# 智慧化企業整合

## Project3-

## 應用 Bidirectional LSTM 於 NLP 之 BBC 文章分類

**108034535** 施美全

目錄

一、背景介紹----------p.1

二、模型架構----------p.2

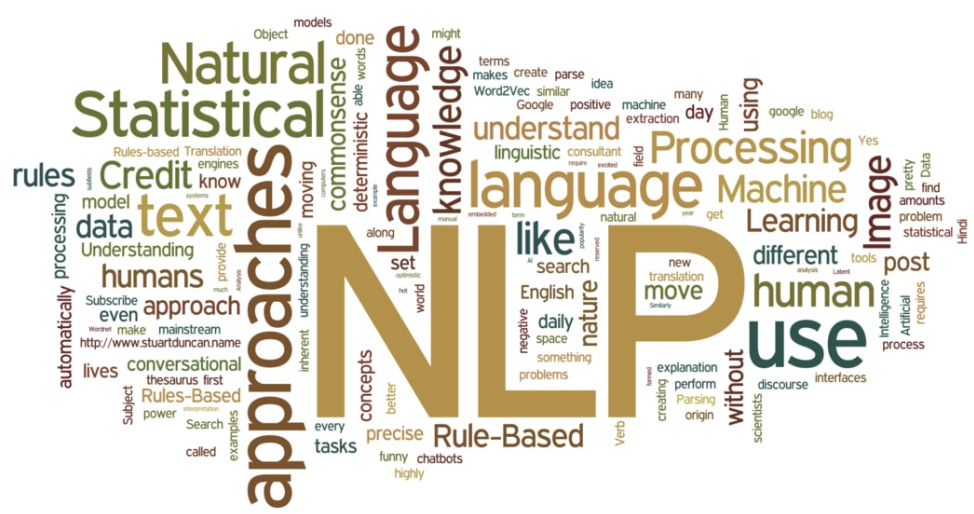
三、模型分析----------p.9

四、結果與討論----------p.11

## 一、　背景介紹

1、<u>情境描述</u>

依據維基百科，NLP 的定義為：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。而此次所探討的主題即是利用深度學習達成文本挖掘之新聞文本分類，而 NLP 目標是設計演算法來讓電腦「理解」自然語言以執行一些任務，如何讓機器學習方法從文字中理解人類語言內含的思想，而此次即是透過深度學習 LSTM 來使電腦進行文本分類，為 NLP 之應用。




2、<u>問題描述 (5W1H)</u>

NLP 的發展歷史非常之久，計算機發明之後，就有以機器翻譯為開端做早期的 NLP 嘗試，但早期做得不是很成功。直到上個世紀八十年代，大部分自然語言處理系統還是基於人工規則的方式，使用規則引擎或者規則系統來做問答、翻譯等功能。第一次突破是上個世紀九十年代，有了統計機器學習的技術，並且建設了很多優質的語料庫之後，統計模型使 NLP 技術有了較大的革新。。接下來的發展基本還是基於這樣傳統的機器學習的技術，從 2006 年深度學習開始，包括現在圖像上取得非常成功的進步之後，已經對 NLP 領域領域影響非常大，希望透過此次報告對 NLP 深度學習領域有其貢獻。

以下將透過 5W1H 進行問題定義：

| | |
|---|---|
| What? | LSTM 文本分類 |
| When? | NLP 時代來臨 |
| Who? | 新聞網站、網路文章、GOOGLE(腥羶色文章) |
| Where? | 服務中心與網路 |
| Why? | 使用電腦即可過濾大量新聞、詞性標註 |
| How? | 資料分析、深度學習 |

## 二、 模型架構

1、資料前處理過程

- 資料來源

為了建立合適的模型，我們採用 Kaggle 內的數據作為資料來源。Kaggle 是一個數據建模和數據分析競賽平台。企業和研究者可在此發布數據，提供統計學家和數據挖掘專家以此資料為基礎，進行競賽以產生最好的模型。

此次報告使用當中“BBC articles fulltext and category”之 BBC News 之文檔(包含來自 BBC 新聞網站的 2225 個文檔，這些文檔與 2004-2005 年五個主題領域的文章)，分類標籤：5（商業，娛樂，政治，體育，科技。

