自然語言處理

第 8 章 文本分類

Text Classification

講師：紀俊男

本章大綱

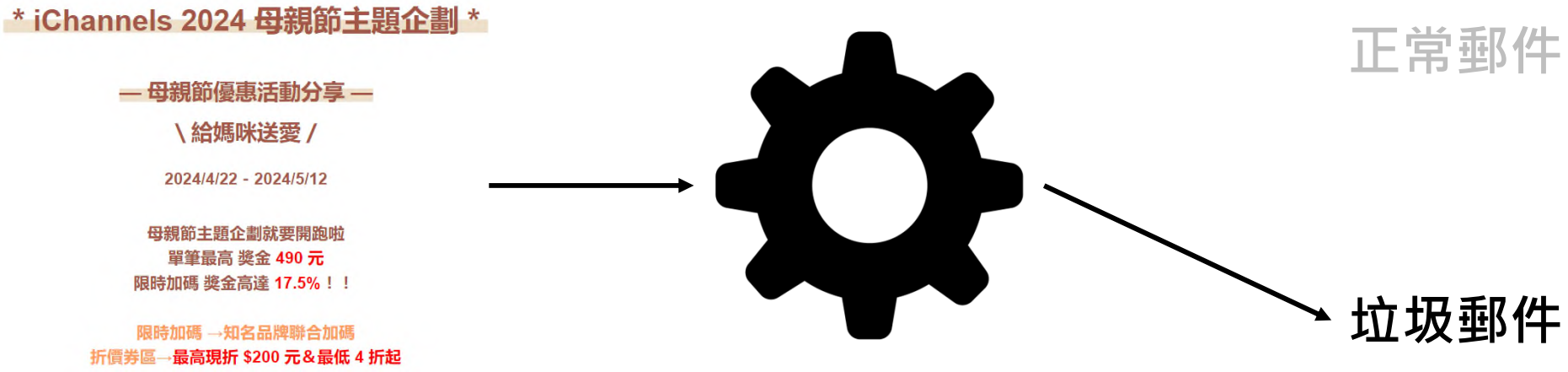
- 文本分類簡介
- 範例一：情感分析（使用 SnowNLP）
- 範例二：情感分析（使用 RNN）
- 本章總結

文本分類簡介

Introduction

何謂「文本分類」

- 輸入一段文字，利用模型將它分到某一個類別的問題



常見的文本分類應用

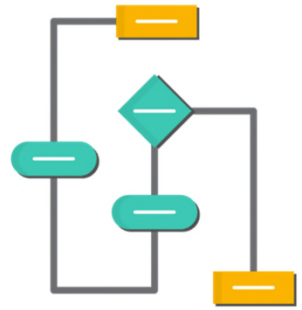
- **郵件分類**
  - 正常郵件 vs. 垃圾郵件（二選一）
  - 個人郵件 vs. 社交郵件 vs. 廣告郵件（多選一）
- **情緒分析**
  - 正面評價 vs. 負面評價
- **新聞分類**
  - 體育 vs. 政治 vs. 娛樂 ...

文本分類常用演算法

- **機器學習演算法**（60~70%）
  - 單純貝氏分類器
    （Naïve Bayesian Classifier）
  - 支援向量機 SVM
    （Support Vector Machine）
  - 隱藏式馬可夫模型 HMM
    （Hidden Markov Model）

- **深度學習演算法**（>80%）
  - 循環神經網路 RNN
    （Recurrent Neural Networks）
  - Transformer 相關演算法 Attention Mechanism
    （BERT、GPT...etc.）

2010 後



文本分類步驟

前處理

- 資料集載入與切分
- 斷句斷詞
- 去除雜訊 Stop Words, Punctuation
- 文字數位化 e.g. 我 → 96
- 序列對齊 (text alignment)
  ↓
  NN 的入口等寬

模型訓練

- 模型建造
- 模型訓練
- 超參數尋優

評估與預測

- 效能評估 (evaluation)
- 模型上線
- 模型修正

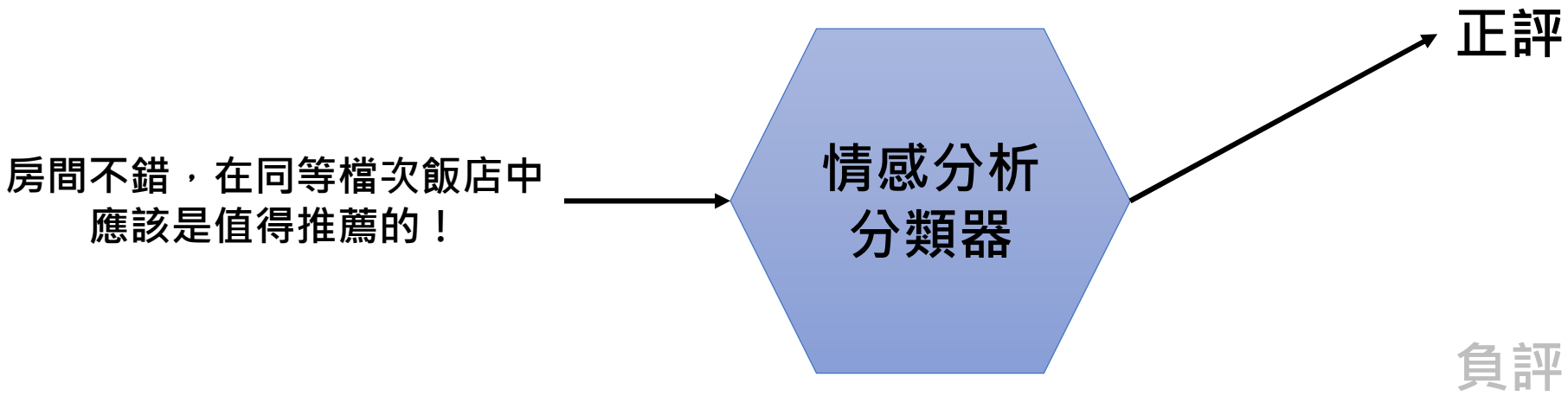
情感分析

使用 SnowNLP 預訓練模型

範例完整原始碼：
https://lurl.cc/YfGTUw

何謂情感分析

- 輸入文字 → 判斷該文字為正面情感或負面情感



何謂 SnowNLP

- 官方網站： https://github.com/isnowfy/snownlp
- SnowNLP 特色

中文特化	簡單易用	無需訓練
• 中文斷詞 • 中文詞性標註 • 中文情感分析	• SnowNLP(文字) • 傳回 0~1 浮點數	• 以買賣評價訓練過 • 無需訓練即可使用

SnowNLP 背後的演算法有哪些

- **單純貝氏分類器（Naive Bayes Classifier）**
  - 基於貝氏定理的簡單機率分類器。
  - 在做出預測時，假設特徵之間各自獨立。 (Naive)
  - 用於判斷文本的情感傾向。

- **支援向量機（Support Vector Machine, SVM）**
  - 一種常用於分類任務的監督式學習模型。
  - SnowNLP 將它與單純貝氏分類器搭配，用於驗證答案。

- **隱藏式馬可夫模型（Hidden Markov Model, HMM）**
  - 能夠記住一定長度的上下文，通常用於處理序列資料（如：文字）。
  - SnowNLP 將它用於詞性標註。 time series

