.5.2 TF-IDF 方法

事實上，如果要實作單純貝氏，上面的數量向量已經可以拿來當我們的資料了
不過其實我還是會習慣把他化成 tf-idf 量度

什麼是 tf-idf 呢？

tf: 這個詞出現在整篇文章的次數，出現越多次，代表這個詞越能代表整篇文章

idf: 這個詞在我蒐集的全部文章出現過的文章數，出現越多次，代表這個詞是一個慣用詞，重要性下降。

上面兩個係數加起來就是我們的語言分析的基礎，衡量這篇文章的關鍵詞是什麼的方法

1.5.3 預先處理

中文的預先處理只有一個最高守則！請記得都轉成簡體或者繁體，不然會被算成不同兩個字！

1.6 資料集

整理過後的新聞語料集，簡體已都轉成繁體了！

<https://drive.google.com/open?id=1zbVKIHMUugqXkKDc4q7CNaWXcwQ7pzFE>

請到上面把整個資料下載

共分三個

chinese_news: 原本的新聞

chinese_trans: 翻譯成繁體的新聞



chinese_tests: 我從原本的新聞每個分類擷取出 10 篇當作測試文章

1.7 預測目標

用單純貝氏預測一篇沒分類過的文章是什麼主題

1. 分類問題
2. 監督式演算法

1.8 需要函式庫

jieba 函式庫: 請使用 pip 或利用 PyCharm 安裝, 可以幫我們做完分詞

<https://github.com/fxsjy/jieba>

可以根據下面的說明用繁體字典替換

1.9 資料預處理

可以根據下面的說明用繁體字典替換

1.9.1 讀入我們的訓練資料

這裡跟我們之前做的比較不一樣，是讀入整個資料夾的一個一個檔案內容

讀者在這裡可以在中間多把東西 (dir_path, file_names 印出來) 來熟悉讀取檔案流程

```
In [1]: import os
        import jieba
        import pandas as pd
```

```
# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)
```

```
# scikit-learn 會有些 deprecation warning，為了顯示漂亮，我刻意地忽略掉
import warnings
warnings.filterwarnings('ignore')
```

```
base_dir = "chinese_news_trans"
test_dir = "chinese_news_test"
```



```
# 因為要處理資料夾很多次，所以定義成函式
def process_dirs(base_dir):
    df = pd.DataFrame(columns = ["類別", "內容"])

    # os.walk 會走到檔案才停下來
    for dir_path, dir_names, file_names in os.walk(base_dir):
        for single_file in file_names:
            if not single_file.startswith("."):
                f = open(os.path.join(dir_path, single_file), "r", encoding = "utf-8")
                content = f.read()
                # 讀完黨以後做出第一步處理，先把換行都去掉
                content = content.replace("\r", "").replace("\n", "")
                split_word = jieba.cut(content)
                # 分詞
                content = " ".join(split_word)
                s = pd.Series([dir_path.split("/")[-1], content], index = ["類別", "內容"])
                df = df.append(s, ignore_index = True)
    df['類別'] = df['類別'].astype('category')
    return df
```

In [12]: df = process_dirs(base_dir)

df

	類別	內容
0	交通	【日期】19960104 【版號】1 【...
1	交通	【日期】19960226 【版號】5 【...
2	交通	大秦鐵路萬噸列車試運成功新華社北京...
3	交通	遼寧省檯安縣村村都通柏油路鄉村公...
4	交通	北京—烏蘭巴托—莫斯科3／4次...
5	交通	福建將建第二條出省鐵路新華社福州5...
6	交通	大秦二期工程中最長的平市東河特大...
...
2630	體育	參加首屆世界盃乒乓球團體賽的中國...
2631	體育	全國健美賽和健美操賽決出6項第...
2632	體育	馬玉芹破女子400米跑全國青...
2633	體育	國際奧委會中國臺北委員吳經國訪問北...
2634	體育	亞奧理事會39個成員組織全部以...

2635 體育

世界盃 乒乓球 團體賽 男子 團體 採用 新賽...

2636 體育

登頂 隊員 簡介 新華社 珠穆朗瑪峰 5月...

[2637 rows x 2 columns]

1.9.2 類別處理

由於 scikit-learn 不接受字串，所以我們一定要把類別轉換成整數

你可以使用 cat.categories 得到所有類別

再使用 cat.codes 轉換成整數

In [3]: # 把類別替換成 code, 並且記錄起來

```
# 走過 categories 同時順便把字典創造起來
saved_map = { cat:df['類別'].cat.categories.get_loc(cat)
               for cat in df['類別'].cat.categories }
saved_map
```

Out[3]: {'交通': 0,

```
'政治': 1,
'教育': 2,
'環境': 3,
'經濟': 4,
'藝術': 5,
'計算機': 6,
'軍事': 7,
'醫藥': 8,
'體育': 9}
```

In [4]: df['類別'] = df['類別'].cat.codes

df

Out[4]: 類別

內容

0	0	【日期】	19960104	【版號】	1	【...】
1	0	【日期】	19960226	【版號】	5	【...】
2	0				大秦鐵路萬噸列車試運成功	新華社北京...
3	0				遼寧省檯安縣村村都通柏油路	鄉村公路...
4	0				北京—烏蘭巴托—莫斯科	3/4次...
5	0				福建將建第二條出省鐵路	新華社福州5...
6	0				大秦二期工程中最長的平市東河特大...	

	
2630	9	參加首屆世界盃乒乓球團體賽的中國...	...
2631	9	全國健美賽和健美操賽決出6項第...	
2632	9	馬玉芹破女子400米跑全國青...	
2633	9	國際奧委會中國臺北委員吳經國訪問北...	
2634	9	亞奧理事會39個成員組織全部以...	
2635	9	世界盃乒乓球團體賽男子團體採用新賽...	
2636	9	登頂隊員簡介新華社珠穆朗瑪峰5月...	

[2637 rows x 2 columns]

1.9.3 讀入測試資料

把我們的測試資料讀取，並且使用剛剛存起來的 category 來 map

```
In [5]: test_df = process_dirs(test_dir)
test_df
```

```
Out[5]:
```

	類別	內容
0	交通	日月光華 --- Traffic_Info 精華區文章閱讀 - - - ...
1	交通	日月光華 --- Traffic_Info 精華區文章閱讀 - - - ...
2	交通	日月光華 --- Traffic_Info 精華區文章閱讀 - - - ...
3	交通	三趟火車停開 乘客可全額退票 瀏覽次數：1180 ...
4	交通	日月光華 --- Traffic_Info 精華區文章閱讀 - - - ...
5	交通	日月光華 --- Traffic_Info 精華區文章閱讀 - - - ...
6	交通	日月光華 --- Traffic_Info 精華區文章閱讀 - - - ...
..
94	體育	中國青島—韓國大邱健美表演賽和對抗賽將舉辦 ...
95	體育	男子健美初登亞運會 中國猛男直指前三 在即將...
96	體育	最優秀選手無緣亞運會健美賽 健美在亞洲運動會 ...
97	體育	各國記者眼中的羽毛球世錦賽 - - - - - ...
98	體育	友好運動會第五天 東道主選手大顯神威 2001年09月03日02...
99	體育	不靠技術比運氣 第二屆奧運會在巴黎舉行，同時這裡也正在舉行...
100	體育	帆板運動簡介(二) 我國在79年由國家...

[101 rows x 2 columns]



1.9.4 替換測試類別

這邊必須使用剛剛存起來的字典來替換

因為如果直接使用 code 可能會發生沒對照到的事故

```
In [6]: test_df['類別'] = test_df['類別'].replace(saved_map)
test_df
```

	類別	內容
0	0 日 月 光 華	-- Traffic _ Info 精華區 文章 閱讀 - - - ...
1	0 日 月 光 華	-- Traffic _ Info 精華區 文章 閱讀 - - - ...
2	0 日 月 光 華	-- Traffic _ Info 精華區 文章 閱讀 - - - ...
3	0 三趟火車停開 乘客可全額退票	瀏覽次數： 1180 ...
4	0 日 月 光 華	-- Traffic _ Info 精華區 文章 閱讀 - - - ...
5	0 日 月 光 華	-- Traffic _ Info 精華區 文章 閱讀 - - - ...
6	0 日 月 光 華	-- Traffic _ Info 精華區 文章 閱讀 - - - ...
..
94	9 中國青島 - 韓國大邱健美表演賽和對抗賽將舉辦	...
95	9 男子健美初登亞運會中國猛男直指前三	在即將...
96	9 最優秀選手無緣亞運會健美賽	健美在亞洲運動會...
97	9 各國記者眼中的羽毛球世錦賽	- - - - - ...
98	9 友好運動會第五天東道主選手大顯神威2001年09月03日02...	
99	9 不靠技術比運氣 第二屆奧運會在巴黎舉行，同時這裡也正在舉行...	
100	9 帆板運動簡介(二) 我國在79年由國家...	

[101 rows x 2 columns]

```
In [7]: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
vec = TfidfVectorizer()
# 注意一定要使用 fit_transform, 才會幫你轉換成詞向量
bag = vec.fit_transform(df['內容'])
print("總共維度:", len(vec.get_feature_names()))
# 讀者可以把註解拿掉，就可以看到 transform 後的 matrix
#for entry in bag:
#    print(entry)
```

總共維度: 96042



1.10 scikit-learn 單純貝氏

1.10.1 合適的選擇

scikit-learn 裡的單純貝氏有對你的資料類型做過優化

sklearn.naive_bayes : Naive Bayes

The `sklearn.naive_bayes` module implements Naive Bayes algorithms. These are supervised learning methods based on applying Bayes' theorem with strong (naive) feature independence assumptions.

User guide: See the [Naive Bayes](#) section for further details.

<code>naive_bayes.BernoulliNB</code> ([alpha, binarize, ...])	Naive Bayes classifier for multivariate Bernoulli models.
<code>naive_bayes.GaussianNB</code> ([priors])	Gaussian Naive Bayes (GaussianNB)
<code>naive_bayes.MultinomialNB</code> ([alpha, ...])	Naive Bayes classifier for multinomial models

1. BernoulliNB: 對於特徵是 True 和 False 二分法優化
2. GaussianNB: 對於特徵是高斯分布的連續數字優化
3. MultinomialNB: 對於特徵是整數，而且是數幾次的分布優化，不過使用就算使用 tf-idf 分數一樣可以達到很好效果

1.10.2 Alpha 的選擇

Alpha 的用意是 Laplace smoothing 的意思

我們現在矩陣裡面很多元素都是等於 0

但等於 0 會造成一個現象，就是你發生的機率會是 0

這時候由於我們的資料並不是完整的資料，所以你如果說 $P(\text{某個字出現} \mid \text{某種分類}) = 0$

反而會造成過度武斷，而導致預測的行為變差

所以 smoothing 最主要的用意是讓 0 機率不要是 0，而是一個接近 0 的微小機率 (有機率跟沒機率差很多！)

$$P(X_j | Y_k) = \frac{\text{Count}(X_j, Y_k) + 1}{\sum_j^V (\text{Count}(X_j, Y_k) + 1)}.$$

我們如果不設定 alpha 的話，預設值就是如同公式的 1

如果你使用的是 CountVectorizer，那沒有問題，就照著預設值設置

但如果你使用的是 TfidfVectorizer，tf-idf 分數算出來都大概在 0.x 而已

你加 1 就會變得不太實在，所以如果使用 tf-idf 的時候，我們通常會設置 $\text{alpha} = 0.001$

Smoothing 的白話文理解：假設你把出現“收益”這詞的信 100% 當成不是垃圾信，那會有一個很嚴重的後果，只要在垃圾信裡加上收益這詞，就永遠不會被你檢測出來，因為 0 乘上去以後，整體機率就是 0，你永遠不會去考慮任何其他特徵，所以你不應該讓機率是 0%，而是用一個極小接近 0 的機率來算，這就是 Smoothing



In [8]: # 只要是文字，我們通常就會選擇 *MultinomialNB*

```
from sklearn.naive_bayes import MultinomialNB
```

```
clf = MultinomialNB(alpha = 0.001).fit(bag, df['類別'])
```

In [9]: # 由於我們用剛剛的 *vec* 訓練他，所以維度會保持跟剛剛一樣

```
test_bag = vec.transform(test_df['內容'])
print("維度:", len(vec.get_feature_names()))
```

維度: 96042

In [10]: from sklearn.metrics import accuracy_score

```
predict = clf.predict(test_bag)
```

```
print("預測:", list(predict))
```

```
print("正確標籤:", list(test_df['類別']))
```

```
print("Naive-Bayes 正確率: ", accuracy_score(test_df['類別'], predict) * 100, "%")
```

預測: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,

正確標籤: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,

Naive-Bayes 正確率: 100.0 %

In [11]: from sklearn import neighbors

```
clf = neighbors.KNeighborsClassifier(n_neighbors=8)
```

```
clf = clf.fit(bag, df['類別'])
```

```
predict = clf.predict(test_bag)
```

```
print("預測:", list(predict))
```

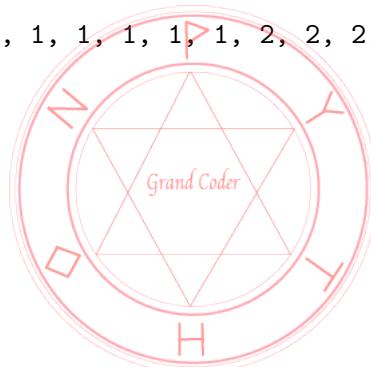
```
print("正確標籤:", list(test_df['類別']))
```

```
print("kNN 正確率: ", accuracy_score(test_df['類別'], predict) * 100, "%")
```

預測: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,

正確標籤: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,

kNN 正確率: 100.0 %



1.10.3 結論

由於訓練資料非常完善，而且目標相對單純，所以你發現不管單純貝氏或者 kNN 都可以順利達到高正確率



PoemClassifier

2018 年 7 月 5 日



1 單純貝氏 (詩詞分類)

1.1 介紹

我們上一章節談到了單純貝氏，以及歸類文章，現在我們再試試看

但是這次我們拿的資料是比較抽象的詩詞資料，我們看看我的單純貝氏是否能研究出三位詩人的常用語

對無名詩詞做出分類

1.2 資料集

自行收集的詩詞資料集

https://drive.google.com/open?id=1KuH3QyTaD7yrqGv8uTPPVgDQD_XW8amC

poem_train.csv: 供你訓練模型

poem_test.csv: 供你測試模型

1.3 目標

標籤總共有三種:



1.3.1 李白



風格 (節錄自 wiki):

李詩富個性，有強烈的主觀抒情色彩，內容表現出蔑視庸俗，反抗和不媚權貴的叛逆精神，歌頌遊俠和仙道，被譽為「詩俠」、「詩仙」，後世亦以詩仙李白稱之。



1.3.2 杜甫



風格 (節錄自 wiki):

杜詩主要風格是沉鬱頓挫，氣魄闊大雄偉，詩歌意象鮮明強烈。風格多樣，豐富多姿，或雄渾奔放，或清新細膩，或沉鬱悲涼、或辭藻富麗、或平易質樸、或通俗自然。杜詩融洽吸收前人藝術技巧，發展成一種獨特的新風格。



1.3.3 白居易



風格 (節錄自 wiki):

開頭破題，在結尾時凸顯全詩要旨；用辭淺顯，使人容易明瞭；用語直接而銳利，使人警惕；敘事可靠可信；體例流暢而可以傳唱於歌曲之中。

1.4 資料預處理

1.4.1 資料讀取

csv 的讀取就直接使用 pandas read_csv 即可

```
In [15]: import pandas as pd
```

```
# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
```

```
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)
```

```
# scikit-learn 會有些 deprecation warning，為了顯示漂亮，我刻意地忽略掉
import warnings
warnings.filterwarnings('ignore')
```



```
df = pd.read_csv("poem_train.csv", encoding = "utf-8")
df
```

Out[15]:

	作者	詩名 \
0	李白	菩薩蠻 · 平林漠漠煙如織
1	李白	把酒問月
2	李白	春思
3	李白	春夜洛城聞笛
4	李白	古風 其十九
5	李白	關山月
6	李白	將進酒 · 君不見黃河之水天上来
...
2724	白居易	秋蝶
2725	白居易	三年為刺史二首
2726	白居易	鬱元九後詠所懷
2727	白居易	早秋曲江感懷
2728	白居易	東墟晚歇 時退居渭村。
2729	白居易	南秦雪
2730	白居易	寄蘄州簟與元九因題六韻 時元九鰥居。

內容

0 平林漠漠煙如織，寒山一帶傷心碧。\\r\\n 曇色入高樓，有人樓上愁。玉階空佇立，宿鳥歸飛急。
1 青天有月來幾時，我今停杯一問之：人攀明月不可得，月行卻與人相隨？皎如飛鏡臨丹闕，綠煙
2 燕草如碧絲，秦桑低綠枝。當君懷歸日，是妾斷腸時。春風不相識，何事入羅帷。
3 誰家玉笛暗飛聲，散入春風滿洛城。此夜曲中聞折柳，何人不起故園情。
4 西上蓮花山，迢迢見明星。（西上 一作：西嶽）素手把芙蓉，虛步躡太清。霓裳曳廣帶，飄拂升
5 明月出天山，蒼茫雲海間。長風幾萬裡，吹度玉門關。漢下白登道，胡窺青海灣。由來征戰地，
6 君不見黃河之水天上来，奔流到海不複回。君不見高堂明鏡悲白發，朝如青絲暮成雪。人生得意
...
2724 秋花紫蒙蒙，秋蝶黃茸茸。花低蝶新小，飛戲叢西東。日暮涼風來，紛紛花落叢。夜深白露冷，
2725 三年為刺史，無政在人口。唯向郡城中，題詩十餘首。慚非甘棠詠，豈有思人不？三年為刺史，
2726 零落桐葉雨，蕭條槿花風。悠悠早秋意，生此幽閒中。況與故人鬱，中懷正無悰。勿雲不相送，
2727 離離暑雲散，嫋嫋涼風起。池上秋又來，荷花半成子。朱顏易銷歇，白日無窮已。人壽不如山，
2728 涼風冷露蕭索天，黃蒿紫菊荒涼田。繞塚秋花少顏色，細蟲小蝶飛翻翻。中有騰騰獨行者，手拄
2729 往歲曾為西邑吏，慣從駱口到南秦。\\r\\n 三時雲冷多飛雪，二月山寒少有春。\\r\\n 我思舊事猶
2730 笛竹出蘄春，霜刀劈翠筠。織成雙鎖簟，寄與獨眠人。卷作筒中信，舒為席上珍。滑如鋪薤葉，

Grand Coder



[2731 rows x 3 columns]

1.4.2 類別處理

由於 scikit-learn 不接受字串，所以我們一定要把類別轉換成整數
 你可以使用 cat.categories 得到所有類別
 再使用 cat.codes 轉換成整數

```
In [16]: df['作者'] = df['作者'].astype('category')
          saved_map = { cat:df['作者'].cat.categories.get_loc(cat) for cat in df['作者'].cat.categories }
          saved_map
```

```
Out[16]: {'李白': 0, '杜甫': 1, '白居易': 2}
```

1.4.3 分詞處理

我們直接把分詞和簡易的內容處理定義成一個函式

```
In [17]: # 定義好等等我們要對所有內容做的 split
          def split_poem(poem):
              return " ".join(jieba.cut(poem))

          def process_poems(df):
              df['內容'] = df['內容'].apply(split_poem)
              df['內容'] = df['內容'].str.replace('\r', '')
              df['內容'] = df['內容'].str.replace('\n', '')
              # 詩名我們今天沒用到，先 drop 掉
              df = df.drop(["詩名"], axis = 1)
              df['作者'] = df['作者'].astype('category')
              return df
```

1.4.4 建立詞向量

我們可以選用 CountVectorizer 或者 TfidfVectorizer 建立你的詞向量
 那建議使用 sklearn 內建的轉換來做，因為這樣她會自動地幫你把詞向量特徵建起來起來，不需要自己建立

```
In [18]: import jieba

df = process_poems(df)
df
```



Out [18]:	作者	內容
0	李白	平林 漠漠 煙如織，寒山 一帶 傷心碧。暝 色入 高樓，有人 樓上 愁。 ...
1	李白	青天有月來 幾時，我今停杯一問之：人攀明月不可得，月行卻與...
2	李白	燕草如碧絲，秦桑低綠枝。當君懷歸日，是妾斷腸時。春風不相識，...
3	李白	誰家玉笛暗飛聲，散入春風滿洛城。此夜曲中聞折柳，何人不起故園情。
4	李白	西上蓮花山，迢迢見明星。（西上一作：西嶽）素手把芙蓉...
5	李白	明月出天山，蒼茫雲海間。長風幾萬裡，吹度玉門關。漢下白登道，...
6	李白	君不見黃河之水天上來，奔流到海不複回。君不見高堂明鏡悲白發，...
...
2724	白居易	秋花紫蒙蒙，秋蝶黃茸茸。花低蝶新小，飛戲叢西東。日暮涼風來...
2725	白居易	三年為刺史，無政在人口。唯向郡城中，題詩十餘首。慚非甘棠詠...
2726	白居易	零落桐葉雨，蕭條槿花風。悠悠早秋意，生此幽閒中。況與故人...
2727	白居易	離離暑雲散，嫋嫋涼風起。池上秋又來，荷花半成子。朱顏易銷歇...
2728	白居易	涼風冷露蕭索天，黃蒿紫菊荒涼田。繞塚秋花少顏色，細蟲小蝶飛翻...
2729	白居易	往歲曾為西邑吏，慣從駱口到南秦。三時雲冷多飛雪，二月山寒...
2730	白居易	笛竹出蘄春，霜刀劈翠筠。織成雙鎖簟，寄與獨眠人。卷作筒中...

[2731 rows x 2 columns]

In [5]: df['作者'] = df['作者'].cat.codes
df

Out [5]:	作者	內容
0	0	平林 漠漠 煙如織，寒山 一帶 傷心碧。暝 色入 高樓，有人 樓上 愁。 ...
1	0	青天有月來 幾時，我今停杯一問之：人攀明月不可得，月行卻與...
2	0	燕草如碧絲，秦桑低綠枝。當君懷歸日，是妾斷腸時。春風不相識，...
3	0	誰家玉笛暗飛聲，散入春風滿洛城。此夜曲中聞折柳，何人不起故園情。
4	0	西上蓮花山，迢迢見明星。（西上一作：西嶽）素手把芙蓉...
5	0	明月出天山，蒼茫雲海間。長風幾萬裡，吹度玉門關。漢下白登道，...
6	0	君不見黃河之水天上來，奔流到海不複回。君不見高堂明鏡悲白發，...
...
2724	2	秋花紫蒙蒙，秋蝶黃茸茸。花低蝶新小，飛戲叢西東。日暮涼風來...
2725	2	三年為刺史，無政在人口。唯向郡城中，題詩十餘首。慚非甘棠詠...
2726	2	零落桐葉雨，蕭條槿花風。悠悠早秋意，生此幽閒中。況與故人...
2727	2	離離暑雲散，嫋嫋涼風起。池上秋又來，荷花半成子。朱顏易銷歇...
2728	2	涼風冷露蕭索天，黃蒿紫菊荒涼田。繞塚秋花少顏色，細蟲小蝶飛翻...
2729	2	往歲曾為西邑吏，慣從駱口到南秦。三時雲冷多飛雪，二月山寒...
2730	2	笛竹出蘄春，霜刀劈翠筠。織成雙鎖簟，寄與獨眠人。卷作筒中...

[2731 rows x 2 columns]

```
In [6]: test_df = pd.read_csv("poem_test.csv")
test_df = process_poems(test_df)
test_df
```

	作者	內容
0	李白	日照香爐生紫煙，遙看瀑布掛前川。飛流直下三千尺，疑是銀河落九天。
1	李白	朝辭白帝彩雲間，千裡江陵一日還。兩岸猿聲啼不住，輕舟已過萬...
2	李白	李白乘舟將欲行，忽聞岸上踏歌聲。桃花潭水深千尺，不及汪倫送我情。
3	李白	故人西辭黃鶴樓，烟花三月下揚州。孤帆遠影碧空盡，唯見長江天際流。
4	李白	危樓高百尺，手可摘星辰。不敢高聲語，恐驚天上人。
5	李白	床前明月光，疑是地上霜。舉頭望明月，低頭思故鄉。
6	李白	天門中斷楚江開，碧水東流至此回。兩岸青山相對出，孤帆一片日...
..
23	白居易	雨砌長寒蕪，風庭落秋果。窗間有閒叟，儘日看書坐。書中見...
24	白居易	睡足肢體暢，晨起開中堂。初旭泛簾幕，微風拂衣裳。二婢扶盥櫛...
25	白居易	履道西門有弊居，池塘竹樹繞君廬。豪華肥壯雖無分，飽暖安閒即有餘...
26	白居易	昨日複今辰，悠悠七十春。所經多故處，卻想似前身。散秩優遊...
27	白居易	不與老為期，因何兩鬢絲？才應免夭促，便已及衰羸。昨夜夢...
28	白居易	暖床斜臥日曛腰，一覺閒眠百病銷。儘日一飧茶兩碗，更無所...
29	白居易	選石鋪新路，安橋壓古堤。似從銀漢下，落傍玉川西。影定欄杆倒...

[30 rows x 2 columns]

1.4.5 替換測試類別

這邊必須使用剛剛存起來的字典來替換

因為如果直接使用 code 可能會發生沒對照到的事故

```
In [7]: test_df['作者'] = test_df['作者'].replace(saved_map)
test_df
```

	作者	內容
0	0	日照香爐生紫煙，遙看瀑布掛前川。飛流直下三千尺，疑是銀河落九天。
1	0	朝辭白帝彩雲間，千裡江陵一日還。兩岸猿聲啼不住，輕舟已過萬...
2	0	李白乘舟將欲行，忽聞岸上踏歌聲。桃花潭水深千尺， 不及 汪倫送我情。
3	0	故人西辭黃鶴樓，烟花三月下揚州。孤帆遠影碧空盡，唯見長江天際流。

4	0	危樓高百尺，手可摘星辰。不敢高聲語，恐驚天上人。
5	0	床前明月光，疑是地上霜。舉頭望明月，低頭思故鄉。
6	0	天門中斷楚江開，碧水東流至此回。兩岸青山相對出，孤帆一片日...
..
23	2	雨砌長寒蕪，風庭落秋果。窗間有閒叟，儘日看書坐。書中見...
24	2	睡足肢體暢，晨起開中堂。初旭泛簾幕，微風拂衣裳。二婢扶盥櫛...
25	2	履道西門有弊居，池塘竹樹繞君廬。豪華肥壯雖無分，飽暖安閒即有餘...
26	2	昨日複今辰，悠悠七十春。所經多故處，卻想似前身。散秩優遊...
27	2	不與老為期，因何兩鬢絲？才應免夭促，便已及衰羸。昨夜夢...
28	2	暖床斜臥日曛腰，一覺閒眠百病銷。儘日一飧茶兩碗，更無所...
29	2	選石鋪新路，安橋壓古堤。似從銀漢下，落傍玉川西。影定欄杆倒...

[30 rows x 2 columns]

1.5 開始訓練

1.5.1 建立詞向量並預測

我們可以選用 CountVectorizer 或者 TfidfVectorizer 建立你的詞向量
那建議使用 sklearn 內建的轉換來做，因為這樣她會自動地幫你把詞向量特徵建起來起來，不需要自己建立

```
In [8]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.naive_bayes import MultinomialNB
        vec = TfidfVectorizer()
        bag = vec.fit_transform(df['內容'])
        print("維度:", len(vec.get_feature_names()))
        clf = MultinomialNB(alpha = 0.001)
        clf.fit(bag, df['作者'])
```

維度: 52294

Out [8]: MultinomialNB(alpha=0.001, class_prior=None, fit_prior=True)

```
In [9]: from sklearn.metrics import accuracy_score

        test_bag = vec.transform(test_df['內容'])
        predict = clf.predict(test_bag)
```



```

print("預測:", list(predict))
print("正確標籤:", list(test_df['作者']))
print("Naive-Bayes 正確率: ", accuracy_score(test_df['作者'], predict) * 100, "%")

```

預測: [0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 1]
 正確標籤: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
 Naive-Bayes 正確率: 80.0 %

In [10]: from sklearn.metrics import confusion_matrix
 cm = confusion_matrix(test_df['作者'], predict)
 pd.DataFrame(cm)

Out[10]:

	0	1	2
0	8	1	1
1	0	8	2
2	1	1	8

1.5.2 跟 kNN 比較一下

我們試試看 kNN 在這裡表現的如何

In [11]: from sklearn import neighbors

 clf = neighbors.KNeighborsClassifier(n_neighbors=8)
 clf = clf.fit(bag, df['作者'])
 predict = clf.predict(test_bag)

 print("預測:", list(predict))
 print("正確標籤:", list(test_df['作者']))
 print("kNN 正確率: ", accuracy_score(test_df['作者'], predict) * 100, "%")

預測: [0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 2, 0, 1, 1, 2, 2, 0, 0, 2, 0, 2]
 正確標籤: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
 kNN 正確率: 53.333333333333336 %

1.6 結論

你最後發現，對於文字這種資料，還是單純貝氏來的比較合適



那為什麼我們的訓練結果跟上一次我們使用新聞的結果差距比較大呢？

因為我們的詩詞是一首比較短的向量，所以訓練資料相對的更少了，而在訓練資料少的時候我們更可以看到機率算法的好處（對於資料的多少稍微不敏感一點）

1.6.1 優點

1. 啟動資料少的時候，單純貝氏還是可以達到一個不錯的結果
2. 對於文字這種稀疏的特徵，是一個非常好的分類器

1.6.2 缺點

1. 解釋性稍微低了一點
2. 用在普通非稀疏的模型，效果甚至可能不如 kNN 或者決策樹類型的演算法來得好



randomforest

2018 年 7 月 5 日



1 隨機森林 + Kaggle 初參賽

1.1 目標

我們使用 Kaggle 的鐵達尼號資料集來教你一個重量級的演算法，隨機森林
隨機森林是一個非常方便的演算法，可以應用在很多現實生活的問題並且得到還不錯的結果
我們順便在資料科學的大本營 (Kaggle)，參加我們第一個練習賽
當然，雖然只是學習一個簡單的演算法並且參加一個練習賽，我們還是希望能在這個比賽上面
拿到一個不錯的名次

1.2 隨機森林

徹底地貫徹『三個臭皮匠，勝過一個諸葛亮』的概念，
我們蒐集很多不用到最佳的樹，讓他們用多數決來決定答案是哪個分類
但這裡有一點要特別強調
上面那句話是對的，但要加上一點特別的聲明，要是三個擁有不同經歷的臭皮匠
如果三個臭皮匠根本長得一模一樣，無論如何都不會勝過一個諸葛亮的
所以如果我們每個決策樹都長得一模一樣，我們的森林永遠建造不起來
那如何讓他不一樣呢？
很簡單，就每個決策樹在創建的時候不使用全部的訓練資料，只使用其中的一部份就好
這招我們叫 bagging，在不使用全部資料的情況下，每個建造出來的樹自然就會不一樣



還有另外一種建造組合的方式叫作 **boosting**, 我們剛剛的 **bagging** 是並行的建造很多樹, 但 **boosting** 是把每一次的結果錯的地方修正 **boosting** 和 **bagging** 可以參考
<https://www.jianshu.com/p/708dff71df3a>

隨機森林雖然簡單, 但在許多的分類問題都可以得到非常非常好的效果

1.3 資料集位置

<https://www.kaggle.com/c/titanic/data>

1. 需要登入才能下載
2. train.csv: 訓練的資料集
3. test.csv: 測試的資料集

1.4 資料處理

1.4.1 Step 0. 讀入鐵達尼訓練集

```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import train_test_split
        %matplotlib inline

%matplotlib inline

# 為了顯示的漂亮, 我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)

In [2]: df = pd.read_csv("train.csv")
        df
```

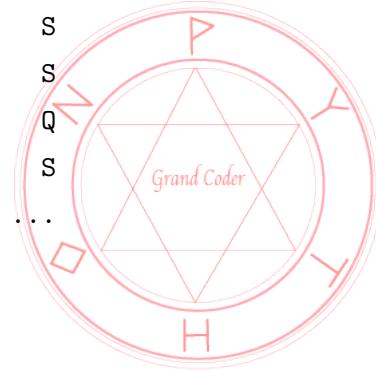
```
Out[2]:    PassengerId  Survived  Pclass \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
```



5	6	0	3
6	7	0	1
..
884	885	0	3
885	886	0	3
886	887	0	2
887	888	1	1
888	889	0	3
889	890	1	1
890	891	0	3

		Name	Sex	Age	SibSp	\
0		Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th... 2	Heikkinen, Miss. Laina	female	38.0	1	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0		
4	Allen, Mr. William Henry	male	35.0	1		
5	Moran, Mr. James	male	NaN	0		
6	McCarthy, Mr. Timothy J	male	54.0	0		
..	
884		Suthehall, Mr. Henry Jr	male	25.0	0	
885		Rice, Mrs. William (Margaret Norton)	female	39.0	0	
886		Montvila, Rev. Juozas	male	27.0	0	
887		Graham, Miss. Margaret Edith	female	19.0	0	
888	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1		
889		Behr, Mr. Karl Howell	male	26.0	0	
890		Dooley, Mr. Patrick	male	32.0	0	

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S
5	0	330877	8.4583	NaN	Q
6	0	17463	51.8625	E46	S
..



