作者序

近幾年間，大數據、機器學習、人工智慧等字眼已經是家喻戶曉，在各應用領域遍地開花，更有許多在不知不覺中已融入我們的生活。事實上，這三者間並非獨立，簡而言之，若把大數據當成食材，則機器學習可視為食譜，而人工智慧即為烹調後的佳餚。有了食材與食譜，也需要有稱手的廚具，正所謂「工欲善其事，必先利其器」，而這裡的廚具指的就是程式語言。在琳瑯滿目的程式語言中，Python 除了開源、免費、跨平台等特色外，更有許多辛勤耕耘的先進貢獻各種套件，開拓出一條條通往眾多應用領域的道路。因此，本書以 Python 為工具實作常見的機器學習模型，並將焦點放在如何妥善地運用模型與解讀結果。

本書是筆者擷取課程教材的一部分，再擴充編撰而成。如書名所示，內容聚焦在介紹機器學習的基礎知識的同時，也以實際數據動手嘗試，培養實務應用的技能。這中間也融入筆者在教與學、實務過程所得的經驗與技巧，並點出實作上容易犯錯的地方。同時，書中範例以.ipynb 的形式提供下載，可搭配內容一邊動手操作，以加深學習印象與提升興趣。而在每章最後的綜合範例與習題，解題步驟皆以程序化的方式呈現，不論是在教導與學習上皆能作為解題思維的樣板。

由於是側重在基礎與實務應用，因此書中盡量以視覺化圖表呈現模型的特性並進行比較，這些大多仰賴 scikit-learn 提供程式碼，望能以此讓讀者更直觀地了解機器學習模型的特色。本書適用有 Python 程式撰寫經驗，且用過 numpy、pandas、matplotlib 等套件的讀者，若能有機率統計、線性代數、微積分的基礎更佳，而設定的目標是達到機器學習的入門程度。儘管本書對模型的理論著墨不多，仍鼓勵有興趣的讀者繼續深入探討，定能發掘出更多奧妙之處。

在本書撰寫的過程中受到許多啟發，不管是來自諸位先進的文章、書籍或是網路資源皆帶給筆者莫大助益，能從簡潔且直觀的角度來詮釋模型，正如「寫中學、學中寫」。同時，也感謝編輯團隊對教材編輯、排版樣式等提供許多意見回饋，而家人的鼓勵與支持更是讓筆者無後顧之憂，非常感謝。

機器學習相關議題包含甚廣，本書也只是初窺門徑，難以面面俱到，有遺漏或錯誤之處也歡迎大家協助指教與討論。

林英志

2021 年 1 月

CHAPTER

7

機器學習應用

機器學習應用

最近幾年的人工智慧（AI）已經成為最潮的字眼，而機器學習是 AI 領域中最重要且成果豐碩的分支項目，它的相關應用如雨後春筍般湧現在各種業務部門，也不斷產生更強大與更簡便的工具、應用框架以及大量研究論文。推波助瀾下，更讓機器學習深化結合到各應用場景，形成一波波自動化與智慧化的浪潮。

機器學習的「學習」是透過外界刺激來建立與改善數學模型，從中抽取出隱藏的規律或模式，以推論未來並輔助決策。在之前的章節已經看過許多監督與非監督式學習方法，也了解如何調校超參數及評估學習模型的效能。本章將綜合這些策略與技巧，應用到探討 7-1 節的自然語言處理（Natural Language Processing、NLP）以及 7-2 節的序列資料處理。

7-1 自然語言處理

自然語言顧名思義是我們日常生活中使用到的各種語言形式，包括文字、語音等，而文字再細分下去諸如新聞、部落格文章、聊天訊息、論壇留言、FB 或 Twitter 短貼文、甚至是古代中國使用的文言文與詩詞等也都屬於自然語言的範疇。根據維基百科的解釋，自然語言處理（NLP）是計算機科學以及人工智慧的子領域，專注在如何讓計算機處理並分析大量（人類的）自然語言數據。NLP 常見的應用與挑戰有語音辨識、自然語言理解、機器翻譯以及自然語言的生成等。

7-1-1 基本操作

文字敘述屬於非結構化資料，處理過程相當繁瑣且各式各樣的處理手法也與之後的分析效能息息相關。儘管如此，在前處理階段還是有些基本且常用的操作，熟悉這些操作技巧能讓自然語言處理的流程更加順暢。