SnowNLP 常用函數

```
model = SnowNLP(“這是要被分析的句子。”)
```

- model.sentiments
  - 傳回 0~1 之間的浮點數。
  - 0: 負向情緒 / 1: 正向情緒。
- model.sentences
  - 斷句用。傳回 Python List。
  - 傳回值會把最後之標點符號去除。
- model.words
  - 斷詞用。傳回 Python List。

call

call

TF-IDF

- model.tags
  - 詞性標註用。
  - 以（“詞彙”, PoS標籤）傳回。
- model.summary(limit=5)
  - 傳回關鍵句。
  - limit= 可以限制最多傳回幾句。
- model.keywords(limit=5)
  - 傳回關鍵詞。
  - limit= 可以限制最多傳回幾個詞。

環境設定

1
```
# 安裝必要套件
!pip install snownlp
```

2
```
# 下載資料集檔案
import os
Dataset_File = "Comments_Hotels.xlsx"

if not os.path.isfile(Dataset_File):
  os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)
```

3
```
# 下載講師自製的 HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://github.com/cnchi/HappyML.git")
```

隨堂練習：環境設定

- 請將前一頁的原始碼拷貝貼上，並且執行看看：

```
# 安裝必要套件
!pip install snownlp

# 下載資料集檔案
import os
Dataset_File = "Comments_Hotels.xlsx"

if not os.path.isfile(Dataset_File):
  os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)

# 下載講師自製的 HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://github.com/cnchi/HappyML.git")
```

資料前處理

1

```python
# 載入資料集
import pandas as pd
dataset = pd.read_excel(Dataset_File)

# 印出前幾筆做為驗證
dataset.head()
```

2

```python
# 切分自變數 X 與應變數 Y
import HappyML.preprocessor as pp

X, Y = pp.decomposition(dataset, x_columns=[1], y_columns=[0])
```

	label	review
0	1	距離川沙公路較近,但是公車指示不對,如果是"搭陸線"的話,會非常麻煩.建議用別的路線.房間較...
1	1	商務大床房，房間很大，床有2M寬，整體感覺經濟實惠不錯!
2	1	早餐太差，無論去多少人，那邊也不加食品的。酒店應該重視一下這個問題了。房間本身很好。
3	1	賓館在小街道上，不大好找，但還好北京熱心同胞很多~賓館設施跟介紹的差不多，房間很小，確實挺小...
4	1	CBD中心,周圍沒什麼店鋪,說5星有點勉強.不知道為什麼洗手間沒有吹風機

隨堂練習：資料前處理

- 請撰寫前一頁的原始碼，並且執行看看：

```
# 載入資料集
import pandas as pd
dataset = pd.read_excel(Dataset_File)

# 印出前幾筆做為驗證
dataset.head()

# 切分自變數 X 與應變數 Y
import HappyML.preprocessor as pp

X, Y = pp.decomposition(dataset, x_columns=[1], y_columns=[0])
```

以 SnowNLP 做情感分析

```python
# 引入 SnowNLP
from snownlp import SnowNLP

# 建立一個空的串列，儲存預測結果
snownlp_results = []

# 將自變數 X 中的每一條評論，迭代出來
for review in X['review']:
    # 使用 SnowNLP 進行情感分析
    analysis_result = SnowNLP(review)
    # 將結果轉換成 0 或 1
    score = 1 if analysis_result.sentiments > 0.5 else 0
    # 將結果添加到串列中
    snownlp_results.append(score)

# 將真實值與預測值合併成一個 DataFrame 觀察之
import pandas as pd

results = pd.DataFrame({'Y_real': Y.values.ravel(), 'Y_pred': snownlp_results})
results.head()
```

	label	review
0	1	距離川沙公路較近,但是公車指示不對,如果是"蔡陸線"的話,會非常麻煩.建議用別的路線.房間較...
1	1	商務大床房，房間很大，床有2M寬，整體感覺經濟實惠不錯!
2	1	早餐太差，無論去多少人，那邊也不加食品的。酒店應該重視一下這個問題了。房間本身很好。
3	1	賓館在小街道上，不大好找，但還好北京熱心同胞很多~賓館設施跟介紹的差不多，房間很小，確實挺小...
4	1	CBD中心,周圍沒什麼店鋪,說5星有點勉強.不知道為什麼洗手間沒有吹風機

負向情感 0.14567358 ↓ 0

正向情感 0.83419917 ↓ 1

	Y_real	Y_pred
0	1	0
1	1	0
2	1	0
3	1	1
4	1	0

隨堂練習：情感分析

- 請撰寫前一頁的原始碼，並且執行看看：

```python
# 引入 SnowNLP
from snownlp import SnowNLP

# 建立一個空的串列，儲存預測結果
snownlp_results = []

# 將自變數 X 中的每一條評論，迭代出來
for review in X['review']:
  # 使用 SnowNLP 進行情感分析
  analysis_result = SnowNLP(review)
  # 將結果轉換成 0 或 1
  score = 1 if analysis_result.sentiments > 0.5 else 0
  # 將結果添加到串列中
  snownlp_results.append(score)

# 將真實值與預測值合併成一個 DataFrame 觀察之
import pandas as pd

results = pd.DataFrame({'Y_real': Y.values.ravel(), 'Y_pred': snownlp_results})
results.head()
```

計算模型效能

```
1  # 使用 HappyML 計算模型效能
2  from HappyML.performance import ClassificationPerformance
3
4  pfm = ClassificationPerformance(results['Y_real'], results['Y_pred'])
5
6  print("Confusion Matrix:\n", pfm.confusion_matrix())
7  print(f"Accuracy: {pfm.accuracy():.2%}")
8  print(f"Recall: {pfm.recall():.2%}")
9  print(f"Precision: {pfm.precision():.2%}")
10 print(f"F1 Score: {pfm.f_score():.2%}")
```

1. 匯入 HappyML 的 ClassificationPerformance 模組
2. 建立 ClassificationPerformance 物件，傳入真實值與預測值
3. 呼叫 confusion_matrix()、accuracy()、recall()、precision()、f_score() 計算模型效能

```
Confusion Matrix:
 [[2421   22]
 [4489  833]]
Accuracy: 41.91%
Recall: 57.38%
Precision: 66.23%
F1 Score: 39.37%
```

隨堂練習：計算模型效能

- 請撰寫前一頁的原始碼，並且執行看看：

```python
# 使用 HappyML 計算模型效能
from HappyML.performance import ClassificationPerformance

pfm = ClassificationPerformance(results['Y_real'], results['Y_pred'])

print("Confusion Matrix:\n", pfm.confusion_matrix())
print(f"Accuracy: {pfm.accuracy():.2%}")
print(f"Recall: {pfm.recall():.2%}")
print(f"Precision: {pfm.precision():.2%}")
print(f"F1 Score: {pfm.f_score():.2%}")
```

```
Confusion Matrix:
 [[2421   22]
 [4489  833]]
Accuracy: 41.91%
Recall: 57.38%
Precision: 66.23%
F1 Score: 39.37%
```

效能低落的原因

- **太多正向情感被預測為負向**

```
Confusion Matrix:
 [[2421   22]
 [4489  833]]
Accuracy: 41.91%
Recall: 57.38%
Precision: 66.23%
F1 Score: 39.37%
```

- **訓練資料 vs. 預測資料不匹配**
  - SnowNLP 原始訓練資料：3C 產品線上買賣評價
  - 當前測試資料：旅館評價資料

SnowNLP 的模型可以訓練嗎

- 可以！

斷詞訓練

```
1  from snownlp import seg
2  seg.train('data.txt')
3  seg.save('seg.marshal')
```

← 一個詞一列
← 一定要叫此名

詞性標註（PoS）訓練

```
1  from snownlp import tag
2  tag.train('pos_data.txt')
3  tag.save('tag.marshal')
```