884	0	SOTON/0Q	392076	7.0500	NaN	S
885	5		382652	29.1250	NaN	Q
886	0		211536	13.0000	NaN	S
887	0		112053	30.0000	B42	S
888	2	W./C.	6607	23.4500	NaN	S
889	0		111369	30.0000	C148	C
890	0		370376	7.7500	NaN	Q

[891 rows x 12 columns]

1.4.2 Step 1. 遺漏值的處理

在還沒建立模型，處理現實資料的時候，我們通常會端上的第一道處理是遺失值的處理
因為遺失值不可避免會對我們建立模型造成一定的影響

我們先檢查一下到底有哪些欄位是有缺失值，而且缺失幾個

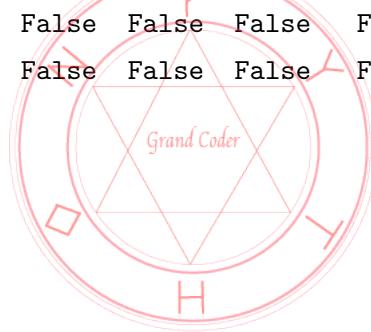
如果缺失太多，我們會選擇直接放棄這個欄位

如果缺失只有少數，我們會在不影響資料整體的前提下，補上遺漏值

In [3]: df.isnull()

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	\
0		False	False	False	False	False	False	False	False	False
1		False	False	False	False	False	False	False	False	False
2		False	False	False	False	False	False	False	False	False
3		False	False	False	False	False	False	False	False	False
4		False	False	False	False	False	False	False	False	False
5		False	False	False	False	True	False	False	False	False
6		False	False	False	False	False	False	False	False	False
..	
884		False	False	False	False	False	False	False	False	False
885		False	False	False	False	False	False	False	False	False
886		False	False	False	False	False	False	False	False	False
887		False	False	False	False	False	False	False	False	False
888		False	False	False	False	True	False	False	False	False
889		False	False	False	False	False	False	False	False	False
890		False	False	False	False	False	False	False	False	False

Fare Cabin Embarked



```

0    False   True   False
1    False  False  False
2    False   True  False
3    False  False  False
4    False   True  False
5    False   True  False
6    False  False  False
...
884  False   True  False
885  False   True  False
886  False   True  False
887  False  False  False
888  False   True  False
889  False  False  False
890  False   True  False

```

[891 rows x 12 columns]

In [4]: # sum 會針對上面得到的結果加出一個答案
`df.isnull().sum()`

Out[4]: PassengerId 0
Survived 0
Pclass 0
Name 0
Sex 0
Age 177
SibSp 0
Parch 0
Ticket 0
Fare 0
Cabin 687
Embarked 2
dtype: int64

1.1 連續數值的缺失處理 如果選擇補上的話

這裡在補上缺失值的時候，通常大家會有兩種不同的做法
做法 1: 補上 mean, 平均值



做法 2: 補上 median, 中位數值

我個人比較不喜歡補平均值，因為平均值會被極高或者極小值影響，所以我會補上中位數值
你可以直接對整個 df 使用 df.fillna(數字)，就會幫你把所有數字型態的欄位直接補上那個數字

```
In [5]: df = df.fillna(df.median())
```

```
df
```

```
Out[5]:   PassengerId  Survived  Pclass \
```

0	1	0	3
1	2	1	1
2	3	1	3
3	4	1	1
4	5	0	3
5	6	0	3
6	7	0	1
..
884	885	0	3
885	886	0	3
886	887	0	2
887	888	1	1
888	889	0	3
889	890	1	1
890	891	0	3

		Name	Sex	Age	SibSp	\
0		Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th... 2	Heikkinen, Miss. Laina	female	38.0	1	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0		
4	Allen, Mr. William Henry	female	35.0	1		
5	Moran, Mr. James	male	35.0	0		
6	McCarthy, Mr. Timothy J	male	28.0	0		
..		
884		Suthehall, Mr. Henry Jr	male	25.0	0	
885	Rice, Mrs. William (Margaret Norton)	female	39.0	0		
886	Montvila, Rev. Juozas	male	27.0	0		
887	Graham, Miss. Margaret Edith	female	19.0	0		
888	Johnston, Miss. Catherine Helen "Carrie"	female	28.0	1		

889			Behr, Mr. Karl Howell	male	26.0	0
890			Dooley, Mr. Patrick	male	32.0	0
<hr/>						
Parch	Ticket	Fare	Cabin	Embarked		
0	A/5 21171	7.2500	Nan	S		
1	PC 17599	71.2833	C85	C		
2	STON/O2. 3101282	7.9250	Nan	S		
3	113803	53.1000	C123	S		
4	373450	8.0500	Nan	S		
5	330877	8.4583	Nan	Q		
6	17463	51.8625	E46	S		
..		
884	0 SOTON/OQ 392076	7.0500	Nan	S		
885	5 382652	29.1250	Nan	Q		
886	0 211536	13.0000	Nan	S		
887	0 112053	30.0000	B42	S		
888	2 W./C. 6607	23.4500	Nan	S		
889	0 111369	30.0000	C148	C		
890	0 370376	7.7500	Nan	Q		

[891 rows x 12 columns]

In [6]: # 補完數字以後只剩 *Cabin*(字串) 和 *Embarked*(字串) 有遺漏值了

```
df.isnull().sum()
```

Out[6]:

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2
dtype:	int64



1.2 類別(字串)數值的缺失處理 如果選擇補上的話，這裡的字串就會直接補上最常出現的字串
但是 Cabin 實在遺漏太多了 687/891，所以我直接選擇放棄這個欄位

```
In [7]: df = df.drop(["Cabin"], axis = 1)
```

Embarked 我選擇補上最常出現的值

```
In [8]: df['Embarked'].value_counts()
```

```
Out[8]: S    644
         C    168
         Q     77
Name: Embarked, dtype: int64
```

```
In [9]: print("Embarked 最常出現:", df['Embarked'].value_counts().idxmax())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].value_counts().idxmax())
print("補完後的 nan:")
print(df.isnull().sum())
```

Embarked 最常出現: S

補完後的 nan:

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         0
dtype: int64
```

1.4.3 Step 2. 特徵的處理

在隨機森林的時候，數值特徵並不需要特別的處理，因為我們是選擇一個數，分成左右兩份，所以就算你把數做標準化(濃縮成 0 到 1)也不會影響任何事物



但是在處理類別(字串)特徵的時候，我們通常會選擇一種處理方式叫做 One-Hot Encoding(獨熱編碼)

為什麼要特別處理類別(字串)，因為事實上我們的 scikit-learn 和大多數的機器學習函式庫在處理特徵的時候，都只會把特徵值當成數值來處理

所以我們會把我們的字串轉成整數表示

這時候就有一個問題了，假設你的特徵值是 0(紅色) 1(藍色) 2(綠色)

但其實紅藍綠根本沒有一個誰大誰小的順序這就會造成問題了

這時候我們會選擇把這個欄位分成三個欄位

One-Hot Encoding 例子

id	顏色
0	紅
1	藍
2	綠

id	紅
0	1
1	0
2	0

我們這裡使用 get_dummies 就會幫我們創造出 one-hot encoding 之後再把它連接回去原本的 df 就好

```
In [10]: # 創造出 one-hot 欄位
dummy = pd.get_dummies(df['Embarked'])
# concat 是連結的意思，axis = 1 指的是水平的連接
df = pd.concat([df, dummy], axis=1)
df = df.drop(['Embarked'], axis = 1)
```

```
dummy = pd.get_dummies(df['Sex'])
df = pd.concat([df, dummy], axis=1)
df = df.drop(['Sex'], axis = 1)
```

df

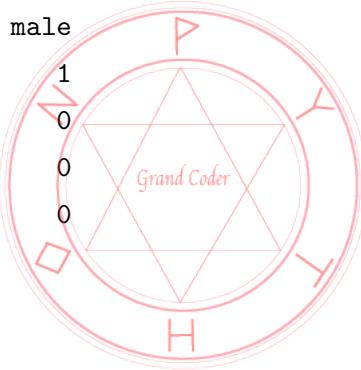
```
Out[10]:      PassengerId  Survived  Pclass \
0                 1         0       3
```



1	2	1	1
2	3	1	3
3	4	1	1
4	5	0	3
5	6	0	3
6	7	0	1
..
884	885	0	3
885	886	0	3
886	887	0	2
887	888	1	1
888	889	0	3
889	890	1	1
890	891	0	3

		Name	Age	SibSp	Parch	\
0		Braund, Mr. Owen Harris	22.0	1	0	
1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	38.0 26.0	1 0	0 0	0 0	
2	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	0	
3	Allen, Mr. William Henry	35.0	0	0	0	
4	Moran, Mr. James	28.0	0	0	0	
5	McCarthy, Mr. Timothy J	54.0	0	0	0	
6	
..		Suttehall, Mr. Henry Jr	25.0	0	0	
884		Rice, Mrs. William (Margaret Norton)	39.0	0	5	
885		Montvila, Rev. Juozas	27.0	0	0	
886		Graham, Miss. Margaret Edith	19.0	0	0	
887		Johnston, Miss. Catherine Helen "Carrie"	28.0	1	2	
888		Behr, Mr. Karl Howell	26.0	0	0	
889		Dooley, Mr. Patrick	32.0	0	0	
890						

	Ticket	Fare	C	Q	S	female	male
0	A/5 21171	7.2500	0	0	1	0	1
1	PC 17599	71.2833	1	0	0	1	0
2	STON/O2. 3101282	7.9250	0	0	1	1	0
3	113803	53.1000	0	0	1	1	0



```

4           373450   8.0500  0  0  1      0  1
5           330877   8.4583  0  1  0      0  1
6           17463    51.8625  0  0  1      0  1
...
...          ...     ...  ...  ...  ...
884  SOTON/OQ 392076   7.0500  0  0  1      0  1
885          382652  29.1250  0  1  0      1  0
886          211536  13.0000  0  0  1      0  1
887          112053  30.0000  0  0  1      1  0
888        W./C. 6607   23.4500  0  0  1      1  0
889          111369  30.0000  1  0  0      0  1
890          370376   7.7500  0  1  0      0  1

```

[891 rows x 14 columns]

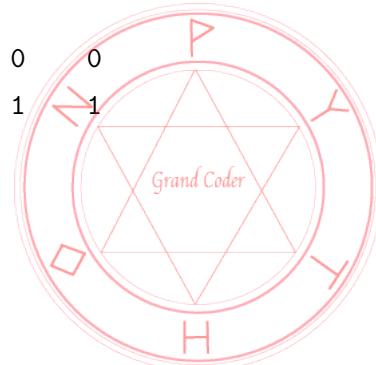
接下來處理名字，名字對我而言有用的東西是稱謂。
 這裡就一定要畫個圖看一下到底每個稱謂出現了幾次
 我畫完了以後發現 Mr Mrs Miss 是三個出現最多而且看起來真的有影響的稱謂，其餘都是一些稀少稱謂
 所以我就只留 Mr Mrs Miss，其餘直接丟掉

```
In [11]: s = df['Name'].str.split(", ", expand = True)[1]
s = s.str.split(" ", expand = True)[1]
# 這裡我要產生 pdf 的時候 style 會印製不出來，所以我用最素的
# 讀者可以把下面的註解拿掉替換
# pd.crosstab(s, df['Survived']).T.style.background_gradient(cmap = "autumn")
pd.crosstab(s, df['Survived']).T
```

```
Out[11]: 1      Capt.  Col.  Don.  Dr.  Jonkheer.  Lady.  Major.  Master.  Miss.  \
Survived
0           1      1      1      4      1      0      1      17      55
1           0      1      0      3      0      1      1      23     127

1      Mlle.  Mme.  Mr.  Mrs.  Ms.  Rev.  Sir.  the
Survived
0           0      0    436     26      0      6      0      0
1           2      1    81     99      1      0      1      1
```

```
In [12]: def name_filter(data):
    if data == 'Mr.':
```



```

        return 'Mr'
    elif data == 'Mrs.':
        return 'Mrs'
    elif data == 'Miss.':
        return 'Miss'
    else:
        return 'Unknown'

df['Name'] = s.apply(name_filter)

```

```

dummy = pd.get_dummies(df['Name'])
df = pd.concat([df, dummy], axis=1)
df = df.drop(["Name"], axis = 1)
df = df.drop(["Unknown"], axis = 1)

```

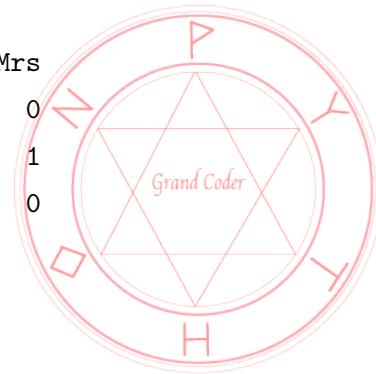
df

```

Out[12]:   PassengerId  Survived  Pclass  Age  SibSp  Parch            Ticket \
0              1         0      3  22.0     1      0          A/5 21171
1              2         1      1  38.0     1      0             PC 17599
2              3         1      3  26.0     0      0  STON/O2. 3101282
3              4         1      1  35.0     1      0             113803
4              5         0      3  35.0     0      0             373450
5              6         0      3  28.0     0      0             330877
6              7         0      1  54.0     0      0             17463
...
884            885         0      3  25.0     0      0  SOTON/OQ 392076
885            886         0      3  39.0     0      5             382652
886            887         0      2  27.0     0      0             211536
887            888         1      1  19.0     0      0             112053
888            889         0      3  28.0     1      2       W./C. 6607
889            890         1      1  26.0     0      0             111369
890            891         0      3  32.0     0      0             370376

```

	Fare	C	Q	S	female	male	Miss	Mr	Mrs
0	7.2500	0	0	1	0	1	0	1	0
1	71.2833	1	0	0	1	0	0	0	1
2	7.9250	0	0	1	1	0	1	0	0



```

3    53.1000  0  0  1      1    0    0    0    1
4    8.0500  0  0  1      0    1    0    1    0
5    8.4583  0  1  0      0    1    0    1    0
6    51.8625  0  0  1      0    1    0    1    0
...
884   7.0500  0  0  1      0    1    0    1    0
885  29.1250  0  1  0      1    0    0    0    1
886 13.0000  0  0  1      0    1    0    0    0
887 30.0000  0  0  1      1    0    1    0    0
888 23.4500  0  0  1      1    0    1    0    0
889 30.0000  1  0  0      0    1    0    1    0
890  7.7500  0  1  0      0    1    0    1    0

```

[891 rows x 16 columns]

Ticket 的數字太多，而且沒有一個很直覺的規律，所以我先將其丟棄
PassengerId 只是一個一直增加的數，也並不影響我們的結果，所以也先將其丟棄

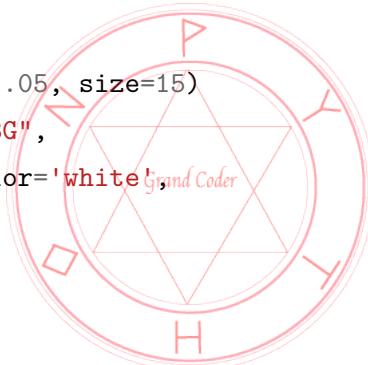
```
In [13]: df = df.drop(["Ticket"], axis = 1)
#df = df.drop(["Name"], axis = 1)
df = df.drop(["PassengerId"], axis = 1)
```

```
In [14]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練
from sklearn.model_selection import train_test_split
# 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
data_train, data_test, target_train, target_test = train_test_split(
                                                    df.drop(["Survived"], axis = 1),
                                                    df['Survived'],
                                                    test_size=0.1)
```

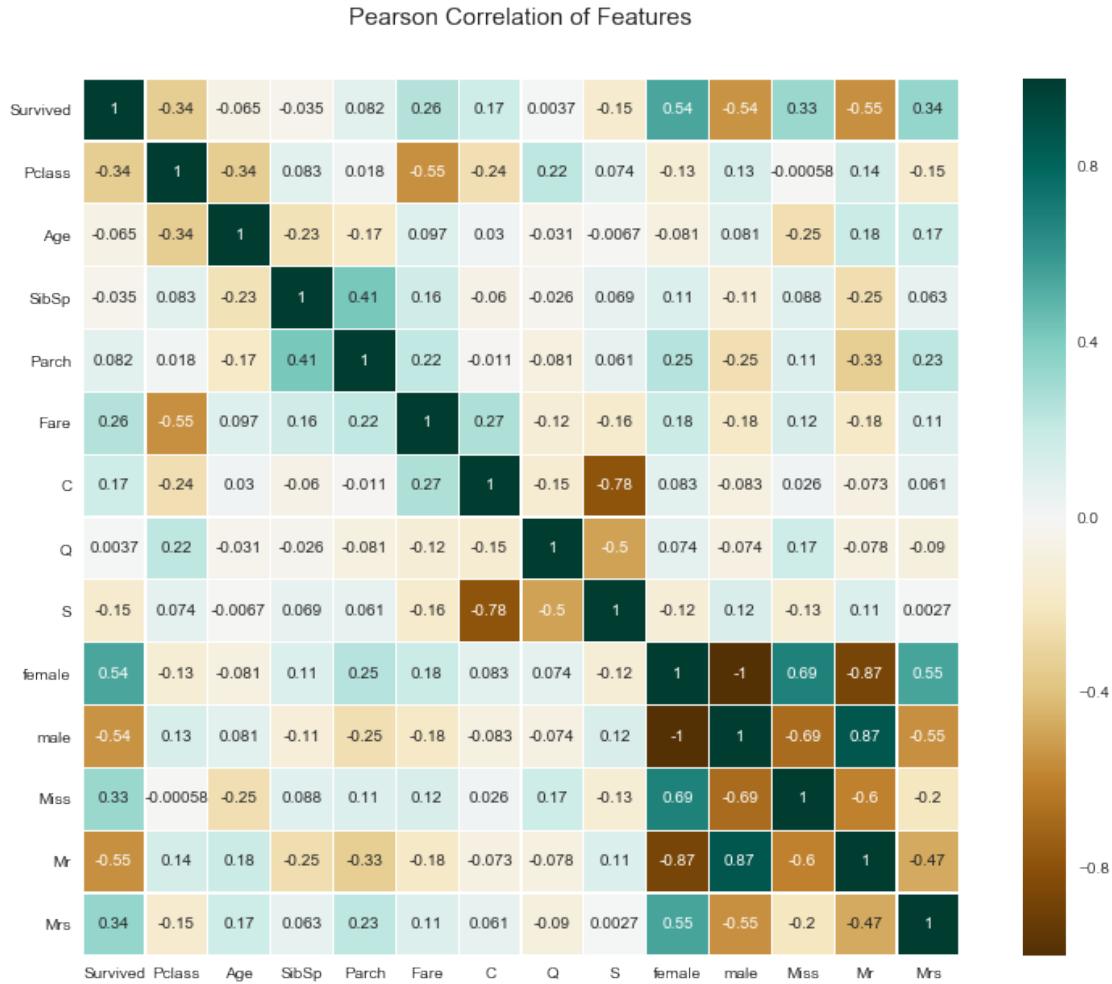
1.4.4 Step 3. 初步感覺

畫個 heatmap 感覺一下什麼是比較重要的，基本你發現了，性別大概就是最重要的差別
在那個事件上，大部分男士都很紳士的讓女士先上救生艇了！

```
In [15]: plt.figure(figsize=(14,10))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(df.astype(float).corr(), cmap = "BrBG",
            linewidths=0.1, square=True, linecolor='white',
            annot=True)
```



Out [15]: <matplotlib.axes._subplots.AxesSubplot at 0x11c094cc0>



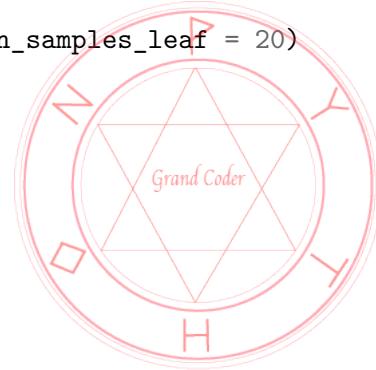
1.5 開始預測

1.5.1 Step 1. 初步感覺

我們先使用之前說過的 DecisionTreeClassifier 看看一個分類樹會怎麼處理我們的問題

```
In [16]: from sklearn.tree import DecisionTreeClassifier
        clf = DecisionTreeClassifier(max_depth = 15, min_samples_leaf = 20)
        clf = clf.fit(data_train, target_train)
```

```
In [17]: from sklearn.tree import export_graphviz
        import graphviz
```



```

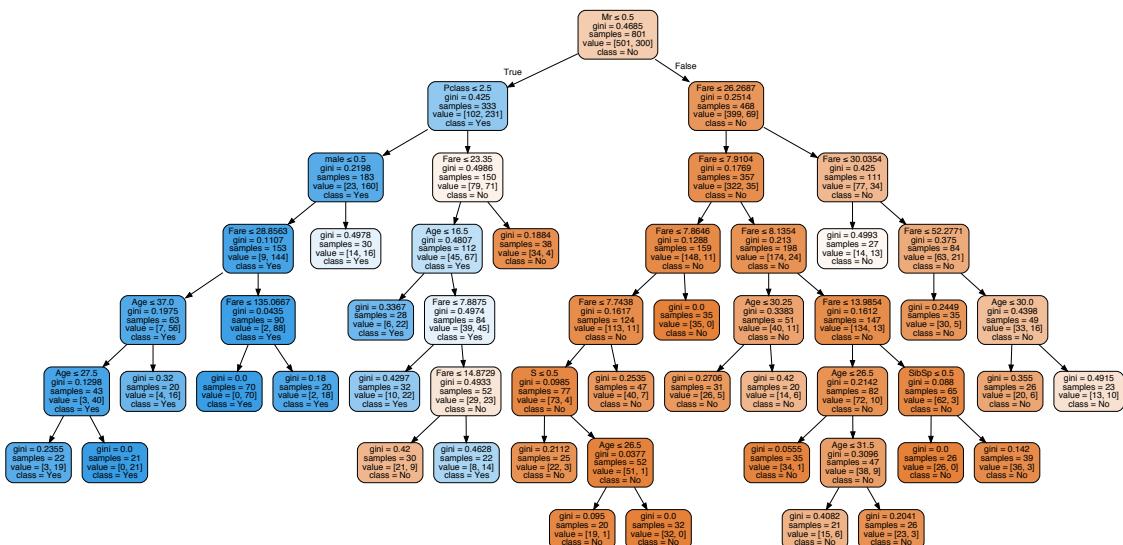
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=df.drop(["Survived"], axis=1).columns,
                           class_names=["No", "Yes"],
                           filled=True, rounded=True,
                           special_characters=True)

graph = graphviz.Source(dot_data)
# 這行可以輸出一個 pdf, 讀者可以自行把註解拿掉試試看
# graph.render("iris2")

graph

```

Out [17] :

In [18]: `from sklearn.metrics import accuracy_score`

```

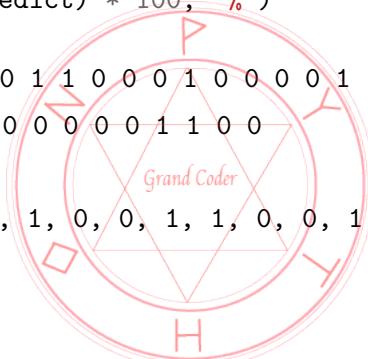
predict = clf.predict(data_test)

print("預測:", predict)
print("正確標籤:", list(target_test))
print("正確率: ", accuracy_score(target_test, predict) * 100, "%")

```

預測: [0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 1
1 0 0 1 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0]

正確標籤: [0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]



正確率： 82.2222222222221 %

1.5.2 Step 2. 建立模型

開始使用隨機森林來建立模型，不過這次我們不使用隨機切割了

我們做一件事叫做交叉驗證

交叉驗證指的是每次把一部分的 samples 拿出來當訓練，另外一部分拿來當測試
最後算出每次驗證的平均

交叉驗證可以較好的表示我們模型的好壞

RandomForest 的參數調整我們通常我們會調整

`n_estimators`: 要有幾顆樹，理論上越多越好，但是如果你的資料集沒有這麼大，產生太多樹反而沒那麼有意義，因為你會有很多同類型的臭皮匠

`max_depth`: 理論上不需要被調整，但我還是會建議調整一下，不要讓你每個臭皮匠過擬合
這裡經過我的調整我選擇 26 顆樹，每棵樹六層的深度

```
In [19]: from sklearn.ensemble import RandomForestClassifier
        clf = RandomForestClassifier(n_estimators = 26, max_depth = 6)
```

```
In [20]: # 這裡就不用 fit 了，fit 和 predict 會由交叉驗證幫你做，cv 參數代表要幾次的交叉驗證
        from sklearn.model_selection import cross_val_score
        scores = cross_val_score(clf, df.drop(["Survived"], axis = 1),
                                df['Survived'], cv = 10)
        print("十次分數:", scores)
        # 由於 score 是 ndarray，可以直接使用 average 來計算平均
        import numpy as np
        print("平均:", np.average(scores))
```

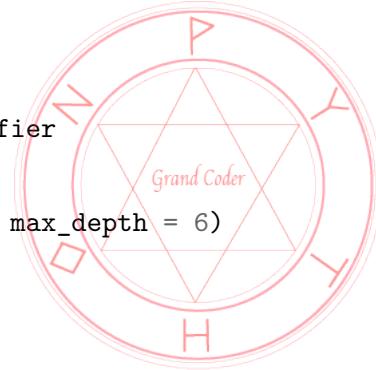
十次分數: [0.81111111 0.85555556 0.7752809 0.88764045 0.85393258 0.83146067
0.83146067 0.7752809 0.86516854 0.85227273]

平均: 0.8339164113040518

1.5.3 Step 3. 正式預測

正式的預測我們的 test.csv，並且把結果輸出並上傳

```
In [21]: from sklearn.ensemble import RandomForestClassifier
        # 6 or 7 is good
        clf = RandomForestClassifier(n_estimators = 26, max_depth = 6)
```



```

clf = clf.fit(df.drop(["Survived"], axis = 1), df['Survived'])

test_df = pd.read_csv("test.csv")

result_df = pd.DataFrame(columns = ["PassengerId", "Survived"])
result_df["PassengerId"] = test_df["PassengerId"]

test_df = test_df.fillna(df.median())
test_df = test_df.drop(["Cabin"], axis = 1)
test_df['Embarked'] = test_df['Embarked'].fillna("S")

dummy = pd.get_dummies(test_df['Embarked'])
test_df = pd.concat([test_df, dummy], axis=1)
test_df = test_df.drop(["Embarked"], axis = 1)
test_df = test_df.drop(["Ticket"], axis = 1)
test_df = test_df.drop(["PassengerId"], axis = 1)

dummy = pd.get_dummies(test_df['Sex'])
test_df = pd.concat([test_df, dummy], axis=1)
test_df = test_df.drop(["Sex"], axis = 1)

s = test_df['Name'].str.split(", ", expand = True)[1]
s = s.str.split(" ", expand = True)[1]

test_df['Name'] = s.apply(name_filter)

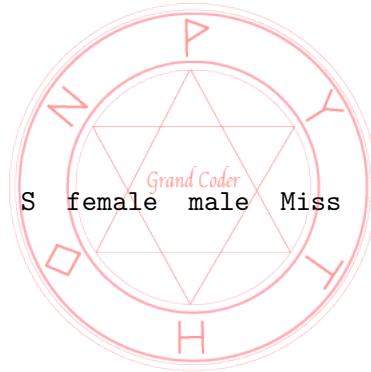
dummy = pd.get_dummies(test_df['Name'])
test_df = pd.concat([test_df, dummy], axis=1)
test_df = test_df.drop(["Name"], axis = 1)
test_df = test_df.drop(["Unknown"], axis = 1)

test_df

```

Out[21]:

	Pclass	Age	SibSp	Parch	Fare	C	Q	S	female	male	Miss	Mr	\



0	3	34.5	0	0	7.8292	0	1	0	0	1	0	1
1	3	47.0	1	0	7.0000	0	0	1	1	0	0	0
2	2	62.0	0	0	9.6875	0	1	0	0	1	0	1
3	3	27.0	0	0	8.6625	0	0	1	0	1	0	1
4	3	22.0	1	1	12.2875	0	0	1	1	0	0	0
5	3	14.0	0	0	9.2250	0	0	1	0	1	0	1
6	3	30.0	0	0	7.6292	0	1	0	1	0	1	0
..
411	1	37.0	1	0	90.0000	0	1	0	1	0	0	0
412	3	28.0	0	0	7.7750	0	0	1	1	0	1	0
413	3	28.0	0	0	8.0500	0	0	1	0	1	0	1
414	1	39.0	0	0	108.9000	1	0	0	1	0	0	0
415	3	38.5	0	0	7.2500	0	0	1	0	1	0	1
416	3	28.0	0	0	8.0500	0	0	1	0	1	0	1
417	3	28.0	1	1	22.3583	1	0	0	0	1	0	0

Mrs

0	0
1	1
2	0
3	0
4	1
5	0
6	0
..	...
411	1
412	0
413	0
414	0
415	0
416	0
417	0

[418 rows x 13 columns]

In [22]: pre = clf.predict(test_df)



```
result_df["Survived"] = pre  
result_df.to_csv("result_test.csv", index = False)
```

上傳你的第一個競賽的結果吧!!

你會發現我們的模型在沒看過的資料上面大概是 78% - 81% 的正確率

排名在 11000 多個排名排名約 700

已經表現得非常不錯了!!

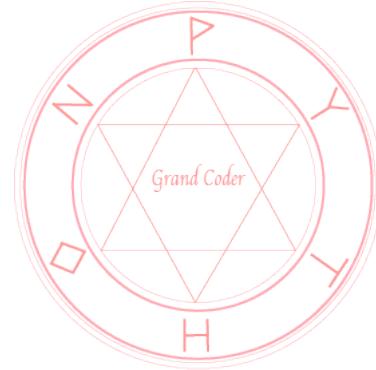
714

▼ 56

Elwing



0.80861



ANN_Basic

2018 年 7 月 18 日

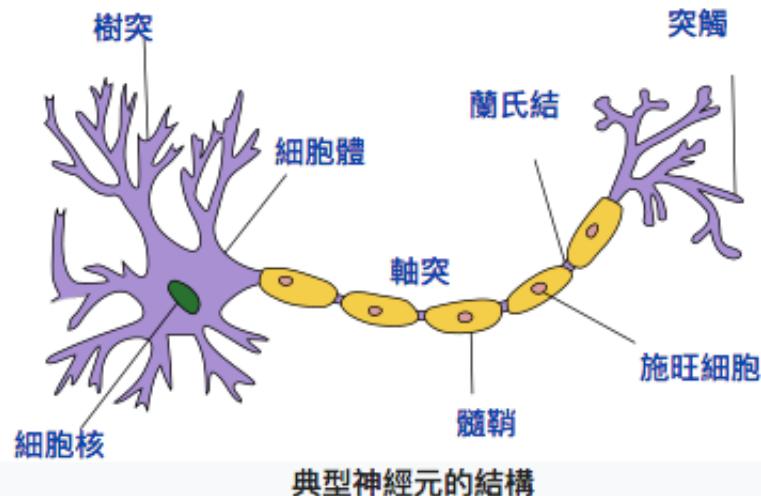


1 初探神經網路 (感知器)

1.1 介紹

在前面的樹類和機率演算法如火如荼的發展和使用的時候，有一脈的演算法也正快速的發展中
這種類的演算法我們稱之為神經網路

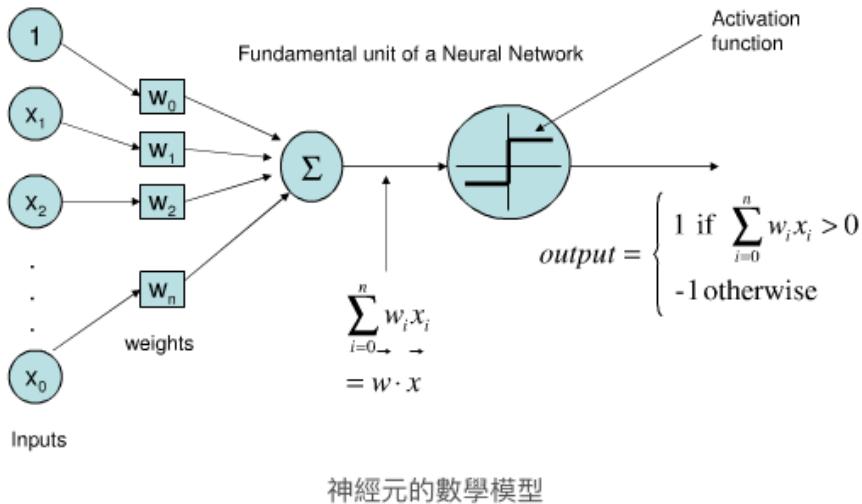
為何叫神經網路呢？因為這類的演算法的基礎是一個很類似我們人的神經的一個判斷方式



樹突接受外界刺激，達到一定程度就會激發突觸，繼續往下一層傳遞

最初的神經網路就是我們今天講的感知器 (Perceptron)，模仿單一的神經元





簡單來說，就是把每個特徵(刺激)乘上一定的係數再加起來
超過一定的限度就歸類為正類，否則為負類
所以整個步驟是這樣的：

1. 我們有一個特徵向量 $[a, b, c]$
2. 乘上係數 $score = w_1 * a + w_2 * b + w_3 * c$
3. 讓我們的 $score$ 經過一個啟動(激勵)函數(Activation Function)，這啟動函數我們先設定成最基本的超過一定的量輸出 1，否則輸出 0

1.2 需要函式庫

1. mlxtend: 可以幫我們快速畫出分類線的函式庫

1.3 資料集

我們利用之前已經用過的鳶尾花數據集，但我們這次做一點比較特別的事
我們只用花瓣和花萼的長度來分類就好
因為我們希望讓你看到決策的邊界

1.4 開始撰寫程式

1.4.1 Step 0. 讀入我們的鳶尾花數據集作為練習

```
In [1]: from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
```



```

import seaborn as sns
%matplotlib inline

# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示 10 個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)

# 為了顯示的漂亮，有時候 sklearn 會提示一下 Future Warning
# 我也把關掉了
import warnings
warnings.filterwarnings('ignore')

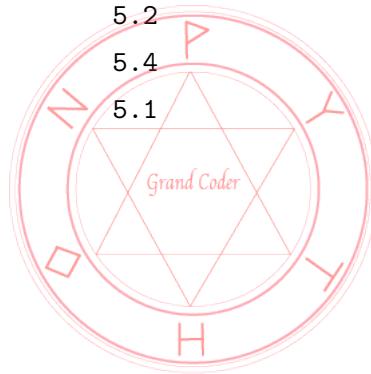
# 使用 scikit-learn 提供的鳶尾花資料庫
iris = load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df["target"] = iris["target"]
df

```

Out[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	
5	5.4	3.9	1.7	0.4	
6	4.6	3.4	1.4	0.3	
..
143	6.8	3.2	5.9	2.3	
144	6.7	3.3	5.7	2.5	
145	6.7	3.0	5.2	2.3	
146	6.3	2.5	5.0	1.9	
147	6.5	3.0	5.2	2.0	
148	6.2	3.4	5.4	2.3	
149	5.9	3.0	5.1	1.8	

target

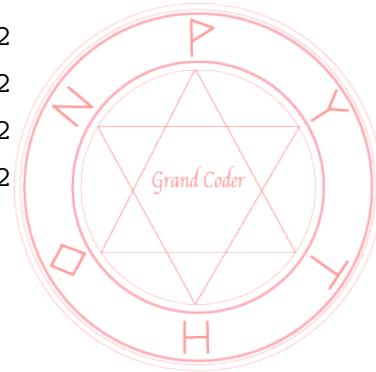


0	0
1	0
2	0
3	0
4	0
5	0
6	0
..	...
143	2
144	2
145	2
146	2
147	2
148	2
149	2

[150 rows x 5 columns]

```
In [2]: df = df.drop(["petal width (cm)", "sepal width (cm)"], axis = 1)  
df
```

```
Out[2]:      sepal length (cm)  petal length (cm)  target  
0           5.1              1.4          0  
1           4.9              1.4          0  
2           4.7              1.3          0  
3           4.6              1.5          0  
4           5.0              1.4          0  
5           5.4              1.7          0  
6           4.6              1.4          0  
..          ...              ...          ...  
143         6.8              5.9          2  
144         6.7              5.7          2  
145         6.7              5.2          2  
146         6.3              5.0          2  
147         6.5              5.2          2  
148         6.2              5.4          2  
149         5.9              5.1          2
```



```
[150 rows x 3 columns]
```

In [3]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練

```
from sklearn.model_selection import train_test_split
# 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
data = df.drop(["target"], axis = 1)
data_train, data_test, target_train, target_test = train_test_split(data,
                                                               df['target'],
                                                               test_size=0.1)
```

1.4.2 Step 1. 建立模型

我們使用 sklearn 的 Perceptron 來建造我們的感知器

這裡我並沒有調整過多的參數，目的只是為了讓你看看感知器是如何做出分類的

In [4]: from sklearn.linear_model import Perceptron
clf = Perceptron()
clf = clf.fit(data_train, target_train)

In [5]: from sklearn.metrics import accuracy_score

```
predict = clf.predict(data_test)
print("預測:", list(predict))
print("正確標籤:", list(target_test))
print("正確率: ", accuracy_score(target_test, predict) * 100, "%")
```

預測: [2, 2, 2, 0, 2, 2, 0, 0, 0, 2, 0, 2, 0, 0]

正確標籤: [2, 2, 1, 0, 1, 1, 2, 0, 0, 0, 2, 0, 1, 0, 0]

正確率: 73.3333333333 %

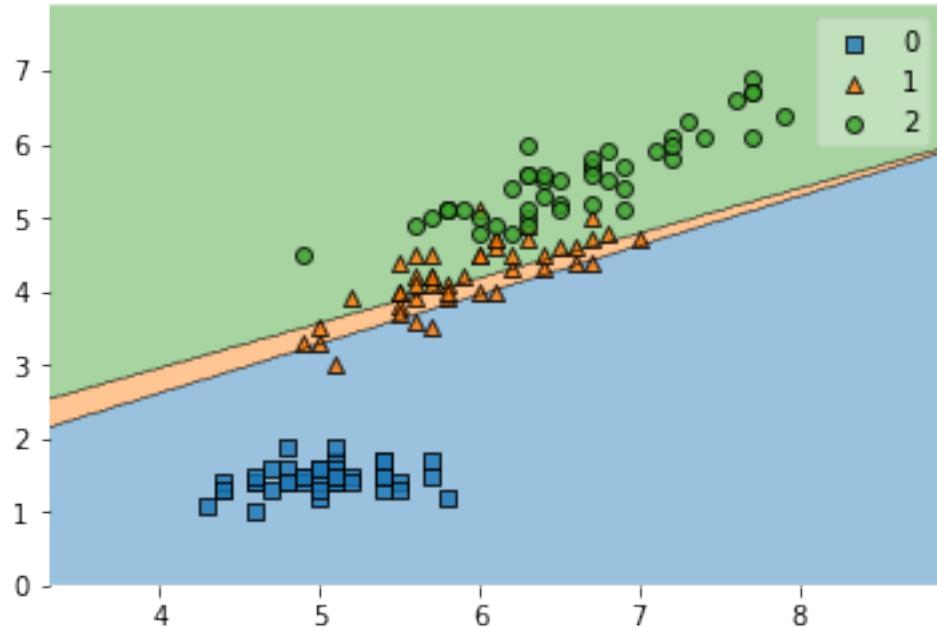
1.4.3 Step 2. 畫出決策邊界

接著我們用剛剛的分類器畫出他的決策邊界

In [6]: from mlxtend.plotting import plot_decision_regions
import numpy as np
plot_decision_regions(X=np.array(data_train),
y=np.array(target_train),
clf=clf)



Out [6]: <matplotlib.axes._subplots.AxesSubplot at 0x10a63cb70>



你應該發現了，感知器畫出一條直線來分出兩個區間，所以那種無法用直線分開的混濁類別我們就無法分類了

現在我們將分類或回歸分成兩種

1. 線性分類器: 分類畫出的標準是直來直去的，二維畫出的分類是一條線，三維畫出的分類是一個平面，簡單來說就是每個特徵的衡量都是一次方！用數學的方式表示就是 $f(x) = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 \dots$
2. 非線性分類器: 去模擬非線性分布的方法很多，最直覺的方式就是使用一個多次方的函式去擬合你的資料，ex: $f(x) = w_1 * x_1^{12} + w_2 * x_2^{12} \dots$ ，來模擬橢圓形，這種方式類似我們下面的單純貝氏畫出來的方式(但用單純貝氏舉例比較不直覺，你可以想成貝氏做了很多特徵機率的相乘，所以並不是個一次方的擬合)。另外一種方式是多條直線切割模擬非線性分類，最容易理解的例子就是下面的決策樹，決策樹分類非線性的方法是不斷二切，就可以切出一個用很多直線構成的非線性擬合

單純貝氏擬合非線性的決策邊界如下

In [14]: # 由於我們的特徵不是不連續的整數，我們必須使用 GaussianNB 來