從檔案或是透過解析網頁取得非結構化的文件資料，再進行基本的清理工作（如去掉句子頭尾的空白字元、字元或字串的替換等）後，接著就開始針對自然語言的處理程序。因為英文文句中的詞彙間有空白隔開，很容易就能進行斷字，因此接下來會考慮移除標點符號。除了在一些情況下的標點符號可能帶有資訊，能產生有用的特徵外（例如：問號表示為疑問句、句號用在結構完整的句子結束後、驚嘆號用於驚訝與感嘆的語句以加強句子的情緒等），大部分時候可直接移除。

移除標點符號的直觀做法是透過字串處理的 replace()直接以空白取代標點符號，此舉雖然簡單且可挑選想要移除的標點符號，缺點除了難以詳列所有標點符號外，執行速度也比較慢。在底下範例中先產生一個內含三個字串的串列，接著建構一個字典用所有標點符號的 Unicode 為鍵（key），且以 None 為值（value），再透過 translate()依照字典內容快速進行轉換。

範例程式 ex7-1-1_2.ipynb

[1]:
```
texts = ['Pikachu is a short, Electric-type Pokémon
introduced in Generation I!!!',
         'It is covered in yellow fur with two horizontal
brown stripes on its back. It has a small mouth, long,
pointed ears with black tips, and brown eyes.',
         'It evolves from Pichu when leveled up with high
friendship and evolves into Raichu.']
len(texts)
```

[1]:
```
3
```

[2]:
```
import unicodedata
import sys

punctuation = dict.fromkeys(i for i in
         range(sys.maxunicode) if
         unicodedata.category(chr(i)).startswith('P'))
texts_no_punct = [s.translate(punctuation) for s in
                                                texts]
texts_no_punct
```

[2]:
```
['Pikachu is a short Electrictype Pokémon introduced in
Generation I',
 'It is covered in yellow fur with two horizontal brown stripes
on its back It has a small mouth long pointed ears with black
tips and brown eyes',
 'It evolves from Pichu when leveled up with high friendship
and evolves into Raichu']
```

再來是進行記號化（tokenization），尤其是詞或句子的記號化，經常用來將文句轉化以建構有用特徵的起手式。由於我們已經移除所有標點符號，再加上英文文句以空白分隔的特性，所以詞的記號化可簡單利用 split()切割空白字元來達成。這裡介紹一個常見方法是透過自然語言處理工具包 NLTK（Natural Language Toolkit

for Python），有許多強大的文句操作功能。在底下範例程式中第一次匯入 nltk 時，需要耗費一點時間下載一些文件資料，而匯入使用 nltk 前也要先安裝。

```
[3]:
import nltk

# 第一次載入 nltk時，要先下載一些文件(需要等一會)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# 下載一組停止詞
nltk.download('stopwords')
```

```
[3]:
True
```

```
[4]:
from nltk.tokenize import word_tokenize

words_lst = [word_tokenize(t) for t in texts_no_punct]
print(words_lst[0])
```

```
[4]:
['Pikachu', 'is', 'a', 'short', 'Electrictype', 'Pokémon',
'introduced', 'in', 'Generation', 'I']
```

```
[5]:
from nltk.tokenize import sent_tokenize
sent_tokenize(texts[1])
```

```
[5]:
['It is covered in yellow fur with two horizontal brown stripes
on its back.',
 'It has a small mouth, long, pointed ears with black tips,
and brown eyes.']
```

隨手瀏覽幾個短句或文章，不難發現其中有經常出現的字或詞，比如英文的 i、a、the、and 等，中文裡也有「我」、「和」、「的」等。這些資訊含量少但出現頻率高的詞稱為停止詞（stop word）。NLTK 中有一張常用停止詞的列表，可用來比對並移除停止詞，要注意的是這個列表只針對小寫字母。

```
[6]:
from nltk.corpus import stopwords

stop_words = stopwords.words('english')
print('停止詞：', stop_words[:5])
print('停止詞數量 =', len(stop_words))

for i in range(len(words_lst)):
```

```
    words_lst[i] = [w for w in words_lst[i] if w not in
                                                stop_words]

print(texts_no_punct[0])
print(words_lst[0])
```

```
停止詞： ['i', 'me', 'my', 'myself', 'we']
停止詞數量 = 179
Pikachu is a short Electrictype Pokémon introduced in
Generation I
['Pikachu','short', 'Electrictype', 'Pokémon', 'introduced',
'Generation', 'I']
```