- 數據分析

(1) 首先，將需要的工具匯入(如:csv、Tokenizer、pad_sequences、stopwords… …等等以下註解為其說明。

```
import csv
#讀取與寫入CSV 檔案
import tensorflow as tf
#TensorFlow 是一個採用資料流程圖(data flow graphs)，用於數值計算的開源軟體庫。
import numpy as np
#NumPy是Python語言的一個擴充程式庫。 支援高階大量的維度陣列與矩陣運算，此外也針對陣列運算提供大量的數學函數函式庫
from tensorflow.keras.preprocessing.text import Tokenizer
#keras的Tokenizer進行文本預處理，序列化，向量化
#直接用texts_to_matrix方法就可以直接把texts中的每個text，也就是每個string都轉成LSTM輸入所需要的向量，於是直接輸
from tensorflow.keras.preprocessing.sequence import pad_sequences
#keras只能接受長度相同的序列輸入。因此如果目前序列長度參差不齊，這時需要使用pad_sequences()。該函數是將序列轉化為
#官方語法如下1：
#keras.preprocessing.sequence.pad_sequences(sequences,
  #maxlen=None,
  #dtype='int32',
  #padding='pre',
  #truncating='pre',
  #value=0.)
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))
#NLTK 的 stopwords 語料庫支援了 21 種語言，但仍以英文為主
#停用詞大致分為兩類
#1)人類語言中包含的功能詞，如'the'、'is'、'at'、'which'、'on'等。
#2)詞彙詞，比如'want'等，這些詞應用十分廣泛，但是對這樣的詞搜尋引擎無法保證能夠給出真正相關的搜索結果。
#使用英文版本的停用字
```

(2) 定義參數:

將此次深度學習之參數，統一將其至上，以方便統一修改。

```
vocab_size = 5000
embedding_dim = 512
max_length = 200
trunc_type = 'post'
padding_type = 'post'
oov_tok = '<OOV>'
training_portion = .6
test_portion= .2
```

- 數據清理

（1）首先，刪除 stopwords，停用詞大致分為兩類，一類為人類語言中包含的功能詞，如'the'、'is'、'at'、'which'、'on'等。第二類為詞彙詞，比如'want'等，這些詞應用十分廣泛。

（2）此 data 中共有 2225 條新聞文章。

```
articles = [] #定義文章的陣列
labels = [] #定義標籤的陣列

#主要是利用迴圈，刪除停用字
with open("bbc-text.csv", 'r') as csvfile: #開啟csv檔案
    reader = csv.reader(csvfile, delimiter=',') #讀取csv檔案內容，以逗號分隔欄位
    next(reader)#next()用來讀取下一行的資料
    for row in reader:  #以迴圈輸出每一列
        labels.append(row[0]) #將row[0]加入到label陣列
        article = row[1]
        for word in STOPWORDS:
            token = ' ' + word + ' '
            article = article.replace(token, ' ')
            #前者是要被替換的(token)，後者是替換的新字串
            article = article.replace('  ', ' ')
        articles.append(article)

print(len(labels))  #顯示陣列長度
print(len(articles)) #顯示陣列長度
#可以得知我們數據中有2225條新聞文章
```

- 訓練集：驗證集：測試集

透過各種文獻之建議，本研究將訓練集：驗證集：測試集的劃分比例為 6:2:2(1335:445:445)，並依序顯示其文章長度及標籤長度。

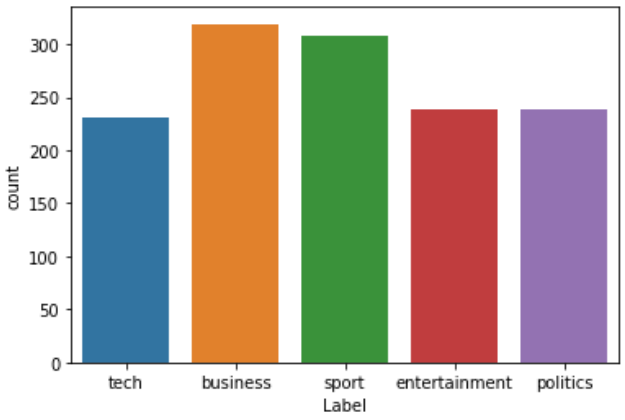
```
#訓練集：驗證集：測試集的劃分比例為6:2:2
#training_portion=0.6  test_portion=0.2 已在上面參數列設置
train_size = int(len(articles) * training_portion)
test_size=int(len(articles) * test_portion)
valid_size=int(len(articles) * test_portion)


train_articles = articles[0: train_size]
train_labels = labels[0: train_size]

test_articles = articles[train_size:(train_size+test_size)]
test_labels = labels[train_size:(train_size+test_size)]

validation_articles = articles[(train_size+test_size):]
validation_labels = labels[(train_size+test_size):]

print(train_size)
print(test_size)

print(len(train_articles))
print(len(train_labels))
print(len(test_articles))
print(len(test_labels))
print(len(validation_articles))
print(len(validation_labels))
```

```
1335
445
1335
1335
445
445
445
445
```