```
from sklearn.naive_bayes import GaussianNB
```

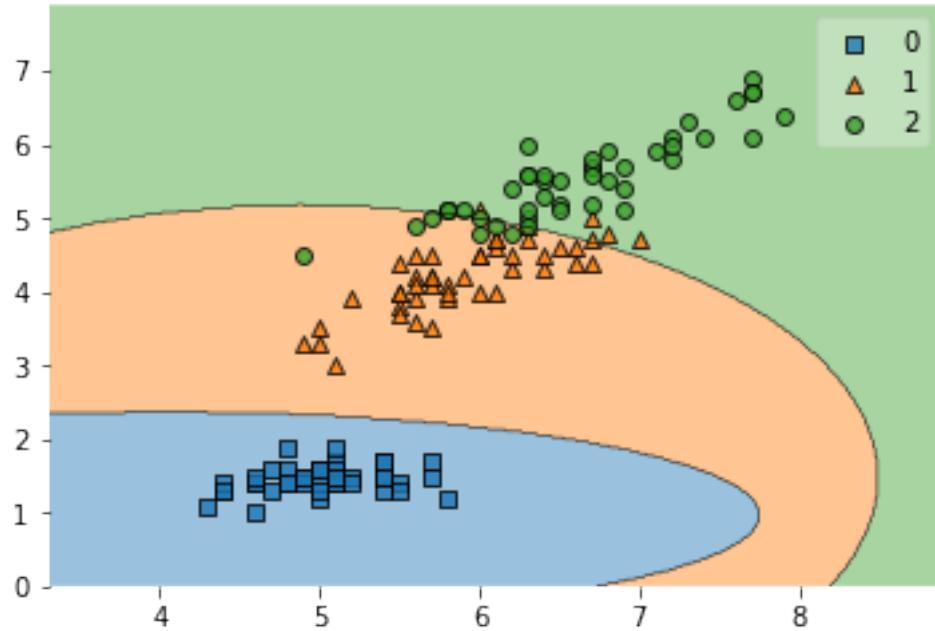


```

clf = GaussianNB()
clf = clf.fit(data_train, target_train)
plot_decision_regions(X=np.array(data_train),
                      y=np.array(target_train),
                      clf=clf)

```

Out [14]: <matplotlib.axes._subplots.AxesSubplot at 0x11227f2b0>



決策樹擬合非線性的決策邊界如下

In [12]: `from sklearn.tree import DecisionTreeClassifier`

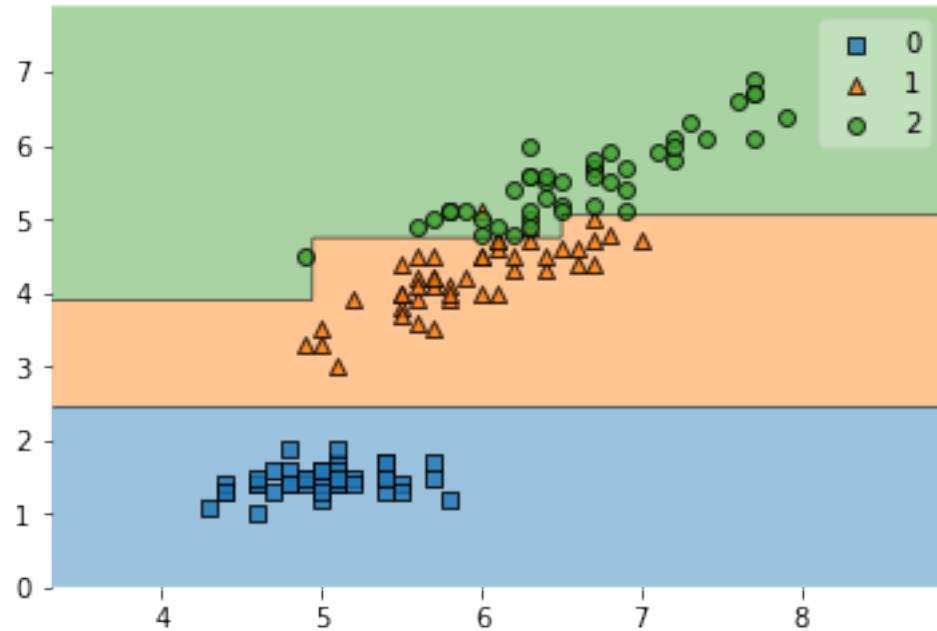
```

# 讀者可以試試看從 max_depth = 1 慢慢往上調整，觀看決策邊界的變化
clf = DecisionTreeClassifier()
clf = clf.fit(data_train, target_train)
plot_decision_regions(X=np.array(data_train),
                      y=np.array(target_train),
                      clf=clf)

```

Out [12]: <matplotlib.axes._subplots.AxesSubplot at 0x112301dd8>





1.5 問題 1

我們發現感知器有一點點小小的問題

因為只有正負的分類，並沒有把相對於分類線的距離做成人類比較容易理解的量度，所以會比較難跟人解釋（你無法說出：降雨的機率是多少，你只能說出：會不會降雨）

為了解決這個問題，我們發展出邏輯斯回歸 (Logistic Regression)，我們在下一章節介紹這個問題

1.6 問題 2

感知器身為神經網路的鼻祖，現在多是一個精神象徵的地位，而不是直接拿來實際使用了，因為我們在現實的問題大部分都是非線性問題居多，所以我們繼續往下擴展，來繼續看神經網路的發展。順便讓大家看看那代人對感知器最後下的結論，『對於 XOR(Exclusive OR) 問題完全沒有辦法分類』

XOR 問題：類似 OR 問題，不過在左右都是 True 的時候會是 False。

舉例：一個女生對於帥或者有才華只要有一個就可以了，兩者都有更好，這叫 OR 問題。但是如果他對於又帥又有才華的男生有恐懼感，這就叫 XOR 問題。



OR	True	False
True	True	True
False	True	False

XOR	True	False
True	False	True
False	True	False

```
In [27]: from numpy import random
# 可以用 numpy 快速產生隨機，第一個參數是你產生有多少種類
# 第二個參數是你要幾個
# x1 是我們的第一特徵，你可以想像成帥
# x2 是我們的第二特徵，你可以想像成有才華
x1 = random.choice([True, False], 100)
x2 = random.choice([True, False], 100)
# y 是我們的 target，你可以想像成會不會喜歡
y = np.logical_xor(x1, x2)
df = pd.DataFrame(columns = ["x1", "x2", "y"])
df["x1"] = x1
df["x2"] = x2
df["y"] = y
df = df.astype(int)
df
```

Out[27]:

	x1	x2	y
0	0	0	0
1	0	0	0
2	1	1	0
3	1	1	0
4	0	0	0
5	0	1	1
6	1	0	1
..
93	0	1	1
94	1	0	1
95	0	0	0

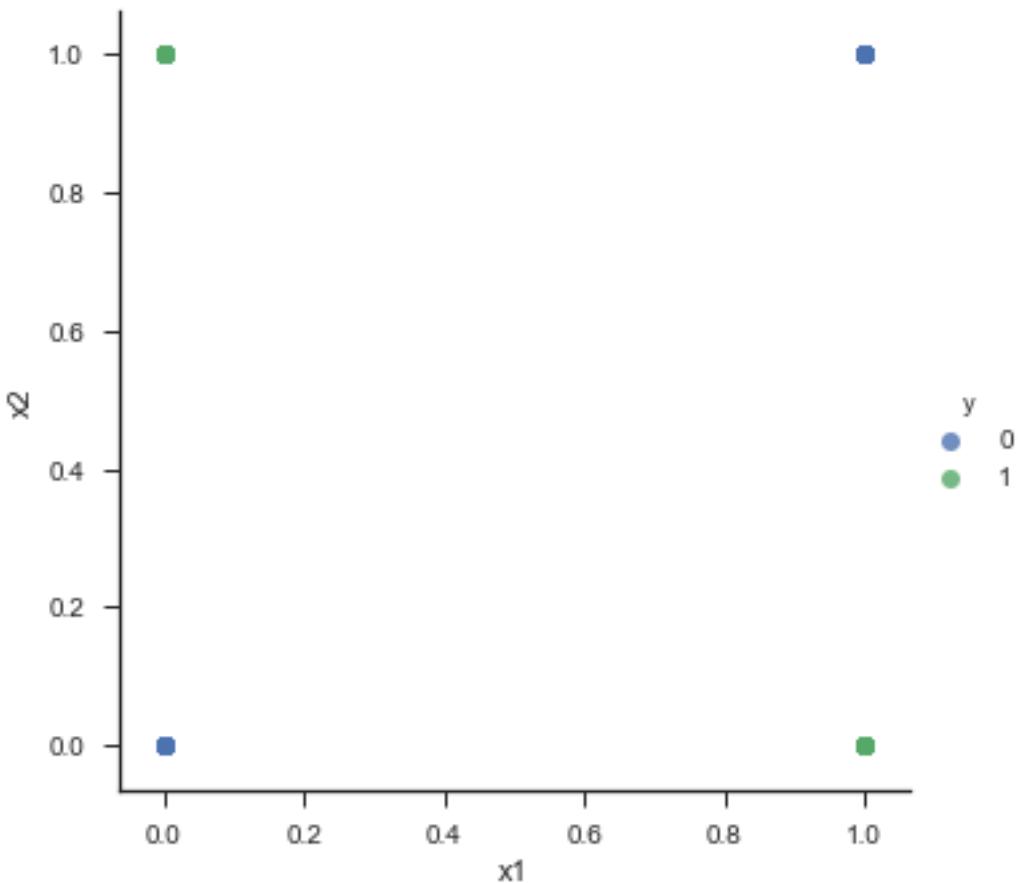


```
96    1    1    0  
97    0    0    0  
98    0    1    1  
99    0    1    1
```

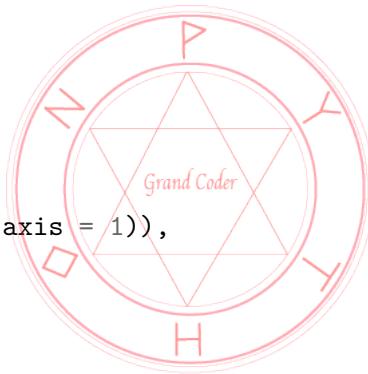
[100 rows x 3 columns]

```
In [28]: import seaborn as sns  
sns.lmplot(x = "x1", y = "x2", hue = "y", data=df, fit_reg = False)
```

Out[28]: <seaborn.axisgrid.FacetGrid at 0x1132fd278>

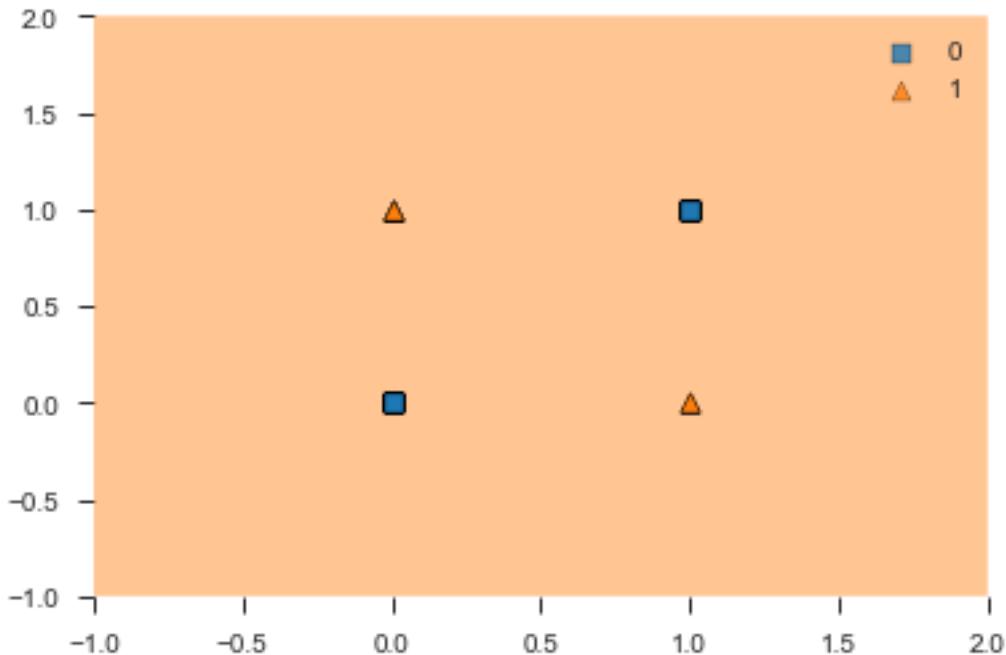


```
In [30]: clf = Perceptron()  
clf = clf.fit(df.drop(["y"], axis = 1), df["y"])  
plot_decision_regions(X=np.array(df.drop(["y"], axis = 1)),
```



```
y=np.array(df["y"]),
clf=clf)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x113e92e48>



你可以看到完全分不出來，這分類器將所有的類別都當成 True。我們下一章節要就這個問題來一起看看神經網路的發展



ANN_Basic_2

2018 年 7 月 18 日



1 初探神經網路 (邏輯斯迴歸)

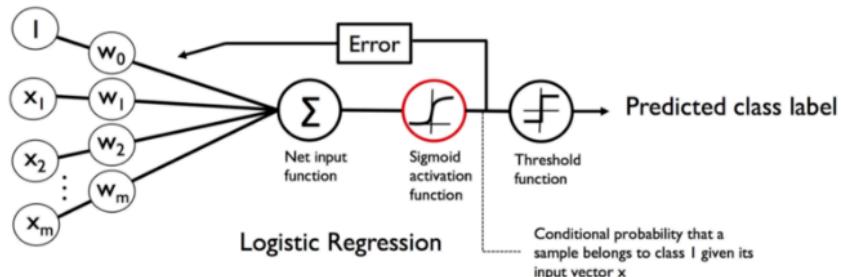
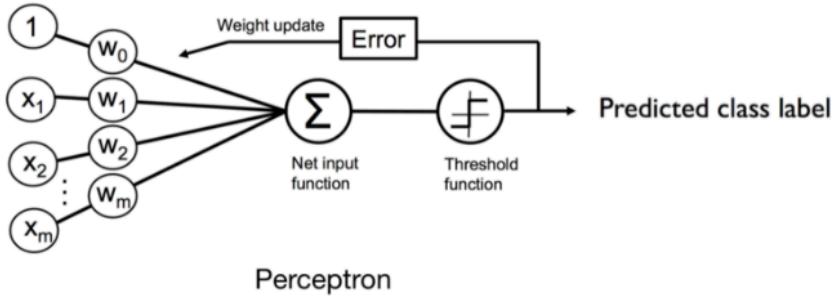
1.1 介紹

前面我們說的問題 1，對於解釋性的部分我們的感知器有點稍嫌弱，有人就提出一個變體，我們在進到我們的分類前，先轉換成一個機率值

那怎麼轉換機率值呢？我們經過一個 0~1 連續分布的函數不就得了

所以我們把我們的模型轉換成下面這樣



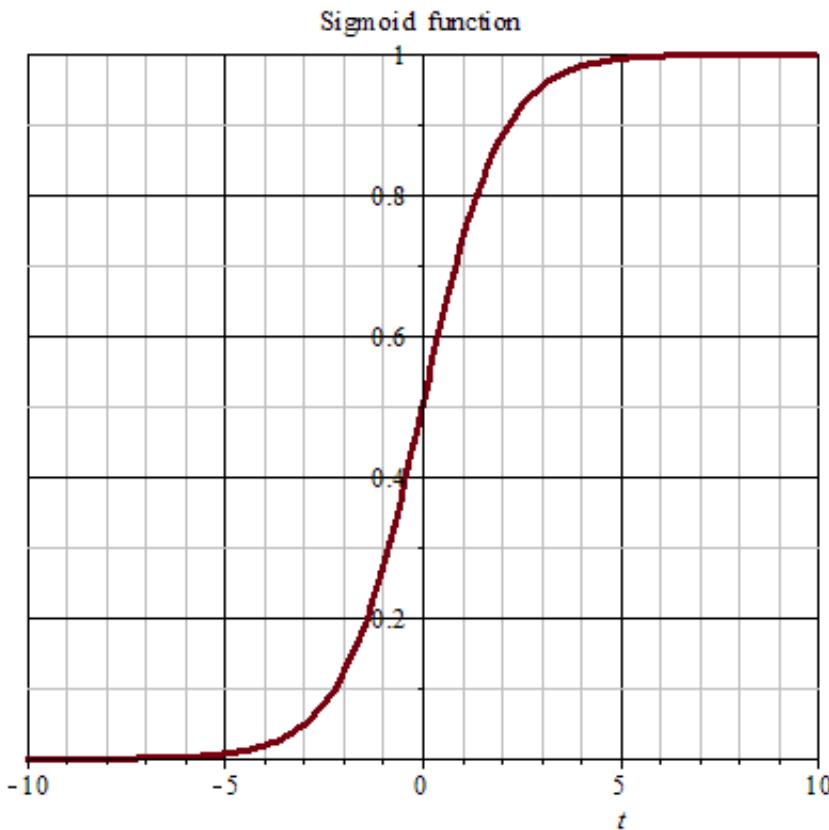


Perceptron以及Logistic Regression模型的差異

這個函數我們常用所謂的 S 函數 (sigmoid function) 來轉換，之所以叫 S 函數是因為他長得像 S 型，同時他也有個別名叫做 Logistic 函數

$$S(t) = \frac{1}{1 + e^{-t}}.$$





經過這樣轉換，我們就也可以跟人家說出機率了！

這個就叫我們的邏輯斯迴歸 (Logistic Regression)

注意：邏輯斯回歸雖然有個迴歸但是他是一個分類模型！這迴歸字眼指的是我們用這個 S 函數去迴歸 (擬合) 的意思

所以步驟是這樣的

1. 我們有一個特徵向量 $[a, b, c]$
2. 乘上係數 $score = w_1 * a + w_2 * b + w_3 * c$
3. (跟之前不一樣的一步) 再將我們算出的 score 經過 S 函數，得到一個新 score 值，這值就可以拿來當作機率了
4. 讓我們的 score 經過一個啟動函數 (Activation Function)，這啟動函數我們先設定成最基本的
超過一定的量輸出 1，否則輸出 0

1.2 需要函式庫

1. mlxtend: 可以幫我們快速畫出分類線的函式庫



1.3 資料集

我們利用之前已經用過的鳶尾花數據集，但我們這次做一點比較特別的事
 我們只用花瓣和花萼的長度來分類就好
 因為我們希望讓你看到決策的邊界

1.4 開始撰寫程式

由於步驟和之前差不多，這裡就直接從頭寫到結束，首先先準備資料集

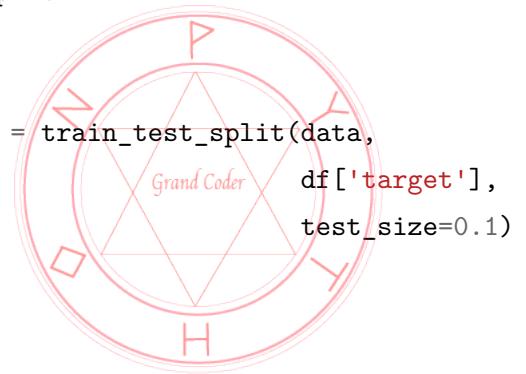
```
In [1]: from sklearn.datasets import load_iris
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        # 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示 10 個
        # 大家練習的時候可以去掉下面兩行
        pd.set_option('display.max_rows', 15)
        pd.set_option('display.max_columns', 10)

        # 為了顯示的漂亮，有時候 sklearn 會提示一下 Future Warning
        # 我也把關掉了
        import warnings
        warnings.filterwarnings('ignore')

        # 使用 scikit-learn 提供的鳶尾花資料庫
        iris = load_iris()
        df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
        df["target"] = iris["target"]
        df = df.drop(["petal width (cm)", "sepal width (cm)"], axis = 1)

        # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練
        from sklearn.model_selection import train_test_split
        # 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
        data = df.drop(["target"], axis = 1)
        data_train, data_test, target_train, target_test = train_test_split(data,
                                                                           df['target'],
                                                                           test_size=0.1)
```



接著訓練我們的 Logistic Regression

```
In [4]: from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf = clf.fit(data_train, target_train)
```

預測試試看，但這裡我們順便把機率印出來

```
In [6]: from sklearn.metrics import accuracy_score
```

```
predict = clf.predict(data_test)
print("預測:", list(predict))
print("正確標籤:", list(target_test))
print("正確率: ", accuracy_score(target_test, predict) * 100, "%")
```

預測: [2, 1, 0, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 0]

正確標籤: [1, 1, 0, 2, 1, 2, 1, 0, 1, 0, 1, 2, 2, 0, 0]

正確率: 86.6666666667 %

```
In [8]: # 你可以看到我們的 sample 分屬 0, 1, 2 的機率
```

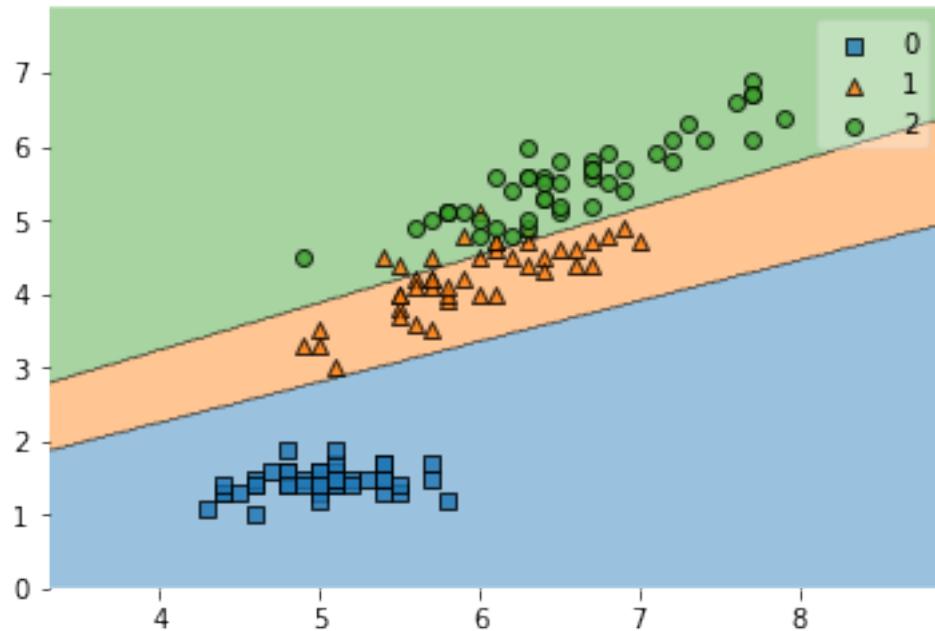
```
proba = pd.DataFrame(clf.predict_proba(data_test))
pd.DataFrame(proba)
```

	0	1	2
0	0.009954	0.482254	0.507793
1	0.015853	0.500694	0.483452
2	0.821032	0.178457	0.000511
3	0.010367	0.494120	0.495513
4	0.029885	0.528887	0.441228
5	0.000611	0.336665	0.662724
6	0.006807	0.418578	0.574614
7	0.823367	0.176253	0.000380
8	0.068456	0.627174	0.304370
9	0.821987	0.177550	0.000463
10	0.053154	0.634643	0.312203
11	0.000623	0.327043	0.672334
12	0.002899	0.379808	0.617293
13	0.821987	0.177550	0.000463
14	0.841304	0.158456	0.000240



```
In [9]: from mlxtend.plotting import plot_decision_regions
import numpy as np
plot_decision_regions(X=np.array(data_train),
                      y=np.array(target_train),
                      clf=clf)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x10aedf048>



1.5 結論

透過上面的決策邊界，你可以看到，Logistic Regression 雖然解決了第一個問題，但卻沒解決我們的第二個問題，非線性的分類。所以實用性也不算高，也屬於精神象徵類型，下一章節我們繼續往前邁進，對於非線性分類給出一個好的答案！



ANN_BASIC_3

2018 年 7 月 18 日



1 再探神經網路 - SVM

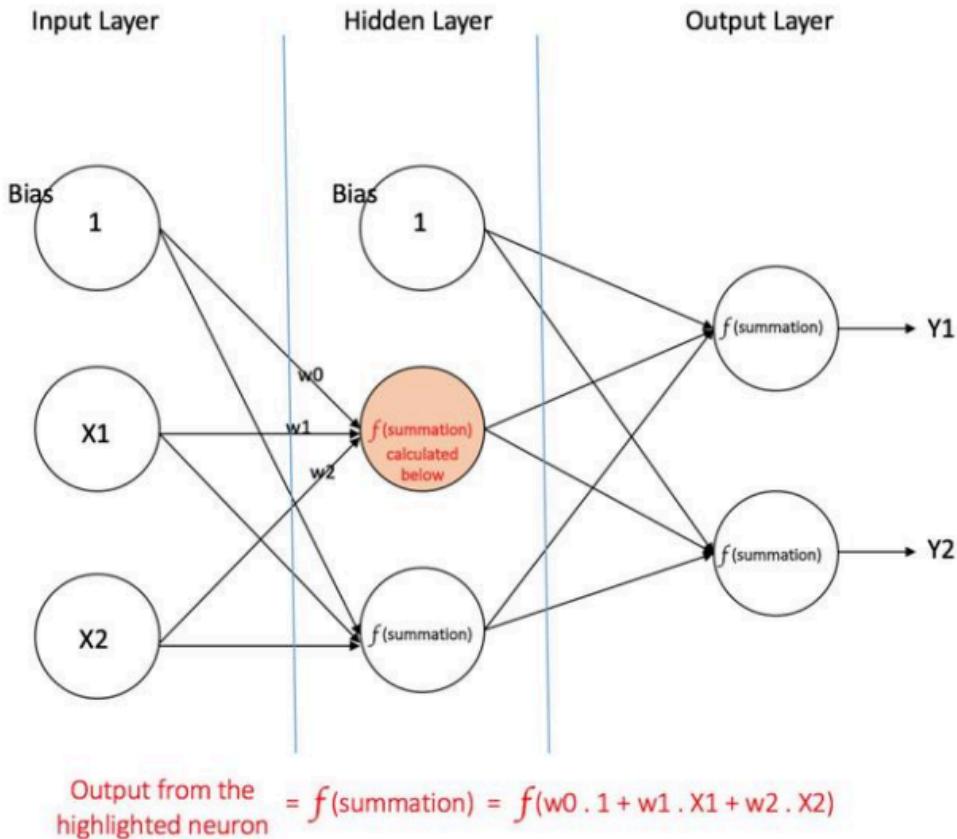
1.1 介紹

上一個章節我們介紹了感知器，我們也知道感知器有其限制，只能分類線性可劃分的類別。所以在歷史的發展中，感知器一度被捨棄，最明顯的辯論就是感知器對於 XOR 問題極度的差，接下來神經網路大神們就著重在這問題提出兩種解決方式

1.1.1 解法 1: 多層感知器 (Multi-layer Perceptron)

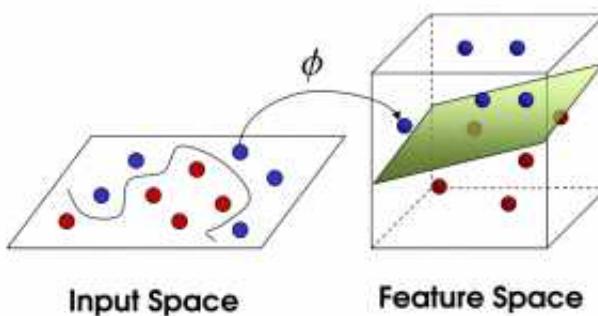
模擬神經的方式，利用多個神經元組合下去，也就是所謂的 MLP(Multi-layer Perceptron)，我們留待下一章節再介紹





1.1.2 解法 2:

先把你的數據點升高維度，也許就可以變成線性可分，例子如下圖

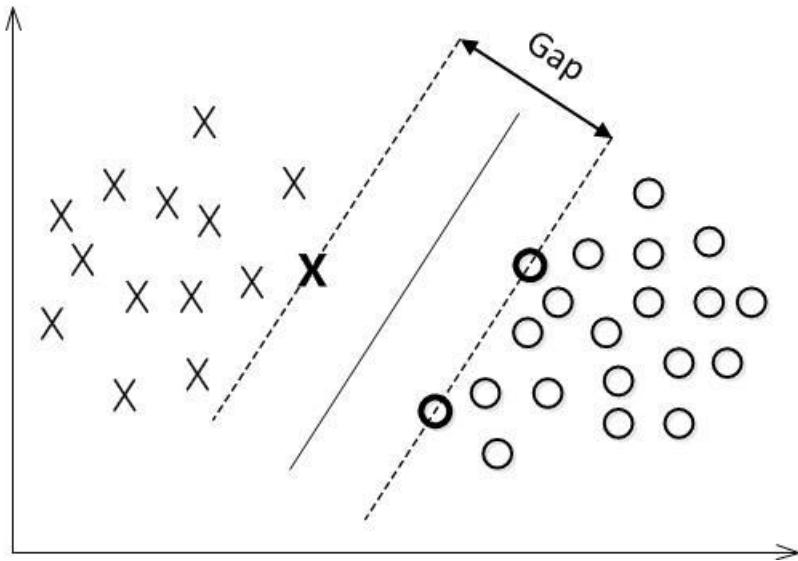


但是，怎麼把你的網路變成非線性的分類器呢？

我們先從一個線性的分類器說起，但這次我們不使用之前的感知器或者邏輯斯回歸，我們使用一種方法叫做 SVM(支援向量機)



1.2 支援向量機 (Support Vector Machine)



之前我們的線性分類器都只著重在畫出一條可以分割的線就好了，但有時候你畫出的那條線只是把兩邊分開，卻不是最中間，把大家分最開的一條線，所以下一個要判別的資料進來的時候就有可能分錯，於是我們有了支援向量機

支援向量機最重要的一件事，就是畫出一條把正類和負類分最開的線

而我們的問題就變成找到負類距離最近點的平行線 (上方經過 X 的虛線) 和正類距離最近點的平行線 (上方經過 O 的虛線)，當他們距離最大的時候，中間的實線就是我們要找的線了！

1.3 線性 SVM

由於步驟和之前差不多，這裡就直接從頭寫到結束，首先先準備資料集

```
In [3]: from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from mlxtend.plotting import plot_decision_regions
%matplotlib inline
# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示 10 個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)
```



```
# 為了顯示的漂亮，有時候 sklearn 會提示一下 Future Warning
# 我也把關掉了

import warnings
warnings.filterwarnings('ignore')
```

In [4]: # 使用 scikit-learn 提供的鳶尾花資料庫

```
iris = load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df["target"] = iris["target"]
df = df.drop(["petal width (cm)", "sepal width (cm)"], axis = 1)
df
```

Out[4]:

	sepal length (cm)	petal length (cm)	target
0	5.1	1.4	0
1	4.9	1.4	0
2	4.7	1.3	0
3	4.6	1.5	0
4	5.0	1.4	0
5	5.4	1.7	0
6	4.6	1.4	0
..
143	6.8	5.9	2
144	6.7	5.7	2
145	6.7	5.2	2
146	6.3	5.0	2
147	6.5	5.2	2
148	6.2	5.4	2
149	5.9	5.1	2

	sepal length (cm)	petal length (cm)	target
0	5.1	1.4	0
1	4.9	1.4	0
2	4.7	1.3	0
3	4.6	1.5	0
4	5.0	1.4	0
5	5.4	1.7	0
6	4.6	1.4	0
..
143	6.8	5.9	2
144	6.7	5.7	2
145	6.7	5.2	2
146	6.3	5.0	2
147	6.5	5.2	2
148	6.2	5.4	2
149	5.9	5.1	2

[150 rows x 3 columns]

使用 sklearn 的 LinearSVC 做出判斷，並且畫出邊界

SVM 也可以用來做迴歸：擬合出方程式來預測下一個連續的數值

> 迴歸: Support Vector Regression(SVR)

> 分類: Support Vector Classification(SVC)

有興趣的讀者可以往前翻一下，你可以看出 SVC 畫出的邊界和 Logistic Regression 畫出的邊界截然不同

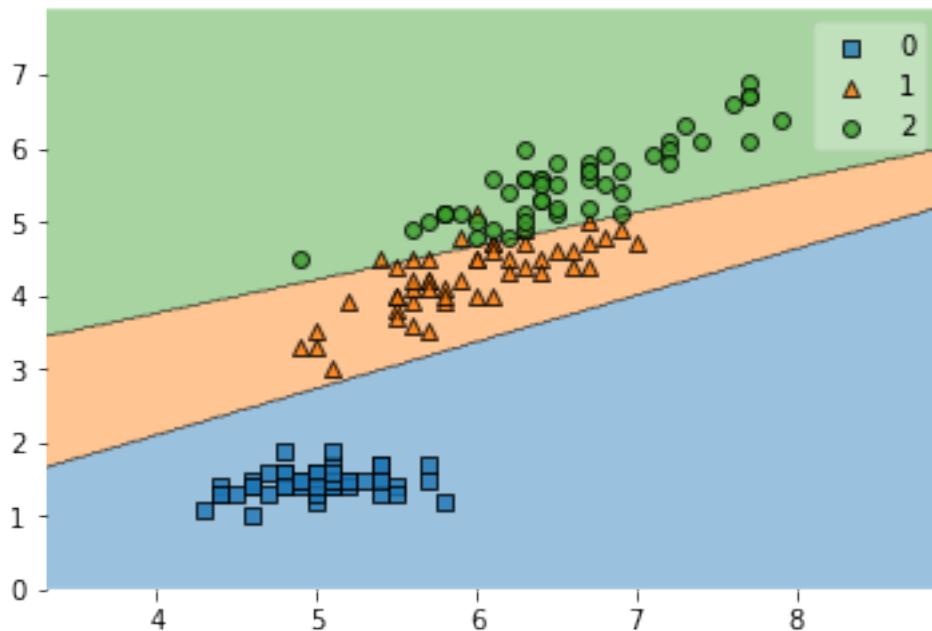


```
In [5]: from sklearn.svm import LinearSVC
        from sklearn.metrics import accuracy_score

        clf = LinearSVC()
        clf = clf.fit(df.drop(["target"], axis = 1), df["target"])

In [6]: plot_decision_regions(X=np.array(df.drop(["target"], axis = 1)),
                           y=np.array(df["target"]),
                           clf=clf)