← 每列：詞彙 詞性
← 一定要叫此名

情感分析模型訓練

```
1  from snownlp import sentiment
2  sentiment.train('neg.txt', 'pos.txt')
3  sentiment.save('sentiment.marshal.3')
```

← 負向/正向句子檔案
← 一定要叫此名

訓練流程：

1. 將資料依照規定擺好。
2. 撰寫左側程式碼，執行之。
3. 將產生出來的模型檔（seg.marshal, tag.marshal, sentiment.marshar.3），拿到 Python 安裝路徑 → Lib → site-packages → snownlp 之下。
4. 找到同名檔案，將之覆蓋。
5. 完成！

情感分析

使用 RNN 從頭訓練

範例完整原始碼：
https://lurl.cc/o1JqYN

為何要從頭訓練

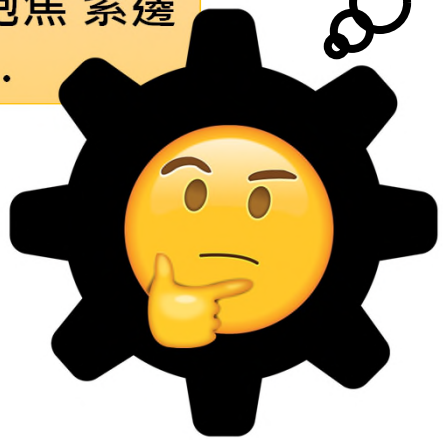
- 為了能得到與當前資料集匹配的模型

鏡頭 焦距 成像
銳利 跑焦 紫邊
卡頓 ...

索尼的對焦就是快到沒有朋友 → 正評

大廳？氣派？
早餐？

鏡頭 焦距 成像
銳利 跑焦 紫邊
卡頓 ...

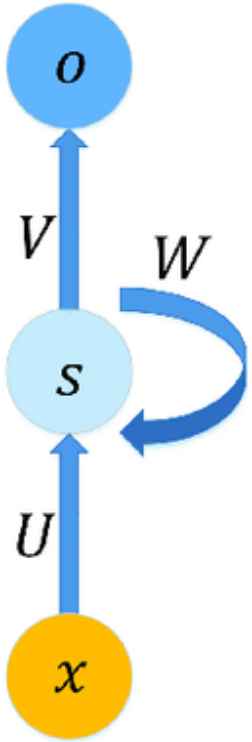


大廳很氣派！早餐好吃！ → ？？

情緒分析常用演算法

Recurrent Neural Networks

（循環神經網路）



RNN 簡單介紹

- 猜猜看，大街上的人們穿什麼？

RNN = 會記住前 N 筆
輸入資料的神經網路

得對前一個輸入「有記憶」，
才能正確地推論！

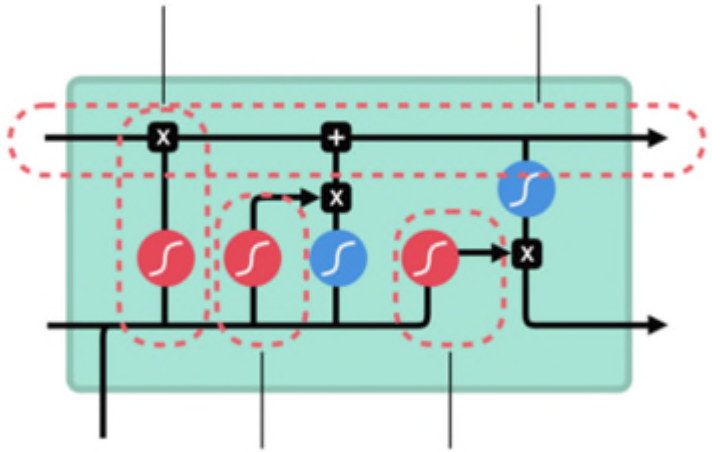
今天氣溫 37 度

穿短袖 5% 95%

穿外套 5% 5%

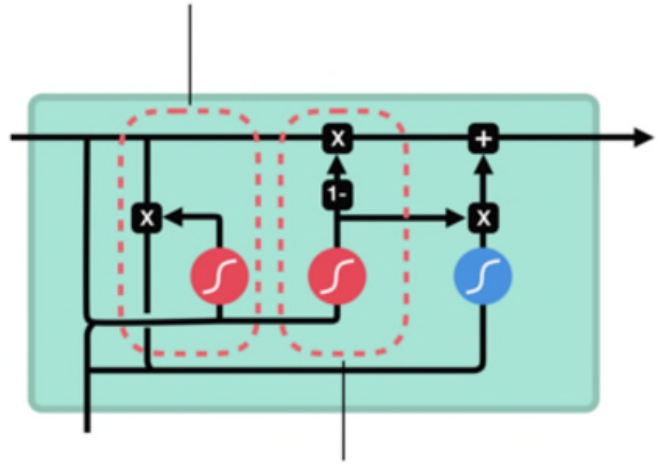
RNN 神經網路的種類

- **LSTM（長短期記憶）**
  - Long Short-Term Memory



結構複雜、精確度高、運算時間久

- **GRU（閘道循環單元）**
  - Gated Recurrent Unit



結構簡單、精確度稍低、運算時間較短

「自然語言處理」程式設計流程

自然語言「前處理」

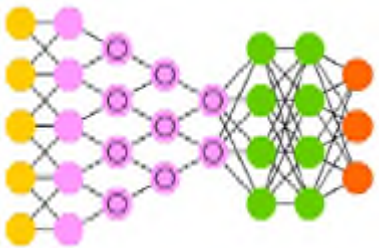
斷詞 → 數位化 → 序列對齊 (sequence alignment) → 編碼 → 模型訓練

compile

斷詞	數位化	序列對齊	編碼
['I', 'love', 'jogging']	[1, 2, 3]	[1, 2, 3]	[[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]]
['and', 'you']	[4, 5]	[0, 4, 5]	
['我', '愛', '跑步']			[[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]]
['你', '呢']			

補 (padding) 在前面較有效率

切 (truncate) 在後面 〃



環境設定

1
2
3
4

```
# 下載資料集檔案
import os
Dataset_File = "Comments_Hotels.xlsx"

if not os.path.isfile(Dataset_File):
  os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)

# 下載講師自製的 HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://github.com/cnchi/HappyML.git")

# 載入必要的函式庫
import torch
import torch.nn as nn

import HappyML.preprocessor as pp
from HappyML.pytorch import Sequential

# 檢查是否有可用的 GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"device: {device}")
```

環境設定

1
2
3
4

```
# 安裝斷詞相關的函式庫、檔案
!pip install jieba

# 取得斷詞專用、繁體中文自定義辭典
import os
Dictionary_File = 'dict.txt.big'

if not os.path.isfile(Dictionary_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dictionary_File)

# 取得繁體中文停止詞辭典
StopWords_File = "stopWords_big5.txt"

if not os.path.isfile(StopWords_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + StopWords_File)

StopWords_Set = set()
with open(StopWords_File, "rt", encoding="utf-8") as f:
  for line in f:
    line = line.strip() # Remove trailing \n
    StopWords_Set.add(line)

# 設定中文標點符號集合
Punctuation_Set = set("$!&#%\()+-*/_,.    ?:;'\"<=>^`|~[]{}’0123456789?_“”、。《》！，　：；？「」（） ")
```