若要進行文句或文章間的詞彙比對，則能藉由辨識與移除詞綴（affixes）的方式提取出詞幹（word stemming），這是一個將字詞轉變為字根形式的過程，以保留詞彙的原意。比方說在上述範例裡的 introduced 與 introduce 都有詞幹 introduc，代表這兩個詞雖然不同，但表示的概念卻相通。透過詞幹提取可將詞轉換成可讀性稍差一點的形式，但比較適合用來在各文章間進行比對。NLTK 提供被廣泛使用的波特詞幹提取演算法（Porter stemming algorithm），可移除或取代字詞內常見的前後綴以得到詞幹。

```
from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()
stem_lst = []

for words in words_lst:
   stem_lst.append([porter.stem(w) for w in words])

print(words_lst[0])
print(stem_lst[0])
```

```
['Pikachu', 'short', 'Electrictype', 'Pokémon',
'introduced', 'Generation', 'I']
['pikachu', 'short', 'electrictyp', 'pokémon', 'introduc',
'gener', 'I']
```

在自然語言處理的前期階段還有一個相當繁瑣且重要的工作，就是為每一個字詞加上詞性標籤（Part-of-Speech tag、POS tag），可透過 NLTK 已訓練好的詞性標記器 pos_tag()來對單詞的詞性進行標記，而標記後的結果是陣列格式，陣列裡的每

個元素是一個元組，包含有詞與其詞性。此外，NLTK 使用的是 PennTreebank 詞性標籤，例如在底下的範例中，NNP 代表專有名詞，JJ 是形容詞，而 VBD 則是動詞的過去式。

```
[8]:
1 from nltk import pos_tag
2
3 words_tag_lst = [pos_tag(w) for w in words_lst]
4 print(words_tag_lst[0])
```

```
[8]:
[('Pikachu', 'NNP'), ('short', 'JJ'), ('Electrictype', 'NNP'), ('Pokémon', 'NNP'), ('introduced', 'VBD'), ('Generation', 'NNP'), ('I', 'PRP')]
```

```
[9]:
1 # 搜尋特定詞類
2 [w for w, tag in words_tag_lst[0] if tag in ['NNP']]
```

```
[9]:
['Pikachu', 'Electrictype', 'Pokémon', 'Generation']
```

通常 NLTK 提供的詞性標記器對於英文文章已能有不錯的標記效果，但如果是特定的專門領域的標註效果可能大打折扣，此時 NLTK 也提供能訓練自己使用的標籤器。然而，在訓練前要先準備一個很大的語料庫（corpus），並提供每一個詞的詞性，建構過程需要投入大量心力。

接著能以詞性的特徵向量來表達一個字句、一份推文或是一篇文章，也就是若文句內有該詞性則特徵值為 1，否則特徵值為 0。這個功能可透過 scikit-learn 提供的 MultiLabelBinarizer()來達成，底下是範例程式：

```
[10]:
1 from sklearn.preprocessing import MultiLabelBinarizer
2
3 tag_lst= []
4 for words_tag in words_tag_lst:
5     tag_lst.append([tag for word, tag in words_tag])
6
7 mlb = MultiLabelBinarizer()
8 mlb.fit_transform(tag_lst)
```

```
[10]:
array([[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0],
       [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1]])
```

```
[11]:
1 # 顯示特徵名稱
2 mlb.classes_
```

```
[11]:
array(['CD', 'IN', 'JJ', 'NN', 'NNP', 'NNS', 'PRP', 'RB',
'VBD', 'VBP', 'VBZ'], dtype=object)
```

```
[12]:
1 data = mlb.fit_transform([{'皮卡丘', '雷丘'}, {'小火龍',
2                                          '噴火龍'}, {'傑尼龜'}])
3 print(data)
4 list(mlb.classes_)
```

```
[13]:
[[0 0 0 1 1]
 [0 1 1 0 0]
 [1 0 0 0 0]]
['傑尼龜', '噴火龍', '小火龍', '皮卡丘', '雷丘']
```