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x10ae8b978>
```



1.4 非線性 SVM

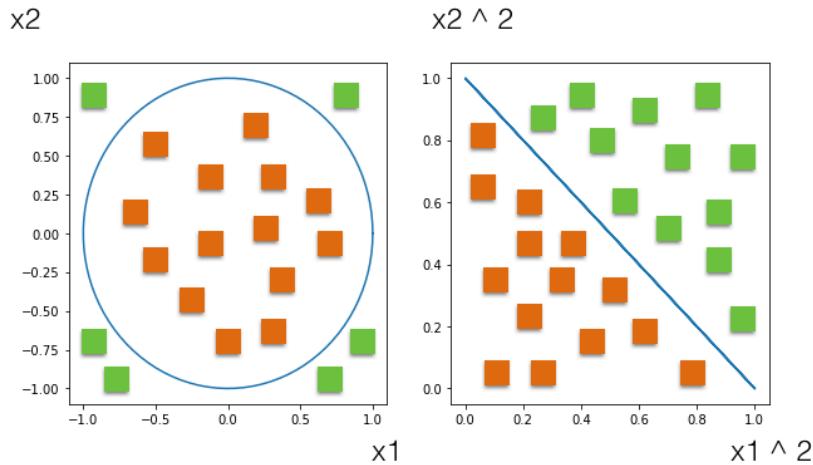
會了 S V M我們就可以開始解決剛剛非線性的問題了

剛剛說到，我們在 S V M解決非線性的問題使用的是“提高維度”的方法

我們先用一個最簡單的例子來簡單描述一下什麼是提高維度



1.4.1 例子 1



我們從左邊的圖看到我們的資料可以被一個二次曲線(圓)劃分

圓的曲線表示是 $x_1^2 + x_2^2 = r^2$ (r 為半徑)

用我們之前的線性分類器，當然行不太通

但是如果我們製造出兩個新的特徵 x_1^2 和 x_2^2

然後使用這兩個製造出來的特徵分類，你會發現！這兩個新的特徵可以簡單地用一個直線擬合

1.4.2 例子 2

我們用比較生活的方式再來跟你說一次提高維度

假設有一個情況是今天我們有兩個特徵來決定一個人會不會愛這個食物

1. x_1 : 食物的軟硬度 (越軟越不愛)
2. x_2 : 食物的臭味 (越臭越不愛)

但你驚訝的發現當兩個很不愛的東西加起來的時候，反而很多人很愛，這是什麼食物呢？臭豆腐！

那我們如果只用 x_1 和 x_2 這兩個特徵來看的話，只能用一個二次以上的曲線來擬合，但是如果我們製造出一個新的特徵叫做 $x_1 * x_2$

這個特徵也許就可以幫我們來辨別上面的 case(乘起來極度小的時候代表愛)

1.4.3 總結

你可以發現，上面我們只做了一件事情，利用已有的特徵製造新的特徵，但如果你把它當成資料的維度的時候，你可以把它想像成資料的維度增加了！(從兩個特徵變成三或四個特徵)

這就是所謂的增加維度!!!

那增加維度有什麼意義呢？



經由較為嚴謹的數學證明: 當維度增加的時候, 能夠線性劃分的機率會增加
用比較想當然爾的想法想: 當維度增加也就代表你在空間的分佈變廣了, 那就越有機會做出線性的劃分

注意, 這裡只是能夠劃分的機率增加, 並不是 100% 保證能線性劃分, 但就算不能完全劃分, 因為分佈變廣, 你的線性分割還是可以取得比較好的效果

1.5 核函數

剛剛我們已經大概敘述了我們解決非線性的核心思維了
但還有個問題沒解決, 我們轉換過的維度有時候實在太大了, 如果對每個轉換過的向量做 w^*x 來算分數的話計算量會非常非常的大
那怎麼辦呢?
我們先把之前的 $wx + b$ 的優化問題拿出來討論
我們在解這個問題的時候, 主要在解 w 向量, 但是在一個限制條件下, 就是不能有任何落在支援向量以內的空間
嚴格的證明你要將這優化問題轉化成『對偶問題』解決
但我們今天用比較直覺的想法來想這個問題

w 應該是根據你餵進去的訓練資料所決定的!!! 那就應該是跟你訓練資料的特徵和分類有關

所以我們大概得到下面的簡化

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\ &= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \end{aligned}$$

簡單來說 $w =$ 常數 (a) \times 每一筆資料的特徵 (x_i) \times 資料的分類 (正類或負類 y_i)
但 y 不是向量, 只是個常數, 所以我們可以把他拉出來, 最後就只剩 x_i 乘上 x (向量的乘法我們叫做內積)

所以對於你要判斷的新資料, 只要傳進來跟所有舊的資料做內積就結束了!

你會說! 疑, 我們的原始問題還沒解決啊! 維度的數目並沒有被縮小啊!

是的, 但如果你在這世界上能夠找到一個函式

$$k(x, y) = \Phi(x) \Phi(y)$$

指的是一個某種低維向高維的轉換



如果你能找到兩個轉換過的向量的內積 = 沒轉換過的向量的某種計算結果

上面的問題得到解決了!!! 因為所有的轉換過的內積計算都可以換成沒轉換過的向量計算
最後我們的式子變成

$$g(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b$$

那最後最後的疑問只剩，真的有這個函式嗎？有的話長怎樣呢？

- linear: $K(x_i, x_j) = x_i^T x_j.$
- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0.$
- radial basis function (RBF): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0.$
- sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r).$

這些都是我們可以使用的核函數，不過我們最常使用的一個核函數是 RBF 函數，原因是 RBF 是一個往無窮維度的轉換，所以能找出線性切割的機率較大

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

e 的展開是一個無窮的維度，所以 rbf 展開當然也是一個無窮的維度

最後根據上面的說明，你發現在使用 RBF 核函數的時候，兩個高維度向量的內積變成原維度向量的距離度量！

多麼美好的一件事啊！

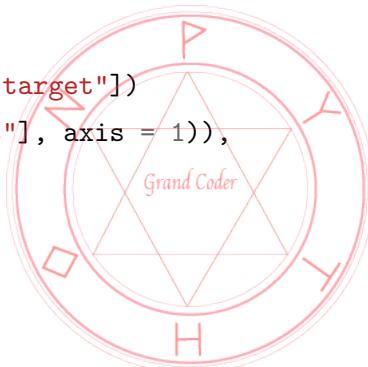
1.6 非線性 SVM 例子 1

我們先拿之前的鳶尾花的例子繼續來看看非線性的分類，你可以看到我的 kernel 參數選擇的是 rbf

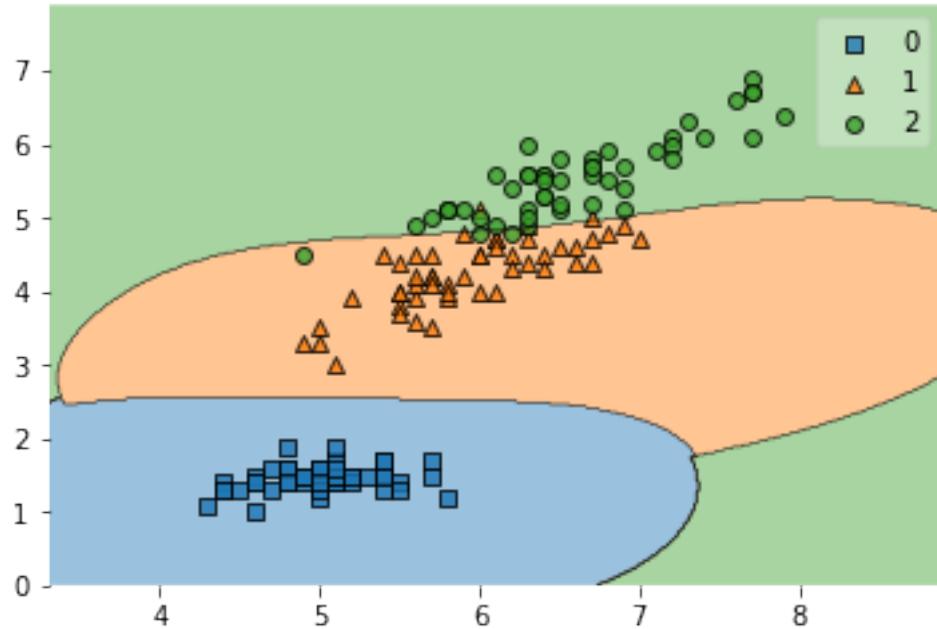
最後我們得到一個非線性的邊界！！！

In [7]: `from sklearn.svm import SVC`

```
clf = SVC(kernel="rbf")
clf = clf.fit(df.drop(["target"], axis = 1), df["target"])
plot_decision_regions(X=np.array(df.drop(["target"], axis = 1)),
                      y=np.array(df["target"]),
                      clf=clf)
```



Out [7]: <matplotlib.axes._subplots.AxesSubplot at 0x118186358>



1.7 非線性 SVM 例子 2

把我們上一節懸而未解的問題: XOR 問題拿出來，一樣選擇 rbf 核

看一下分類的效果

你可以看到我們完美的分出了 XOR 問題

```
In [8]: from numpy import random
        # 可以用 numpy 快速產生隨機，第一個參數是你產生有多少種類
        # 第二個參數是你要幾個
        # x1 是我們的第一特徵，你可以想像成帥
        # x2 是我們的第二特徵，你可以想像成有才華
        x1 = random.choice([True, False], 100)
        x2 = random.choice([True, False], 100)
        # y 是我們的 target，你可以想像成會不會喜歡
        y = np.logical_xor(x1, x2)
        df = pd.DataFrame(columns = ["x1", "x2", "y"])
        df["x1"] = x1
        df["x2"] = x2
```



```
df[\"y\"] = y
df = df.astype(int)
df
```

Out [8]:

	x1	x2	y
0	1	0	1
1	0	1	1
2	0	0	0
3	0	1	1
4	1	0	1
5	1	1	0
6	1	0	1
..
93	1	0	1
94	0	0	0
95	1	1	0
96	0	1	1
97	0	1	1
98	0	0	0
99	1	1	0

[100 rows x 3 columns]

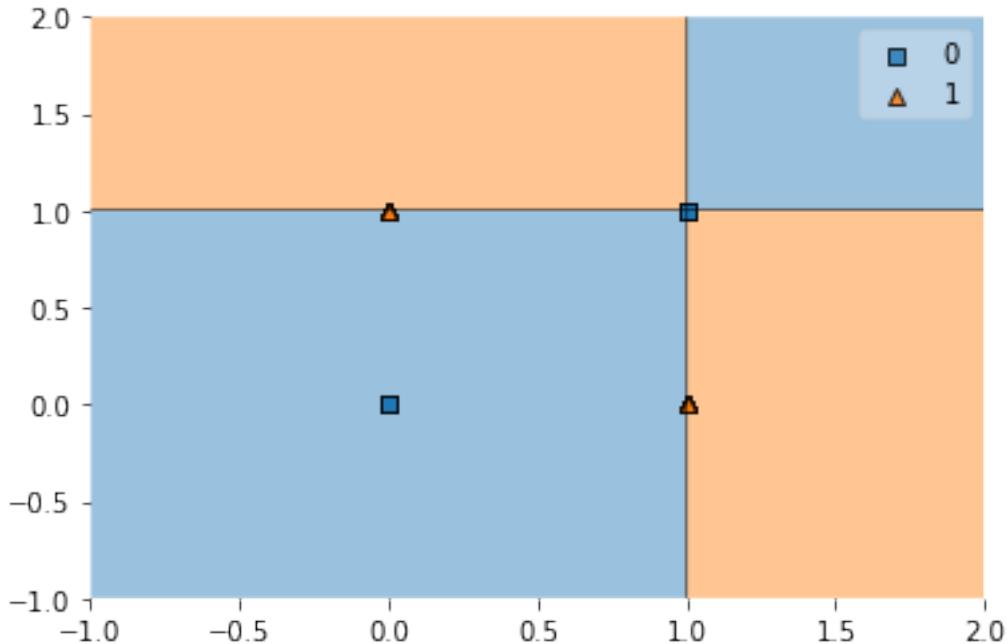
In [9]:

```
from mlxtend.plotting import plot_decision_regions
from sklearn.svm import SVC

clf = SVC(kernel="rbf")
clf = clf.fit(df.drop(["y"], axis = 1), df["y"])
plot_decision_regions(X=np.array(df.drop(["y"], axis = 1)),
                      y=np.array(df["y"]),
                      clf=clf)
```

Out [9]: <matplotlib.axes._subplots.AxesSubplot at 0x10c376358>





1.8 總結

當初非線性 S V M被提出的時候，造成一陣轟動，因為所有的問題都可以試著用非線性 S V M提出一個數學方程式來分類，而且分類的效果非常好，甚至不輸我們之前最常用的隨機森林，而 S V M也成為機器學習界使用好一陣子的主要演算法。但最後我們碰到了一個瓶頸了：抽象問題的分類，介紹到現在的演算法在分類抽象問題都有其極限！

究竟我們要怎麼解決比較抽象的問題呢？我們下一節繼續看下去！一起來進入深度學習的世界吧!!!



MLP

2018 年 7 月 18 日



1 深度學習基礎 - MLP

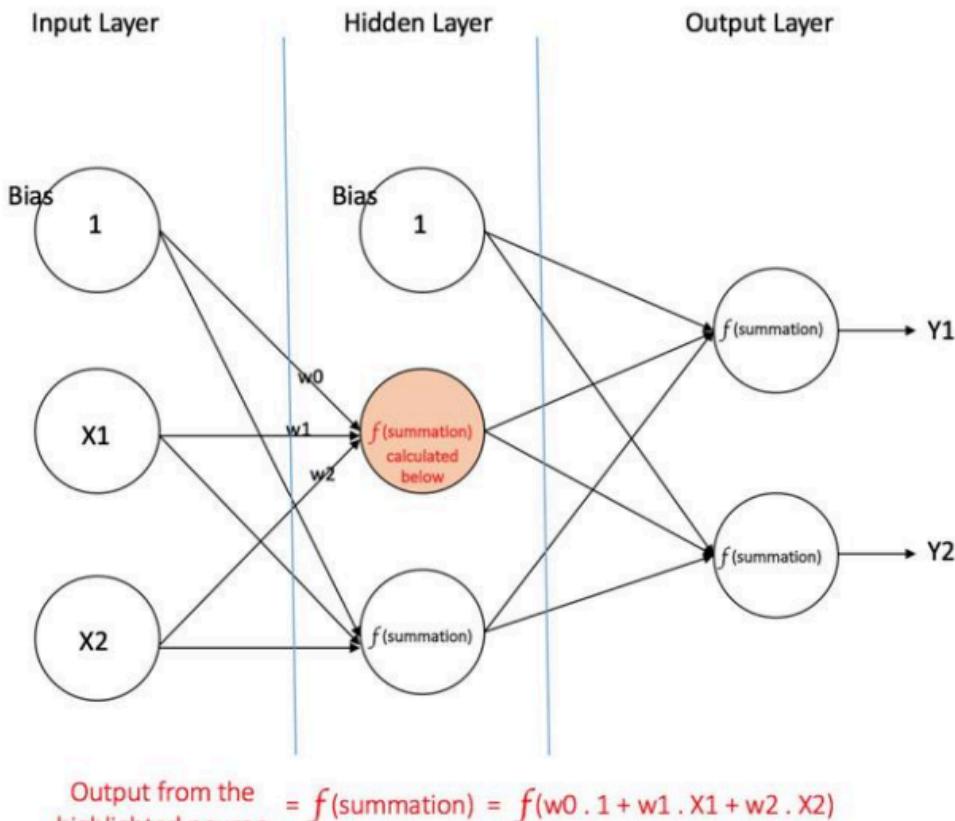
1.1 介紹

上一次我們介紹到了 SVM，利用核函數來模擬非線性的分類，當時還有另外一派的學者說我們何不讓神經元變成一個多層的結構，這樣整個模型會很類似我們人類的神經系統，而且加上一層層，自然就不會是一條直線到底，非線性的問題自然就不攻而破。



1.2 多層感知器 (Multi-layer Perceptron)

1.2.1 第一步: 緣起



這圖就是當初我們設想的多層神經元，每一層的輸入都會乘上一個權重 (w)，相加以後傳入下一層神經元，不過這裡要特別注意到的是，傳進去前我們還要經過一次的激勵函數 (Activation Function)，也就是我們之前提到的 Logistic Function，簡單來說，每次往下一層的傳遞就是決定下一個神經元是否應該被“激活”。

這個模型看似美好，但是第一個問題要被解決，就是這麼多的係數 (w) 應該怎麼被決定呢？

1.2.2 第二步: 梯度下降

我們先回到一個神經元來說一下梯度的概念

斜率各位應該沒有問題，我們常說一句話，斜率 = 0 的地方就代表我們的極大值或者極小值，那如果我換一句講法，把函數當成一個 2D 山谷，你的斜率就是你傾向走的方向，走到底點或者高點的時候自然就傾向留在那了，所以斜率 = 0，傾向不走。

那梯度只是把斜率擴充到高維度的概念，最大的差別就是!! 我們的梯度是一個向量，是有方向性的!! 再換句話說，我們現在的維度空間提高了，所以你在走的時候要說出你每個維度要走的量。

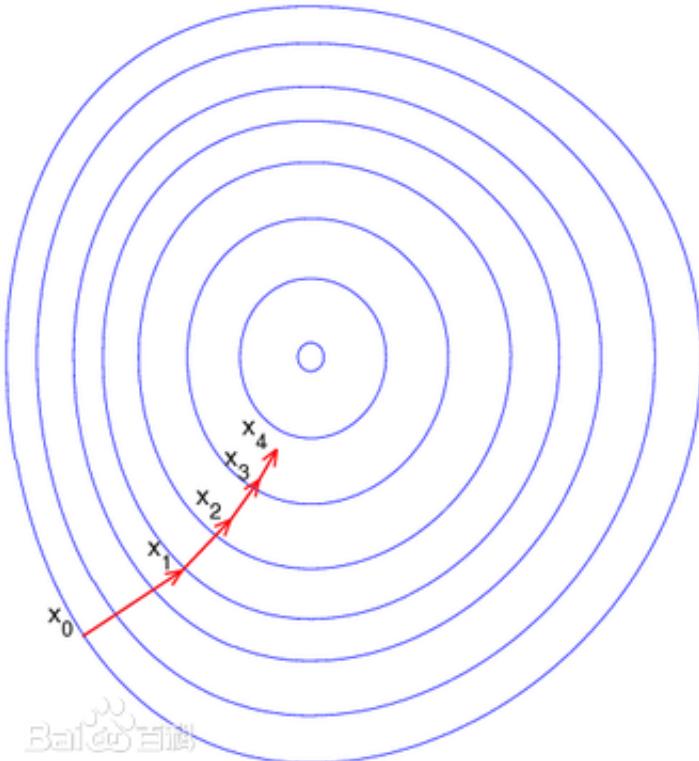
假設我們有 $f(x, y, z) \rightarrow$ 那我們走的傾向就是 (x 方向, y 方向, z 方向) = (對 x 偏微分, 對 y 偏微



分，對 z 偏微分)

$$\nabla \varphi = \left(\frac{\partial \varphi}{\partial x}, \frac{\partial \varphi}{\partial y}, \frac{\partial \varphi}{\partial z} \right)$$

回到一層的神經元，如何求出一層神經元的 w (權重) 呢？很簡單，假設把我們所有 w 的選擇和 Loss(跟目標的差距) 做出一個圖，我們開始擁有一個等高線圖了



假設比較裡面的圈圈是我們的損失最小，我們要往它前進，該如何做呢？

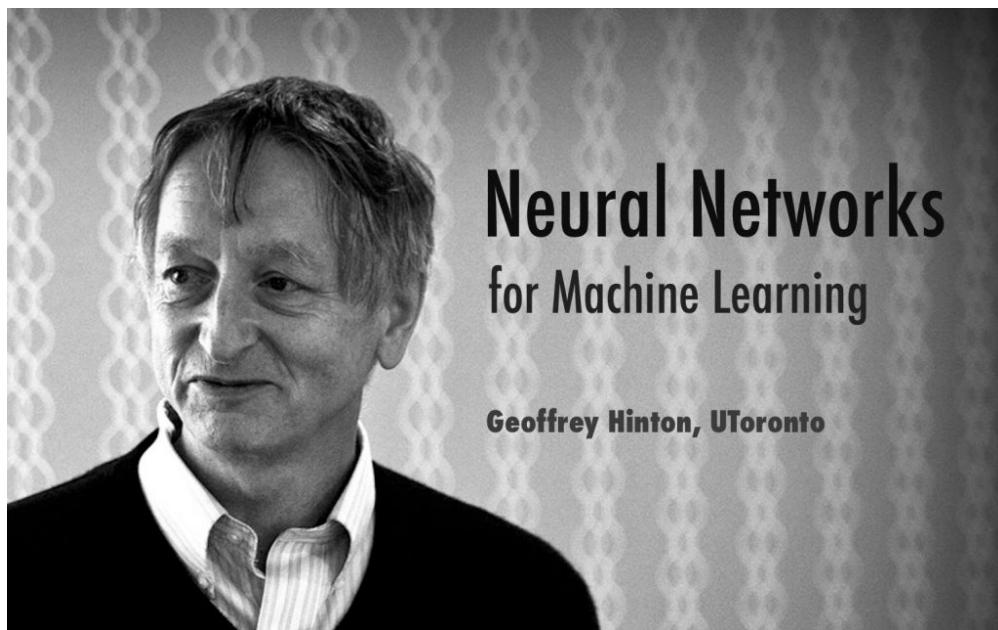
很簡單，隨便選個 w ，這時候你的損失一定不是最小，但是我們可以沿著“負梯度”的方向前進，我們就可以走到整個等高線的最低點，這個就叫做我們的踢度下降！

1.2.3 第三步：反向傳播

剛剛我們理解了一個神經元如何決定 w ，那怎麼決定多層的 w 呢？

這裡就不得不提到我們深度學習的鼻祖 Hinton 教授了





Hinton 教授在二十年前根據梯度的概念提出了誤差的反向傳播 (Backpropagation) 造成轟動
我們先聊聊簡單微分，假設今天我們有 $f(x) = 4x^2$ ，微分應該告訴你微分後 $f'(x) = 8x$
但是如果我換一個方法問你呢？

我們的 x 先經過 g 函數 $g(x) = 2x$ 再把整個 g 算出的結果丟進 $f f(g) = g^2$
這樣我如果 f 函數對 x 的微分是多少呢？

微分原理的鏈式法則告訴你 f 對 x 微分 = f 對 g 微分 * g 對 x 微分 *

我們先算 g 對 x 微分: 2 再來我們算 f 對 g 微分 = 2 g

所以 f 對 x 微分 = 4 g 囉 $g = 2x$, 一樣 = 8 x

為什麼我們會說到這呢？因為我們的多層神經元不就是這樣寫的嗎？
W: 權重加總函數
L: Logistic

$result = L2(W2(L1(W1(數據特徵))))$

利用上面的鏈式法則，你可以輕鬆算出你想要更新權重層的梯度

W1 層更新: result 對 W1 微分

W2 層更新: result 對 W2 微分

Hinton 教授在 20 年前提出誤差反向傳播後到今天我們都還在使用這方法來更新我們的權重，
但深度學習在 20 年前卻式微了，為什麼呢？

因為有一個嚴重的問題沒有被解決

你想想看上面的式子，我們總有一天會算到 Logistic 函數的微分，麻煩的是邏輯函數的微分你
可以看到上面的兩條基本上微分 = 0

中間的曲線微分後大概介於 0 ~ 1 之間

一層的神經元的時候，這個微分乘上去還沒有什麼

但你能想像多層的時候，我們就必須把這 0 ~ 1 之間的值乘上多次！



大概就是你每層的更新幅度到下一層都打個八折的感覺！總有一天你的更新幅度會趨近於 0
這就是我們所說的梯度消失，也就是最前面幾層的權重更新根本沒在動!!!
也因此當初大家都認為就算多層，我們也只能做到兩三層，兩三層我們還不如使用之前所說的
S V M，還可以達到更好的精確度！

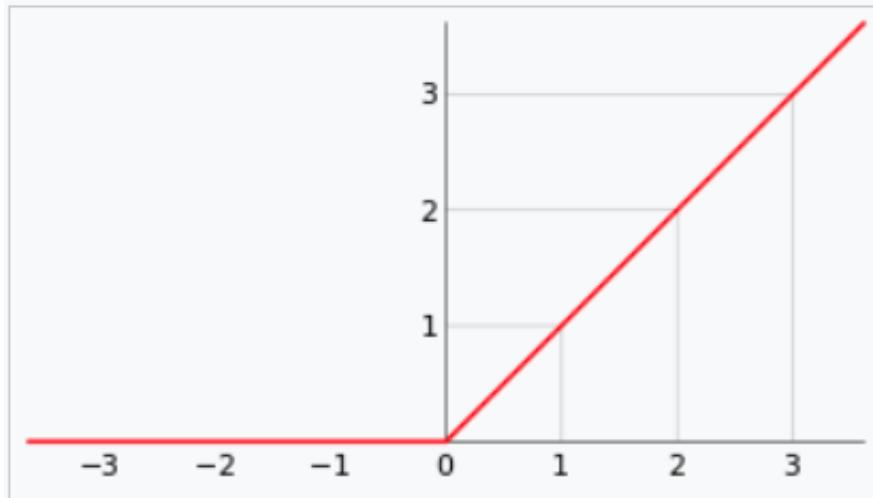
1.2.4 第四步：梯度消失的解決

Hinton 教授在經過十數年的深度學習黑暗期，始終沒有放棄他的多層神經網路的想法，終於，他提出了深度信念網路來解決問題。終於讓深度學習再度露出一絲曙光，站在他的肩膀上，我們再度開始探討如何解決梯度消失的問題，雖然今天我們已經不用 Hinton 教授當初提出的方法，但我們始終感謝 Hinton 教授這二十年對於深度學習的貢獻。

那我們今天怎麼解決梯度消失的問題呢？有兩種想法

1. 想法 1：讓你的激勵函數的微分等於 1 就不會打折了
2. 想法 2：打折就好像我們人在忘記東西，那不要讓事情太容易忘就好，也就是跨層的神經元連接

想法 1：微分等於 1 於是我們使用 relu 函數來替代 Logistic 函數



relu 簡單來說就是讓函數 (負數) = 0 函數 (正數) = 數自己

你可以看到在正數的範圍我們都是微分 1!! 解決

但是 relu 也是有幾個小問題

1. 激勵幅度的中心不是 0，不平均
2. 負類別微分永遠 =0，代表很多神經元不會更新，相當於死掉了！所以後續有一些改進，如 leaky relu

但是這些小問題對於他的表現影響很小!! 也是我們今天採用的重要方法!!



想法 2: 跨層連接 我們後續會在 LSTM(長短期記憶模型) 和 ResNet(進階的圖像辨識模型) 看到這種特別的想法

1.3 需要函式庫

機器學習的函式庫有很多, ex. Caffe, PyTorch, Keras, Theano... 等等, 我們的選擇如下:

1. TensorFlow: 請使用 PyCharm 或者 pip 安裝 tensorflow, 我們後續才會教你如何直接使用 tensorflow
2. Keras: 請使用 PyCharm 或者 pip 安裝 keras, 將底層實作再包一層, 讓你快速建立模型, 當然在彈性上會有些損失, 支援 TensorFlow 和 Theano 兩種底層

1.4 Step1. 資料預處理

這裡我們選用內建的 mnist 手寫數字資料庫來, mnist 提供共 70000 筆手寫數字, 而且用 keras 讀取的時候會直接幫你分成訓練和測試兩份資料

```
In [1]: import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense, Dropout
        import matplotlib.pyplot as plt
        %matplotlib inline
        # 我們會使用到一些內建的資料庫, MAC 需要加入以下兩行, 才不會把對方的 ssl 憑證視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context

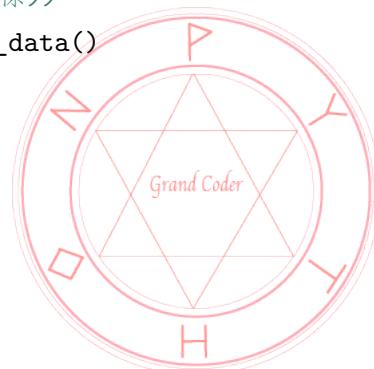
Using TensorFlow backend.
```

這裡讀取的時候比較不一樣, load_data 紿給你的是 tuple 中的 tuple, load_data 回傳值 ((訓練特徵, 訓練目標), (測試特徵, 測試目標))

所以你要接的時候必須再加上 () 來接裡面的 tuple

```
In [2]: # 回傳值: ((訓練特徵, 訓練目標), (測試特徵, 測試目標))
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
In [3]: print("訓練資料筆數:", len(y_train))
        print("測試資料筆數:", len(y_test))
```



訓練資料筆數: 60000

測試資料筆數: 10000

In [4]: `print("特徵的維度:", x_train.shape)`

特徵的維度: (60000, 28, 28)

稍微用 `dataframe` 看一下我們資料，我選擇其中的第一筆特徵來看，這是一個黑白圖像
所以你可以看到我們每張圖是 $28 * 28$ pixel，每個 pixel 最大值 255(白) 最小值 0(黑)

In [5]: `import pandas as pd`

為了顯示的漂亮，我刻意的把印出來的 `row` 只顯示 15 個和 `column` 只顯示 10 個
大家練習的時候可以去掉下面兩行

```
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)
pd.DataFrame(x_train[0])
```

Out [5]:

	0	1	2	3	4	...	23	24	25	26	27
0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	...	0	0	0	0	0
5	0	0	0	0	0	...	127	0	0	0	0
6	0	0	0	0	0	...	64	0	0	0	0
...
21	0	0	0	0	0	...	0	0	0	0	0
22	0	0	0	0	0	...	0	0	0	0	0
23	0	0	0	0	55	...	0	0	0	0	0
24	0	0	0	0	136	...	0	0	0	0	0
25	0	0	0	0	0	...	0	0	0	0	0
26	0	0	0	0	0	...	0	0	0	0	0
27	0	0	0	0	0	...	0	0	0	0	0

[28 rows x 28 columns]

這裡我們會習慣做標準化 (把所有的資料縮放到 0 - 1 間)，標準化的兩個原因



1. 希望所有特徵影響的幅度一樣，如果有個特徵區間明顯比別人大，那在梯度下降的時候就比較會傾向往它走，調整它
2. 如果你的特徵區間太大，一個梯度下降的步幅相對來說就比較大，很容易超過你的最低點

這裡我們標準化的原因主要是 2，不過建議你不管如何，都養成標準化的好習慣

另外記得將你的輸出做成 One-Hot Encoding，因為我們在使用深度學習的時候，最後一層可以是多個神經元

每個神經元就代表了一件事的正負，像我們現在的判定有 0 - 9，最後一層就可以是 10 個神經元，看誰被激發的幅度比較大，答案就是誰！

```
In [6]: from keras.utils import np_utils
        # reshape 讓他從 32 * 32 變成 784 * 1 的一維陣列
        # 除以 255 讓我們標準化到 0-1 區間
        x_train_shaped = x_train.reshape(60000, 784).astype("float32") / 255
        x_test_shaped = x_test.reshape(10000, 784).astype("float32") / 255
        # keras 要求你的分類輸出必須換成 One-hot 模式
        y_train_cat = np_utils.to_categorical(y_train)
        y_test_cat = np_utils.to_categorical(y_test)
```

取一個目標讓你看看 One-Hot Encoding 前後的改變

```
In [7]: print("One-hot 前:", y_train[0])
        print("One-hot 後:", y_train_cat[0])
```

One-hot 前: 5

One-hot 後: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

利用 `input` 讓你看看你想看的圖片，記得輸入完要按下 Enter 才可以繼續往下跑 Cell

```
In [8]: a = int(input("請輸入你想可視化的圖片 [0-59999]:"))
        print("你想可視化的圖片號碼是", a)
        print("圖片答案是", y_train[a])
```

請輸入你想可視化的圖片 [0-59999]:45

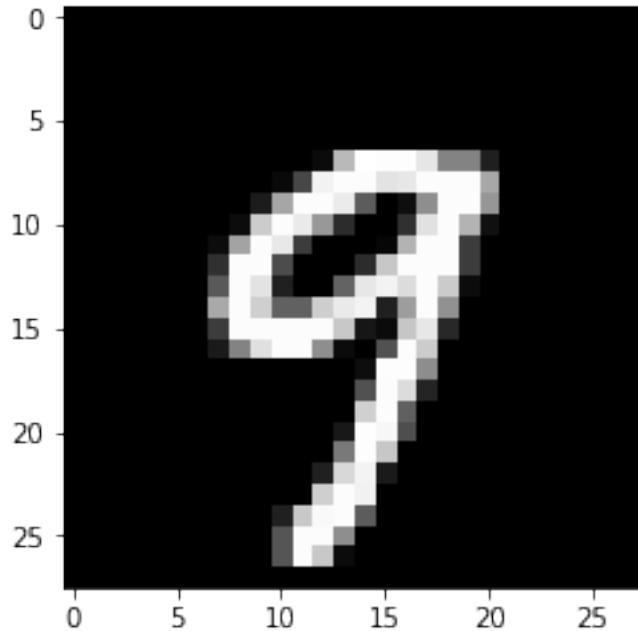
你想可視化的圖片號碼是 45

圖片答案是 9

```
In [9]: plt.imshow(x_train[a], cmap='gray')
```



Out [9]: <matplotlib.image.AxesImage at 0x1052cb7f0>



1.5 Step2. 建立模型

在建立模型的時候你必須選擇一個 Model 然後將你的一層一層神經元疊上去，我們接下來先解釋一下我們重要的選擇和參數設定

1.5.1 Model 選擇

Keras 的 Model 有兩種模型，Sequential 和 Functional，Sequential 使用上比較簡單，就一層一層 Layer 疊上去就可以，也可以滿足我們大部分的需要，Functional 的模型可以做出更多特別的設置，在使用上會比較彈性

1.5.2 Layer 選擇

我們現在只用到最簡單的一個層，叫做全連接層 (Dense Layer)，就是每一個神經元都會貢獻給下一層的神經元

這裡一個比較大的問題就是你的每一層應該有多少神經元，我們分三個部分討論

1. 輸入層 (最初層): 無庸置疑，你有多少個特徵就應該有多少神經元
2. 輸出層 (最後層): 無庸置疑，你有多少個分類要判斷就應該有幾個最後的神經元



3. 隱藏層(中間層): 這裡沒有公式，有的只有經驗法則，只能試出一個比較佳的值，還好前人已經給你一些經驗可以借鑑，見下圖的三種選擇你都可以先試試看再微調

$$\begin{aligned} m &= \sqrt{n+l} + \alpha \\ m &= \log_2 n \\ m &= \sqrt{nl} \\ m &\text{: 隱含层节点数} \\ n &\text{: 输入层节点数} \\ l &\text{: 输出层节点数} \\ \alpha &\text{: 1--10 之间的常数。} \end{aligned}$$

1.5.3 W 權重初始

因為我們要使用梯度下降，所以我們需要選擇一組隨機的權重開始，通常我會喜歡使用常態分佈 (random_normal) 來選擇我的初始權重

1.5.4 激勵函數

如同我們之前所說的，在層和層之間我們會選擇 relu 當我們的激勵函數，因為這樣才可以解決梯度消失的問題

問題在最後一層我們要選擇什麼激勵函數呢？以前我們選擇 sigmoid 也就是 Logistic 函數來激勵，Logistic 在單一的判斷的時候沒有問題

但在多個判斷的時候會有個小問題，就是所有的類別的判斷值加起來不等於 1，跟我們對於機率的認知不同 (所有機率加起來 = 1)

所以我們改使用 softmax 函數，長個跟 sigmoid 有 87% 像，但是加起來會 = 1

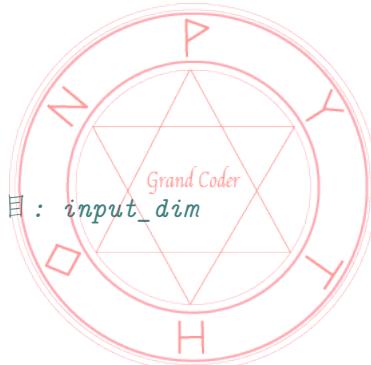
假設我們有兩個神經元，算出分數分別是 a 和 b， $\text{softmax}(a) = e^a / (e^a + e^b)$, $\text{softmax}(b) = e^b / (e^a + e^b)$

讀者可以輕易看出 $\text{softmax}(a) + \text{softmax}(b) = 1$

在這裡列出詳細的數學定義

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

```
In [10]: from keras.models import Sequential
        from keras.layers import Dense
        model = Sequential()
        # 一層隱藏層的模型，第一個隱藏層記得要寫出特徵的數目: input_dim
```



```

h_layer = Dense(units = 256,
                 input_dim = 784,
                 kernel_initializer = "random_normal",
                 activation = "relu")

model.add(h_layer)

o_layer = Dense(units = 10,
                 kernel_initializer = "random_normal",
                 activation = "softmax")

model.add(o_layer)

```

你可以畫出方塊圖 (記得要安裝 graphviz)

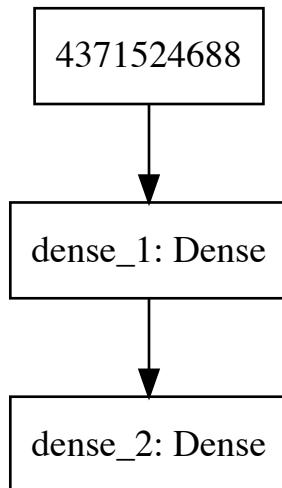
```

In [11]: from IPython.display import SVG
         from keras.utils.vis_utils import model_to_dot

         SVG(model_to_dot(model).create(prog='dot', format='svg'))

```

Out[11] :



更仔細的參數設定你可以藉由 `summary` 來看到，這裡解釋一個東西，權重 (Param) 的個數因為是全連接，所以至少需要

第一隱藏層: $784 * 256 = 200704$ 輸出層: $256 * 10 = 2560$

但是每一個輸出，應該要配置一個 bias 常數 ($f = wx + b$ 的 b 常數) 來讓你的激勵函數還是可以保持一樣，所以總數是

第一隱藏層: $784 * 256 + 256(\text{bias 個數}) = 200960$ 輸出層: $2560 + 10(\text{bias 個數}) = 2570$



In [12]: `model.summary()`

Layer (type)	Output Shape	Param #
<hr/>		
dense_1 (Dense)	(None, 256)	200960
<hr/>		
dense_2 (Dense)	(None, 10)	2570
<hr/>		
Total params: 203,530		
Trainable params: 203,530		
Non-trainable params: 0		
<hr/>		

設置完記得 `compile` 一下你的模型

`optimizer` 參數：梯度下降的方式，傳統的梯度下降有一個小缺點，就是你如果走太小步，要訓練很久，走太大步容易錯過最小值，所以我們做出動量的修改，也就是在可以走大步（你可以想像成斜率還很大的時候）的時候盡量大步，走小步的時候小步一點，這裡我們通常使用 `adam` 優化過的梯度下降就可以了

參考: <https://zhuanlan.zhihu.com/p/22252270>

`metrics` 參數：預設在輸出的時候只會告訴你 `loss` 是多少，你需要什麼量度的話要額外告訴他，我這裡告訴他的量度是正確率

In [13]: `model.compile(loss="categorical_crossentropy",
optimizer = "adam",
metrics = ['accuracy'])`

開始 `fit` 你的模型，這裡也解釋一下重要的參數

`validation_split`: 多少用來訓練，多少用來測試

`batch_size`: 我們使用的是梯度下降，所以現在的問題是，到底看過多少個樣本更新一次梯度，如果只看一個樣本就更新一次，會有亂走的問題，因為一個樣本跟你整體的趨勢差距太大了！如果看完全部樣本才更新一次，也會有一點問題，第一個問題是內存可能沒辦法一次載入全部樣本，所以計算時間會變得很大，第二個問題是全部樣本才更新一次，但你還是需要足夠的更新次數才能走到低點，假設你需要 10 次更新次數，那就是要把整個樣本看十次才能走到你的低點，這是一個非常非常大的計算量！所以我們做了一個折衷，我們一次看完很多個（但不是全部）就做一次權重更新，這裡我設置成看完 200 個樣本做出一次權重更新，建議你在這裡設置成 100~200 間的數值

epochs: 每一個樣本要看過幾次，這裡我設成 10，也就是整份樣本我必須看過 10 次，建議從 10~20 開始試

verbose: 0(不輸出訓練 log) 1(只輸出進度條) 2(每一個 epoch 輸出訓練信息)，我們通常就選擇 2