隨堂練習：環境設定

- 請將前兩頁的原始碼拷貝貼上，並且執行看看：

```python
# 下載資料集檔案
import os
Dataset_File = "Comments_Hotels.xlsx"

if not os.path.isfile(Dataset_File):
  os.system("wget https://raw.githu

# 下載講師自製的 HappyML
import os

if not os.path.isdir("HappyML"):
  os.system("git clone https://gith

# 載入必要的函式庫
import torch
import torch.nn as nn

import HappyML.preprocessor as pp
from HappyML.pytorch import Sequent

# 檢查是否有可用的 GPU
device = torch.device("cuda" if tor
print(f"device: {device}")
```

```python
# 安裝斷詞相關的函式庫、檔案
!pip install jieba

# 取得斷詞專用、繁體中文自定義辭典
import os
Dictionary_File = 'dict.txt.big'

if not os.path.isfile(Dictionary_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + Dictionary_File)

# 取得繁體中文停止詞辭典
StopWords_File = "stopWords_big5.txt"

if not os.path.isfile(StopWords_File):
    os.system('wget https://raw.githubusercontent.com/cnchi/datasets/master/' + StopWords_File)

StopWords_Set = set()
with open(StopWords_File, "rt", encoding="utf-8") as f:
  for line in f:
    line = line.strip() # Remove trailing \n
    StopWords_Set.add(line)

# 設定中文標點符號集合
Punctuation_Set = set("$!&#%()+-*/_,.   ?:;'\"<=>^`|~[]{}’0123456789?_“”、。《》！，：；？「」（） ")
```

前處理：載入與切分 X、Y

```
1  # 載入資料集
2  import pandas as pd
3  dataset = pd.read_excel(Dataset_File)
4
5  # 印出前幾筆做為驗證
6  dataset.head()
7
8  # 切分自變數 X 與應變數 Y
9  X, Y = pp.decomposition(dataset, x_columns=[1], y_columns=[0])
```

	label	review
0	1	距離川沙公路較近,但是公車指示不對,如果是"蔡陸線"的話,會非常麻煩.建議用別的路線.房間較...
1	1	商務大床房，房間很大，床有2M寬，整體感覺經濟實惠不錯!
2	1	早餐太差，無論去多少人，那邊也不加食品的。酒店應該重視一下這個問題了。房間本身很好。
3	1	賓館在小街道上，不大好找，但還好北京熱心同胞很多~賓館設施跟介紹的差不多，房間很小，確實挺小...
4	1	CBD中心,周圍沒什麼店鋪,說5星有點勉強.不知道為什麼洗手間沒有吹風機

review
距離川沙公路較近,但是公車指示不對,如果是"蔡陸線"的話,會非常麻煩.建議用別的路線.房間較...
商務大床房，房間很大，床有2M寬，整體感覺經濟實惠不錯!
早餐太差，無論去多少人，那邊也不加食品的。酒店應該重視一下這個問題了。房間本身很好。
賓館在小街道上，不大好找，但還好北京熱心同胞很多~賓館設施跟介紹的差不多，房間很小，確實挺小...
CBD中心,周圍沒什麼店鋪,說5星有點勉強.不知道為什麼洗手間沒有吹風機

label
1
1
1
1
1

前處理：斷詞

```
11  # 載入斷詞函式庫
12  import jieba
13
14  # 設定使用繁體中文自訂辭典
15  jieba.set_dictionary(Dictionary_File)
16
17  # 定義斷詞、去除標點符號、停止詞函數
18  def jieba_tokenizer(text):
19      # 斷詞
20      tokenized_results = list(jieba.cut(text))
21      # 去除標點符號
22      tokenized_results = [token for token in tokenized_results if token not in Punctuation_Set]
23      # 去除停止詞
24      tokenized_results = [token for token in tokenized_results if token not in StopWords_Set]
25
26      # 輸出斷詞結果
27      return " ".join(tokenized_results)
28
29  # 用 Pandas 的 .apply() 函數，把斷詞套用到每個字串上
30  import pandas as pd
31
32  # 套用斷詞函數
33  X["review"] = X["review"].apply(jieba_tokenizer)
34
35  # 印出前幾筆做為驗證
36  X.head()
```

距離川沙公路較近,但是公車指示不對,如果是"蔡陸線"的話,會非常麻煩.建議用別的路線

Python object iterator

距離 川沙 公路 較近 公車 指示 蔡陸線 會 非常 麻煩 建議 路線

pandas 允許你用 .apply()，
將特定函數套用到指定行列。

review
距離 川沙 公路 較近 公車 指示 蔡陸線 會 非常 麻煩 建議 路線 房間 簡單
商務 大床 房 房間 很大 床有 2M 寬 整體 感覺 經濟 實惠 不錯
早餐 太差 不加 食品 酒店 應該 重視 一下 問題 房間
賓館 街道 不大好 找 還好 北京 熱心 同胞 很多 賓館 設施 介紹 差不多 房間 很小 ...
CBD 中心 周圍 沒什麼 店鋪 說 是 有點 勉強 知道 洗手間 沒有 吹風機

隨堂練習：載入、切分、斷詞

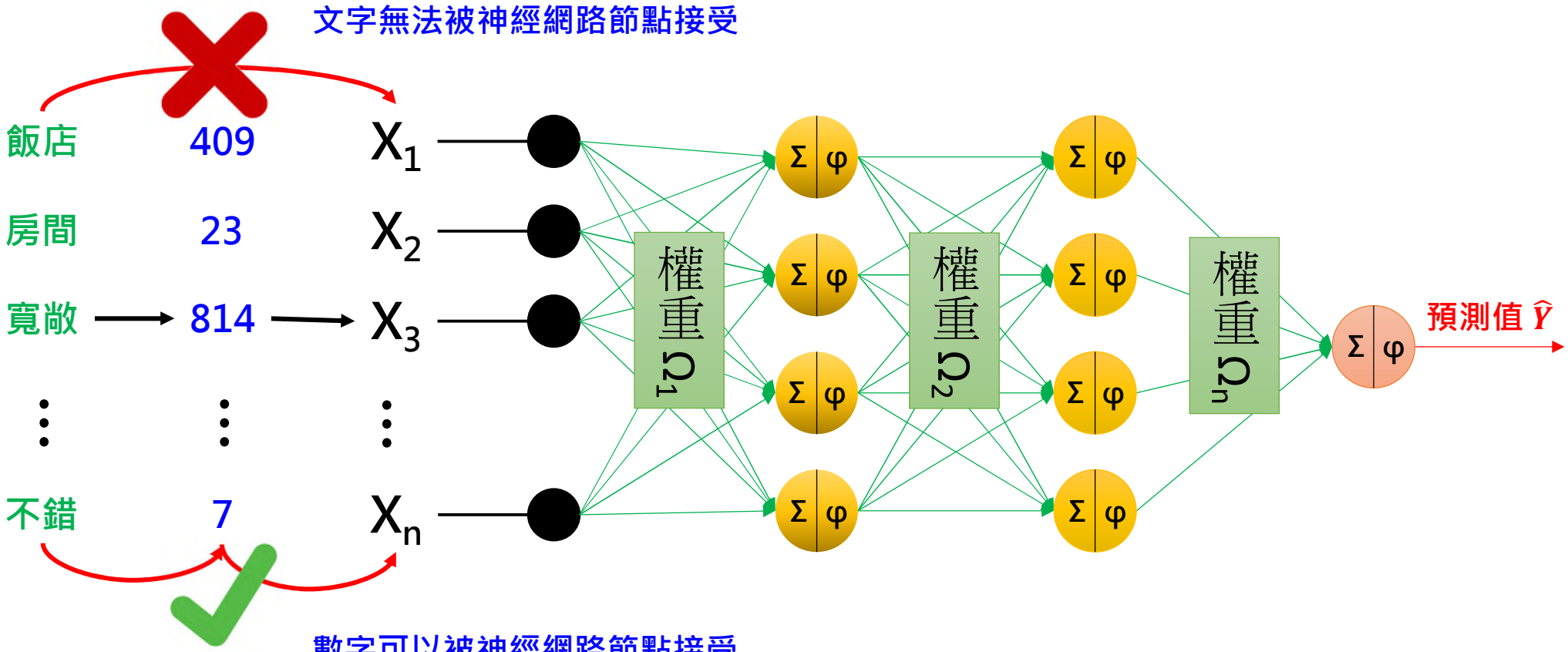
- 請將前兩頁的原始碼拷貝貼上，並且執行看看：

```python
# 載入資料集
import pandas as pd
dataset = pd.read_excel(D

# 印出前幾筆做為驗證
dataset.head()

# 切分自變數 X 與應變數 Y
X, Y = pp.decomposition(d
```

```python
# 載入斷詞函式庫
import jieba

# 設定使用繁體中文自訂辭典
jieba.set_dictionary(Dictionary_File)

# 定義斷詞、去除標點符號、停止詞函數
def jieba_tokenizer(text):
  # 斷詞
  tokenized_results = list(jieba.cut(text))
  # 去除標點符號
  tokenized_results = [token for token in tokenized_results if token not in Punctuation_Set]
  # 去除停止詞
  tokenized_results = [token for token in tokenized_results if token not in StopWords_Set]

  # 輸出斷詞結果
  return " ".join(tokenized_results)

# 用 Pandas 的 .apply() 函數，把斷詞套用到每個字串上
import pandas as pd

# 套用斷詞函數
X["review"] = X["review"].apply(jieba_tokenizer)

# 印出前幾筆做為驗證
X.head()
```