- 訓練集標籤檢視

  利用可視化圖表，可以看出訓練集，各 label 量沒有哪一項特別突出或特別少。

```
#查看訓練集有哪些標籤
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure()
sns.countplot(train_labels)
plt.xlabel('Label')
plt.show()
```



- 將所有新聞內容轉換成電腦容易處理的數字序列:

(1) word_index 就是我們的字典，我們利用該字典，將 1 句話轉成包含多個數字的序列，而每個數字實際上代表著一個 Toke。

```
#Tokenizer 即是將一段文字轉換成一系列的詞彙（Tokens），並為其建立字典。
tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_articles)
word_index = tokenizer.word_index #每個詞的索引
dict(list(word_index.items())[0:10])
#OOV是指有特殊單字及含意，則歸在這類
#可以看到“ <OOV>”是我們語料庫中最常見的標記，其次是“ said”，其次是“ mr”，依此類推。
```

```
{'<OOV>': 1,
 'also': 6,
 'mr': 3,
 'new': 8,
 'one': 10,
 'people': 7,
 'said': 2,
 'us': 9,
 'would': 4,
 'year': 5}
```

(2) 將標記轉化成序列列表

```
#標記化後，下一步是將這些標記轉換為序列列表
train_sequences = tokenizer.texts_to_sequences(train_articles)
print(train_sequences[10])
```

```
[2389, 1, 257, 4144, 19, 633, 524, 257, 4144, 1, 1, 1589, 1, 1, 2389, 19, 496, 1, 1, 134, 265, 1, 134, 26
```

- 序列的 Zero Padding

每篇標題的序列長度並不相同，利用 pad_sequences，將長度超過此數字的序列尾巴會被刪掉；而針對原來長度不足的序列，我們則會在詞彙前面補零。

```
#當為NLP訓練神經網絡時，我們需要序列大小相同
#序列預處理pad_sequences()序列填充
#max_length設定為長度200，上面參數已設定
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding=padding_type, truncating=trun
print(len(train_sequences[0]))
print(len(train_padded[0]))
#第一篇文章的長度為425，變成200

print(len(train_sequences[1]))
print(len(train_padded[1]))

print(len(train_sequences[10]))
print(len(train_padded[10]))
```

```
425
200
192
200
186
200
```

例如:對於第 11 條，長度為 186，我們填充為 200，最後填充，即添加 14 個零。

```
print(train_padded[10])
##padding_type和truncating_type，都有post，例如，對於第11條，長度為186，我們填充為200，最後填充，即添加14個零。
```

```
[2389    1  257 4144   19  633  524  257 4144    1    1 1589    1    1
 2389   19  496    1    1  134  265    1  134  265  883  731  662 2632
    1 1019 1732    1 1485    1    1    1    1    1    1 3941    1    1
  104 4402    1    2 4403 1378  320 4145    1   56  375    1  320 2555
 3730   39   19 3553    1    1    1    1  515    1    1    1  794  761
 2154  409 4146    1  325   19    1  746 2390    1    1  142   10    1
 4404  663 4147    1   19    1  477  731  662    1   83   13  593    1
  257 4144    1  576    1 1732  943    1    1  795 2026  115    1    1
    1 3219   19    1  126  265    1 1774    1  515  504    1 1482 4405
  775 1166    1 1873   10   34  633  256    1   70  511  496  285 1557
   19  499    1    1 1909    1  822    1 3220    1 1283    6    1 2389
  496   19 3214    1    1    1    1    1    1  823   43 1969  586  287
   24  893    1  809   19  365   19   13  285 1557 1379  510   20   67
  776 1189 4148  274    0    0    0    0    0    0    0    0    0    0
    0    0    0    0]
```

- 對驗證集和測試集作相同資料前處理

```
#對驗證集和測試集作相同資料前處理
validation_sequences = tokenizer.texts_to_sequences(validation_articles)
validation_padded = pad_sequences(validation_sequences, maxlen=max_length, padding=padding_type, truncat

print(len(validation_sequences))
print(validation_padded.shape)

test_sequences = tokenizer.texts_to_sequences(test_articles)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_ty

print(len(test_sequences))
print(test_padded.shape)
```

```
445
(445, 200)
445
(445, 200)
```