```
In [14]: train_history = model.fit(x = x_train_shaped, y = y_train_cat,
                                 validation_split = 0.1,
                                 epochs = 10,
                                 batch_size = 200,
                                 verbose = 2)
```

Train on 54000 samples, validate on 6000 samples

Epoch 1/10

- 1s - loss: 0.4229 - acc: 0.8873 - val_loss: 0.1857 - val_acc: 0.9503

Epoch 2/10

- 1s - loss: 0.1843 - acc: 0.9474 - val_loss: 0.1282 - val_acc: 0.9658

Epoch 3/10

- 1s - loss: 0.1290 - acc: 0.9632 - val_loss: 0.1025 - val_acc: 0.9700

Epoch 4/10

- 1s - loss: 0.0978 - acc: 0.9724 - val_loss: 0.0900 - val_acc: 0.9745

Epoch 5/10

- 1s - loss: 0.0772 - acc: 0.9786 - val_loss: 0.0797 - val_acc: 0.9760

Epoch 6/10

- 1s - loss: 0.0623 - acc: 0.9826 - val_loss: 0.0809 - val_acc: 0.9768

Epoch 7/10

- 1s - loss: 0.0517 - acc: 0.9852 - val_loss: 0.0773 - val_acc: 0.9773

Epoch 8/10

- 1s - loss: 0.0426 - acc: 0.9883 - val_loss: 0.0754 - val_acc: 0.9768

Epoch 9/10

- 1s - loss: 0.0346 - acc: 0.9913 - val_loss: 0.0705 - val_acc: 0.9790

Epoch 10/10

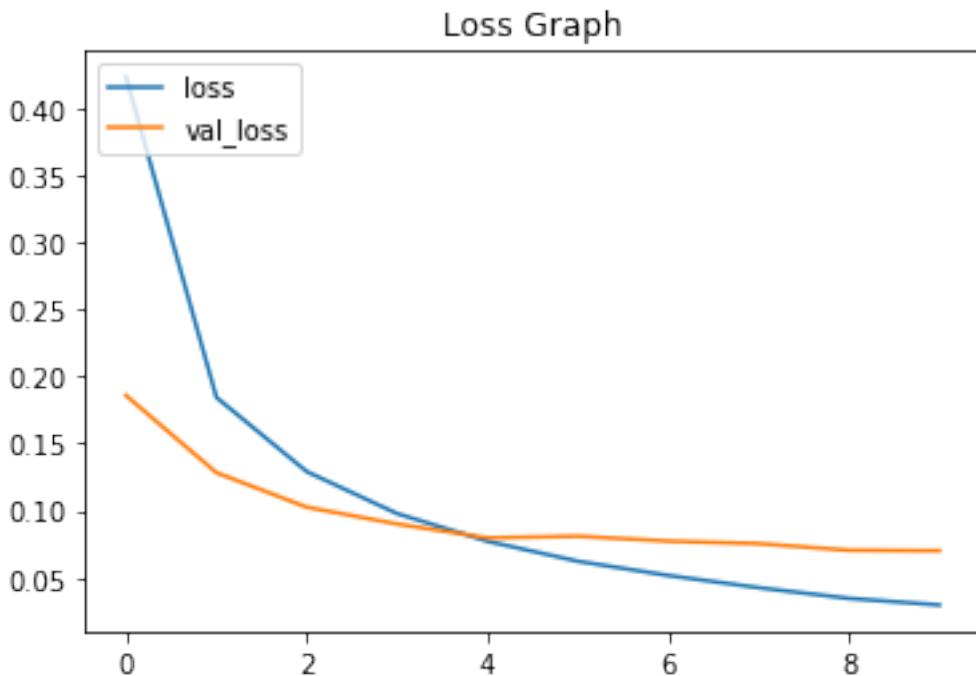
- 1s - loss: 0.0297 - acc: 0.9925 - val_loss: 0.0701 - val_acc: 0.9800

畫個很重要的圖，loss 和 val_loss 的圖，簡單來說，loss 是你的模型跟所有訓練資料（我們剛剛切出 9/10 訓練）正確答案的差距，val_loss 是你的模型跟你切出來的驗證資料（我們剛剛切出 1/10 驗證）正確答案的差距，別忘了我們在前面的課程跟你強調過“過擬合”的問題，就是過度訓練的意思，那通常我看 val_loss 沒什麼大的變動的時候（再訓練也對沒看過的資料沒啥幫助）我就會停

來了，去避免過度訓練，你可以看到其實我們的 10 epoch 是差不多的，val_loss 已經達到他能下降的極致了。

```
In [15]: plt.plot(train_history.history["loss"])
plt.plot(train_history.history["val_loss"])
plt.title("Loss Graph")
plt.legend(['loss', 'val_loss'], loc="upper left")
```

```
Out[15]: <matplotlib.legend.Legend at 0x1283c3be0>
```



你可以輸出預測的分類

```
In [20]: # 取五筆給你看
pre = model.predict_classes(x_test_shaped)
print("預測標籤:", list(pre[:5]))
print("正確標籤:", list(y_test)[:5])
```

預測標籤: [7, 2, 1, 0, 4]

正確標籤: [7, 2, 1, 0, 4]



```
In [18]: e = model.evaluate(x_test_shaped, y_test_cat)
    print("衡量係數:", e)
    print("正確率:", e[1] * 100, "%")
```

10000/10000 [=====] - 0s 17us/step

衡量係數: [0.07355162749246229, 0.9778]

正確率: 97.78 %

看看是什麼被分類錯，我們使用之前說過的混淆矩陣，你也可以使用之前說過 sklearn 的一些衡量唷！

```
In [22]: from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(y_test, pre))
```

	0	1	2	3	4	5	6	7	8	9
0	971	0	1	1	1	1	1	1	2	1
1	0	1119	4	0	0	1	2	2	7	0
2	5	1	1011	1	1	0	2	5	6	0
3	2	0	5	992	0	1	0	4	4	2
4	1	0	6	0	964	0	2	1	0	8
5	4	0	0	14	1	864	4	0	2	3
6	4	2	1	1	4	3	941	0	2	0
7	2	3	10	3	0	0	0	1002	4	4
8	4	0	5	4	4	2	1	3	949	2
9	3	4	0	8	15	3	0	6	5	965

列標籤是正確的標籤，行標籤是你的預測

你可以看到 9 由於和 7 和 4 長得非常像，所以被判斷錯的機率就還蠻高的！0 長得獨樹一幟，所以分類錯誤的機率真的還蠻小的！



SimpleCNN

2018 年 8 月 3 日



1 圖像的深度學習 - CNN

1.1 介紹

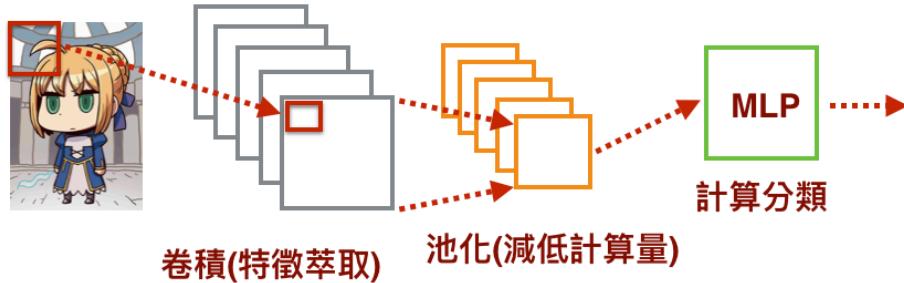
儘管之前已經使用了 MLP 來辨識簡單的手寫圖像，但是如果來到複雜的圖像，我們發現兩個問題

1. 把每一個像素平鋪開以後，在全連接下，要訓練的參數太多，要花費超級多時間訓練
2. 圖片素材取得不易，很難建立大量的資料

所以我們針對這兩個問題進行改進，發展成另外一種針對圖像的深度學習演算法：卷積神經網路 (Convolutional Neural Network)



1.2 CNN 理論基礎



CNN 跟之前唯一不同的就是加入卷積和池化來解決上面的兩個問題

1.2.1 卷積層

我們先來解決第二個問題: 圖片素材取得不易, 難以建立大量的資料

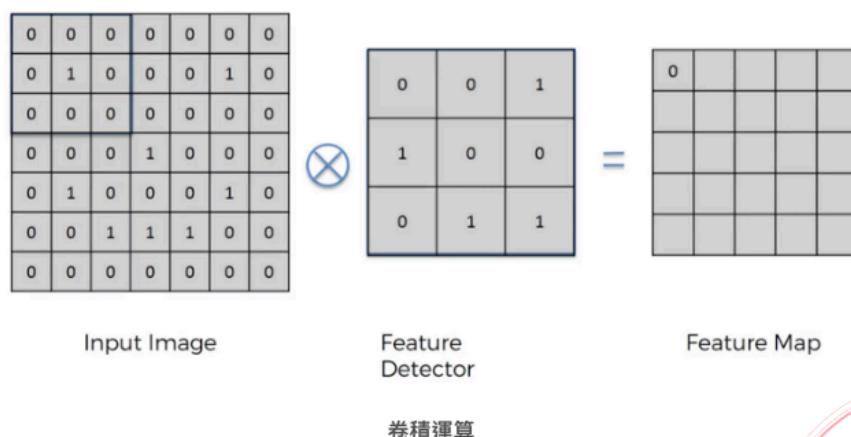
事實上, 這個問題起因應該是我們對於整個圖片的使用率太低

什麼意思呢? 一個圖片事實上應該有很多可以看的地方, 譬如: 顏色, 外框之類的等等

但是在我們之前的 M L P , 我們並未做出這些特徵的抓取, 而是直接暴力的把所有像素進去

這裡為了讓圖片的應用效果最大化, 我們利用 filter 的概念, filter 的意思就是把原始圖片經過某些流程, 只留下我們感興趣的特徵 (ex. 邊角, 外框...之類)

有了這些特徵圖, 我們就可以改用這些特徵圖來做我們的分類, 一個具體的想像就是把圖片擴展成跟我們之前不管是鳶尾花還是鐵達尼號的分類一樣, 擁有許多許多的特徵, 利用這些特徵來分類



我們會選一個特徵圖 (filter)(ex. 可能是一個人臉特徵...之類) 不斷的一個一個像素平移, 檢查那個區域有沒有我們想要的特徵



上面的檢測 = 每一個像素相乘加起來 = $0 \times 0 + 0 \times 0 + 0 \times 1 + 0 \times 1 + 1 \times 0 + 0 \times 0 + 0 \times 0$
 $+ 0 \times 1 + 0 \times 1 = 0$
= 沒有相關特徵在這區域

透過很多的特徵圖，我們就可以得到很多不同的特徵！最大化一張圖訓練的效益

1.2.2 池化層

池化是為了解決第一個問題，計算量太大，那怎麼讓計算量減小呢？

就是圖片的壓縮和縮小，簡單來說，假設我有一個 2×2 的像素區域，假設是

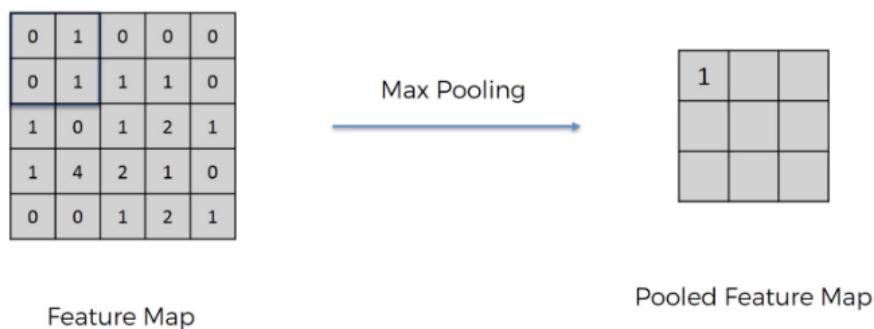
$$\begin{array}{r} \overline{0 \quad 1} \\ \hline 0 \quad 4 \quad 1 \\ 1 \quad 1 \quad 0 \\ \hline \end{array}$$

我們如果把他壓縮成 1×1 像素，取最大值(4)，會不會失真太多呢？答案是不會的，只要區域不要太大，理論上是不會失真太多的

池化通常有兩種：

1. Max Pooling(最大值池化): 取一個區域的最大值
2. Mean Pooling(平均值池化): 取一個區域的平均值

我們在這裡用最簡單的最大值池化就好了



1.3 CNN 架構

現在的問題是，我們究竟要幾層的卷積和池化呢？池化的 Size 應該多少呢？Filter 的數量應該是多少個呢？

事實上在這裡完全沒有式子可以告訴你最佳答案

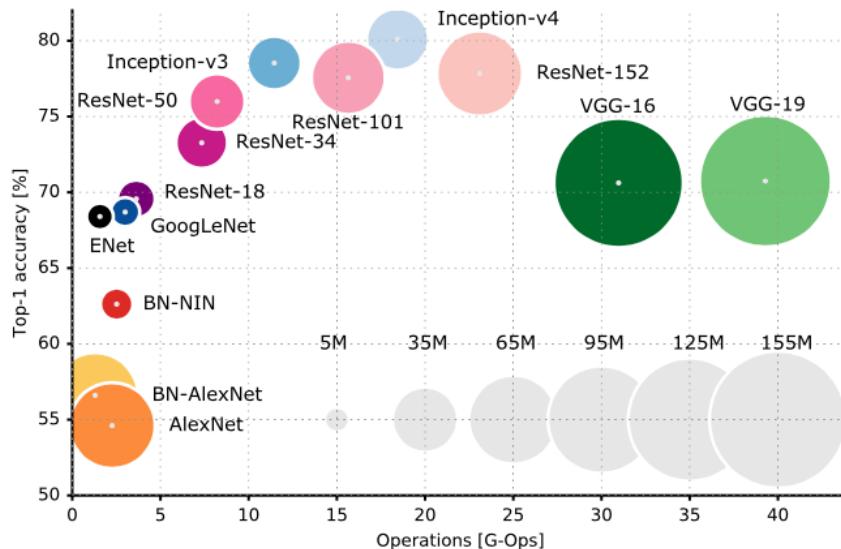
我們也很難自己試出最佳答案到底是多少，所以我們在圖片分類的時候通常會做 ~~order~~ 一件事：參考別人的架構！



Imagenet 是 Hinton 和他的學生建立起來的一個龐大圖像資料庫，許多大公司都會在上面驗證他們的演算法好壞

每年還會舉辦比賽，參加的是全球的科技巨頭

在這些年的比賽中，出現了許多非凡的演算法，也就是下面的這些



橫軸的 Ops 你可以想成計算量，縱軸的 Top-1 Accuracy 是指說這模型只猜一個類別，正確率是多少

偶爾我們會看到 Top-5 Accuracy，指的是模型可以猜五個他覺得最可能的猜測，只要有一個對，就算對

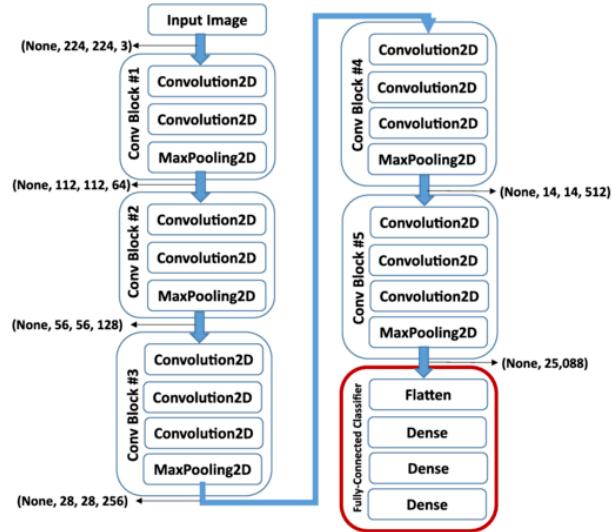
1.3.1 VGG-16

VGG-16 是第一個展露頭角的 CNN 網路，而且是非常傳統的 CNN 網路，只用卷積以及池化和全連結層構成了整個架構

今天我們就參考 VGG-16 網路來建立我們的 CNN 網路

這裡我們先來一個簡化的圖





我們再來討論一下卷積和池化

首先，我們進來的圖片是 (224, 224, 3) 三個維度 -> 224 像素 (寬) x 224 像素 (高) x 3(RGB 三通道)

再論卷積 你把 RGB 當成原圖的三個特徵，卷積其實就是特徵的萃取和再製，所以卷積只會去改變第三個維度 (在這裡就是 3)

而不會改動到寬和高這兩個維度，VGG-16 的卷積基本就是每一次的卷積把特徵數 * 2

從 3, 128, 256 到最後的 512

再論池化 池化剛好相反，他是去取像素的最大值 (最大池化)，所以反而一定不會改到第三個維度，只會更改到前面的寬和高

這裡從 224, 112, 56, 28 到最後的 14

全連接 做完特徵的萃取和計算量的減低，一樣攤開經由我們的 MLP 開始判斷



Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
dense_1 (Dense)	(None, 16)	16016
<hr/>		
Total params: 138,373,560		
Trainable params: 138,373,560		
Non-trainable params: 0		

完整參數

1.4 簡化版 VGG-16

在這裡我們不可能從頭開始訓練 VGG-16 的所有參數，所以我們先取前面的幾層來訓練得到一個初步的印象

在實務上，有時候我們為了時間考慮，會拿取已經訓練好的模型，再 fine-tune 其中幾層的參數

理論上，有了基礎的分辨，稍加訓練，雖然不及從頭訓練來得好，但是已經有還不錯的分辨能力了

1.5 Step1. 資料預處理

這裡我們選用 cifar10 圖像資料庫來，cifar10 提供共 60000 筆較為小的圖片，並且總共有 10 種分類

你甚至可以把它當成不清晰版的 imagenet 了

In [1]: `from keras.datasets import cifar10`



```
# MAC 一定要加入此行，才不會把對方伺服器的 SSL 證書視為無效
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

import matplotlib.pyplot as plt
%matplotlib inline

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Using TensorFlow backend.

把測試資料和訓練資料印給你看，訓練資料是 50000 萬筆的 32×32 的 RGB 圖片

```
In [2]: print(x_train.shape)
print(x_test.shape)

(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

十種類別分別如下，我先創造出來，等等可以把圖片的類別印給你看

```
In [3]: label = {0:"飛機", 1:"車", 2:"鳥", 3:"貓", 4:"鹿",
5:"狗", 6:"青蛙", 7:"馬", 8:"船", 9:"卡車"}
```

利用 input 你可以決定你要秀出哪張照片

```
In [4]: a = int(input("請輸入你想可視化的圖片 [0-49999]:"))
print("你想可視化的圖片號碼是", a)
print("圖片答案是", label[y_train[a][0]])
plt.imshow(x_train[a])
```

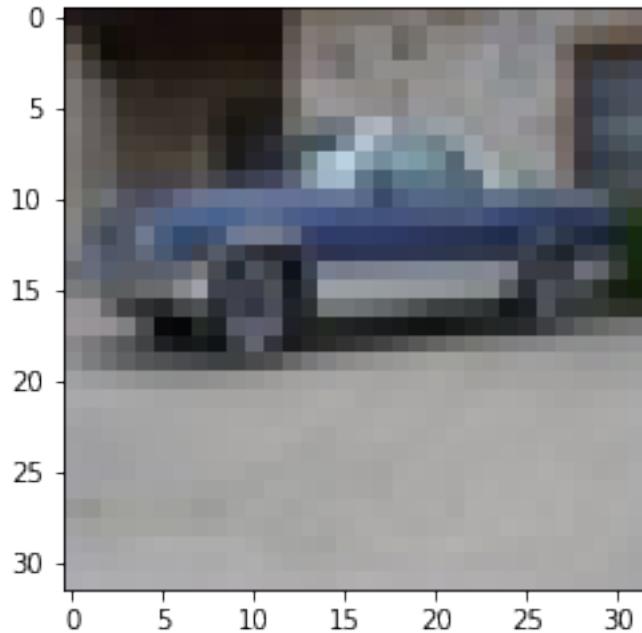
請輸入你想可視化的圖片 [0-49999]:10000

你想可視化的圖片號碼是 10000

圖片答案是 車

```
Out[4]: <matplotlib.image.AxesImage at 0x124a0c630>
```





在上一張我們提過，使用深度神經網路的一個好習慣就是標準化，這樣對於我們的 w 的更新會比較好！

另外一個你需要做的是，如果判斷是多個類別，請把它轉成 One-hot Encoding 形式

```
In [5]: from keras.utils import np_utils
x_train_shaped = x_train.astype("float32") / 255
x_test_shaped = x_test.astype("float32") / 255
y_train_cat = np_utils.to_categorical(y_train)
y_test_cat = np_utils.to_categorical(y_test)
```

1.6 Step2. 建立模型

這裡我們一樣使用 Sequential 模型

但我們開始用三種不同的 Layer

1.6.1 Conv2D (卷積層)

我們使用 Conv2D 層來卷積，要設定的最重要參數是 filters，我們這裡參照了上面的 VGG16 並做了一點小小的簡化，我們只做了兩次的卷積，並且第一次只擴充到 32 通道，kernel_size 則是你的過濾器的寬和高，我們設值成 (3, 3)，也就是跟上面的介紹一樣的過濾器寬高

啟動函數跟 MLP 一樣，只要在中間層我們就會選擇 relu，避免梯度的消失



原來我們的 filter 也是透過訓練 w 決定的!!!

這正是深度神經網路和之前的不一樣，我們的過濾器不再是提早決定好了，而是透過深度神經網路的訓練來決定

Padding 的設置是為了讓卷積後的寬高保持不變 (最後一次的卷積會只遇到 1 寬度，我們會幫他 Padding 成 2 寬度)

1.6.2 MaxPooling2D (最大池化層)

我們選擇池化窗是 (2, 2) 的話也就意味著 2 個寬度取一個最大值，2 個高度取一個最大值就相當於本來的寬高 $(w, h) \rightarrow$ 池化後 $\rightarrow (w/2, h/2)$

1.6.3 Dropout (Dropout 層)

在訓練中，我們很怕遇到過擬合的情況，那在我們的神經網路裡，怎麼防止模型對於資料過擬合呢？

Hinton 提出了一個很簡單但卻非常有效的方法，就是每一次在訓練的時候，不要用全部的神經元來訓練

而是隨機的斷開一些神經元 (斷開不代表消失，下次斷開的不一定是他)，也就是我們每次訓練的模型稍微都有所不同 (很像隨機森林裡的每棵樹對吧)

這樣就會自然的不這麼擬合了！

依照前人的經驗，通常在這裡 drop 掉 25%(0.25)~50%(0.5) 個神經元會是一個不錯的選擇

```
In [1]: from keras.models import Sequential
        from keras.layers import Dense, Dropout, Activation, Flatten
        from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D

model = Sequential()

# 第一次卷積和第一次池化
model.add(Conv2D(filters=32,
                  kernel_size=(3, 3),
                  input_shape=(32, 32, 3),
                  activation='relu',
                  padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 斷開 25% 的連接
# 並且加入第二次卷積和第二次池化
```



```

model.add(Dropout(0.25))
model.add(Conv2D(filters=64,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 把你處理過的東西攤開成為一維
model.add(Flatten())
model.add(Dropout(rate=0.25))

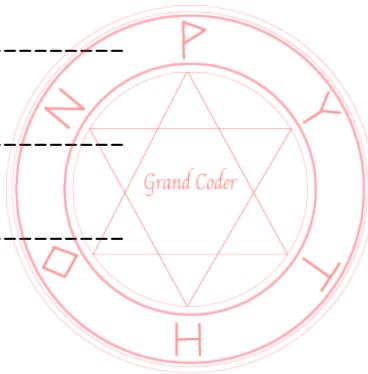
# 全連接層
model.add(Dense(128, activation='relu'))
model.add(Dropout(rate=0.25))

model.add(Dense(10, activation='softmax'))
model.summary()

```

Using TensorFlow backend.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dropout_2 (Dropout)	(None, 4096)	0



```

dense_1 (Dense)           (None, 128)      524416
-----
dropout_3 (Dropout)       (None, 128)      0
-----
dense_2 (Dense)           (None, 10)       1290
=====
Total params: 545,098
Trainable params: 545,098
Non-trainable params: 0
-----
```

1.6.4 Param 個數

我們一起看看上方 summary 的參數個數

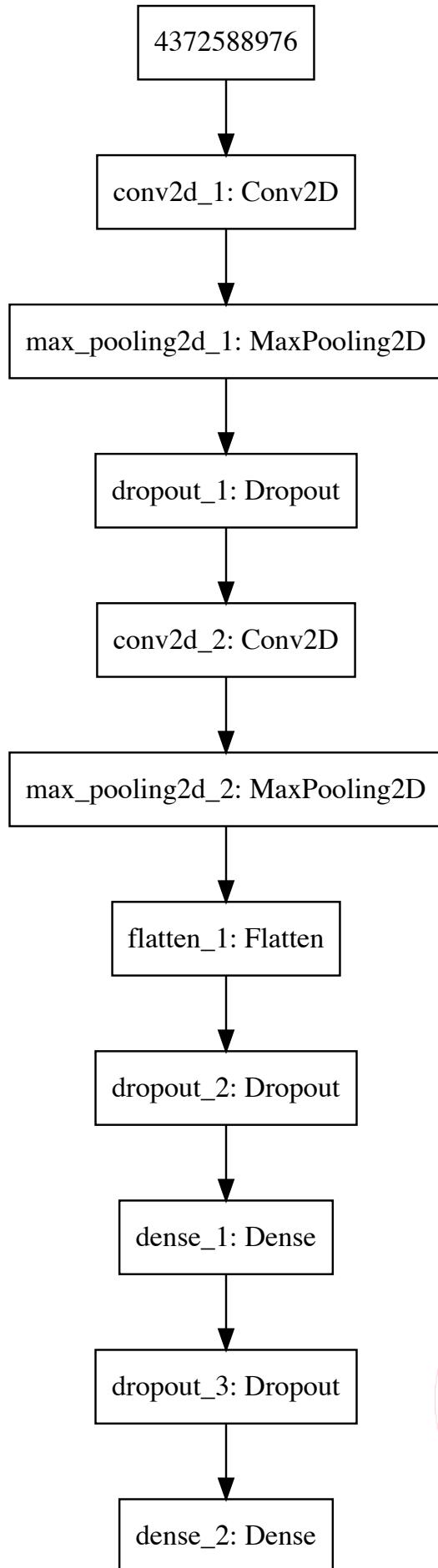
- 卷積層 1 的 896: 32 個濾波器，每一個 $3(\text{寬度}) \times 3(\text{高度}) \times 3(\text{input 通道 RGB}) + 1(\text{bias}) = 28$ 個參數， $32 \times 28 = 896$ 個參數
- 卷積層 2 的 18496: 64 個濾波器，每一個 $3(\text{寬度}) \times 3(\text{高度}) \times 32(\text{input 通道數}) + 1(\text{bias}) = 289$ 個參數， $64 \times 289 = 18496$ 個參數
- 全連接層的 524416: $4096 \times 128 + 128(\text{bias}) = 524416$ 個參數

```
In [7]: from IPython.display import SVG
        from keras.utils.vis_utils import model_to_dot

        SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[7] :





1.6.5 開始訓練

你可以看到我們 10 個 epoch 大概花了 10 分鐘訓練，最後 val_loss 差不多已經趨於平穩，所以其實 10 個 epoch 已經差不多了

```
In [8]: model.compile(loss="categorical_crossentropy",
                      optimizer = "adam",
                      metrics = ['accuracy'])

train_history = model.fit(x = x_train_shaped, y = y_train_cat,
                           validation_split = 0.1,
                           epochs = 10,
                           batch_size = 128,
                           verbose = 2)
```

Train on 45000 samples, validate on 5000 samples

Epoch 1/10

- 57s - loss: 1.7013 - acc: 0.3832 - val_loss: 1.3603 - val_acc: 0.5134

Epoch 2/10

- 53s - loss: 1.3428 - acc: 0.5182 - val_loss: 1.1724 - val_acc: 0.5946

Epoch 3/10

- 54s - loss: 1.2093 - acc: 0.5706 - val_loss: 1.1136 - val_acc: 0.6146

Epoch 4/10

- 52s - loss: 1.1301 - acc: 0.5984 - val_loss: 1.0042 - val_acc: 0.6474

Epoch 5/10

- 58s - loss: 1.0636 - acc: 0.6219 - val_loss: 0.9457 - val_acc: 0.6702

Epoch 6/10

- 55s - loss: 1.0149 - acc: 0.6396 - val_loss: 0.9024 - val_acc: 0.6942

Epoch 7/10

- 53s - loss: 0.9735 - acc: 0.6567 - val_loss: 0.8784 - val_acc: 0.7012

Epoch 8/10

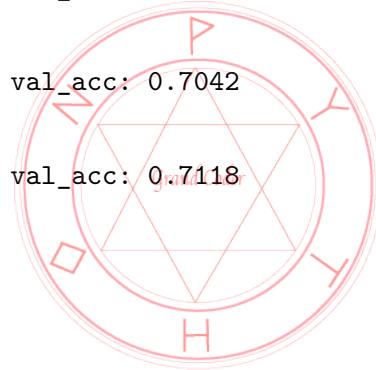
- 52s - loss: 0.9385 - acc: 0.6712 - val_loss: 0.8536 - val_acc: 0.7104

Epoch 9/10

- 53s - loss: 0.9075 - acc: 0.6806 - val_loss: 0.8530 - val_acc: 0.7042

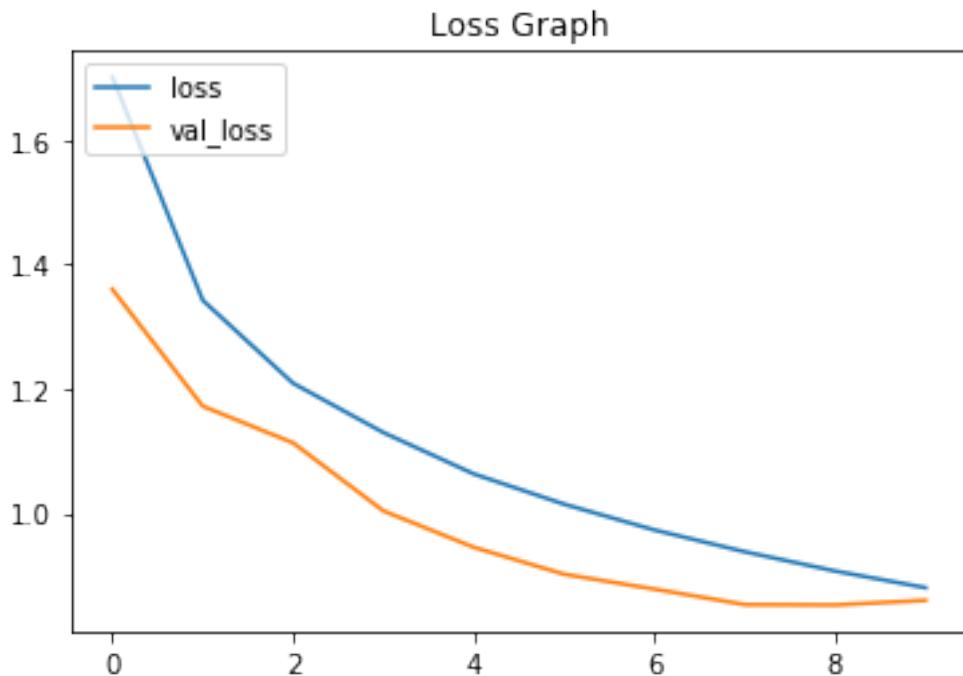
Epoch 10/10

- 68s - loss: 0.8807 - acc: 0.6892 - val_loss: 0.8605 - val_acc: 0.7118



```
In [9]: plt.plot(train_history.history["loss"])
plt.plot(train_history.history["val_loss"])
plt.title("Loss Graph")
plt.legend(['loss', 'val_loss'], loc="upper left")
```

Out [9]: <matplotlib.legend.Legend at 0x150e054a8>



List 的第二個是我們的正確率，你可以看到正確率大概是 70% 左右，以這麼簡單的卷積網路來說已經不錯了

In [10]: model.evaluate(x_test_shaped, y_test_cat)

10000/10000 [=====] - 5s 520us/step

Out [10]: [0.8914391684532166, 0.6947]

1.7 Step3. 儲存模型

我們可以藉由 `save` 直接儲存成 `hdf5` 類型的檔案，`hdf5` 是一種資料庫形式，他會以階層式(就像我們電腦的資料夾)來儲存你的資料，你的一個一個儲存的資料就會像電腦裡的檔案位於不同的資料夾，之後你就可以使用 `load` 把你當初訓練好的所有參數載入回來



In [11]: `model.save('cnn1.h5')`

我們可以藉由軟體 hdfview 來開啟 hdf5 檔案格式

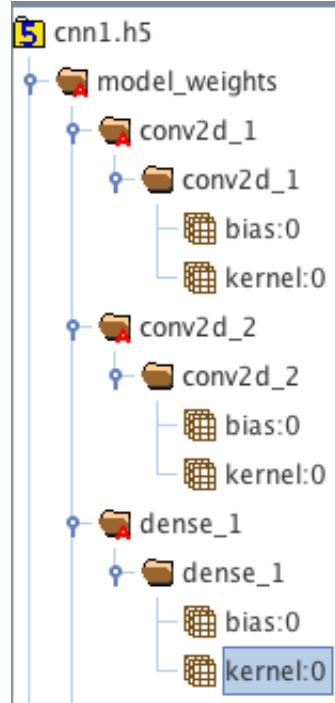
<https://support.hdfgroup.org/products/java/release/download.html>

請在這裡下載



以下展示一下我開啟 `cnn1.h5` 檔案的畫面

我們可以看到每一層都被儲存下來了



點進去你可以看到確實訓練出來的參數都被儲存了



kernel:0 at /model_weights/dense_1/dense_1/ [cnn1.h5 in /Users/Elwi

Table

	0	1	2	3	4
4030	0.000442...	-0.01749...	-0.05094...	-0.02302...	-0.0051...
4031	-0.00340...	0.055477...	-0.00523...	-0.04247...	-0.01333...
4032	0.016113...	0.021479...	0.026292...	-0.00316...	0.014386...
4033	-0.01049...	-0.01994...	0.004785...	0.014793...	0.008700...
4034	0.020475...	-0.03334...	0.004371...	0.002015...	0.029571...
4035	0.030232...	-0.02547...	-0.00751...	0.024257...	0.009110...
4036	-0.00291...	0.060014...	0.004878...	-1.24107...	-0.03728...
4037	0.034150...	0.020552...	-0.00290...	-0.01285...	-0.02622...
4038	0.007243...	-0.06513...	-0.03890...	0.004310...	0.009421...
4039	-0.03771...	0.073945...	0.035994...	-0.01278...	-0.00329...
4040	-0.02035...	0.009032...	-0.00185...	0.002331...	0.015627...
4041	-0.00887...	-0.14406...	-0.03788...	-0.03566...	0.007929...
4042	0.036383...	-0.21050...	0.008463...	0.019900...	0.007325...
4043	-0.01477...	0.042035...	-0.00531...	0.017164...	-0.03203...
4044	-0.00780...	-0.02041...	0.007992...	1.789570...	-0.03511...
4045	-0.00716...	-0.02020...	0.026667...	-0.05026...	-0.01205...
4046	-0.00755...	-0.10623...	-0.01257...	0.020442...	-0.00273...
4047	0.015341...	-0.08324...	-0.01765...	-0.00374...	-0.00394...
4048	-0.00733...	0.068243...	-0.02780...	0.010626...	0.027988...
4049	-0.01486...	0.056750...	-0.01469...	-0.02104...	-0.02754...