文字數位化

- 為何需要文字數位化



文字數位化

- 可以將遇到的每個詞，都賦予獨一無二的數字

1
```
# 載入文字數位化物件
from tensorflow.keras.preprocessing.text import Tokenizer
```
(deprecated)

2
```
# 設定字典總字數最大上限（None = 不設限）
MAX_NUM_WORDS = 10000
```
← 指定「文/數」對照表最多能夠承受的詞彙量（否則會爆掉）

3
```
# 產生一個數位化物件
tk = Tokenizer(
        num_words=MAX_NUM_WORDS,
        filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
        lower=True,
        split=' ',
        char_level=False,
        oov_token='NiD'
    )
```
← 建立一個 0~9999 的對照表

← 這些文字直接跳掉，不編號

← 若遇到英文，一律轉成小寫

← 以空白做為斷詞依據

← 是否斷詞至字母層級

↑ 對照表內沒有編號的話，用哪個字串代替？（NiD = Not in Dictionary）

Out of Vocabulary

"商務 大床 房間 很大"
→ "商務" "大床" "房間" "很大"

文字數位化

```
17  # 將文字轉成數字
18  tk.fit_on_texts(X["review"])
19
20  # 印出文字數字對照辭典
21  print(tk.word_index)
22  print(tk.index_word)
23
24  # 試著將文本轉成數字
25  seq = tk.texts_to_sequences(X["review"])
26
27  # 印出前幾筆做為驗證
28  print(seq[:10])
29
30  # 試著將數字轉為文字
31  text = tk.sequences_to_texts(seq)
32
33  # 印出前幾筆做為驗證
34  print(text[:10])
```

1. "距離 川沙 公路 較近 公車 指示 蔡陸線 會 非常 麻煩 建議 路線 房間 簡單"
2. {'NiD': 1, '酒店': 2, '房間': 3, '不錯': 4, '服務': 5, '沒有': 6, ...}
   {1: 'NiD', 2: '酒店', 3: '房間', 4: '不錯', 5: '服務', 6: '沒有', ...}

   依照詞彙重要性編號，
   罕見詞彙則放棄編號。
3. [[283, 1, 3691, 2405, 677, 2520, 1, 33, 13, 678, 68, 598, 3, 456], ...]
4. ['距離 NiD 公路 較近 公車 指示 NiD 會 非常 麻煩 建議 路線 房間 簡單', ...]

隨堂練習：文字數位化

- 請將前兩頁的原始碼拷貝貼上，並且執行看看：

```python
# 載入文字數位化物件
from tensorflow.keras.preprocessing.text import Tokenizer

# 設定字典總字數最大上限（None = 不設限）
MAX_NUM_WORDS = 10000

# 產生一個數位化物件
tk = Tokenizer(
        num_words=MAX_NUM_WORDS,
        filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
        lower=True,
        split=' ',
        char_level=False,
        oov_token='NiD'
    )
```

```python
# 將文字轉成數字
tk.fit_on_texts(X["review"])

# 印出文字數字對照辭典
print(tk.word_index)
print(tk.index_word)

# 試著將文本轉成數字
seq = tk.texts_to_sequences(X["review"])

# 印出前幾筆做為驗證
print(seq[:10])

# 試著將數字轉為文字
text = tk.sequences_to_texts(seq)

# 印出前幾筆做為驗證
print(text[:10])
```

序列對齊（Sequence Alignment）

- 什麼是「序列對齊」
  - 將所有「**數位化**」過的句子，**截長補短**成**相同長度**。
- 為何要「序列對齊」
  - 方便**輸入**神經網路的「**輸入層**」。
- 如何「序列對齊」

```
[[2, 3, 4],
 [5, 2, 3, 7, 1]]
```

```
[[0, 0, 2, 3, 4]
 [5, 2, 3, 7, 1]]
```

```python
# 引入序列對齊套件
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 指定要截長補短至多少個單字
MAX_SEQUENCE_LENGTH = 32

# 建立一個序列對齊物件
padded_seq = pad_sequences(
        sequences=seq,    ← 要被對齊的序列
        maxlen=MAX_SEQUENCE_LENGTH, ← 調成32個字
        dtype="int32", ← 使用整數做為輸出型態
        padding="pre", ← 要補字的話，補在前面
        truncating="post", ← 要砍字的話，砍後面
        value=0  ← 不足一律補 0
    )

# 印出前幾筆做為驗證
print(padded_seq[:10])
```

計算過程：

1. 先算所有句子的平均詞數 = 42.7
2. 想找 2^n、且小一點的數字 → 32

找 2^ 較快

1	“NiD”
2	“酒店”
3	“房間”
4	“不錯”
5	“服務”
6	“沒有”
7	…

序列對齊（Sequence Alignment）

python f-string

```python
# 重新合成為自變數 X
column_names = [f"word_{i}" for i in range(MAX_SEQUENCE_LENGTH)]
X = pd.DataFrame(padded_seq, columns=column_names)

# 印出前幾筆做為驗證
X.head()
```

	word_0	word_1	word_2	word_3	word_4	word_5	word_6	word_7	word_8	word_9	...	word_22	word_23	word_24	word_25	word_26	word_27	word_28	word_29	word_30	word_31
0	0	0	0	0	0	0	0	0	0	0	...	677	2520	1	33	13	678	68	598	3	456
1	0	0	0	0	0	0	0	0	0	0	...	3	77	9890	6436	1730	193	11	642	441	4
2	0	0	0	0	0	0	0	0	0	0	...	10	156	7757	1331	2	67	1300	74	49	3
3	20	2521	3436	104	198	340	1731	1	53	20	...	24	4	1627	132	326	813	1	1569	704	859
4	0	0	0	0	0	0	0	0	0	0	...	199	6438	12	106	34	968	50	35	6	898