7-1-2 詞袋模型

在經過前一節的斷詞、移除標點符號、提取詞幹以及標註詞性之後，經常會進行向量化的動作，而在介紹向量化之前先來了解詞袋模型（Bag-of-Words、BoW）。這是將文章轉成特徵常見的方法之一，模型忽略文章內詞與詞間的上下文關係，只考慮每個詞的權重，而權重最簡單的設定即是該詞的出現次數。換言之，BoW 在斷詞之後以每個詞作為特徵，再以該詞在每篇文章樣本中的出現次數為該樣本的特徵值，而將某篇文章的所有詞與對應詞頻放在一起即為文章的向量化。要特別注意的是 BoW 在使用上有其局限性，因為它只考慮詞頻而漠視上下文中詞與詞的關係，因此會遺失一部分文章的語義。Scikit-learn 實作的 CountVectorizer()能方便地將文章內的詞轉換為詞頻矩陣，底下用參考範例具體說明：

```
[14]:
1  from sklearn.feature_extraction.text import
2  CountVectorizer
3
4  corpus = ['This is a small document.',
5            'Pokémon document is the second document.',
6            'Pikachu is a short and Electric-type Pokémon.
7                                      It has a small mouth.',
8            'Is this the first document?']
9
10 vectorizer = CountVectorizer(stop_words='english')
11 X = vectorizer.fit_transform(corpus)
12 print(vectorizer.get_feature_names())
```

```
[14]: ['document', 'electric', 'pikachu', 'pokémon', 'second',
'small', 'type']
```

```
[15]:
1 import pandas as pd
2
3 df_vec = pd.DataFrame(X.toarray(),
4                       columns=vectorizer.get_feature_names())
5 df_vec
```

[15]:

	document	electric	pikachu	pokémon	second	small	type
0	1	0	0	0	0	1	0
1	2	0	0	1	1	0	0
2	0	1	1	1	0	1	1
3	1	0	0	0	0	0	0

```
[16]:
1 vectorizer2 = CountVectorizer(stop_words='english',
2                                ngram_range=(2, 3))
3 X2 = vectorizer2.fit_transform(corpus)
4 print(vectorizer2.get_feature_names())
```

```
[16]: ['document second', 'document second document', 'electric
type', 'electric type pokémon', 'pikachu small', 'pikachu
small electric', 'pokémon document', 'pokémon document
second', 'second document', 'small document', 'small
electric', 'small electric type', 'type pokémon']
```

使用 CountVectorizer()時可透過設定一些參數讓整個轉換過程更順暢，例如：

- 除了以字串或字串串列，透過參數 input 設定也能直接輸入檔案。
- 參數 lowercase 設定是否要轉換為小寫字母，預設值為是。
- 設定參數 stop_words 可搭配內建或自訂的列表來移除停止詞，由於內建的停止詞包含許多來源（可參考 https://github.com/igorbrigadir/stopwords）且 nltk 僅是其中之一，因此在前例中可發現 first 被當成停止詞移除，但是 first 並沒有在 nltk 的停止詞庫裡。此外，也可依照詞在文章間出現的次數定義停止詞，比方說設定 max_df = 0.8 代表在超過八成文章中出現的詞為停止詞，而如果設定 max_features = 5 則只挑選詞頻最大的前五個。

- 不僅是單詞，也可考慮雙詞（稱為二元，2-gram）或甚至多詞（多元，multi-gram）的組合。前例中設定 ngram_range = (2, 3)即是回傳所有二元與三元的特徵。
- 運用參數 vocabulary 可將要處理的詞彙侷限在自訂的列表中。

除此之外，實際分析時的文章長度遠超過前述範例，即使有移除停止詞，BoW 為每個詞產生的特徵數量仍然相當驚人，若是直接儲存將會占據相當多記憶體空間。然而，每篇文章僅包含少部分詞彙，使得詞頻矩陣中大多數的值皆為 0，稱之為稀疏矩陣（sparse matrix），此時只記錄非零值並搭配特殊存取方式可大幅降低儲存空間，而 CountVectorizer()預設即是輸出稀疏矩陣。

在前述的 BoW 中簡單以詞頻來代表詞的權重，但這似乎不太客觀，畢竟一個詞出現在許多文章中，則該詞對個別文章的重要性就會降低；反觀，一個詞在文章內的出現頻率越多，通常意謂著該詞對該文章的重要性越高。比方說，若 pokemon 很常出現，則該篇文章描述寶可夢相關內容的機會就高，而非僅僅舉例說明。因此，「詞頻-逆文件頻率」（Term Frequency-Inverse Document Frequency、TF-IDF）即綜合這兩個統計量，用以描述詞的權重。具體而言，TF-IDF 由以下兩個部分組成：