1.8 結語

我們已經完成了第一個簡單的 CNN，但要說深度，是完全完全不夠的，我們下一章繼續增加深度看一下我們的模型的轉變



CNN-Copy1

2018 年 8 月 3 日



1 再論 CNN

1.1 介紹

上一章節我們已經試過了超簡化版的 VGG-16，我們再來加上一些深度，看看加上深度到底對我們整個模型的影響為何

```
In [1]: from keras.datasets import cifar10
        # MAC 一定要加入此行，才不會把對方伺服器的 SSL 證書視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
        import matplotlib.pyplot as plt
        %matplotlib inline

        (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Using TensorFlow backend.

一樣看看 shape

```
In [2]: print(x_train.shape)
        print(x_test.shape)
```



```
(50000, 32, 32, 3)
```

```
(10000, 32, 32, 3)
```

```
In [3]: label = {0:"飛機", 1:"車", 2:"鳥", 3:"貓", 4:"鹿",
5:"狗", 6:"青蛙", 7:"馬", 8:"船", 9:"卡車"}
```

可視化你想看的圖片

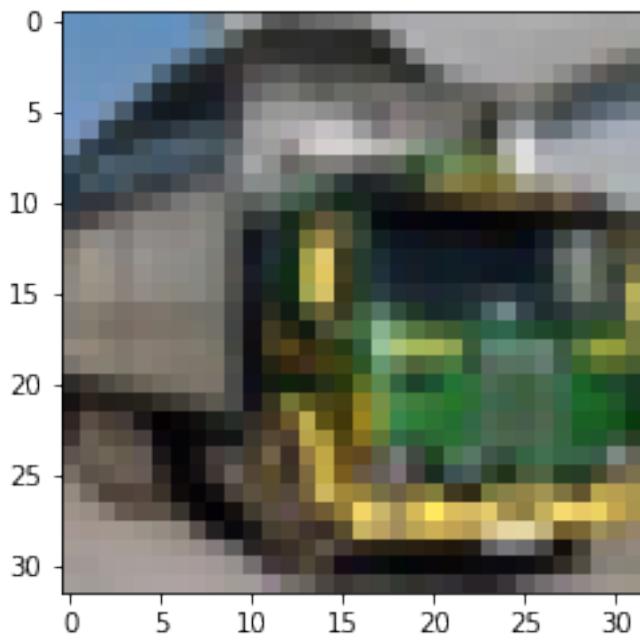
```
In [4]: a = int(input("請輸入你想可視化的圖片 [0-49999]:"))
print("你想可視化的圖片號碼是", a)
print("圖片答案是", label[y_train[a][0]])
plt.imshow(x_train[a])
```

請輸入你想可視化的圖片 [0-49999]:14

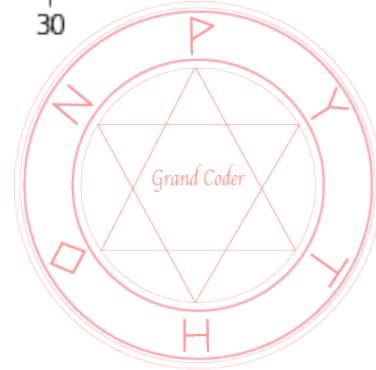
你想可視化的圖片號碼是 14

圖片答案是 卡車

```
Out[4]: <matplotlib.image.AxesImage at 0x12490d6d8>
```



一樣對特徵做出標準化，並且對目標做出 One-hot 編碼



```
In [5]: from keras.utils import np_utils  
x_train_shaped = x_train.astype("float32") / 255  
x_test_shaped = x_test.astype("float32") / 255  
y_train_cat = np_utils.to_categorical(y_train)  
y_test_cat = np_utils.to_categorical(y_test)
```

這次我們真的遵照 VGG-16 的結構來構建卷積層，我們總共做了 6 次的卷積(特徵萃取)，三次的池化(減少計算量)，最後在接上全連接層做出分類

```
In [6]: from keras.models import Sequential  
from keras.layers import Dense, Dropout, Activation, Flatten  
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D  
  
model = Sequential()  
  
  
model.add(Conv2D(filters=64,  
                 kernel_size=(3, 3),  
                 input_shape=(32, 32, 3),  
                 activation='relu',  
                 padding='same'))  
model.add(Conv2D(filters=64,  
                 kernel_size=(3, 3),  
                 activation='relu',  
                 padding='same'))  
model.add(MaxPooling2D(pool_size=(2, 2)))  
  
model.add(Dropout(0.25))  
model.add(Conv2D(filters=128,  
                 kernel_size=(3, 3),  
                 activation='relu',  
                 padding='same'))  
model.add(Conv2D(filters=128,  
                 kernel_size=(3, 3),  
                 activation='relu',  
                 padding='same'))  
model.add(MaxPooling2D(pool_size=(2, 2)))
```



```

model.add(Dropout(0.25))
model.add(Conv2D(filters=256,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding='same'))
model.add(Conv2D(filters=256,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

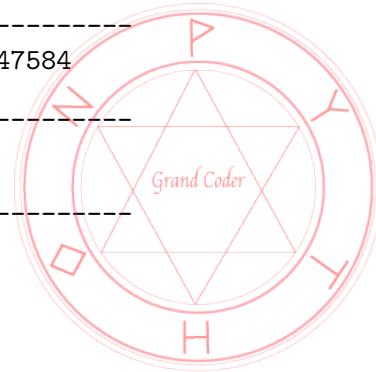
model.add(Flatten())
model.add(Dropout(rate=0.25))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(rate=0.25))

model.add(Dense(10, activation='softmax'))
model.summary()

```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 64)	1792
conv2d_2 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_4 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0



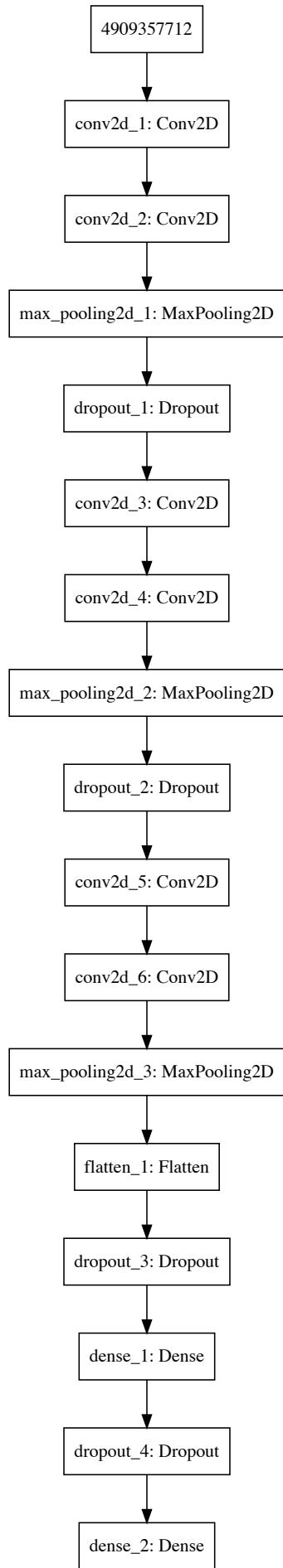
```
dropout_2 (Dropout)           (None, 8, 8, 128)          0
-----
conv2d_5 (Conv2D)             (None, 8, 8, 256)         295168
-----
conv2d_6 (Conv2D)             (None, 8, 8, 256)         590080
-----
max_pooling2d_3 (MaxPooling2D) (None, 4, 4, 256)        0
-----
flatten_1 (Flatten)           (None, 4096)              0
-----
dropout_3 (Dropout)           (None, 4096)              0
-----
dense_1 (Dense)               (None, 1024)             4195328
-----
dropout_4 (Dropout)           (None, 1024)              0
-----
dense_2 (Dense)               (None, 10)                10250
=====
Total params: 5,350,986
Trainable params: 5,350,986
Non-trainable params: 0
```

```
In [ ]: from IPython.display import SVG
        from keras.utils.vis_utils import model_to_dot

        SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out [None]:





```
In [ ]: model.compile(loss="categorical_crossentropy",
                      optimizer = "adam",
                      metrics = ['accuracy'])

train_history = model.fit(x = x_train_shaped, y = y_train_cat,
                          validation_split = 0.1,
                          epochs = 25,
                          batch_size = 200,
                          verbose = 2)

Train on 45000 samples, validate on 5000 samples
Epoch 1/25
- 506s - loss: 1.7535 - acc: 0.3524 - val_loss: 1.3401 - val_acc: 0.5048
Epoch 2/25
- 1332s - loss: 1.2987 - acc: 0.5338 - val_loss: 1.1006 - val_acc: 0.6030
Epoch 3/25
- 530s - loss: 1.0836 - acc: 0.6148 - val_loss: 0.9409 - val_acc: 0.6720
Epoch 4/25
- 524s - loss: 0.9304 - acc: 0.6695 - val_loss: 0.8318 - val_acc: 0.7104
Epoch 5/25
- 548s - loss: 0.8294 - acc: 0.7091 - val_loss: 0.7503 - val_acc: 0.7418
Epoch 6/25
- 572s - loss: 0.7383 - acc: 0.7391 - val_loss: 0.6894 - val_acc: 0.7628
Epoch 7/25
- 577s - loss: 0.6673 - acc: 0.7651 - val_loss: 0.6526 - val_acc: 0.7760
Epoch 8/25
- 562s - loss: 0.6147 - acc: 0.7840 - val_loss: 0.6549 - val_acc: 0.7792
Epoch 9/25
- 560s - loss: 0.5583 - acc: 0.8043 - val_loss: 0.6476 - val_acc: 0.7808
Epoch 10/25
- 549s - loss: 0.5181 - acc: 0.8192 - val_loss: 0.5955 - val_acc: 0.8032
Epoch 11/25
```

```
In [ ]: plt.plot(train_history.history["loss"])
plt.plot(train_history.history["val_loss"])
```



```
plt.title("Loss Graph")
plt.legend(['loss', 'val_loss'], loc="upper left")
```

```
In [ ]: model.evaluate(x_test_shaped, y_test_cat)
```



Embedding

2018 年 8 月 3 日



1 文字的深度學習 - 詞嵌入 + MLP

1.1 介紹

對於文字的深度學習，我們一樣遇到一個很嚴重的問題，當時我們在單純貝氏的時候，我們是把文字變成一個超高維度的向量，假設直接拿來這裡使用的話，可以想見一定是個不適用的方法，因為太稀疏了，所以你每一個訓練都跟沒訓練一樣，而且參數 w 的數量也多到不可思議

所以這裡我們要做一件很特別的事，也就是『降低維度』

1.2 降低維度

事實上，仔細研究我們的文字，我們會發現，其實我們可能不需要這麼大的維度，因為很多文字的意思是一樣的啊，譬如在討論好感度的文章，『喜歡』和『愛』應該是同一個維度的東西，只是程度的不同吧

沒錯，我們在處理文字的時候，通常做的第一件事就是降低維度，下面我們來看看我們對於降低維度(詞嵌入)的處理步驟

1.3 詞嵌入 (Embedding)

詞嵌入是我們給每個詞降低維度比較專業的稱呼，我們一起來看看最基本的處理步驟



1.3.1 選擇你的輸入維度

首先，你要決定你的最大輸入維度，白話的來說，就是建立一個跟維度一樣大的字典，舉個例子，假設我有一個句子

This is an apple.

我會建立一個四個字的字典，而且賦予他們一個數字

1	2	3	4
This	is	an	apple

我們就可以把上面的句子換成數字

1 2 3 4

跟之前單純貝氏不一樣的是，我們現在要對每一個詞處理，而不是對一整篇文章處理
所以其實上面的數字序列你可以表示成跟字典維度一樣大的向量序列

[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]

接下來再來做出我們下一步的處理

1.3.2 降低輸入維度

舉個例子

如果你可以有一個轉換的方式，不管是根據辭意或者根據討論議題，把它轉換成

[2, 1], [1, 2], [3, 3], [3, 2]

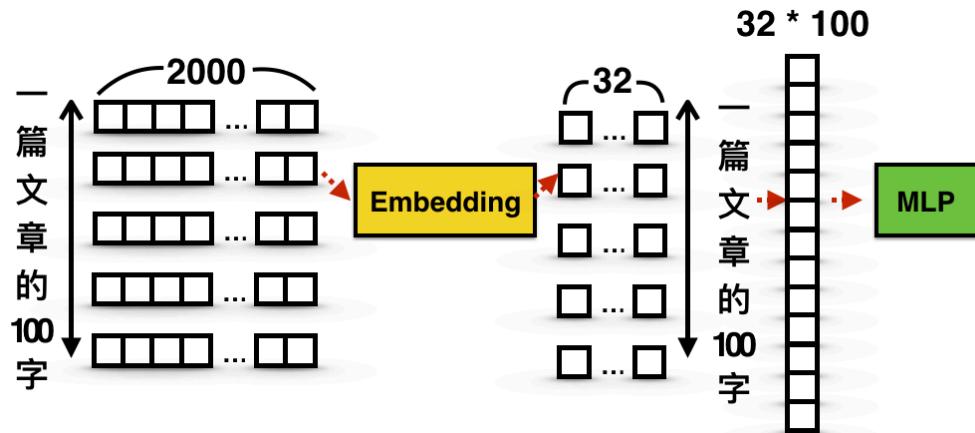
每一個詞都開始變成兩個維度的向量，你就可以開始做很多事了

譬如你的 kNN 會開始有比較好的效果(稀疏度減少)，可以開始把詞歸類成不同群體
又或者你可以把特徵攤開，做一個深度的 MLP

1.4 加入 MLP 做出分類

利用我們詞嵌入和 MLP 加在一起，就變成下面這樣





1.5 RNN 記憶單元

利用完詞嵌入和 MLP，你已經可以得到一個不錯的結果了

但是你應該會有一個疑惑，我們有些詞是要根據上下文來推斷他的意思

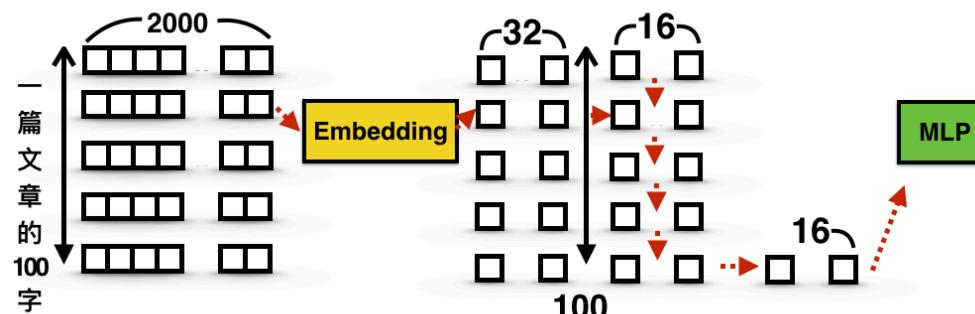
或者要加在一起考慮 (看到後面的『美食』應該把剛剛記憶中的『英國』拿出來)

這是我們的 RNN 記憶單元所負責的事物

簡單來說，就是把每一個詞化成狀態 (可能是地點，喜歡...之類) 神經元

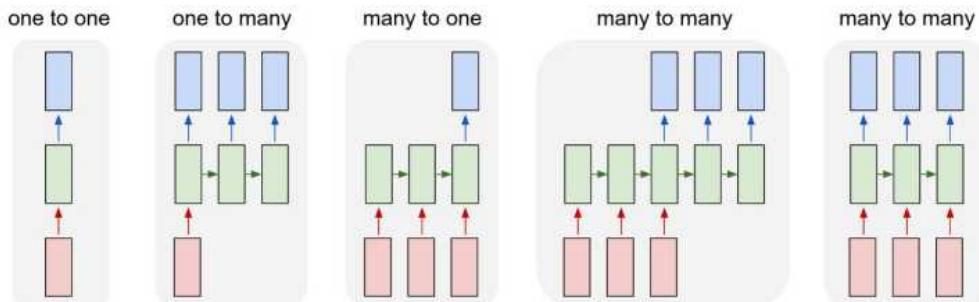
然後把狀態一直累加下去，直到最後把最後累加的狀態輸出

我們最常使用的 RNN 由於只輸出最後的狀態，所以我們叫它 many-to-one 的 RNN



另外還有不同的記憶單元 (ex. many-to-many 每個都輸出，代表我想知道看到每個字的記憶狀態)





我們就不在這介紹了

1.6 Step1. 資料預處理

這裡我們選用 imdb 的資料庫，但我們不直接用 keras 裡面的 imdb，因為裡面已經把原始的文字處理掉了，我們從網路下載原版的 imdb 資料庫

imdb 資料集已經幫你把每一篇 imdb 影評標註成『正面』和『負面』

imdb 資料集裡總共有 25000 訓練資料 (一半正一半負) 25000 測試資料 (一半正一半負)

我們就要藉由 Embedding 加上 MLP 做出二元的分類

下載網址是: http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

In [1]: # urlretrieve 是一個方便的東西

```
# 他直接結合 urlopen + file.write 幫你做完儲存工作
from urllib.request import urlretrieve
import os
# MAC 要加入這段，SSL 證書才不會被視為無效
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

url = "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
# 如果 data 資料夾不存在就創一下
if not os.path.exists("./data"):
    print("data 資料夾不存在，現在幫你創唷")
    os.mkdir("./data")
# 還沒下載過就下載一下
filepath = "./data/imdb.tar.gz"
if not os.path.exists(filepath):
    print("還沒下載過資料，現在幫你下載唷")
    urlretrieve(url, filepath)
else:
    print("已下載過")
```



已下載過

我們可以使用內建的 `tarfile` 幫我們解壓縮

```
In [2]: import tarfile
if not os.path.exists("data/aclImdb"):
    print("還沒解壓縮過，現在幫你解壓縮")
    tfile = tarfile.open(filepath, 'r')
    tfile.extractall('data')
else:
    print("已解壓縮過")
```

已解壓縮過

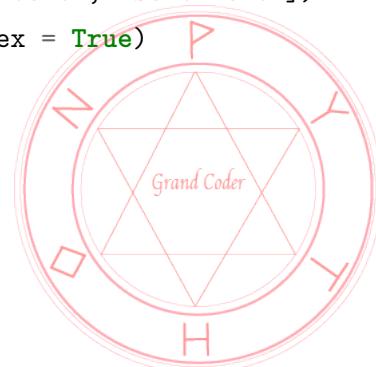
整理一下我們的訓練資料

```
In [3]: import pandas as pd
# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)

train_df = pd.DataFrame(columns = ["content", "sentiment"])

# 走過 pos 的資料夾，把資料夾整理好
pos_path = "data/aclImdb/train/pos"
for fpath in os.listdir(pos_path):
    if not fpath.startswith("."):
        fpath = os.path.join(pos_path, fpath)
        f = open(fpath, "r", encoding = "utf-8")
        content = f.read()
        s = pd.Series([content, 1], index = ["content", "sentiment"])
        train_df = train_df.append(s, ignore_index = True)

# 走過 neg 的資料夾，把資料夾整理好
neg_path = "data/aclImdb/train/neg"
for fpath in os.listdir(neg_path):
```



```

if not fpath.startswith("."):
    fpath = os.path.join(neg_path, fpath)
    f = open(fpath, "r", encoding = "utf-8")
    content = f.read()
    s = pd.Series([content, 0], index = ["content", "sentiment"])
    train_df = train_df.append(s, ignore_index = True)

train_df

```

Out[3]:

	content	sentiment
0	Bromwell High is a cartoon comedy. It ran at t...	1
1	Homelessness (or Houselessness as George Carli...	1
2	Brilliant over-acting by Lesley Ann Warren. Be...	1
3	This is easily the most underrated film inn th...	1
4	This is not the typical Mel Brooks film. It wa...	1
5	This isn't the comedic Robin Williams, nor is ...	1
6	Yes its an art... to successfully make a slow ...	1
...
24993	Although the production and Jerry Jameson's di...	0
24994	Capt. Gallagher (Lemmon) and flight attendant ...	0
24995	Towards the end of the movie, I felt it was to...	0
24996	This is the kind of movie that my enemies cont...	0
24997	I saw 'Descent' last night at the Stockholm Fi...	0
24998	Some films that you pick up for a pound turn o...	0
24999	This is one of the dumbest films, I've ever se...	0

[25000 rows x 2 columns]

整理測試資料

In [4]: test_df = pd.DataFrame(columns = ["content", "sentiment"])

```

pos_path = "data/aclImdb/test/pos"
for fpath in os.listdir(pos_path):
    if not fpath.startswith("."):
        fpath = os.path.join(pos_path, fpath)
        f = open(fpath, "r", encoding = "utf-8")
        content = f.read()

```



```

        s = pd.Series([content, 1], index = ["content", "sentiment"])
        test_df = test_df.append(s, ignore_index = True)

neg_path = "data/aclImdb/test/neg"
for fpath in os.listdir(neg_path):
    if not fpath.startswith("."):
        fpath = os.path.join(neg_path, fpath)
        f = open(fpath, "r", encoding = "utf-8")
        content = f.read()
        s = pd.Series([content, 0], index = ["content", "sentiment"])
        test_df = test_df.append(s, ignore_index = True)

test_df

```

Out [4] :

	content	sentiment
0	I went and saw this movie last night after bei...	1
1	Actor turned director Bill Paxton follows up h...	1
2	As a recreational golfer with some knowledge o...	1
3	I saw this film in a sneak preview, and it is ...	1
4	Bill Paxton has taken the true story of the 19...	1
5	I saw this film on September 1st, 2005 in Indi...	1
6	Maybe I'm reading into this too much, but I wo...	1
...
24993	This is one dreary, inert, self-important bore...	0
24994	Awful, awful, awful times a hundred still does...	0
24995	I occasionally let my kids watch this garbage ...	0
24996	When all we have anymore is pretty much realit...	0
24997	The basic genre is a thriller intercut with an...	0
24998	Four things intrigued me as to this film - fir...	0
24999	David Bryce's comments nearby are exceptionall...	0

[25000 rows x 2 columns]

把我們訓練資料找出最常見的 2000 字拿來建立出 2000 維度的字典

In [5]: `from keras.preprocessing.text import Tokenizer`
`token = Tokenizer(num_words=2000)`
`token.fit_on_texts(train_df["content"])`



```
# 我省略了這裡的印出，讀者可以把註解秀出字典的樣子
# token.word_index
```

Using TensorFlow backend.

利用字典把我們整理好的 DataFrame 每一個詞變成數字

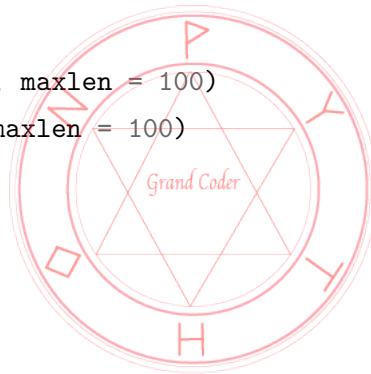
```
In [6]: x_train_seq = token.texts_to_sequences(train_df["content"])
x_test_seq = token.texts_to_sequences(test_df["content"])
pd.DataFrame(x_train_seq)
```

	0	1	2	3	4	...	1706	1707	1708	1709	1710
0	309	6	3	1069	209	...	NaN	NaN	NaN	NaN	NaN
1	39	14	739	44	74	...	NaN	NaN	NaN	NaN	NaN
2	526	117	113	31	1957	...	NaN	NaN	NaN	NaN	NaN
3	11	6	711	1	88	...	NaN	NaN	NaN	NaN	NaN
4	11	6	21	1	797	...	NaN	NaN	NaN	NaN	NaN
5	11	215	1	1714	1693	...	NaN	NaN	NaN	NaN	NaN
6	419	91	32	495	5	...	NaN	NaN	NaN	NaN	NaN
...
24993	259	1	362	2	1514	...	NaN	NaN	NaN	NaN	NaN
24994	2	23	3	1501	890	...	NaN	NaN	NaN	NaN	NaN
24995	946	1	127	4	1	...	NaN	NaN	NaN	NaN	NaN
24996	11	6	1	240	4	...	NaN	NaN	NaN	NaN	NaN
24997	10	216	233	311	30	...	NaN	NaN	NaN	NaN	NaN
24998	46	105	12	22	1259	...	NaN	NaN	NaN	NaN	NaN
24999	11	6	28	4	1	...	NaN	NaN	NaN	NaN	NaN

[25000 rows x 1711 columns]

你發現每一篇文章的長度不一樣，所以我們截長補短把每一篇文章取 100 個字來做訓練（訓練的時候無法處理長短不同的文章）

```
In [7]: from keras.preprocessing import sequence
x_train_pad = sequence.pad_sequences(x_train_seq, maxlen = 100)
x_test_pad = sequence.pad_sequences(x_test_seq, maxlen = 100)
pd.DataFrame(x_train_pad)
```



```
Out[7]:      0    1    2    3    4    ...    95   96   97   98   99
      0    30    1  169    55   14    ...    48    3   12    9  215
      1    27  553     7     7  134    ...    77   22     5  335  405
      2     8 1640    23  330     5    ...     9   60     6  176  396
      3    88    19     1  249    91    ...    35   73    14     3  482
      4    73   326    71    88     4    ...   196   253    65  528    70
      5    24     7     7   79  1180    ...   155    36     7     7     1
      6     0     0     0     0     0    ...     2   839    3  343    62
      ...
      ...
      ...
      ...
      24993  222    54  757     4     9    ...  1246  1586   848    30    36
      24994  203     2  841   130    23    ...    37   12     2     7     7
      24995     3  195   135    10   298    ...     5   103    3  411    76
      24996    75    14    10    83   194    ...    41  1568    37     4     1
      24997     9     6    49   846    18    ...    80   11    17    96    75
      24998     1    55     9   211     5    ...   105    8  260  1195  794
      24999   683    49  344    39   106    ...   126    55    11     6  1350
```

[25000 rows x 100 columns]

秀個大家看一下經過我們 padding 成為 100 以後的資料長成如何

```
In [8]: x_train_pad[0]
```

```
Out[8]: array([ 30,     1,  169,    55,    14,    46,    82,    41,   392,   110,   138,
       14,    58,  150,     8,     1,   482,    69,     5,   261,    12,     6,
       73,     5,  632,    71,     6,     1,     5,     1,  1534,    34,    67,
       64,   205,  140,    65,  1230,     1,     4,     1,   223,   901,    29,
       69,     4,     1,    10,   693,     2,    65,  1534,    51,    10,   216,
       1,   387,     8,    60,     3,  1467,    800,     5,   177,     1,   392,
      10,  1237,    30,   309,     3,   353,    344,   143,   130,     5,    28,
       4,   126,  1467,     5,   309,    10,   532,    12,   108,  1468,     4,
       58,   555,   101,    12,   309,     6,   227,    48,     3,    12,     9,
      215], dtype=int32)
```

1.7 Step2. 建立模型

這裡我們一樣使用 Sequential 模型

但我們開始用一個新的 Layer



1.7.1 Embedding 層

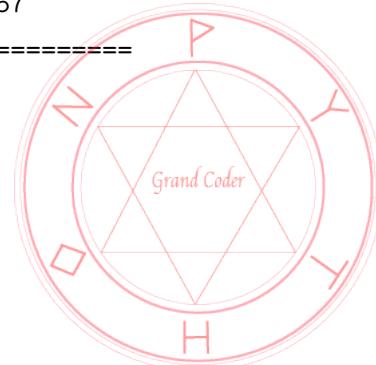
Keras 內建詞嵌入層，可以幫你把 2000 維度的每一個詞向量換成 32 維度的詞向量

```
In [9]: from keras.models import Sequential
        from keras.layers.core import Dense, Dropout, Flatten
        from keras.layers.embeddings import Embedding

        model = Sequential()
        model.add(Embedding(output_dim=32, input_dim=2000, input_length=100))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(units=256, activation='relu'))
        model.add(Dropout(0.35))
        # 注意一下，因為我們是二元分類，最後的激勵函數選擇 sigmoid
        # sigmoid(正 + 負 =100%) softmax(類別全部 =100%)
        model.add(Dense(units=1, activation='sigmoid'))
```

```
In [10]: model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
embedding_1 (Embedding)	(None, 100, 32)	64000
<hr/>		
dropout_1 (Dropout)	(None, 100, 32)	0
<hr/>		
flatten_1 (Flatten)	(None, 3200)	0
<hr/>		
dense_1 (Dense)	(None, 256)	819456
<hr/>		
dropout_2 (Dropout)	(None, 256)	0
<hr/>		
dense_2 (Dense)	(None, 1)	257
<hr/>		
Total params: 883,713		
Trainable params: 883,713		
Non-trainable params: 0		



1.7.2 Param 個數

我們一起看看上方 summary 的參數個數

1. Embedding 層的 64000: 2000(輸入詞維度) \times 32(輸入詞維度) = 64000
2. Flatten 層的 3200 輸入維度: 32(詞維度) * 100(文章的字數) = 3200

In [11]: `import numpy as np`

```
# 特別注意一下，因為我們只是二元分類，所以這裡的 loss 選擇 binary_crossentropy
model.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])

from keras.datasets import imdb
y_train = train_df['sentiment']
train_history = model.fit(x_train_pad, y_train,
                           batch_size = 100,
                           epochs = 3,
                           verbose = 2,
                           validation_split = 0.2)
```

Train on 20000 samples, validate on 5000 samples

Epoch 1/3

- 4s - loss: 0.4770 - acc: 0.7572 - val_loss: 0.5236 - val_acc: 0.7576

Epoch 2/3

- 4s - loss: 0.2715 - acc: 0.8897 - val_loss: 0.5791 - val_acc: 0.7472

Epoch 3/3

- 4s - loss: 0.1595 - acc: 0.9420 - val_loss: 0.5526 - val_acc: 0.7932

根據 val_loss，我們可以知道大概在 3-4 個 epoch 的時候，模型就已經完成了訓練
繼續訓練下去只是增加過擬合的程度

In [12]: `y_test = test_df['sentiment']`
`# 正確率是 list 第二個元素`
`model.evaluate(x_test_pad, y_test)`

25000/25000 [=====] - 1s 43us/step

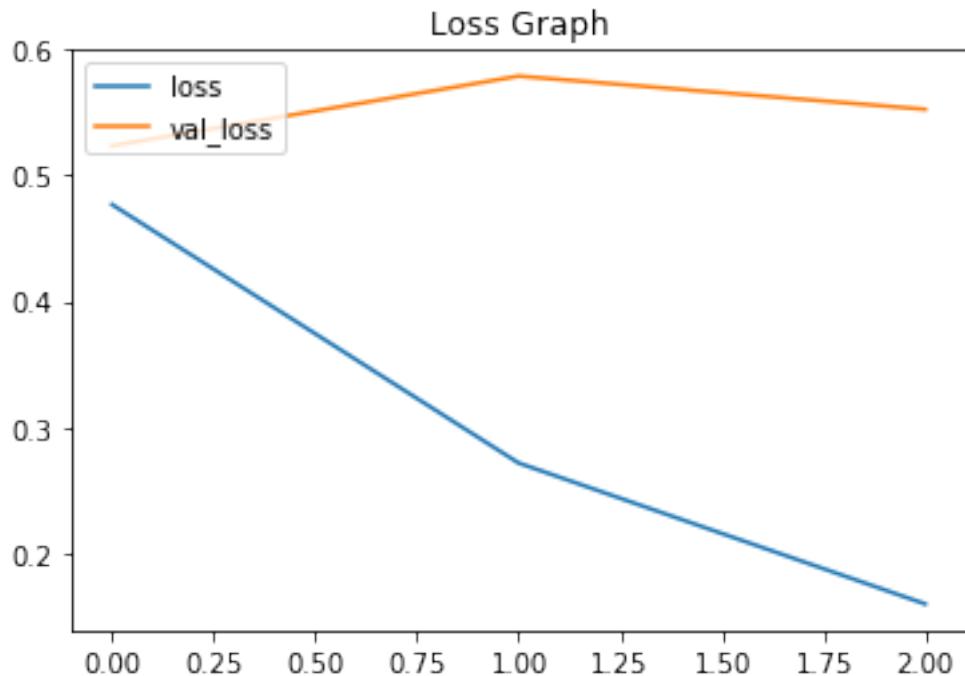


Out [12]: [0.45064449694633485, 0.82524]

你可以看到正確率大概已經在 83%，是一個非常不錯的結果了

```
In [13]: import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(train_history.history["loss"])
plt.plot(train_history.history["val_loss"])
plt.title("Loss Graph")
plt.legend(['loss', 'val_loss'], loc="upper left")
```

Out [13]: <matplotlib.legend.Legend at 0x138f91550>



1.7.3 中間層輸出

我想讓你看看到底 2000 維的向量是怎麼被轉成 32 維的向量，又是長怎麼樣的
keras 可以藉由 Model 拿出你剛剛模型的某幾層，創立出一個新的模型

```
In [14]: # 模型的 Layer List
model.layers
```



```
Out[14]: [<keras.layers.embeddings.Embedding at 0x11c447e10>,
<keras.layers.core.Dropout at 0x11c447198>,
<keras.layers.core.Flatten at 0x104a62b00>,
<keras.layers.core.Dense at 0x11ae4d780>,
<keras.layers.core.Dropout at 0x11c4653c8>,
<keras.layers.core.Dense at 0x11c4882e8>]
```

In [15]: # 拿最初的 *input* 和第一層 (*embedding*) 的輸出拿來當新模型

```
from keras.models import Model
embedding_layer_model = Model(inputs=model.input,
                               outputs=model.layers[0].output)
# 把第一篇文章拿來給你看轉換後的維度
em = embedding_layer_model.predict(x_test_pad[0:1])
em
```

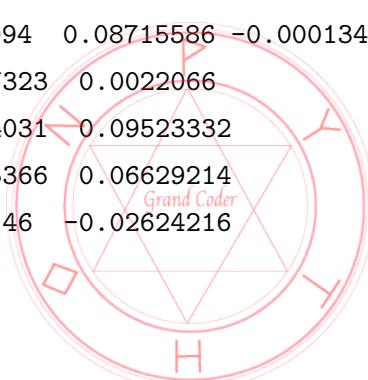
```
Out[15]: array([[[ 0.04700758,  0.01377721,  0.10867094, ..., -0.02624216,
                   0.03142255, -0.09424717],
                  [-0.00131393, -0.02347459,  0.00862844, ...,  0.01718462,
                   0.00600311, -0.01461102],
                  [ 0.05301251,  0.06678709, -0.00761796, ..., -0.01124587,
                   -0.02221869,  0.01999949],
                  ...,
                  [ 0.02881096, -0.03530543, -0.01820028, ..., -0.0320297 ,
                   -0.0836475 , -0.07444255],
                  [-0.06361311, -0.01159843,  0.03845826, ..., -0.01422905,
                   0.0371515 , -0.09605152],
                  [ 0.08404704, -0.07338575,  0.00265952, ...,  0.01804166,
                   0.01961992, -0.04136086]]], dtype=float32)
```

In [16]: `print("維度:", em.shape)`
`print("第一個詞被轉換過的向量:", em[0][0])`

維度: (1, 100, 32)

第一個詞被轉換過的向量: [0.04700758 0.01377721 0.10867094 0.08715586 -0.00013417 0.02513819

0.0326918	0.03093791	-0.01146789	0.05069069	-0.00337323	0.0022066
0.01615707	0.03905306	-0.02227833	0.019749	-0.09394031	0.09523332
0.08358718	-0.0468246	0.03339849	-0.00626512	0.04425366	0.06629214
0.03313974	0.06694432	0.01610038	0.00765152	0.0225146	-0.02624216



0.03142255 -0.09424717]

1.8 Step3. RNN

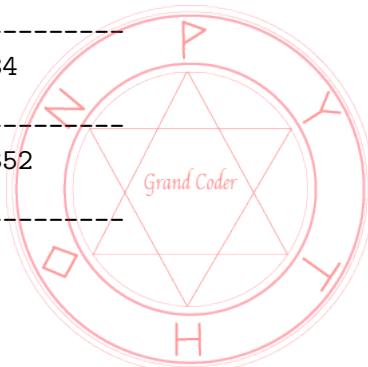
試試看 RNN 吧

RNN 參數個數: 32×16 (每一個詞都由 32 維度轉成 16 維度狀態) + 16×16 (往下傳遞的記憶) + 16 (輸出的參數) = 784

In [26]: `from keras.layers import SimpleRNN`

```
model = Sequential()
model.add(Embedding(output_dim=32, input_dim=2000, input_length=100))
model.add(Dropout(0.2))
# RNN: 記憶 16 個狀態
model.add(SimpleRNN(units=16))
model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.35))
# 注意一下，因為我們是二元分類，最後的激勵函數選擇 sigmoid
# sigmoid(正 + 負 =100%) softmax(類別全部 =100%)
model.add(Dense(units=1, activation='sigmoid'))
# 特別注意一下，因為我們只是二元分類，所以這裡的 loss 選擇 binary_crossentropy
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

Layer (type)	Output Shape	Param #
<hr/>		
embedding_7 (Embedding)	(None, 100, 32)	64000
<hr/>		
dropout_13 (Dropout)	(None, 100, 32)	0
<hr/>		
simple_rnn_5 (SimpleRNN)	(None, 16)	784
<hr/>		
dense_13 (Dense)	(None, 256)	4352
<hr/>		



```

dropout_14 (Dropout)           (None, 256)          0
-----
dense_14 (Dense)              (None, 1)            257
=====
Total params: 69,393
Trainable params: 69,393
Non-trainable params: 0
-----
```

你可以看到透過 RNN，我們還蠻常可以得到一個更好的結果
 這裡就得到了約 1% 的正確率提升 (0.84)

```
In [27]: train_history = model.fit(x_train_pad, y_train,
                                 batch_size = 100,
                                 epochs = 3,
                                 verbose = 2,
                                 validation_split = 0.2)

y_test = test_df['sentiment']
# 正確率是 list 第二個元素

model.evaluate(x_test_pad, y_test)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/3
- 5s - loss: 0.4810 - acc: 0.7669 - val_loss: 0.6841 - val_acc: 0.6846
Epoch 2/3
- 4s - loss: 0.3380 - acc: 0.8588 - val_loss: 0.4044 - val_acc: 0.8340
Epoch 3/3
- 4s - loss: 0.2888 - acc: 0.8803 - val_loss: 0.3504 - val_acc: 0.8634
25000/25000 [=====] - 4s 163us/step
```

Out [27]: [0.37006479842185974, 0.84144]

1.9 結語

我們已經完成了文字的基本預測，利用詞嵌入模型加上我們之前學會的深度學習，完成情緒的預測

但其實我們的選取都選的比較小 (計算量較小)



你可以自行調整字典的大小 (ex: 2000 -> 5000), 調整我們文章取出的字數 (ex: 100 -> 500) 再預測看看



GAN_Final

2018 年 8 月 20 日



1 生成式對抗網路 - GAN

1.1 介紹

之前介紹的 CNN 和 RNN 都是屬於監督式的分類任務，但是現實我們想做到的事更多也更宏大

譬如：讓電腦學會寫字，讓電腦學會填補圖畫之類

但是這種任務都是屬於無中生有的事，我們之前所教的單純分類任務是做不到的
於是就有人提出了 GAN(Generative Adversarial Network)

1.2 GAN 介紹

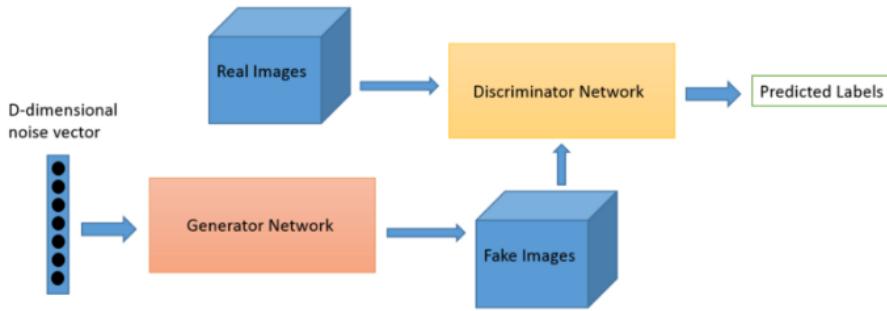
GAN 一個重點就是 Generative(生成)，他試圖讓電腦開始有創作的可能性

第二個重點就是 Adversarial(對抗)，因為創作，一定不可能是標注好答案的(監督式)

那也不可能完全無憑藉(非監督式)的創造

所以他使用的方法是半監督式，藉由環境的反饋來決定一個創作好不好





上圖是整個 GAN 的架構圖

其實概念非常的簡單

我們會有一個正常的分類器，來當作鑑賞家，這個鑑賞家 (Discriminator) 只要學會分類兩件事情，真(我們真實的 mnist 資料)和偽(我們隨手創造出來的數字)