隨堂練習：序列對齊

- 請將前兩頁的原始碼拷貝貼上，並且執行看看：

```python
# 引入序列對齊套件
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 指定要截長補短至多少個單字
MAX_SEQUENCE_LENGTH = 32

# 建立一個序列對齊物件
padded_seq = pad_sequences(
        sequences=seq,
        maxlen=MAX_SEQUENCE_LENGTH,
        dtype="int32",
        padding="pre",
        truncating="post",
        value=0
    )

# 印出前幾筆做為驗證
print(padded_seq[:10])

# 重新合成為自變數 X
column_names = [f"word_{i}" for i in range(MAX_SEQUENCE_LENGTH)]
X = pd.DataFrame(padded_seq, columns=column_names)

# 印出前幾筆做為驗證
X.head()
```

資料集切分與轉換

1
2

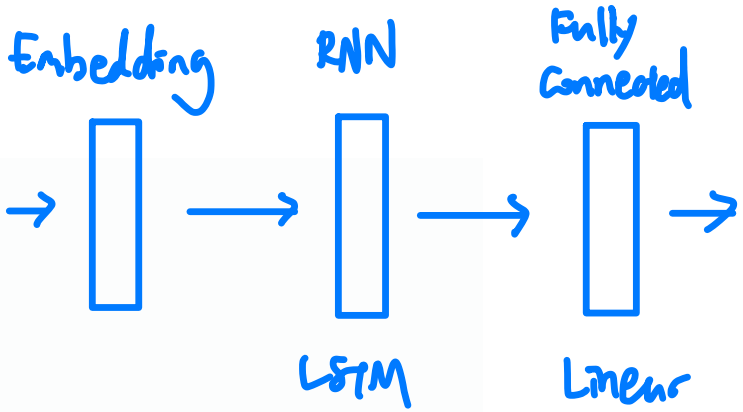
```
1  # 切分訓練集與測試集
2  X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)
3
4  # 將數值轉換成 PyTorch 張量
5  X_train_tensor = torch.tensor(X_train.values, dtype=torch.long).to(device)
6  Y_train_tensor = torch.tensor(Y_train.values, dtype=torch.float).to(device)
7  X_test_tensor = torch.tensor(X_test.values, dtype=torch.long).to(device)
8  Y_test_tensor = torch.tensor(Y_test.values, dtype=torch.float).to(device)
```

隨堂練習：切分與轉換

- 請輸入前一頁的原始碼，並且執行看看：

```
# 切分訓練集與測試集
X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)

# 將數值轉換成 PyTorch 張量
X_train_tensor = torch.tensor(X_train.values, dtype=torch.long).to(device)
Y_train_tensor = torch.tensor(Y_train.values, dtype=torch.float).to(device)
X_test_tensor = torch.tensor(X_test.values, dtype=torch.long).to(device)
Y_test_tensor = torch.tensor(Y_test.values, dtype=torch.float).to(device)
```

模型建造




```
1  # 建立一個以 PyTorch 撰寫的模型
2  class RNN(nn.Module):
3      def __init__(self, input_size, hidden_size, output_size):
4          # 先初始化 nn.Module（父類別）
5          super(RNN, self).__init__()
6          # 將隱藏層初始層數記下來備用
7          self.hidden_size = hidden_size
8          # 建立嵌入層，將 10000 字的辭典，重新編碼為以 hidden 這麼多節點表示
9          self.embedding = nn.Embedding(input_size, hidden_size)
10         # 使用 LSTM 做為 RNN 層
11         self.rnn = nn.LSTM(hidden_size, (hidden_size+output_size)//2, batch_first=True)
12         # 建立全連接層，做為分類之用
13         self.fc = nn.Linear((hidden_size+output_size)//2, output_size)
14         # 二選一，故出口激活函數使用 Sigmoid
15         self.sigmoid = nn.Sigmoid()
```

10000　128　1

嵌入層（Embedding）後述

$$\frac{\text{上一層節點數} + \text{下一層節點數}}{2}$$

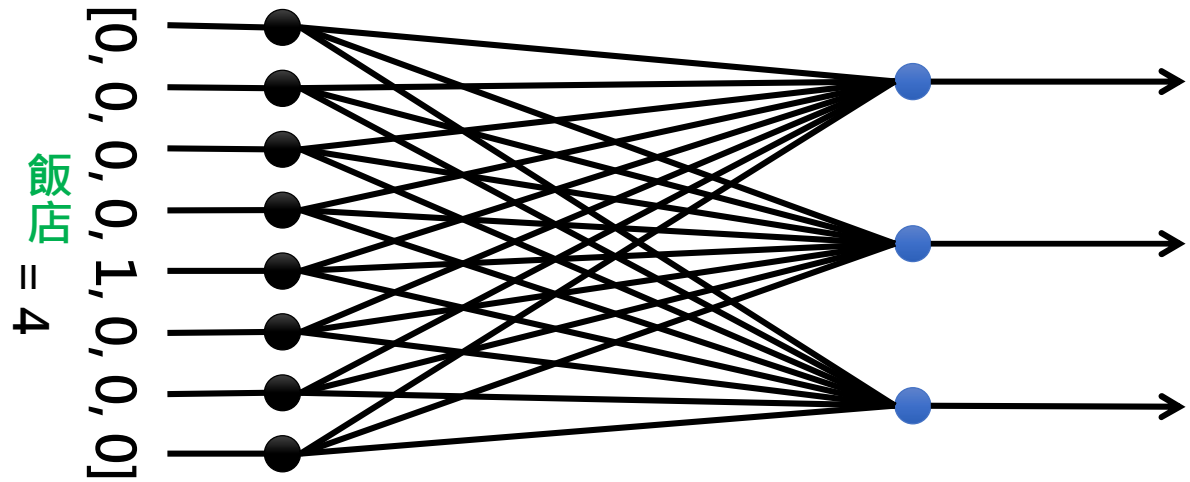
模型建造

```
17      # 前向傳播，真正讓張量通過每一層神經層
18      def forward(self, x):
19          embedded = self.embedding(x)
20          output, hidden = self.rnn(embedded)
21          output = self.fc(output[:, -1, :])
22          output = self.sigmoid(output)
23          return output
24
25  # 產生 RNN 模型的實體
26  model = RNN(MAX_NUM_WORDS, 128, 1).to(device)
27
28  # 指定使用二選一專用的 Binary Cross Entroy 做為損失函數
29  criterion = nn.BCELoss()
30
31  # 指定使用 Adam 做為優化器
32  optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

1
2
3
4

詞向量嵌入法（Word Embedding）

- 利用神經網路，將詞編碼降維後，得到一個壓縮過後的「詞向量」



以後，”飯店”

=

[-0.00180965 0.00570974 -0.01133073]

詞編碼

[0, 0, 0, 0, 1, 0, 0, 0] ⊙

權重 W
（可自行習得）