- 詞頻（Term Frequency、TF）：指的是某個詞 w 在特定文章 doc 中出現的頻率，以底下的公式 $tf(w, doc)$呈現，其中 $count(w, doc)$是 w 在 doc 中出現的次數，而 $size(doc)$則是文章的總詞數。這個比值的作法是對詞的出現次數進行正規化，以避免偏向較長的文章，畢竟同一個詞彙在長文章裡可能會比短文章有更高的出現次數，但不一定與該詞的重要性有關。

$$tf(w,doc)=\frac{count(w,doc)}{size(doc)}$$

- 逆文件頻率（Inverse Document Frequency、IDF）：用以量測一個詞語在各文件的重要性，計算公式 $idf(w)$如下，其中 n_{doc} 是文件數量，$df(w)$則是有包含 w 的文件數量。IDF 公式有幾個大同小異的定義，而這裡所列的是 scikit-learn 實作的方式。公式中，在分母加上 1 是考量訓練集詞語的 $df(w)$ 為 0 的情況，取對數則為了避免頻率過低的詞語被賦予太大的權重，而整個式子加 1 則是為了不讓 $idf(w)$變成 0。

$$idf(w)=\ln\frac{1+n_{doc}}{1+df(w)}+1$$

將上述兩個統計量組合成詞頻-逆文件頻率 $tf\text{-}idf(w, doc) = tf(w, doc) \times idf(w)$，用以評估一個詞對一份文件的重要性。由公式可看出 TF-IDF 主要的想法是某個詞在某文件有較大詞頻（即 TF 較大），且該詞也很少出現在其他文章中（即 IDF 較大），則認為這個詞具有很好的文件代表性，其重要性也就越高。因此，TF-IDF 傾向於過濾在文件中不常出現或者經常在各文件中看到的詞語。

不論是資訊檢索、文字探勘（text mining）或自然語言處理，TF-IDF 都是一種常見的加權策略，且延伸的各種加權形式也常應用於搜尋引擎，作為評估用戶與查詢文件之間的相關程度。透過 TF-IDF 計算得到的詞語重要性，隨著在某文件的詞頻成正比，而與在各文件出現的頻率成對數反比。Scikit-learn 提供兩種計算 TF-IDF 方法，第一種是在使用 CountVectorizer()進行向量化後，再套用 TfidfTransformer()計算；另一種則是直接用 TfidfVectorizer()進行處理。底下是參考範例：

[17]:
```
from sklearn.feature_extraction.text import
                                        TfidfVectorizer

tf_idf = TfidfVectorizer(stop_words='english')
X = tf_idf.fit_transform(corpus)
print(tf_idf.get_feature_names())

# 輸出每個詞的 IDF 值
tf_idf.idf_
```

[17]:
```
['document', 'electric', 'pikachu', 'pokémon', 'second',
'small', 'type']
array([1.22314355, 1.91629073, 1.91629073, 1.51082562,
1.91629073, 1.51082562, 1.91629073])
```

[18]:
```
df_tf_idf = pd.DataFrame(X.toarray(),
                         columns=tf_idf.get_feature_names())
df_tf_idf
```

[18]:

	document	electric	pikachu	pokémon	second	small	type
0	0.629228	0.000000	0.000000	0.000000	0.000000	0.777221	0.000000
1	0.707981	0.000000	0.000000	0.437249	0.554595	0.000000	0.000000
2	0.000000	0.485461	0.485461	0.382743	0.000000	0.382743	0.485461
3	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

[19]:

```
import numpy as np
from sklearn.preprocessing import normalize

index = ['document', 'pokémon', 'second']
tf =np.array(df_vec.loc[1,index])/df_vec.loc[1,:].sum()
idf = np.array([tf_idf.idf_[
  tf_idf.get_feature_names().index(w)] for w in index])
tf_idf_doc1 = tf*idf
normalized = normalize(tf_idf_doc1.reshape(1,-1),
                                        norm='l2').ravel()
dct = {'TF': tf,
       'IDF': idf,
       'TF-IDF': tf_idf_doc1,
       '正規化': normalized}
df_doc1 = pd.DataFrame(dct, index=index)
df_doc1
```

[19]:

	TF	IDF	TF-IDF	正規化
document	0.50	1.223144	0.611572	0.707981
pokémon	0.25	1.510826	0.377706	0.437249
second	0.25	1.916291	0.479073	0.554595