接著我們會有一個反向(神經元越來越多)的深度網路，來當作創作家，這個創作家 (Generator) 會依據神經元的權重來創造數字

接著我們進行以下的步驟，每個 batch 都要這麼進行

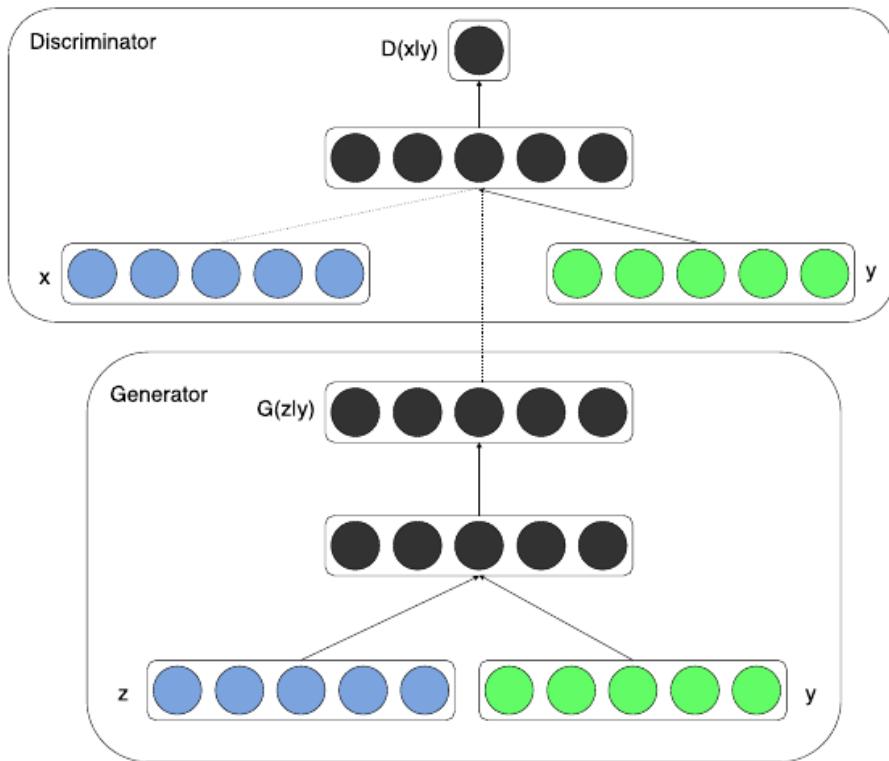
1. 把真實資料的一個 batch 標示為真丟給鑑賞家，也讓創作家創作一個 batch 標示為鑑賞家，並且訓練，讓鑑賞家學會真假
2. 讓創作家再創作一個 batch，這時候開始訓練創作家，讓創作家的作品接近鑑賞家現在的真

不斷的進行這個步驟，創作家和鑑賞家的水品都會提升，來到最後，創作家就可以創作幾可亂真的作品了

1.3 變種 GAN

除了最基本的 GAN，還有許多不同的 GAN，譬如 CGAN(Conditional Generative Adversarial Net)，把機率換成了條件機率，就可以指定創作某一類的數字





1.4 變種 GAN 參考

<https://github.com/eriklindernoren/Keras-GAN>

1.5 Step1. 資料預處理

這裡我們選用內建的 mnist 手寫數字資料庫來，mnist 提供共 70000 筆手寫數字，而且用 keras 讀取的時候會直接幫你分成訓練和測試兩份資料

```
In [1]: from keras.layers import Input
        from keras.models import Model, Sequential
        from keras.layers.core import Reshape, Dense, Dropout, Flatten
        from keras.layers import Embedding, BatchNormalization
        from keras.datasets import mnist
        import numpy as np
        %matplotlib inline
        # 我們會使用到一些內建的資料庫，MAC 需要加入以下兩行，才不會把對方的 ssl 憑證視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
```



Using TensorFlow backend.

```
In [2]: # 回傳值：((訓練特徵，訓練目標)，(測試特徵，測試目標))
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
In [3]: x_train.shape
```

```
Out[3]: (60000, 28, 28)
```

```
In [4]: from keras.utils import np_utils
# reshape 讓他從 32 * 32 變成 784 * 1 的一維陣列
# 讓我們標準化到 -1~1 區間
x_train_shaped = (x_train.reshape(60000, 784).astype("float32") - 127.5)/127.5
x_test_shaped = (x_test.reshape(10000, 784).astype("float32") - 127.5)/127.5
```

1.6 Step2. 建立創作家

我們做一個跟我們以前反向的深度網路，神經元隨著層數越來越大，最後的神經元數目要等於你要創作的圖片的維度 (28×28)

這裡大家在 activation 因為不是要二分機率，所以還蠻喜歡有極正和極負的輸出，所以還蠻常使用 tanh 當作激活函數 (-1 ~ 1)，不過你也可以使用 sigmoid 當你的激活函數

1.6.1 BatchNormalization

這裡我們使用了一個在原始 GAN 論文裡沒使用的技巧

因為 GAN 的 Generator 和 Discriminator 都極其的脆弱

1. 因為是一個反向的神經元擴大，所以可以想見一點影響都會被擴大
2. 你會發現創作的圖片很容易讓鑑賞家走到 relu 的『死亡區』，就是為 0，而且斜率為 0 的區域，一旦來到這區域，代表梯度更新為 0，沒機會從死亡區回來，這時候我們就說這神經元已經死掉了

這裡我們想起了一件事，我們再傳入我們的圖片的時候，通常會喜歡做一次 Normalization 到 0~1，優點是可以好好配合 Keras 隨機的 Weights，不會隨意的亂走

但第二層以後都是神經網路算出來的值，那我們現在可不可以也採納這個概念呢？讓第二層以後的所有算出值也做出標準化呢？

可以的，而且由於我們是批次 (batch) 的訓練，所以我們也希望可以直接對整個 batch 做一次 Normalization 就好



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

上面是 Batch Normalization 的公式

前三行很簡單，就是普通的標準化，平移到均值為 0 的位置，縮放成標準差為 1

整個精華在第四行，因為我們的特徵可能本來就不該均值 0 和標準差 1 啊，所以他加了個縮放參數在這裡 (Gamma 和 Beta)

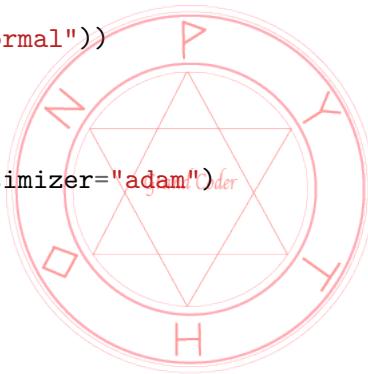
而且讓神經網路自己學習每一層的縮放參數是多少！

經過這美美的 Batch Normalization，我們達成兩個優點

1. 每一層都有經過適度的縮放和平移，可以很好的配合初始的權重
2. 因為經過縮放和平移，所以不會整組落入 relu 死亡區，就算這次落入死亡區，下一次還有機會經過 BN 被拉回來

記得在 GAN 的創作家每一層，我們都可以放上 Batch Normalization 這個技巧！會讓你的結果變得比較美！

```
In [5]: random_dim = 100
generator = Sequential()
generator.add(Dense(256, input_dim=random_dim,
                    activation='relu',
                    kernel_initializer="random_normal"))
generator.add(BatchNormalization())
generator.add(Dense(512, activation='relu',
                    kernel_initializer="random_normal"))
generator.add(BatchNormalization())
generator.add(Dense(784, activation='tanh'))
generator.compile(loss='binary_crossentropy', optimizer="adam")
generator.summary()
```

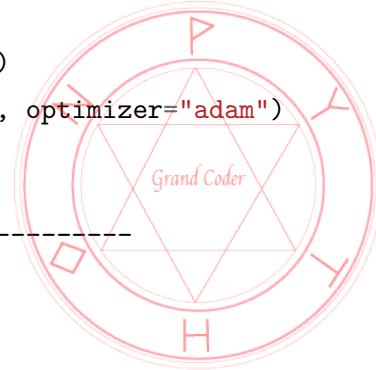


```
-----  
Layer (type)          Output Shape         Param #  
=====-----  
dense_1 (Dense)      (None, 256)          25856  
-----  
batch_normalization_1 (Batch (None, 256)) 1024  
-----  
dense_2 (Dense)      (None, 512)           131584  
-----  
batch_normalization_2 (Batch (None, 512)) 2048  
-----  
dense_3 (Dense)      (None, 784)           402192  
=====-----  
Total params: 562,704  
Trainable params: 561,168  
Non-trainable params: 1,536  
-----
```

1.7 Step3. 建立鑑賞家

一個專門來負責看揪出創作家創造的假作品的鑑賞家，我使用最簡單的 MLP 當作我們鑑賞家

```
In [6]: discriminator = Sequential()  
        discriminator.add(Dense(1024, input_dim=784,  
                               activation='relu',  
                               kernel_initializer="random_normal"))  
        discriminator.add(Dropout(0.25))  
        discriminator.add(Dense(512, activation='relu',  
                               kernel_initializer="random_normal"))  
        discriminator.add(Dropout(0.25))  
        discriminator.add(Dense(256, activation='relu',  
                               kernel_initializer="random_normal"))  
        discriminator.add(Dropout(0.25))  
        discriminator.add(Dense(1, activation='sigmoid'))  
        discriminator.compile(loss='binary_crossentropy', optimizer="adam")  
        discriminator.summary()
```



Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 1024)	803840
dropout_1 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524800
dropout_2 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 1)	257

Total params: 1,460,225
Trainable params: 1,460,225
Non-trainable params: 0

1.8 Step4. 組合網路

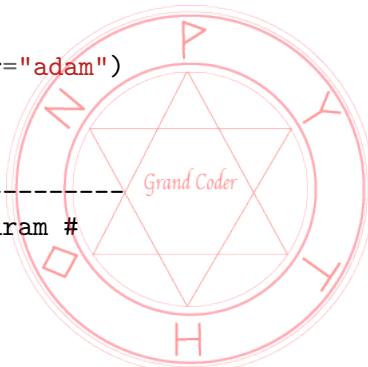
這裡為了方便訓練創作家，把它們組合在一起

那由於只是訓練創作家，所以我們要把鑑賞家的參數固定住，直接設置 trainable = False 即可
不過要在 compile 前就設定，compile 完了就不會改變

所以我們只有在組合網路有將鑑賞家的參數固定(可以看 Non-trainable params 確定)

```
In [7]: discriminator.trainable = False
gan_input = Input(shape=(random_dim,))
x = generator(gan_input)
gan_output = discriminator(x)
gan = Model(inputs=gan_input, outputs=gan_output)
gan.compile(loss='binary_crossentropy', optimizer="adam")
gan.summary()
```

Layer (type)	Output Shape	Param #
--------------	--------------	---------



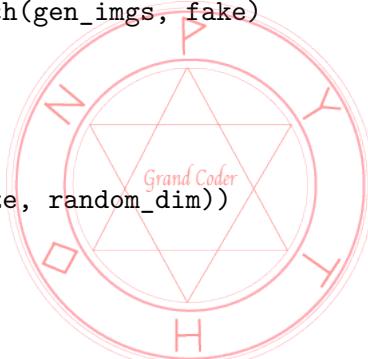
```
=====
input_1 (InputLayer)      (None, 100)      0
-----
sequential_1 (Sequential) (None, 784)      562704
-----
sequential_2 (Sequential) (None, 1)        1460225
=====
Total params: 2,022,929
Trainable params: 561,168
Non-trainable params: 1,461,761
-----
```

```
In [8]: batch_size = 200
epoch_count = 10
d_loss_list = []
g_loss_list = []
for epoch in range(0, epoch_count):
    for batch_count in range(0, 300):
        idx = np.random.randint(0, x_train.shape[0], batch_size)
        imgs = x_train_shaped[idx]

        valid = np.ones((batch_size, 1))
        fake = np.zeros((batch_size, 1))
        # 步驟 0: 讓創作家製造出 fake image
        noise = np.random.normal(0, 1, (batch_size, random_dim))
        gen_imgs = generator.predict(noise)

        discriminator.trainable = True
        # 步驟 1: 讓鑑賞家鑑賞對的 image
        d_loss_real = discriminator.train_on_batch(imgs, valid)
        # 步驟 2: 讓鑑賞家鑑賞錯的 image
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = (d_loss_real + d_loss_fake) / 2

        discriminator.trainable = False
        noise = np.random.normal(0, 1, (batch_size, random_dim))
```



```
# 步驟 3: 訓練創作家的創作能力
g_loss = gan.train_on_batch(noise, valid)
dash = "--" * 15
print(dash, "epoch", epoch, dash)
print("Discriminator loss:", d_loss)
print("Generator loss:", g_loss)
d_loss_list.append(d_loss)
g_loss_list.append(g_loss)

----- epoch 0 -----
Discriminator loss: 0.1319003701210022
Generator loss: 8.2713375
----- epoch 1 -----
Discriminator loss: 0.1373519003391266
Generator loss: 5.532821
----- epoch 2 -----
Discriminator loss: 0.41682881116867065
Generator loss: 2.3718922
----- epoch 3 -----
Discriminator loss: 0.462246835231781
Generator loss: 1.5258994
----- epoch 4 -----
Discriminator loss: 0.5252199769020081
Generator loss: 1.3568485
----- epoch 5 -----
Discriminator loss: 0.49116039276123047
Generator loss: 1.2497867
----- epoch 6 -----
Discriminator loss: 0.534600019454956
Generator loss: 1.126162
----- epoch 7 -----
Discriminator loss: 0.554734468460083
Generator loss: 1.0554607
----- epoch 8 -----
Discriminator loss: 0.5893127918243408
Generator loss: 1.1031088
----- epoch 9 -----
```



Discriminator loss: 0.5687193274497986

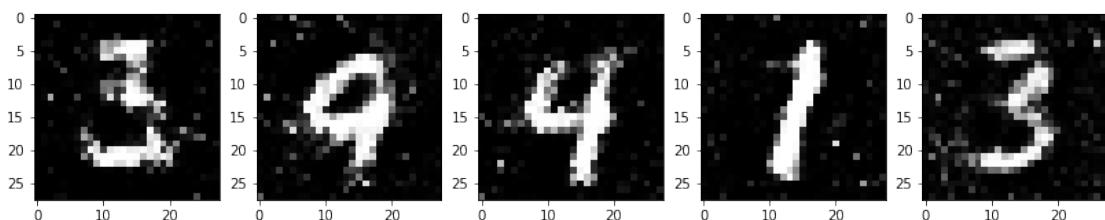
Generator loss: 1.081857

1.9 Step4. 訓練結果

你發現在鑑賞家的逼迫下，我們訓練出來的創作家創造的數字已經有模有樣了，有些數字已經看起來非常的真實了！

```
In [24]: import matplotlib.pyplot as plt
%matplotlib inline
examples = 5
noise = np.random.normal(0, 1, (examples, random_dim))
gen_imgs = generator.predict(noise)

# Rescale images 0 - 1
gen_imgs = 0.5 * gen_imgs + 0.5
gen_imgs = gen_imgs.reshape(examples, 28, 28)
plt.figure(figsize = (14, 14))
for i in range(0, examples):
    plt.subplot(1, examples, i + 1)
    plt.imshow(gen_imgs[i], cmap='gray')
```

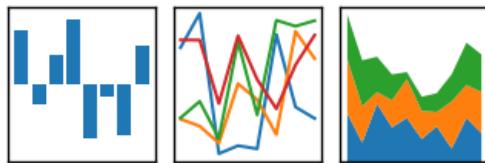


PandasBasic

2018 年 6 月 22 日



$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



1 Pandas 基本使用

1.1 介紹

專門拿來處理表格的函式庫，一生懸命只為了處理表格而生

1.2 用途

1. 處理各種表格 (csv, excel... 等等)
2. 拿來表示機器學習的資料集 (每個列就是一個物品，每個行代表物品的每個特徵)
3. 可以快速結合常用的繪圖函式庫，直接畫出漂亮的圖形

1.3 安裝方法

1. 使用 PyCharm: PyCharm -> Settings -> Project -> Project Interpreter -> + -> (搜索)pandas -> Install Packages
2. 使用命令列: cd 到你安裝 Python 的資料夾 -> 輸入 python -m pip install pandas

1.4 官方文件

<http://pandas.pydata.org/pandas-docs/stable/>

1.5 可以處理的表格形式

<http://pandas.pydata.org/pandas-docs/stable/io.html>



1.6 目標

我們使用 Kaggle 的 TED 資料集來教你 Pandas 的基本操作

1.7 資料集位置

<https://www.kaggle.com/rounakbanik/ted-talks>

1. 需要登入才能下載
2. 只取裡面的 ted_main.csv 來做分析

1.8 Pandas 基本資料

1. 多個行 * 多個列 -> DataFrame
2. 一個行 * 多個列或者一個列 * 多個行 -> Series

```
In [1]: import pandas as pd
```

```
# 為了顯示的漂亮，我刻意的把印出來的 row 和 column 只顯示六個  
# 大家練習的時候可以去掉下面兩行  
pd.set_option('display.max_rows', 6)  
pd.set_option('display.max_columns', 6)
```

1.9 讀取表格操作

使用 read_ 表格形式來讀取，記得最好明確表示用 utf-8 來讀取網站檔案 (網路上的檔案通常使用 utf-8 來儲存)

注意: 如果是 windows 的一些檔案，內建的儲存編碼是 ANSI，用 utf-8 會失效，我們留待編碼篇好好說

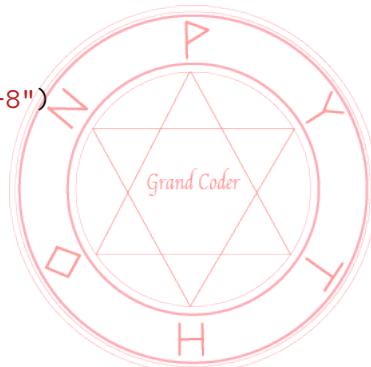
1.9.1 read_csv 重要參數:

1. 必要參數: 檔案位置
2. 選用參數 (有預設值): 讀取使用編碼

1.9.2 read_csv 回傳值:

DataFrame

```
In [2]: df = pd.read_csv("ted_main.csv", encoding = "utf-8")  
df
```



```
Out[2]:      comments                                     description duration \
0          4553 Sir Ken Robinson makes an entertaining and pro...      1164
1           265 With the same humor and humanity he exuded in ...      977
2           124 New York Times columnist David Pogue takes aim...     1286
...
...
2547         10 Science fiction visions of the future show us ...      651
2548         32 In an unmissable talk about race and politics ...    1100
2549         8 With more than half of the world population li...
...
...
0           ...
1           ...
2           ...
...
2547        ...
2548        ...
2549        ...

                           title \
0           ...
1           ...
2           ...
...
2547        ...
2548        ...
2549        ...

                           url      views
0 https://www.ted.com/talks/ken_robinson_says_sc...  47227110
1 https://www.ted.com/talks/al_gore_on_averting_...  3200520
2 https://www.ted.com/talks/david_pogue_says_sim...  1636292
...
...
2547 https://www.ted.com/talks/radhika_nagpal_what_...  375647
2548 https://www.ted.com/talks/theo_e_j_wilson_a_b...  419309
2549 https://www.ted.com/talks/karoliina_korppoo_ho...  391721

[2550 rows x 17 columns]
```

1.10 DataFrame 大小

1. 由於我們有兩個維度，所以以前習慣的 `len` 不能使用了，我們要使用`.shape`來取得兩個維度
2. `.shape` 是一個 tuple，所以第一個元素 [0] 就是你的列數，第二個元素 [1] 就是你的行數

In [3]: `df.shape`

Out[3]: (2550, 17)



1.11 表格行篩選

篩選行的時候，我們就像在操作字典一樣，對你的 DataFrame 加上 []

1.11.1 單行操作

直接在 [] 加上你想要的標籤名字，由於一個維度變成 1，所以你會發現從 DataFrame 變成 Series 了(印出來是不一樣的)

In [4]: df["comments"]

```
Out[4]: 0      4553
        1      265
        2      124
        ...
       2547     10
       2548     32
       2549      8
Name: comments, dtype: int64
```

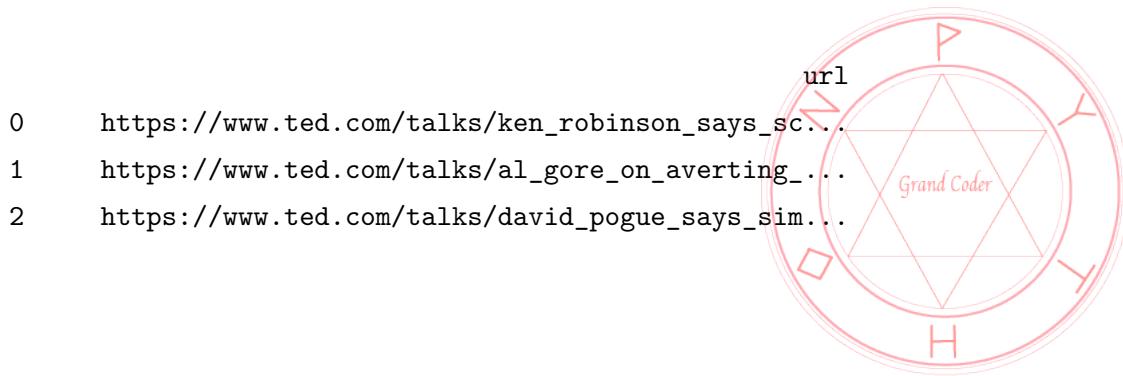
1.11.2 多行操作

你必須把想要的標籤集合成一個 list 傳入行操作的 []，所以這裡兩個 [] 代表截然不同的意思

1. 外面的 []: DataFrame 的行操作
2. 裡面的 []: 把標籤集合起來的 list

In [5]: df[["comments", "description", "url"]]

```
Out[5]:      comments                               description \
0      4553 Sir Ken Robinson makes an entertaining and pro...
1      265 With the same humor and humanity he exuded in ...
2      124 New York Times columnist David Pogue takes aim...
...
2547     10 Science fiction visions of the future show us ...
2548     32 In an unmissable talk about race and politics ...
2549      8 With more than half of the world population li...
```



```

...
2547 https://www.ted.com/talks/radhika_nagpal_what_...
2548 https://www.ted.com/talks/theo_e_j_wilson_a_b...
2549 https://www.ted.com/talks/karoliina_korppoo_ho...

```

[2550 rows x 3 columns]

1.12 表格列篩選

1. 篩選列的時候，我們使用的是.loc(少用，如果有自己創造列標籤才用得上)，.iloc(常用，使用pandas幫你創的0開始的列標籤)
2. 使用.iloc的時候會得到一個像是list的資料，接著就可以使用類似list的操作來操作
3. .iloc -> [“第一筆資料”, “第二筆資料”, “第三筆資料”, ..., “最後一筆資料”]

1.12.1 單列篩選

在.iloc這個列表加上[座號]

In [6]: df.iloc[0]

Out [6]: comments 4553
description Sir Ken Robinson makes an entertaining and pro...
duration 1164
...
title Do schools kill creativity?
url https://www.ted.com/talks/ken_robinson_says_sc...
views 47227110
Name: 0, dtype: object

1.12.2 多列篩選

.iloc後使用[頭部座號(包括):尾部座號(不包括)]

In [7]: # 取 10, 11, 12, 13, 14 共五筆資料

df.iloc[10:15]

Out [7]: comments description duration \\\n10 79 Accepting his 2006 TED Prize, Cameron Sinclair... 1414\\\n11 55 Jehane Noujaim unveils her 2006 TED Prize wish. 1538\\\n12 71 Accepting the 2006 TED Prize, Dr. Larry Brill... 1550

```

13      242 Jeff Han shows off a cheap, scalable multi-tou...      527
14      99 Nicholas Negroponte, founder of the MIT Media ...    1057

                           ...
10      ...           My wish: A call for open-source architecture
11      ...           My wish: A global day of film
12      ...           My wish: Help me stop pandemics
13      ...           The radical promise of the multi-touch interface
14      ...           One Laptop per Child

                           ...
10      https://www.ted.com/talks/cameron_sinclair_on_...  1211416
11      https://www.ted.com/talks/jehane_noujaim_inspi...  387877
12      https://www.ted.com/talks/larry_brilliant_want...  693341
13      https://www.ted.com/talks/jeff_han_demos_his_b...  4531020
14      https://www.ted.com/talks/nicholas_negroponte_...  358304

```

[5 rows x 17 columns]

1.12.3 頭幾列篩選

Pandas 也提供一些讓你偷懶的函式，如果是要篩選頭幾列的話用 `head` 函式來篩選

In [8]: # 頭五列

```
df.head(5)
```

Out[8]:

	comments	description	duration	\
0	4553 Sir Ken Robinson makes an entertaining and pro...		1164	
1	265 With the same humor and humanity he exuded in ...		977	
2	124 New York Times columnist David Pogue takes aim...		1286	
3	200 In an emotionally charged talk, MacArthur-winn...		1116	
4	593 You've never seen data presented like this. Wi...		1190	
...				
0	... Do schools kill creativity?	title \		
1	... Averting the climate crisis			
2	... Simplicity sells			
3	... Greening the ghetto			



```
4     ...      The best stats you've ever seen
```

		url	views
0		https://www.ted.com/talks/ken_robinson_says_sc...	47227110
1		https://www.ted.com/talks/al_gore_on_averting_...	3200520
2		https://www.ted.com/talks/david_pogue_says_sim...	1636292
3		https://www.ted.com/talks/majora_carter_s_tale...	1697550
4		https://www.ted.com/talks/hans_rosling_shows_t...	12005869

[5 rows x 17 columns]

1.12.4 尾幾列篩選

使用 tail 函式來做尾部的篩選

In [9]: # 尾五列

```
df.tail(5)
```

Out[9]: comments

	comments	description	duration	\
2545	17 Between 2008 and 2016, the United States depor...		476	
2546	6 How can you study Mars without a spaceship? He...		290	
2547	10 Science fiction visions of the future show us ...		651	
2548	32 In an unmissable talk about race and politics ...		1100	
2549	8 With more than half of the world population li...		519	

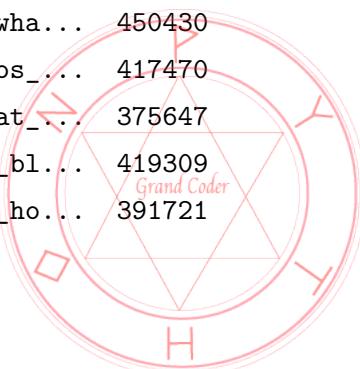
...

title \

	...	title \
2545	... What we're missing in the debate about immigr...	
2546	... The most Martian place on Earth	
2547	... What intelligent machines can learn from a sch...	
2548	... A black man goes undercover in the alt-right	
2549	... How a video game might help us build better ci...	

url views

	url	views
2545	https://www.ted.com/talks/duarte_geraldino_wh...	450430
2546	https://www.ted.com/talks/armando_azua_bustos_...	417470
2547	https://www.ted.com/talks/radhika_nagpal_what_...	375647
2548	https://www.ted.com/talks/theo_e_j_wilson_a_b...	419309
2549	https://www.ted.com/talks/karoliina_korppoo_ho...	391721



```
[5 rows x 17 columns]
```

1.13 表格行+列篩選

1. 只要是 DataFrame 就可以使用上面的行或者列篩選
2. 所以你可以任意組合行列篩選，先 [] 再.iloc[], 或者先.iloc[] 再 []

In [10]: # 如果：後面不寫就是到最底，：前面不寫就是從最頭開始

```
df[ ["comments", "description", "duration"] ].iloc[ :5 ]
```

Out[10]:

	comments	description	duration
0	4553 Sir Ken Robinson makes an entertaining and pro...		1164
1	265 With the same humor and humanity he exuded in ...		977
2	124 New York Times columnist David Pogue takes aim...		1286
3	200 In an emotionally charged talk, MacArthur-winn...		1116
4	593 You've never seen data presented like this. Wi...		1190

1.14 列過濾

1. 過濾操作是把符合我們期待的列留下來，不符合期待的列丟掉的一個操作
2. 核心概念是做一個跟我們的資料筆數一樣大的布林 list，對到 True 的資料留下，對到 False 的資料丟掉
3. (特別) 這時候一樣是對你的 DataFrame 加上 []，把布林 list 丟進你的 [] 裡

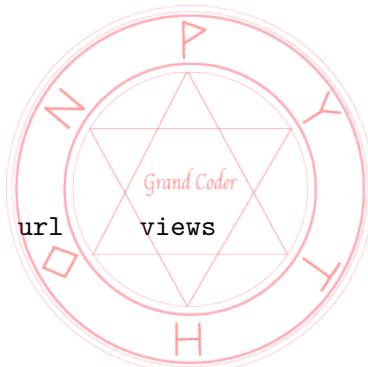
In [11]: # 先做個實驗給你看，只取三列資料

```
test = df.iloc[:3]
test
```

Out[11]:

	comments	description	duration \
0	4553 Sir Ken Robinson makes an entertaining and pro...		1164
1	265 With the same humor and humanity he exuded in ...		977
2	124 New York Times columnist David Pogue takes aim...		1286

	...	title \
0	...	Do schools kill creativity?
1	...	Averting the climate crisis
2	...	Simplicity sells



```
0 https://www.ted.com/talks/ken_robinson_says_sc... 47227110
1 https://www.ted.com/talks/al_gore_on_averting_... 3200520
2 https://www.ted.com/talks/david_pogue_says_sim... 1636292
```

[3 rows x 17 columns]

In [12]: # 過濾，創一個三個大小的 *True, False list*
對到 *True*(第一三筆) 留下，對到 *False*(第二筆) 丟掉
test[[True, False, True]]

Out[12]: comments description duration \
0 4553 Sir Ken Robinson makes an entertaining and pro... 1164
2 124 New York Times columnist David Pogue takes aim... 1286

		title \
0	...	Do schools kill creativity?
2	...	Simplicity sells

		url views
0	https://www.ted.com/talks/ken_robinson_says_sc...	47227110
2	https://www.ted.com/talks/david_pogue_says_sim...	1636292

[2 rows x 17 columns]

In [13]: # 但我們不可能自己用手創一個 2000 多個元素的布林 *list*
所以我們藉由 *pandas* 的函式幫我們
先取出一個 *Series* 取 *str* 屬性得到字串 *list*
藉由 *pandas* 定義的 *contains* 對裡面每個元素做出布林判斷
bool_filter = df["description"].str.contains("Sir")
bool_filter

Out[13]: 0 True
1 False
2 False
...
2547 False
2548 False
2549 False
Name: description, dtype: bool



```
In [14]: # 帶入 DataFrame
```

```
df[bool_filter]
```

```
Out[14]: comments
```

		description	duration	\
0	4553	Sir Ken Robinson makes an entertaining and pro...	1164	
15	325	Violinist Sirena Huang gives a technically bri...	1481	
54	203	Speaking as both an astronomer and "a concerne...	1046	
...
1978	64	The founder of Sirius XM satellite radio, Mart...	1264	
2192	61	Trust: How do you earn it? Banks use credit sc...	491	
2503	20	How smart can our machines make us? Tom Gruber...	586	

```
...
```

		title \
0	...	Do schools kill creativity?
15	...	An 11-year-old's magical violin
54	...	Is this our final century?
...
1978	...	My daughter, my wife, our robot, and the quest...
2192	...	A smart loan for people with no credit history...
2503	...	How AI can enhance our memory, work and social...

```
url views
```

0	https://www.ted.com/talks/ken_robinson_says_sc...	47227110
15	https://www.ted.com/talks/sirena_huang_dazzles...	2702470
54	https://www.ted.com/talks/martin_rees_asks_is_...	2121177
...
1978	https://www.ted.com/talks/martine_rothblatt_my...	1304737
2192	https://www.ted.com/talks/shivani_siroya_a_sma...	1437353
2503	https://www.ted.com/talks/tom_gruber_how_ai_ca...	1139827

```
[11 rows x 17 columns]
```

```
In [15]: # 你仔細對照，你會發現 contains 是只要有 contains 那個字串就可以
```

```
# 並不一定是完整的一個字 (Sirena 也算有 contains Sir)
```

```
# 但我們可以使用格式 (正規表示式) 來結合 contains
```

```
# 記得在你格式字串前加上 r(不轉換任何東西，原始字串)
```

```
# 不然\b 會被當成 backspace, 而不是兩個字
```

```
df[ df["description"].str.contains(r"\bSir\b") ]
```



```
Out[15]:      comments                                     description duration \
0          4553 Sir Ken Robinson makes an entertaining and pro...     1164
54         203 Speaking as both an astronomer and "a concerne...    1046
692        1234 In this poignant, funny follow-up to his fable...    1008
833        473 In this talk from RSA Animate, Sir Ken Robinso...     700
1502       634 Sir Ken Robinson outlines 3 principles crucial...   1151
1802       59 Sir Tim Berners-Lee invented the World Wide We...     403

                                         ...
0             ...           Do schools kill creativity?
54            ...           Is this our final century?
692           ...           Bring on the learning revolution!
833           ...           Changing education paradigms
1502          ...           How to escape education's death valley
1802          ...           A Magna Carta for the web

                                         url      views
0 https://www.ted.com/talks/ken_robinson_says_sc...  47227110
54 https://www.ted.com/talks/martin_rees_asks_is_...  2121177
692 https://www.ted.com/talks/sir_ken_robinson_bri...  7266316
833 https://www.ted.com/talks/ken_robinson_changin...  1854997
1502 https://www.ted.com/talks/ken_robinson_how_to_...  6657858
1802 https://www.ted.com/talks/tim_berners_lee_a_ma...  1054600

[6 rows x 17 columns]
```

1.15 儲存表格

- 非常簡單!!! 就跟 read 一樣, 你想儲存什麼就讓你的 DataFrame 使用 to_ 儲存格式
- 一樣建議指定使用 utf-8 做儲存

1.15.1 to_csv 重要參數

- 必要參數: 檔案位置
- 選用參數 encoding(有預設值): 讀取使用編碼
- 選用參數 index(有預設值 True): 要不要把 pandas 幫你產生的列編號寫進檔案, True: 寫, False: 不寫, 通常我會選 False

In [16]: # 用剛剛的 filter 過後的東西做個例子



```
filter_df = df[ df["description"].str.contains(r"\bSir\b") ]
# 儲存成 csv
filter_df.to_csv("filter.csv", encoding = "utf-8", index = False)
```

In [17]: # 把剛剛儲存的東西讀出來給你看看

```
pd.read_csv("filter.csv", encoding = "utf-8")
```

Out[17]:

	comments	description	duration
0	4553	Sir Ken Robinson makes an entertaining and pro...	1164
1	203	Speaking as both an astronomer and "a concerne...	1046
2	1234	In this poignant, funny follow-up to his fable...	1008
3	473	In this talk from RSA Animate, Sir Ken Robinso...	700
4	634	Sir Ken Robinson outlines 3 principles crucial...	1151
5	59	Sir Tim Berners-Lee invented the World Wide We...	403
	...		
0	...	Do schools kill creativity?	
1	...	Is this our final century?	
2	...	Bring on the learning revolution!	
3	...	Changing education paradigms	
4	...	How to escape education's death valley	
5	...	A Magna Carta for the web	
		url	views
0	https://www.ted.com/talks/ken_robinson_says_sc...	47227110	
1	https://www.ted.com/talks/martin_rees_asks_is_...	2121177	
2	https://www.ted.com/talks/sir_ken_robinson_bri...	7266316	
3	https://www.ted.com/talks/ken_robinson_changin...	1854997	
4	https://www.ted.com/talks/ken_robinson_how_to_...	6657858	
5	https://www.ted.com/talks/tim_berners_lee_a_ma...	1054600	

[6 rows x 17 columns]

1.16 刪除行

1. 你可以使用類似字典的刪除方式，用 `del` 來直接刪除一行
2. 你可以使用 `drop` 來刪除多行，不過記得如果你想要讓 `df` 變成刪除過後的樣子要記得設定回去



In [18]: # 刪除單行，直接就修改了 df

```
del df['title']
df
```

Out[18]:

	comments	description	duration	\
0	4553 Sir Ken Robinson makes an entertaining and pro...		1164	
1	265 With the same humor and humanity he exuded in ...		977	
2	124 New York Times columnist David Pogue takes aim...		1286	
...	
2547	10 Science fiction visions of the future show us ...		651	
2548	32 In an unmissable talk about race and politics ...		1100	
2549	8 With more than half of the world population li...		519	
	...	tags	\	
0	...	['children', 'creativity', 'culture', 'dance', ...]		
1	...	['alternative energy', 'cars', 'climate change...']		
2	...	['computers', 'entertainment', 'interface desi...']		
...	
2547	...	['AI', 'ants', 'fish', 'future', 'innovation', ...]		
2548	...	['Internet', 'TEDx', 'United States', 'communi...']		
2549	...	['cities', 'design', 'future', 'infrastructure...']		
		url	views	
0	https://www.ted.com/talks/ken_robinson_says_sc...	47227110		
1	https://www.ted.com/talks/al_gore_on_averting_...	3200520		
2	https://www.ted.com/talks/david_pogue_says_sim...	1636292		
...		
2547	https://www.ted.com/talks/radhika_nagpal_what_...	375647		
2548	https://www.ted.com/talks/theo_e_j_wilson_a_b...	419309		
2549	https://www.ted.com/talks/karoliina_korppoo_ho...	391721		

[2550 rows x 16 columns]

In [23]: # 刪除多行，axis = 1 指的是刪除行的意思，axis = 0 是刪除列的意思

```
df.drop(["url", "views"], axis = 1)
```

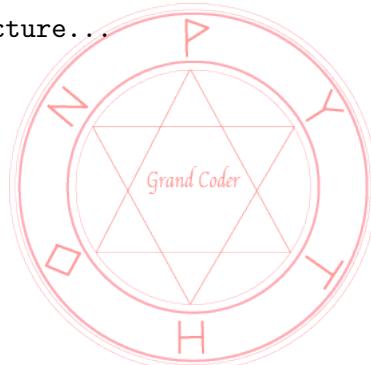
Out[23]:

	comments	description	duration	\
0	4553 Sir Ken Robinson makes an entertaining and pro...		1164	



1	265	With the same humor and humanity he exuded in ...	977
2	124	New York Times columnist David Pogue takes aim...	1286
...
2547	10	Science fiction visions of the future show us ...	651
2548	32	In an unmissable talk about race and politics ...	1100
2549	8	With more than half of the world population li...	519
...			
0	\
1	
2	
...	
2547	
2548	
2549	
related_talks speaker_occupation \			
0	[{'id': 865, 'hero': 'https://pe.tedcdn.com/im...'}]	Author/educator	
1	[{'id': 243, 'hero': 'https://pe.tedcdn.com/im...'}]	Climate advocate	
2	[{'id': 1725, 'hero': 'https://pe.tedcdn.com/i...'}]	Technology columnist	
...
2547	[{'id': 2346, 'hero': 'https://pe.tedcdn.com/i...'}]	Robotics engineer	
2548	[{'id': 2512, 'hero': 'https://pe.tedcdn.com/i...'}]	Public intellectual	
2549	[{'id': 2682, 'hero': 'https://pe.tedcdn.com/i...'}]	Game designer	
tags			
0	['children', 'creativity', 'culture', 'dance', ...]		
1	['alternative energy', 'cars', 'climate change...']		
2	['computers', 'entertainment', 'interface desi...']		
...	...		
2547	['AI', 'ants', 'fish', 'future', 'innovation', ...]		
2548	['Internet', 'TEDx', 'United States', 'communi...']		
2549	['cities', 'design', 'future', 'infrastructure...']		