```
[[ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [-0.00180965  0.00570974 -0.01133073]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]
 [ 0.01023025 -0.01604468  0.04070808]]
```

= [-0.00180965 0.00570974 -0.01133073]

詞向量

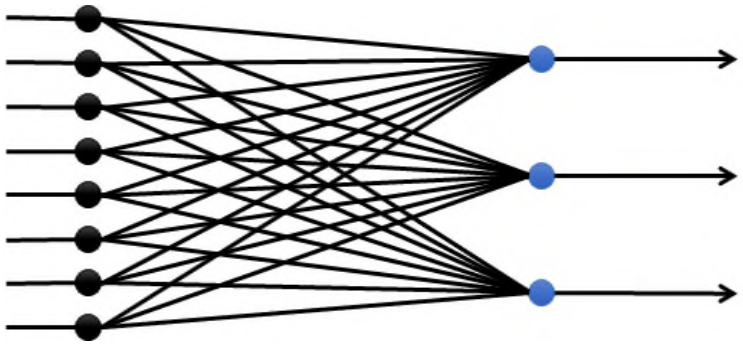
1x8　　8x3　　1x3

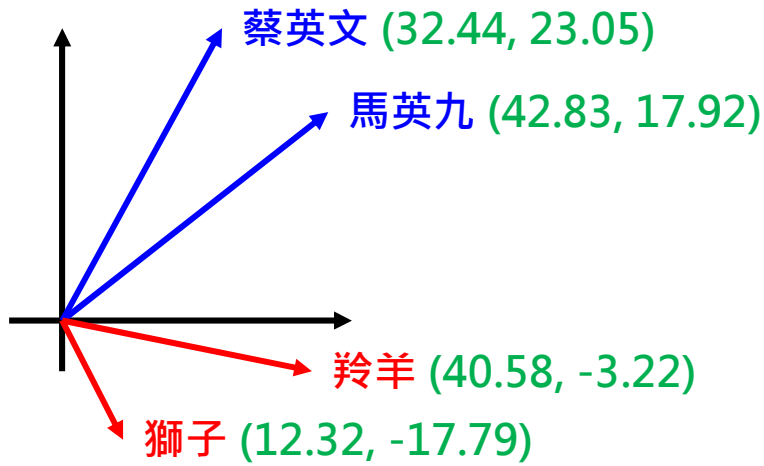
詞向量嵌入法（Word Embedding）

- **詞向量嵌入法**更厲害的地方

容易出現在同一篇文章

詞向量的**歐幾里得距離**接近 = 詞的**語意**接近





隨堂練習：模型建造

- 請輸入前兩頁的原始碼，並且執行看看：

```
# 建立一個以 PyTorch 撰寫的模型
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        # 先初始化 nn.Module（父類別）
        super(RNN, self).__init__()
        # 將隱藏層初始
        self.hidden
        # 建立嵌入層
        self.embedd
        # 使用 LSTM
        self.rnn =
        # 建立全連接
        self.fc = n
        # 二選一，故
        self.sigmoi
        # 前向傳播，真正讓張量通過每一層神經層
        def forward(self, x):
            embedded = self.embedding(x)
            output, hidden = self.rnn(embedded)
            output = self.fc(output[:, -1, :])
            output = self.sigmoid(output)
            return output

# 產生 RNN 模型的實體
model = RNN(MAX_NUM_WORDS, 128, 1).to(device)

# 指定使用二選一專用的 Binary Cross Entroy 做為損失函數
criterion = nn.BCELoss()

# 指定使用 Adam 做為優化器
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

訓練模型

```python
# 指定訓練回數
num_epochs = 500

# 開始訓練模型
for epoch in range(num_epochs):
    # 先將模型設為「訓練模式」
    model.train()

    # 前向傳播，透過自變數 X 估計應變數 Y 的值（Y_pred）
    outputs = model(X_train_tensor)

    # 比較 Y_pred 與 Y_real，取得損失值
    loss = criterion(outputs, Y_train_tensor)

    # 將這次算出來的梯度（預計優化的方向）歸零
    optimizer.zero_grad()
    # 反向傳播，計算出這次的梯度（權重預計優化的方向）
    loss.backward()
    # 依照計算出來的梯度，真正地將權重更新
    optimizer.step()

    # 計算這一次的「確度（Accuracy）」
    predicted = (outputs > 0.5).float()
    correct = (predicted == Y_train_tensor).float().sum()
    accuracy = correct / Y_train_tensor.shape[0]

    # 印出這一回合的損失值與確度
    if (epoch+1) % 1 == 0:
        print(f'Epoch {epoch+1}/{num_epochs}, Loss: {loss.item():.4%}, Accuracy: {accuracy.item():.4%}')
```

```
Epoch 1/500, Loss: 69.0239%, Accuracy: 53.1341%
Epoch 2/500, Loss: 68.3106%, Accuracy: 58.5437%
Epoch 3/500, Loss: 67.6257%, Accuracy: 62.6138%
Epoch 4/500, Loss: 66.9646%, Accuracy: 65.3100%
Epoch 5/500, Loss: 66.3231%, Accuracy: 66.6323%
                    ......
Epoch 497/500, Loss: 0.0919%, Accuracy: 99.9828%
Epoch 498/500, Loss: 0.0917%, Accuracy: 99.9828%
Epoch 499/500, Loss: 0.0914%, Accuracy: 99.9828%
Epoch 500/500, Loss: 0.0912%, Accuracy: 99.9828%
```

隨堂練習：訓練模型

- 請輸入前一頁的原始碼，並且執行看看：

```python
# 指定訓練回數
num_epochs = 500

# 開始訓練模型
for epoch in range(num_epochs):
    # 先將模型設為「訓練模式」
    model.train()

    # 前向傳播，透過自變數 X 估計應變數 Y 的值（Y_pred）
    outputs = model(X_train_tensor)

    # 比較 Y_pred 與 Y_real，取得損失值
    loss = criterion(outputs, Y_train_tensor)

    # 將這次算出來的梯度（預計優化的方向）歸零
    optimizer.zero_grad()
    # 反向傳播，計算出這次的梯度（權重預計優化的方向）
    loss.backward()
    # 依照計算出來的梯度，真正地將權重更新
    optimizer.step()

    # 計算這一次的「確度（Accuracy）」
    predicted = (outputs > 0.5).float()
    correct = (predicted == Y_train_tensor).float().sum()
    accuracy = correct / Y_train_tensor.shape[0]

    # 印出這一回合的損失值與確度
    if (epoch+1) % 1 == 0:
        print(f'Epoch {epoch+1}/{num_epochs}, Loss: {loss.item():.4%}, Accuracy: {accuracy.item():.4%}')
```

模型評估

1
2
3
4
5

```python
# 先將模型設為「評估模式」
model.eval()

# 關閉整個神經網路的梯度計算機制
with torch.no_grad():
    # 向前傳播，透過自變數 X 預測應變數 Y
    outputs = model(X_test_tensor)

    # 計算「確度（Accuracy）」
    outputs = (outputs > 0.5).float()
    accuracy = (outputs == Y_test_tensor).float().mean()

    # 印出測試集的「確度」
    print(f'Accuracy: {accuracy.item():.4%}')
```

threshold

Accuracy: 79.6087%

隨堂練習：模型評估

- 請輸入前一頁的原始碼，並且執行看看：

```python
# 先將模型設為「評估模式」
model.eval()

# 關閉整個神經網路的梯度計算機制
with torch.no_grad():
    # 向前傳播，透過自變數 X 預測應變數 Y
    outputs = model(X_test_tensor)

    # 計算「確度（Accuracy）」
    outputs = (outputs > 0.5).float()
    accuracy = (outputs == Y_test_tensor).float().mean()

    # 印出測試集的「確度」
    print(f'Accuracy: {accuracy.item():.4%}')
```