- 每個標籤列表，轉成 numpy 陣列訓練:Tokenizer()就像是標籤生成器將集 測試集 驗證集都做同樣動作。

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(labels)

training_label_seq = np.array(label_tokenizer.texts_to_sequences(train_labels))
validation_label_seq = np.array(label_tokenizer.texts_to_sequences(validation_labels))
test_label_seq = np.array(label_tokenizer.texts_to_sequences(test_labels))

print(training_label_seq[0])
print(training_label_seq[1])
print(training_label_seq[2])
print(training_label_seq.shape)

print(validation_label_seq[0])
print(validation_label_seq[1])
print(validation_label_seq[2])
print(validation_label_seq.shape)

print(test_label_seq[0])
print(test_label_seq[1])
print(test_label_seq[2])
print(test_label_seq.shape)
```

```
[4]
[2]
[1]
(1335, 1)
[5]
[4]
[3]
(445, 1)
[4]
[5]
[4]
(445, 1)
```

2、模型架構

為使模型的考慮更加全面性並且找到最適切的模型，本研究使用 Bidirectional LSTM 深度學習。一般的 LSTM 只會從前面掃過去，但 Bidirectional LSTM 除了從前面掃過去，也會從後面掃過來，類似從後文來回推前文。

- LSTM

```
model = tf.keras.Sequential([
    #從嵌入層開始，vocab_size = 5000，embedding_dim = 512，參數已設定在上面
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(embedding_dim)),
    #一般的LSTM只會從前面掃過去，但Bidirectional LSTM除了從前面掃過去，也會從後面掃過來，類似從後文來回推前文。
    tf.keras.layers.Dense(embedding_dim, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax'),
    # When we have multiple outputs, softmax convert outputs layers into a probability distribution.
])
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, None, 512)         2560000
_________________________________________________________________
bidirectional_7 (Bidirection (None, 1024)              4198400
_________________________________________________________________
dense_17 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                5130
=================================================================
Total params: 7,288,330
Trainable params: 7,288,330
Non-trainable params: 0
```

- Train 準確率與驗證準確率

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#多分类问题常用:sparse_categorical_crossentropy
num_epochs = 10
history = model.fit(train_padded, training_label_seq, epochs=num_epochs, validation_data=(validation_pad
```

```
Train on 1335 samples, validate on 445 samples
Epoch 1/10
1335/1335 - 502s - loss: 1.7157 - acc: 0.2704 - val_loss: 1.3963 - val_acc: 0.4315
Epoch 2/10
1335/1335 - 493s - loss: 0.7221 - acc: 0.7566 - val_loss: 0.5215 - val_acc: 0.8404
Epoch 3/10
1335/1335 - 494s - loss: 0.1418 - acc: 0.9625 - val_loss: 0.3520 - val_acc: 0.8742
Epoch 4/10
1335/1335 - 496s - loss: 0.0198 - acc: 0.9970 - val_loss: 0.2358 - val_acc: 0.9258
Epoch 5/10
1335/1335 - 493s - loss: 0.0225 - acc: 0.9940 - val_loss: 0.3888 - val_acc: 0.8697
Epoch 6/10
1335/1335 - 493s - loss: 0.0056 - acc: 0.9985 - val_loss: 0.2933 - val_acc: 0.9213
Epoch 7/10
1335/1335 - 489s - loss: 5.6910e-04 - acc: 1.0000 - val_loss: 0.2683 - val_acc: 0.9371
Epoch 8/10
1335/1335 - 489s - loss: 2.4727e-04 - acc: 1.0000 - val_loss: 0.2701 - val_acc: 0.9348
Epoch 9/10
1335/1335 - 485s - loss: 1.4053e-04 - acc: 1.0000 - val_loss: 0.2741 - val_acc: 0.9371
Epoch 10/10
1335/1335 - 483s - loss: 1.0150e-04 - acc: 1.0000 - val_loss: 0.2753 - val_acc: 0.9393
```

- 測試集準確率

```
scores = model.evaluate(test_padded, test_label_seq, verbose=2)
print('Test accuracy:', scores[1])
```

```
445/445 - 29s - loss: 0.2362 - acc: 0.9461
Test accuracy: 0.9460674
```

## 三、 模型結果與分析

- Train 準確率與驗證準確率

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#多分类问题常用:sparse_categorical_crossentropy
num_epochs = 10
history = model.fit(train_padded, training_label_seq, epochs=num_epochs, validation_data=(validation_pad
```