[2550 rows x 14 columns]



1.17 (高階技巧) 列過濾

1. 有時候利用 pandas 有的函式難以完成我想要的過濾, 這時候我可以自定義我的過濾流程
2. 對你的 Series 使用 apply 來過濾
3. (重要) 你的過濾流程最後一定要回傳 True or False

```
In [19]: # 我想把 tag 欄位裡的字串拿出來並且轉換成一個 list
        # 再檢查某個字串也沒有在 list 裡
        # 你的過濾流程的第一個參數, pandas 會幫你傳入
        # 就是你每一格的資料, 也就是 element = 每一格的資料

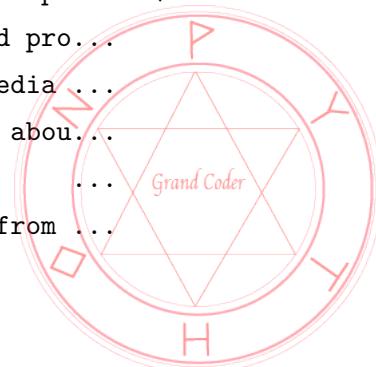
def tag_filter(element):
    # 利用 eval 把字串當成 python 程式執行, 變成一個 list
    tag_list = eval(element)
    if 'children' in tag_list:
        return True
    else:
        return False

# 讓你看看做出來的 bool list
bool_filter = df["tags"].apply(tag_filter)
bool_filter
```

```
Out[19]: 0      True
          1      False
          2      False
          ...
          2547     False
          2548     False
          2549     False
Name: tags, dtype: bool
```

```
In [20]: # 來過濾你的 DataFrame, 並且我們只看 tags, description 兩列
df[bool_filter][ ['description', 'tags'] ]
```

```
Out[20]:                                     description \
0      Sir Ken Robinson makes an entertaining and pro...
14     Nicholas Negroponte, founder of the MIT Media...
152    Author and illustrator Maira Kalman talks abou...
...
2479   Sixty-five million people were displaced from ...
```



```
2482 "We have seen advances in every aspect of our ...
2525 Could it be wrong to help children in need by ...
```

```
tags
0      ['children', 'creativity', 'culture', 'dance',...
14     ['children', 'design', 'education', 'entrepren...
152    ['art', 'children', 'culture', 'design', 'ente...
...
2479   ['TED Books', 'activism', 'big problems', 'chi...
2482   ['children', 'global issues', 'humanity', 'ide...
2525   ['TEDx', 'activism', 'children', 'family', 'po...
```

[143 rows x 2 columns]

In [21]: # 更進階定義，讓你在使用的時候可以再多帶入參數

```
def tag_filter(element, filter_tag):
    tag_list = eval(element)
    if filter_tag in tag_list:
        return True
    else:
        return False

# 你放在 apply 後面的參數，pandas 會幫你丟進你的過濾過程
# 但是參數名字就要對到過濾流程的參數
bool_filter = df["tags"].apply(tag_filter, filter_tag = 'Asia')
bool_filter
```

Out[21]: 0 False

1 False

2 False

...

2547 False

2548 False

2549 False

Name: tags, dtype: bool

In [22]: df[bool_filter][["description", "tags"]]

Out[22]:

4 You've never seen data presented like this. Wi...



```
117 Researcher Hans Rosling uses his cool data too...
359 Reporter Jennifer 8. Lee talks about her hunt ...
...
1621 The developed world holds up the ideals of cap...
1948 The former prime minister of Australia, Kevin ...
2310 Americanization and globalization have basical...
```

```
tags
4      ['Africa', 'Asia', 'Google', 'demo', 'economic...
117    ['Africa', 'Asia', 'Google', 'economics', 'glo...
359    ['Asia', 'business', 'culture', 'exploration',...
...
1621   ['Africa', 'Asia', 'china', 'democracy', 'econ...
1948     ['Asia', 'United States', 'china', 'politics']
2310   ['Africa', 'Asia', 'Europe', 'Foreign Policy',...
```

[26 rows x 2 columns]



PlotBasic

2018 年 6 月 22 日



1 視覺化工具

1.1 使用函式庫

1. Pandas: 我們的特徵資料表達的方式，Pandas 也跟 Matplotlib 有一個很好的連結，可以快速地把 DataFrame 轉成圖
2. Matplotlib: 所有視覺化工具的鼻祖，很多進階的視覺化函式庫都是基於 matplotlib 建立的
3. Seaborn: 進階的視覺化函式庫之一，有一些好用的函式讓你快速地建立一個複雜圖

1.2 介紹

這個章節我們不講一些比較數學的作圖，我們希望讓你了解在做機器學習或者深度學習的時候，我們常用的一些前置的作圖，讓我們了解我們資料特徵之間的關係或者是特徵和標籤之間的關係。

1.3 安裝方法

請用命令列或者 pycharm 安裝好 matplotlib 和 seaborn 函式庫

1.4 官方文件和例子

1. seaborn: <https://seaborn.pydata.org/>
2. matplotlib: <https://matplotlib.org/>



1.5 目標

我們使用 Kaggle 的鐵達尼號資料集來教你繪圖的基本操作

1.6 資料集位置

<https://www.kaggle.com/c/titanic/data>

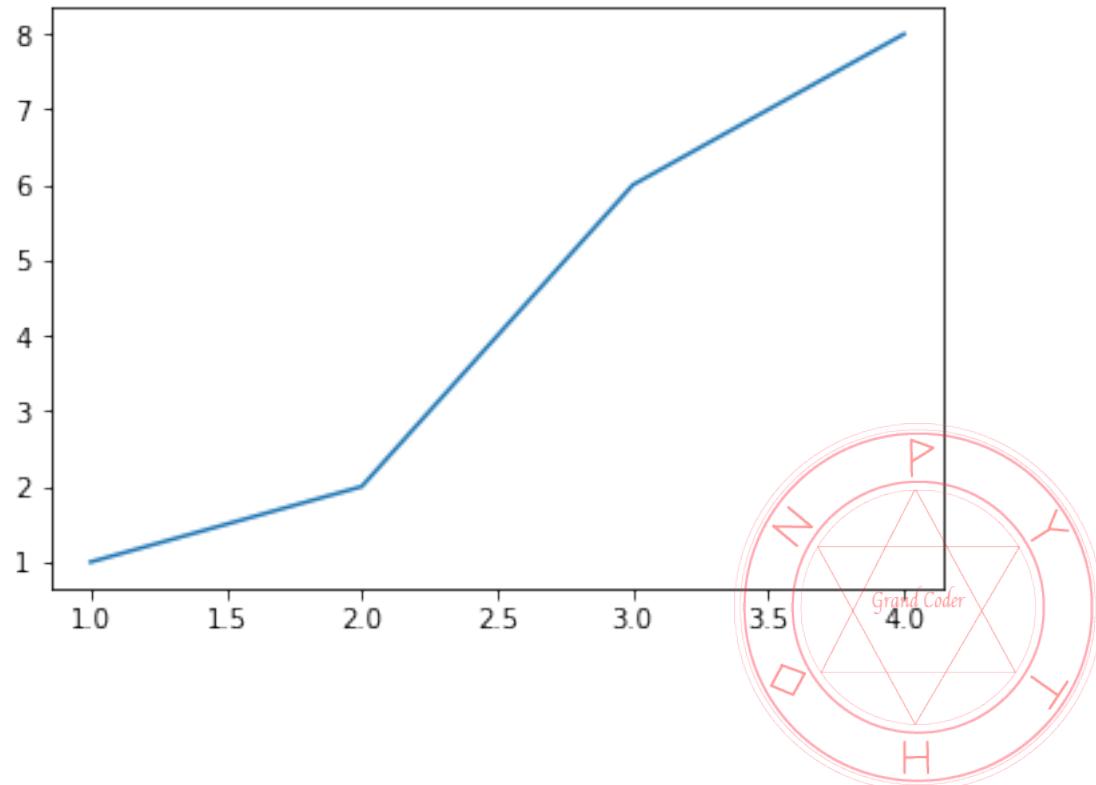
1. 需要登入才能下載
2. 只取裡面的 train.csv 來做繪圖

1.7 Matplotlib 最基本概念

1. 最簡單的繪圖就是給好 x 軸的 list 和給好 y 軸的 list, 然後就可以繪圖了
2. 流程 plot() -> show()
3. plot 的時候你必須把所以想在一張圖上的東西畫好
4. 所有的東西畫好在呼叫 show()
5. (純粹習慣) 大家習慣在 import 以後改叫 plt

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: x_list = [1, 2, 3, 4]
y_list = [1, 2, 6, 8]
plt.plot(x_list, y_list)
plt.show()
```



1.8 選擇線條類型和顏色

繪圖的第三個參數可以以一組類型 + 顏色來客製化

1. 線條



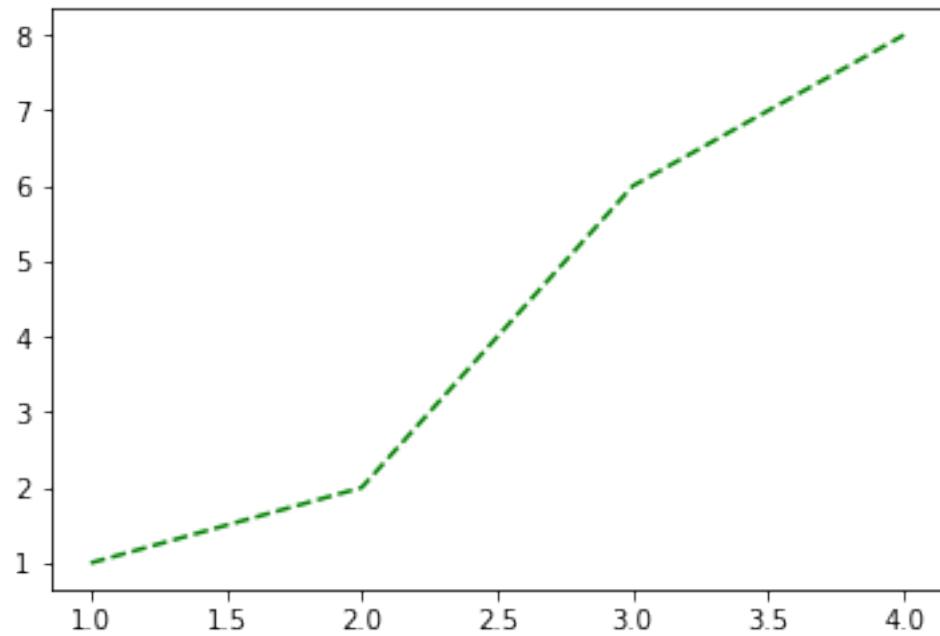
character	description
'_'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
'.'	dotted line style
'. '	point marker
', '	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
' '	vline marker
'—'	hline marker

2. 顏色



character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

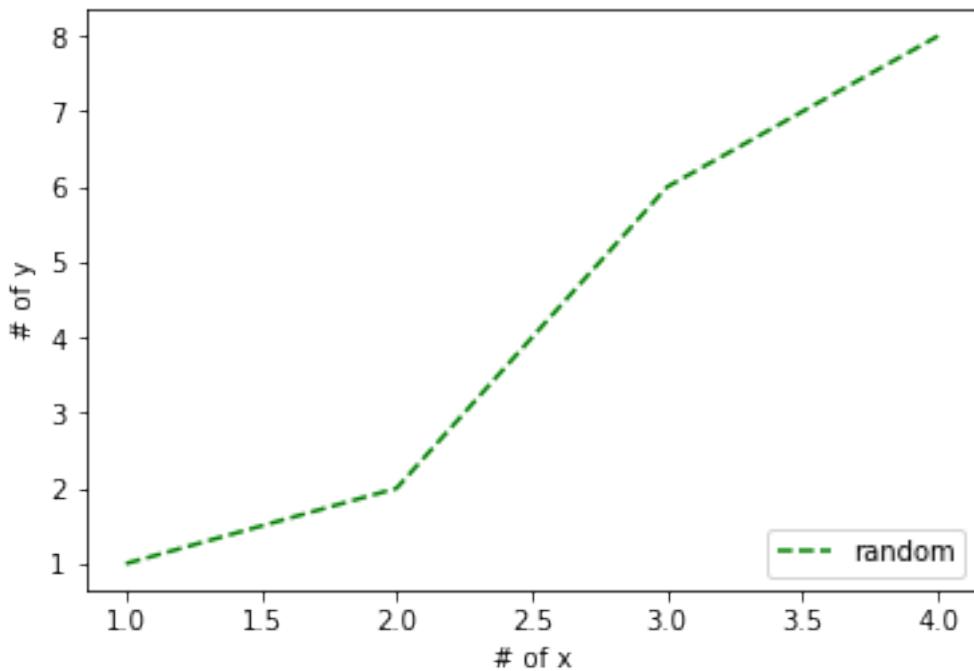
```
In [3]: x_list = [1, 2, 3, 4]
y_list = [1, 2, 6, 8]
# dash-line + green color
plt.plot(x_list, y_list, "--g")
plt.show()
```



1.9 加上標籤

1. 在 plot 的時候加上參數 label
2. 記得要用 legend 放在對的位置上，不然一樣看不到
3. 可以用 xlabel() 和 ylabel() 加上 x 軸標籤和 y 軸標籤

```
In [4]: x_list = [1, 2, 3, 4]
y_list = [1, 2, 6, 8]
# dash-line + green color
plt.plot(x_list, y_list, "--g", label = "random")
plt.legend(loc = "lower right")
plt.xlabel("# of x")
plt.ylabel("# of y")
plt.show()
```



1.10 (盡量不要) 中文顯示

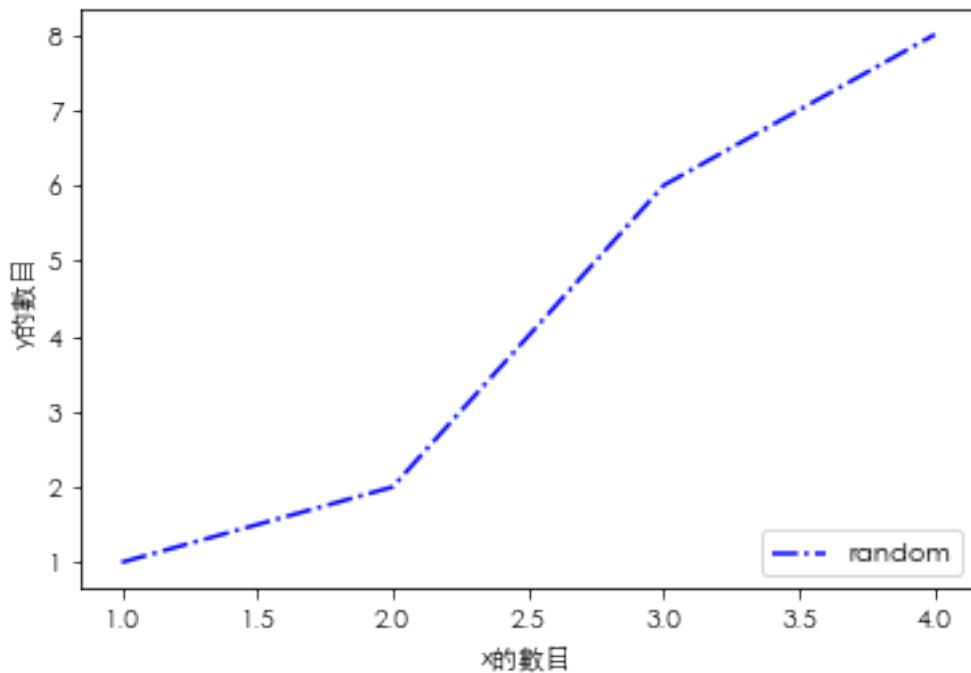
1. Matplotlib 預設只有外文字體，對於外文字體的顯示也比較漂亮一點
2. 如果你真的萬不得已，想要使用中文字體來顯示，要特別將預設的字體修改成中文字體



1.11 使用字體

STHeiti(蘋果黑體): Mac 內建字型，由於這裡 OS 是使用 Mac，所以我選擇了 Mac 內建的字型

```
In [5]: import matplotlib
        matplotlib.rcParams['font.sans-serif'] = 'STHeiti'
        x_list = [1, 2, 3, 4]
        y_list = [1, 2, 6, 8]
        # dash-dot + blue color
        plt.plot(x_list, y_list, "-.b", label = "random")
        plt.legend(loc = "lower right")
        plt.xlabel("x 的數目")
        plt.ylabel("y 的數目")
        plt.show()
```



1.12 鐵達尼號 + Seaborn

1. 使用 Pandas 先將鐵達尼號的資料讀取出來
2. Survived 欄位: 0 代表無法倖存的乘客 1 代表倖存的乘客
3. 以下是每個欄位代表的意思

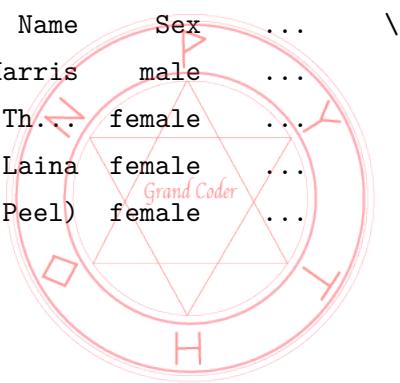


```
In [6]: import pandas as pd
# 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 20 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', 10)

df = pd.read_csv("train.csv", encoding = "utf-8")
df
```

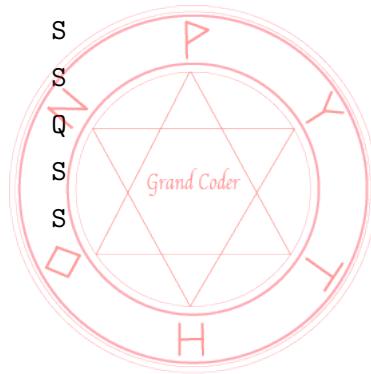
```
Out[6]:   PassengerId  Survived  Pclass \
0                 1         0      3
1                 2         1      1
2                 3         1      3
3                 4         1      1
4                 5         0      3
5                 6         0      3
6                 7         0      1
7                 8         0      3
8                 9         1      3
9                10        1      2
...
881               882        0      3
882               883        0      3
883               884        0      2
884               885        0      3
885               886        0      3
886               887        0      2
887               888        1      1
888               889        0      3
889               890        1      1
890               891        0      3
```

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



4		Allen, Mr. William Henry	male	...
5		Moran, Mr. James	male	...
6		McCarthy, Mr. Timothy J	male	...
7		Palsson, Master. Gosta Leonard	male	...
8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)		female	...
9	Nasser, Mrs. Nicholas (Adele Achem)		female	...
..		
881		Markun, Mr. Johann	male	...
882		Dahlberg, Miss. Gerda Ulrika	female	...
883		Banfield, Mr. Frederick James	male	...
884		Suthehall, Mr. Henry Jr	male	...
885	Rice, Mrs. William (Margaret Norton)		female	...
886		Montvila, Rev. Juozas	male	...
887		Graham, Miss. Margaret Edith	female	...
888	Johnston, Miss. Catherine Helen "Carrie"		female	...
889		Behr, Mr. Karl Howell	male	...
890		Dooley, Mr. Patrick	male	...

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S
5	0	330877	8.4583	NaN	Q
6	0	17463	51.8625	E46	S
7	1	349909	21.0750	NaN	S
8	2	347742	11.1333	NaN	S
9	0	237736	30.0708	NaN	C
..
881	0	349257	7.8958	NaN	S
882	0	7552	10.5167	NaN	S
883	0	C.A./SOTON 34068	10.5000	NaN	S
884	0	SOTON/OQ 392076	7.0500	NaN	S
885	5	382652	29.1250	NaN	Q
886	0	211536	13.0000	NaN	S
887	0	112053	30.0000	B42	Y



```

888      2        W./C. 6607  23.4500   NaN      S
889      0          111369  30.0000  C148      C
890      0          370376   7.7500   NaN      Q

```

[891 rows x 12 columns]

1.13 Import 函式庫以及利用 Notebook

- 由於 Seaborn 是基於 Matplotlib 的函式庫，所以正常寫法下，你一樣得在最後一行加上 `plt.show()` 印出圖形
- 在 Jupyter Notebook 裡使用的時候其實可以透過他提供給我們的簡便工具來簡化我們使用
- (重要，且只有 Notebook 可以用) 加上`%matplotlib inline`這行的話，你就可以在每次做圖的時候少打 `plt.show()`

In [7]: # 大家這裡習慣給他改名成 `sns`

```

import seaborn as sns
# 這行只有在 Jupyter Notebook 可以使用
%matplotlib inline
# sns 每次會跳出 remove_na 的 warning
# 我為了頁面的美化 不讓 warning 印出，但讀者不一定要過濾掉
import warnings
warnings.filterwarnings('ignore')

```

1.14 數量圖

- 數量圖是在類別做圖的時候一個很好用的圖示工具，統計各個類別分別有多少數量
- 使用 `seaborn.countplot`，我們通常選擇只設置其中一軸，另一軸就是數量
- 你可以藉由 `palette` 這個參數來選擇一下你喜歡的整體色系
- `palette`: https://seaborn.pydata.org/generated/seaborn.color_palette.html#seaborn.color_palette

In [8]: # 針對一個類別做數量圖，由於我們 `inline` 了 `matplotlib`，所以不需要 `plt.show()`

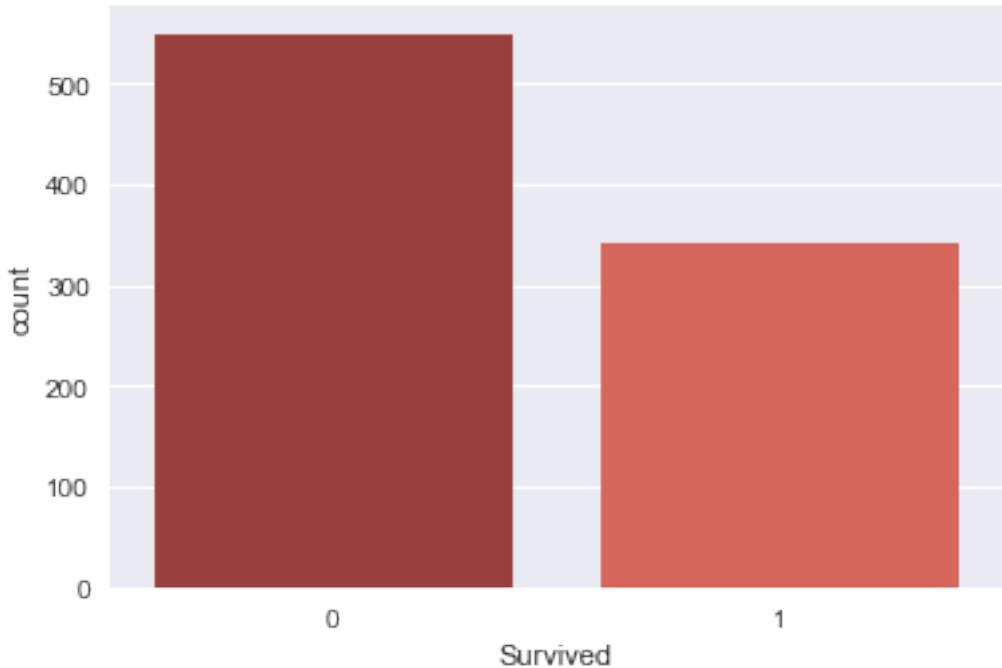
```

# 設置 x 軸的往上長的長條圖
sns.countplot(x = df["Survived"], palette = "Reds_d")

```

Out [8]: <matplotlib.axes._subplots.AxesSubplot at 0x114c73128>

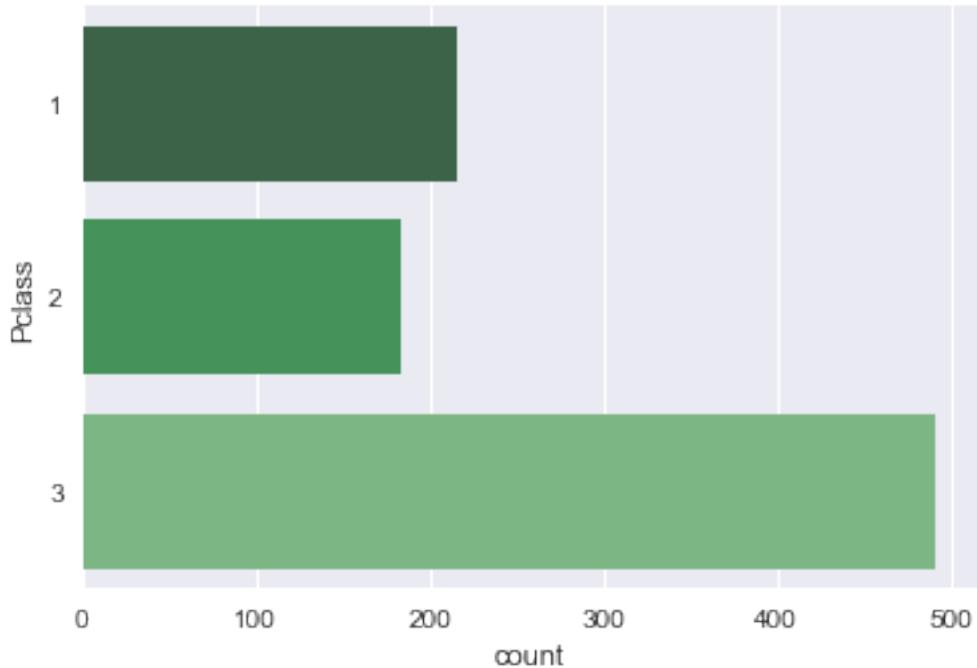




```
In [9]: # 針對一個類別做數量圖，由於我們 inline 了 matplotlib，所以不需要 plt.show()  
# 設置 y 軸的往右長的長條圖  
# 畫出各艙等的人數  
sns.countplot(y = df["Pclass"], palette = "Greens_d")
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x114c9d208>
```

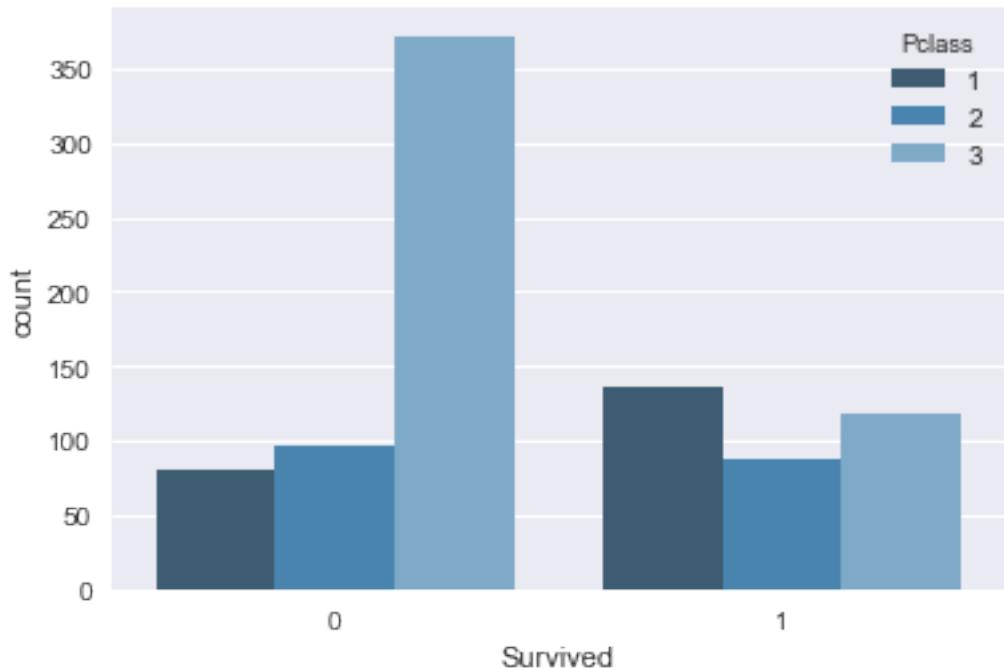




```
In [10]: # 結合上面兩個，把一個區域的長條加入第二個特徵，統計第二個特徵+第一個特徵的數目  
# hue 裡面放的就是第二個特徵  
# 你找到了其中一個相關性：第三艙等的存活率稍微低了一點  
sns.countplot(x = df["Survived"], hue = df["Pclass"], palette = "Blues_d")
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x114d068d0>
```





1.15 FacetGrid 網格圖 + 分布圖

1.15.1 FacetGrid 網格圖

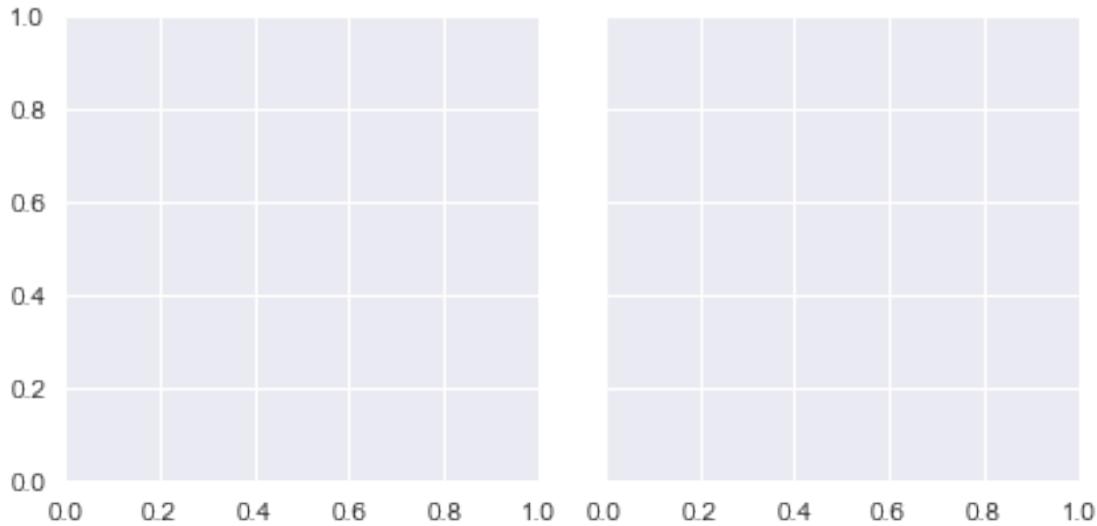
1. FacetGrid 網格圖是你可以固定變數的一種圖形，你設定的欄位會把所有可能的值拿出來，變出這麼多圖出來
2. 可以設定 row 和 col，你的圖的數量就等於 row 可能數 * col 可能數
3. 使用 FacetGrid 來創造，記得最後要給你的每個圖一個畫圖方式 (ex. 分布圖)
4. FacetGrid 是一個跟 pandas 非常友善的設計，你可以將 DataFrame 直接丟給第一個參數
5. 使用 FacetGrid 的主因之一是我們要做出分布圖 (x 軸是連續的)，但連續的東西我們沒辦法像上面的 countplot 分成三條來看，會看不出連續的趨勢

In [11]: # 對每一個網格裡的圖做下面的初始化

由於剛剛已經設定完 `data`，這裡直接給他欄位的名稱當 `x` 軸即可

```
fg = sns.FacetGrid(df, col = "Survived")
```





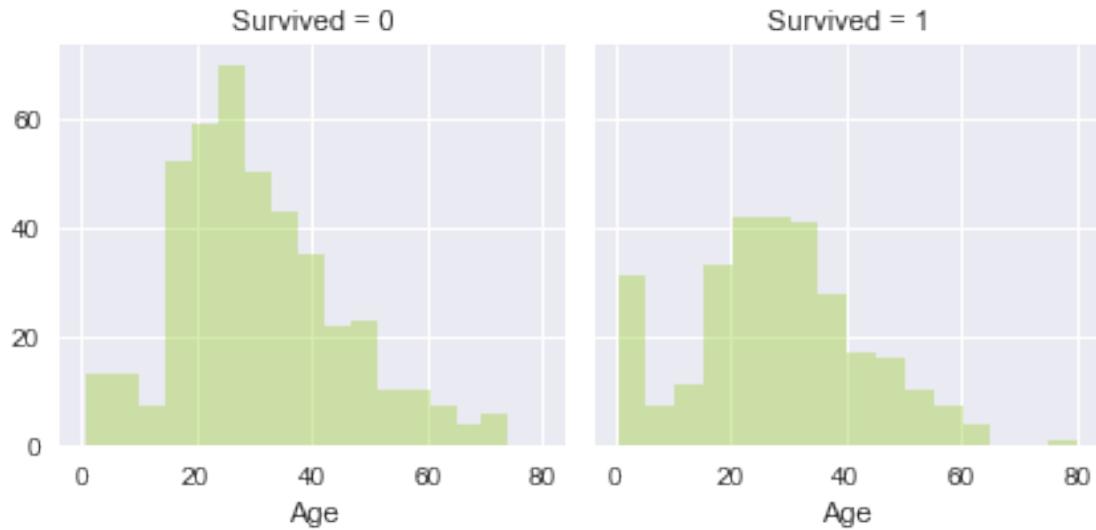
1.15.2 DistPlot 分布圖

1. 通常我們使用了網格圖以後，我們需要對每個圖初始化，我們常用的就是分佈圖
2. 分佈圖要傳入一個一個維度的群集，ex. 一個 Series
3. 他會把你傳入的一維群集轉換成 x 軸的區間
4. 跟 Countplot 一個很大的不同是我們的 x 軸是連續的，不是分類型的
5. 你可以給你的分布圖一個顏色: <https://matplotlib.org/users/colors.html>

```
In [14]: fg = sns.FacetGrid(df, col = "Survived")
fg.map(sns.distplot, 'Age', color = "yellowgreen", kde = False)
```

```
Out[14]: <seaborn.axisgrid.FacetGrid at 0x1152dea90>
```





SimplifyToTraditional

2018 年 7 月 5 日

1 簡繁轉換

1.1 介紹

我們在處理語言處理的時候，常常會遇到一個問題，

1.2 需要函式庫

1.2.1 OpenCC-Python

可以幫我們簡繁轉換

<https://github.com/yichen0831/opencc-python>

1.2.2 chardet(選用)

編碼猜測器，有時候我們真的猜不出來到底是 GBK 還是 windows 編碼的時候，可以選擇使用猜測器猜測一下

<https://github.com/chardet/chardet>

```
In [ ]: import os
        import jieba
        from opencc import OpenCC
        # import chardet

        # 我們要轉換的資料夾
        base_dir = "chinese_news"
        # 我們轉換完輸出的資料夾
        trans_dir = base_dir + "_trans"
        # 如果不存在，我們把資料夾做出來
        if not os.path.exists(trans_dir):
```



```

os.makedirs(trans_dir)

# os.walk 會回出一個 tuple, 而且會輪迴的走過 base_dir 的所有資料夾, 直到他找到檔案為止
for dirPath, dirNames, fileNames in os.walk(base_dir):
    # print(dirPath, dirNames, fileNames)
    # 準備一個簡繁轉換翻譯器
    openCC = OpenCC('s2tw')

    # 針對每一個檔案做出轉換
    for single_name in fileNames:
        if not single_name.startswith('.'):
            # 先不解碼 (rb), 後面再用簡體解碼
            # 當然你也可以在這裡直接使用 encoding="GBK"
            f = open(os.path.join(dirPath, single_name), "rb")
            content = f.read()

            try:
                content = content.decode("GBK")
                f.close()
                # 轉換 !!
                new_content = openCC.convert(content)
                # 把處理的文章路徑印出來
                # print(os.path.join(new_dirPath, single_name))
                # 把轉換好的繁體用 utf-8 編碼寫入文件中
                f = open(os.path.join(new_dirPath, single_name),
                         "w",
                         encoding="utf-8")
                f.write(new_content)
                f.close()
            except:
                # 有時候就是無法正確的轉換, 我們可以選擇直接放棄那一點點無法轉換的文章
                print("[Decode Error] 放棄此篇文章")

```