Accuracy: 79.6087%

模型預測

```python
# 撰寫一個可以丟入一條評論、輸出情緒分析結果的函數
def predict_sentiment(sentence):
    # 先將模型設為「評估模式」
    model.eval()
    # 關閉整個神經網路的梯度計算機制
    with torch.no_grad():
        # 利用先前自己撰寫的斷詞函數斷詞
        sent = jieba_tokenizer(sentence)
        # 將斷詞結果轉換成數字
        seq = tk.texts_to_sequences([sent])
        # 將打算送入神經網路的輸入值「序列對齊」
        padded_seq = pad_sequences(
                sequences=seq,
                maxlen=MAX_SEQUENCE_LENGTH,
                dtype="int32",
                padding="pre",
                truncating="post",
                value=0
            )
        # 將輸入值轉換成 PyTorch Tensor
        X_tensor = torch.tensor(padded_seq, dtype=torch.long).to(device)
        # 丟入模型，做「前向傳播」
        output = model(X_tensor)
        # 把情緒分數與 0.5 比較，轉成布林值 True/False 後，再換成浮點數
        output = (output > 0.5).float()

        # 傳回最後結果
        return output.item()
```

1. 距離川沙公路較近,但是公車指示不對,如果是"蔡陸線"的話,會非常麻煩.建議用別的路線
2.
3. 距離 川沙 公路 較近 公車 指示 蔡陸線 會 非常 麻煩 建議 路線
4. [[283, 1, 3691, 2405, 677, 2520, 1, 33, 13, 678, 68, 598, 3, 456], ...]
5. [[0 0 ... 0 283 1 3691 2405 677 2520 1 33 13 678 68 598 3 456], ...]
6.
7.
8.
9. .item() 可以把 PyTorch 張量，轉回原來純粹的浮點數

模型預測

```
30  # 做為預測範例的兩個測試用評論
31  sentence1 = "這家飯店的服務很好，房間也很乾淨，下次還會再來。"
32  sentence2 = "很糟糕！房間髒，早餐也難吃。下次不會再光臨。"
33
34  # 預測結果
35  sentiment1 = predict_sentiment(sentence1)
36  sentiment2 = predict_sentiment(sentence2)
37
38  # 印出結果
39  print(f'Sentence: {sentence1}, Sentiment: {sentiment1}')
40  print(f'Sentence: {sentence2}, Sentiment: {sentiment2}')
```

1
2
3

```
Sentence: 這家飯店的服務很好，房間也很乾淨，下次還會再來。, Sentiment: 1.0
Sentence: 很糟糕！房間髒，早餐也難吃。下次不會再光臨。, Sentiment: 0.0
```

隨堂練習：模型預測

- 請輸入前兩頁的原始碼，並且執行看看：

```python
# 撰寫一個可以丟入一條評論、輸出情緒分析結果的函數
def predict_sentiment(sentence):
    # 先將模型設為「評估模式」
    model.eval()
    # 關閉整個神經網路的梯度計算機制
    with torch.no_grad():
        # 利用先前自己撰寫的斷詞函數斷詞
        sent = jieba_tokenizer(sentence
        # 將斷詞結果轉換成數字
        seq = tk.texts_to_sequences([se
        # 將打算送入神經網路的輸入值「序列
        padded_seq = pad_sequences(
                sequences=seq,
                maxlen=MAX_SEQUENCE_LEN
                dtype="int32",
                padding="pre",
                truncating="post",
                value=0
            )
        # 將輸入值轉換成 PyTorch Tensor
        X_tensor = torch.tensor(padded_seq, dtype=torch.long).to(device)
        # 丟入模型，做「前向傳播」
        output = model(X_tensor)
        # 把情緒分數與 0.5 比較，轉成布林值 True/False 後，再換成浮點數
        output = (output > 0.5).float()

        # 傳回最後結果
        return output.item()

# 做為預測範例的兩個測試用評論
sentence1 = "這家飯店的服務很好，房間也很乾淨，下次還會再來。"
sentence2 = "很糟糕！房間髒，早餐也難吃。下次不會再光臨。"

# 預測結果
sentiment1 = predict_sentiment(sentence1)
sentiment2 = predict_sentiment(sentence2)

# 印出結果
print(f'Sentence: {sentence1}, Sentiment: {sentiment1}')
print(f'Sentence: {sentence2}, Sentiment: {sentiment2}')
```

課後作業

- 資料集 Comments_Mobile_Phone.xml 收集了 2016 年手機評論如下：

每一則評論由 <Review rid= "X"> 包住

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Reviews>
  <Review rid="1">
    <sentences>
      <sentence id="1:0">
        <text>今天有幸拿到了港版白色iPhone 5真機，</text>
      </sentence>
      <sentence id="1:1">
        <text>真機重量比上一代輕了很多，</text>
        <Opinions>
          <Opinion category="PHONE#DESIGN_FEATURES" polarity="positive"/>
        </Opinions>
      </sentence>
      <sentence id="1:2">
        <text>但 Siri 的答案有時候不是那麼正確。</text>
        <Opinions>
          <Opinion category="SOFTWARE#OPERATION_PERFORMANCE" polarity="negative"/>
        </Opinions>
      </sentence>
    </sentences>
  </Review>
  <Review rid="2">
      ...
  </Review>
</Reviews>
```

每一個句子由 <sentence id= "X:X" > 包住

有情緒的句子，會多個 <Opinion ... polarity="positive"/>
沒情緒的中性句子，就沒有 <Opinion> 這個標籤

句子的情緒標籤：
- 正面評價："positive "
- 負面評價："negative"

課後作業：3C 產品評價情緒分析

- **要求**
  - 請先安裝 **snownlp** 套件，並將資料集 **Comments_Mobile_Phone.xml** 匯入您的程式中。
  - 將 XML 檔案，以 open(檔案名稱, ‘r’, encoding='utf-8') 讀入。
  - 以 BeautifulSoup 剖析 XML 檔案。針對每個句子，抓出下列內容：
    - [[“句子1”, 情緒值], [“句子2”, 情緒值], ...]。其中情緒值 = 1 / 0 / -1
  - 將每個句子，丟入 SnowNLP 取得 0.0~1.0 的情緒分數，若落在 0~1/3，以 -1 計算，若落在 1/3 ~ 2/3，以 0 計算，若落在 2/3 ~ 1，以 1 計算。
  - 計算模型的「**準確度（Accuracy）**」。
  - 參看 Moodle 系統上的作業說明，有**範例程式碼**可以參考。
- **參考輸出**

```
機子的電池不行， -1 -1
一天兩充才能基本上解決， 0 -1
前提還是沒有玩遊戲， 0 0
沒有過長的通話狀態； 0 -1
...
x903 的音質方面沒的說， 1 -1
杜比認證的， 0 0
用了就知道。 0 0
```

```
Accuracy: 35.12%
```

本章總結

- **文本分類**
  - 輸入一段文字，利用模型將它分到某一個類別的問題
- **情感分析**
  - 輸入文字 → 判斷該文字為正面情感或負面情感
- **SnowNLP**
  - 優點：中文特化、簡單易用、無需訓練
  - 演算法：單純貝氏、支援向量機、隱藏式馬可夫模型
  - 語法：
    - model = **SnowNLP**(“這是要被分析的句子。”)
    - model.**sentiments** → 傳回 0.0 ~ 1.0 之間的情緒分數
- **RNN**
  - 斷詞：jieba.cut()
  - 數位化：tensorflow.keras.preprocessing.text.Tokenizer()
  - 序列對齊：tensorflow.keras.preprocessing.sequence.pad_sequences()
  - 詞向量嵌入：torch.nn.Embedding(當前辭典最大數, 壓縮後最大維度)