```
Train on 1335 samples, validate on 445 samples
Epoch 1/10
1335/1335 - 502s - loss: 1.7157 - acc: 0.2704 - val_loss: 1.3963 - val_acc: 0.4315
Epoch 2/10
1335/1335 - 493s - loss: 0.7221 - acc: 0.7566 - val_loss: 0.5215 - val_acc: 0.8404
Epoch 3/10
1335/1335 - 494s - loss: 0.1418 - acc: 0.9625 - val_loss: 0.3520 - val_acc: 0.8742
Epoch 4/10
1335/1335 - 496s - loss: 0.0198 - acc: 0.9970 - val_loss: 0.2358 - val_acc: 0.9258
Epoch 5/10
1335/1335 - 493s - loss: 0.0225 - acc: 0.9940 - val_loss: 0.3888 - val_acc: 0.8697
Epoch 6/10
1335/1335 - 493s - loss: 0.0056 - acc: 0.9985 - val_loss: 0.2933 - val_acc: 0.9213
Epoch 7/10
1335/1335 - 489s - loss: 5.6910e-04 - acc: 1.0000 - val_loss: 0.2683 - val_acc: 0.9371
Epoch 8/10
1335/1335 - 489s - loss: 2.4727e-04 - acc: 1.0000 - val_loss: 0.2701 - val_acc: 0.9348
Epoch 9/10
1335/1335 - 485s - loss: 1.4053e-04 - acc: 1.0000 - val_loss: 0.2741 - val_acc: 0.9371
Epoch 10/10
1335/1335 - 483s - loss: 1.0150e-04 - acc: 1.0000 - val_loss: 0.2753 - val_acc: 0.9393
```

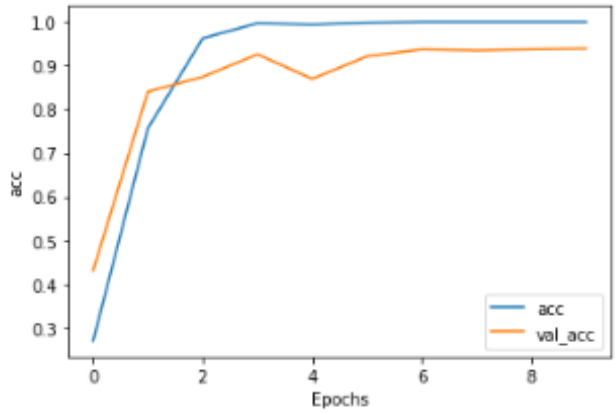
- 測試集準確率

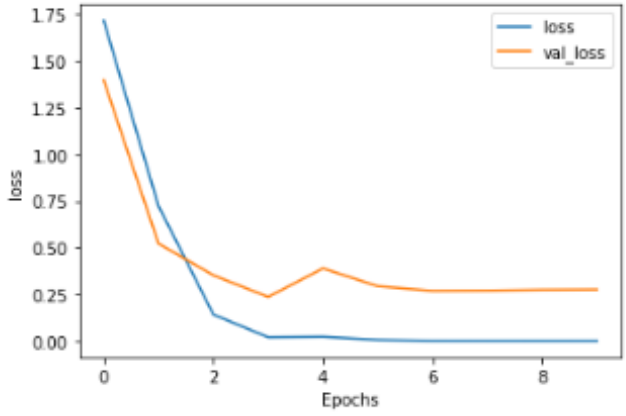
```
scores = model.evaluate(test_padded, test_label_seq, verbose=2)
print('Test accuracy:', scores[1])
```

```
445/445 - 29s - loss: 0.2362 - acc: 0.9461
Test accuracy: 0.9460674
```

- 模型圖示

```python
import matplotlib.pyplot as plt
#顯示訓練準確度 跟 val_accuracy/ loss 和 val_loss

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "acc")
plot_graphs(history, "loss")
```



## 四、　結果與討論

1、結論

此次利用隨機初始化所有詞向量（如前述的隨機轉換），並利用平常訓練神經網路的反向傳播算法（Backpropagation LSTM），讓神經網路自動學到一組適合當前 NLP 任務的詞向量。使其訓練準確率到 100%，驗證集準確率到達 93.9%，測試集準確率到 94.6%，對我來說可說是相當大的突破。

2、未來展望

此次模型僅針對 BBC 新聞文章進行訓練，同時此方法亦可用在其他新聞分類，隨著網際網路日漸發達，網路文章多樣性是必須要過濾器，以保障未成年少年之心靈。並且也可以利用此方法來分類顧客的留言、回饋或客訴留言，從中快速分類客戶之不滿是屬於哪種類型，並且傳送至相關部門，以快速反應顧客需求，就如工業 4.0 所提倡的是以需而製。最後即是 NLP 之領域發展貢獻，可以提供此案例，使得此領域有所微小提升，亦可應用於機器人能讀懂人類的自然語言，以方便與人類溝通，使我們生活更方便。