接著以上述範例的文句 1 的 document 這個字，逐步檢驗它在文句 1 裡的 TF-IDF 是如何被計算出來。首先，從下列計算式可知這個字的 TF = 0.5 且 IDF = 1.223，相乘之後得到 TF-IDF 為 0.612；其次，分別對文句 1 的三個詞語進行計算，得到 TF-IDF 向量[0.612, 0.378, 0.479]；最後，套用 L2 正規化將向量長度變成 1。

$$tf('document', doc_1) = \frac{2}{4} = 0.5, \quad idf('document') = \ln\frac{1+4}{1+3} + 1 = 1.223$$

由上述範例可發現只出現在一個字句的詞語 IDF 值（如 pikachu）比出現在多個字句的詞語（如 documnet）要高，而前者對於第 2 字句的重要性（TF-IDF）卻低於後者對於其他字句的重要性，主要原因在於第 2 字句的詞語較多，使得詞頻相對較小的緣故。事實上，比較不同字句或文章內詞的 TF-IDF 值並無太大意義，因為上述是經過正規化的結果，預設以歐幾里德範數（Euclidean norm、L2 norm）作正規化，所以一個字句所有特徵值的平方和為 1，而若要計算兩個字句或文章的餘弦相似度（cosine similarity）可直接拿兩者的特徵向量進行內積。

7-1-3 情感分析

在眾多 NLP 的應用領域中，情感分析（sentiment analysis）相當實用且受歡迎，主要被用來分析短文句或文章的極性（polarity），即正面或負面評價。在社交媒體盛行的全球村時代，每天皆湧入大量的群眾意見與評論，而這些無疑是了解群眾想法與需求的最佳素材，情感分析正是用來分析這些素材的方法之一，因此廣泛地應用到商品行銷、評論挖掘、電影推薦、投票傾向等。

這裡將運用前兩個小節學到的 NLP 基本操作進行「電影評論數據集」的情感分析，這份評論數據來自網路電影資料庫（Internet Movie Database、IMDb），可由網址（http://ai.stanford.edu/~amaas/data/sentiment/）取得原始評論檔案。該數據集包含有 50,000 筆電影評論與其極性，其中有一半評論的極性標記為正（positive），另一半則為負（negative）。在評價最高為 10 分的前提下，正極性的評論代表其分數超過（含）7 分，而若低於（含）4 分則為負極性，其餘介於中間極性的評論不包含在數據集內，且每個電影的評論不會超過 30 筆，這是因為對同一部電影會傾向有相關的評論與評價。

底下的範例在讀取 IMDb 數據集後，只隨機取出 5,000 筆評論進行分析，目的是為了節省運算時間。從範例中可發現評論裡可能有除了英文字母與數字外的其他字元，必須先清理過後才能進入 NLP 的分析程序。

範例程式 ex7-1-3.ipynb

```
[1]:
 1 import numpy as np
 2 import pandas as pd
 3
 4 size = 5000  # 只取部分樣本，節省運算時間
 5
 6 df = pd.read_csv('IMDb_dataset.csv')
 7 df = df.sample(n=size, random_state=0)
 8 df.reset_index(inplace=True, drop=True)
 9 print(df['sentiment'].value_counts())
10 df.head(3)
```

```
[1]:
negative    2553
positive    2447
Name: sentiment, dtype: int64
```

	review	sentiment
0	John Cassavetes is on the run from the law. He...	positive
1	It's not just that the movie is lame. It's mor...	negative
2	Well, if it weren't for Ethel Waters and a 7-y...	negative

```
[2]:
1 from sklearn.preprocessing import LabelEncoder
2
3 le = LabelEncoder().fit(df['sentiment'])
4 df['sentiment'] = le.transform(df['sentiment'])
5 df.head(3)
```

[2]:

	review	sentiment
0	John Cassavetes is on the run from the law. He...	1
1	It's not just that the movie is lame. It's mor...	0
2	Well, if it weren't for Ethel Waters and a 7-y...	0

```
[3]:
1 # 評論包含 HTML 標籤、標點符號以及其他非字母字元(e.g., (, [)
2 df.loc[0, 'review'][-150:-100]
```

```
[3]:
'ch needed.<br /><br />All the three principle char'
```

為了將評論資料清理乾淨，首先利用 BeautifulSoup 套件移除不帶有用語意資訊的 HTML 標籤。為了簡單起見，接著將移除中括號與在其內的敘述、標點符號以及其他特殊字元，而由於要移除的目標有點多，所以底下使用正規表示式（regular expression）來處理。在底下的程式碼中，第 10 行的正規表示式「\[[^]]*\]」目的在於表示所有中括號及在括號內的敘述；而第 16 行的「r'[^a-zA-Z0-9\s]'」代表的是除了大小寫英文字母、數字與空格外的所有字元，最前面加上 r 是因為字串裡有反斜線（跳脫字元）的緣故。關於 Python 的 re 套件運用，除了可以參考官網資訊外，pythex 網站（https://pythex.org/）也能測試正規表示式的結果是否正確。

```
[4]:
1 from bs4 import BeautifulSoup
2 import re
3
4 def remove_noise(text):
5     # 移除 HTML 標籤
6     bs = BeautifulSoup(text, "html.parser")
7     text = bs.get_text()
```

```
 8
 9     # 移除中括號內的文字
10     text = re.sub('\[[^]]*\]', '', text)
11
12     # 將句點取代為空格
13     text = text.replace('.', ' ')
14
15     # 移除特殊字元、標點符號
16     pattern = r'[^a-zA-Z0-9\s]'
17     text = re.sub(pattern, '', text)
18
19     return text
20
21 df['review'] = df['review'].apply(remove_noise)
22 df.loc[0, 'review'][-150:-100]
```

```
[4]: ' the time was much needed All the three principle '
```

接下來就能進行在前面小節的基本操作裡提到的提取詞幹、移除停止詞，其實移除停止詞的程序也可以到計算 TF-IDF 時透過 TfidfVectorizer()的參數 stop_words 設定停止詞列表即可，而提前進行能稍微加速後續建模的動作。再來則是建立機器學習模型前常用的切割數據集成訓練與測試集，由於一開始已經是從原始評論數據及裡隨機挑選 5,000 筆評論，所以這裡就簡單取前面 80%為訓練集，其餘為測試集。底下範例也分別列出切割後的正、負評論筆數。

```
[5]:  1 from nltk.stem.porter import PorterStemmer
      2
      3 porter = PorterStemmer()
      4
      5 # 提取詞幹
      6 def get_stemming(text):
      7     text = ' '.join([porter.stem(w) for w in
      8                                          text.split()])
      9     return text
     10
     11 df['review'] = df['review'].apply(get_stemming)
     12 df.loc[0, 'review'][:50]
```

```
[5]: 'john cassavet is on the run from the law He is at '
```

[6]:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stopword_lst = stopwords.words('english')

# 移除停止詞
def remove_stopwords(text):
    tokens = word_tokenize(text)
    tokens = [token.strip() for token in tokens]
    filtered_tokens = [token for token in tokens if
                       token.lower() not in stopword_lst]
    filtered_text = ' '.join(filtered_tokens)

    return filtered_text

df['review'] = df['review'].apply(remove_stopwords)
df.loc[0, 'review'][:50]
```

[6]:

```
'john cassavet run law bottom heap see negro sidney'
```

[7]:

```
# 切割訓練集、測試集
train_size = 0.8

X_train = df.loc[:size*train_size-1, 'review'].values
y_train = df.loc[:size*train_size-1, 'sentiment'].values
X_test = df.loc[size*train_size:, 'review'].values
y_test = df.loc[size*train_size:, 'sentiment'].values

dct = {'總筆數': [X_train.shape[0], X_test.shape[0]],
       '正評論筆數': [y_train.sum(), y_test.sum()],
       '負評論筆數': [(y_train==0).sum(),
                                    (y_test==0).sum()]}
pd.DataFrame(dct, index=['訓練集', '測試集'])
```

[7]:

	總筆數	正評論筆數	負評論筆數
訓練集	4000	1954	2046
測試集	1000	493	507

我們打算以邏輯斯迴歸模型進行正、負評論的分類任務，而在套用模型之前要先透過計算 TF-IDF 進行向量化，再加上也要藉由網格搜尋找出合適的超參數組合，所以利用管道化打包這些程序，底下是範例程式。

```
[8]:
from sklearn.feature_extraction.text import
                                        TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

tf_idf = TfidfVectorizer()

pipe = Pipeline([('tfidf', tf_idf),
                 ('clf', LogisticRegression())])

param_grid = [{'tfidf__ngram_range': [(1, 1)],
               'tfidf__stop_words': ['english', None],
               'tfidf__use_idf':[True],
               'tfidf__norm':['l1', 'l2'],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': np.logspace(-2, 2, 10)},
              {'tfidf__ngram_range': [(1, 1)],
               'tfidf__stop_words': ['english', None],
               'tfidf__use_idf':[False],
               'tfidf__norm': ['l1', 'l2'],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': np.logspace(-2, 2, 10)},
              ]

gs = GridSearchCV(pipe, param_grid, n_jobs=-1
                  scoring='accuracy', cv=5, verbose=1)
gs.fit(X_train, y_train)
```

```
[8]:
Fitting 5 folds for each of 160 candidates, totalling 800 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8
                                        concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   19.4s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   49.0s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 800 out of 800 | elapsed:  1.5min
                                                    finished
```

```
[9]:  1  print('Best parameters:', gs.best_params_)
```

```
[9]:  Best parameters: {'clf__C': 4.6415888336127775,
      'clf__penalty': 'l2', 'tfidf__ngram_range': (1, 1),
      'tfidf__norm': 'l2', 'tfidf__stop_words': None,
      'tfidf__use_idf': True}
```

```
[10]:  1  print('Train accuracy:', gs.best_score_)
       2  clf = gs.best_estimator_
       3  print('Test accuracy:', clf.score(X_test, y_test))
```

```
[10]:  Train accuracy: 0.8512500000000001
       Test accuracy: 0.866
```

從上述結果可知最佳參數的組合是使用 NLTK 的停止詞列表，TF-IDF 向量化時要用 IDF 進行加權且用 L2 做正規化，再搭配邏輯斯迴歸對懲罰項的設定，進行建模後對測試集可得到 0.866 的分類準確率。

接著以類似程序嘗試進行中文文句的情感分析，中文短文句與停止詞是從網路上下載的開放資料（https://github.com/UDICatNCHU/UdicOpenData），而對中文 NLP 相當繁瑣的斷字分詞動作則交給知名的 Jieba 工具來進行。原始短文共有 34,880 筆，這裡也只挑選部分短文進行分析以節省運算時間。「範例程式 ex7-1-3.ipynb」使用邏輯斯迴歸針對測試集的分類準確率可超過八成。

此外，Python 也有許多方便的工具包能讓整個文句情感分析的過程更加簡便，例如底下範例使用 TextBlob（官網 https://textblob.readthedocs.io/）對英文字句進行情感分析會得到兩個數字，第一個為情感性，變化範圍是[-1, 1]，且-1 為完全負面，1 則是完全正面；第二個數字代表主觀性程度，介於[0, 1]之間，越接近 1 表示主觀性越強。在範例中的第二句因為有單字 terrible，使得情感極性達到-1，也讓兩段文句的情感偏向負面。

```
[18]:  1  from textblob import TextBlob
       2
       3  text = 'Pokémon is a great game. Gigantamax Pikachu is
       4                                                terrible.'
       5
       6  blob = TextBlob(text)
       7  print(blob.sentences[0].sentiment)
       8  print(blob.sentences[1].sentiment)
```

```
[18]: Sentiment(polarity=0.2, subjectivity=0.575)
      Sentiment(polarity=-1.0, subjectivity=1.0)

[19]: 1 blob.sentiment

[19]: Sentiment(polarity=-0.2, subjectivity=0.71)
```

至於中文文章分析，中研院 CKIP LAB（https://ckip.iis.sinica.edu.tw/demo/）的線上展示網站提供許多中文 NLP 應用，像是詞性標注、中文斷詞、情感分析等。此外，SnowNLP（官網 https://github.com/isnowfy/snownlp）工具包也對中文文章分析實作各種功能，以下是中文情感分析的範例。

```
[20]: 1 from snownlp import SnowNLP
      2
      3 text = u"訓練家小智屢敗屢戰，總算獲得聯盟冠軍。"
      4 s = SnowNLP(text)
      5 for sen in s.sentences:
      6     print(sen, '-> 表達正面情感的機率：',
      7                                 SnowNLP(sen).sentiments)

[20]: 訓練家小智屢敗屢戰 -> 表達正面情感的機率： 0.010584349812285954
      總算獲得聯盟冠軍 -> 表達正面情感的機率： 0.2698977405428781
```