【智慧化企業整合】

Final Project

垃圾郵件識別問題

指導教授：邱銘傳 教授

學生：林鈺婷 109034542

摘要

在資訊快速發展的現今，網路上各種資訊隨手可得，然而我們多數人都是在未經同意的情況下被動地接收資訊，雖然可能換得娛樂與便利，卻也因此接收到許多無用的資訊。本研究藉由建立郵件識別的深度學習模型，依據訊息的特徵進行分類，並以 UCI 上公開的郵件資料集為基準，對訊息做進一步的篩檢，來嘗試預測並排除垃圾郵件，以減少人工識別所接收到的郵件是否為垃圾郵件的時間以及減少垃圾郵件所佔用的空間。

一、緒論

1.1. 研究背景

網路為人們帶來便利，也為人們帶來煩惱，而垃圾郵件就是最為使人頭痛的煩惱。在過去亦有很多學者為解決垃圾郵件的問題提出許多分類方法，對於垃圾郵件製造方，也同樣為應對垃圾郵件的解決方法提出相對應的策略以避開過濾。因此到了如今 2021 年，我們依舊會接收到的大量垃圾郵件，不僅是垃圾郵件越來越多，有害的廣告亦繼續泛濫。新的反垃圾郵件措施總會遇到新的垃圾郵件模式，導致沒有解決垃圾郵件問題的萬靈藥。

1.2. 研究動機與目的

人工智慧的核心是學習，且未來之人工智慧、智慧製造、工業 4.0 的趨勢也漸漸迎來，使用機器學習和其他人工智慧領域的技術，軟體將變得更智能，無需人工協助。因此使用人工智慧防止垃圾郵件將是迫切的議題，單靠人工的方式難以跟上垃圾郵件的成長，而人工智慧的好處在於它能自行適應環境，透過人工智慧亦能幫助我們發現新的垃圾郵件模式。

1.3. 5W1H 分析法

表 1、5W1H

5W1H	描述
Who?	接收郵件的大眾
What?	解決收到大量垃圾郵件的困擾
Why?	大量垃圾郵件造成人工識別的麻煩以及儲存空間的浪費
Where?	任何可接收郵件的地方
When?	收到郵件時
How?	以機器學習、深度學習方法建立垃圾郵件識別模型

二、文獻探討

2.1. 自然語言處理（Natural Language Processing , NLP）

自然語言處理是人工智慧和語言學領域的分支學科，此領域探討如何處理及運用自然語言，包括多方面和步驟，基本有認知、理解、生成等部分。自然語言的認知與理解，是透過複雜的數學模型及演算法來讓電腦把輸入的語言變成有意義的符號和關係，然後根據目的再處理；而自然語言生成系統則是把電腦數據轉化為自然語言。

早期的自然預言處理技術主要基於統計的概念去訓練模型，讓演算法閱讀大量類似字典的文章段落，再讓演算法計算單字、句子出現的機率，然而此種方式無法使系統很好地辨識複雜的文法，同時，這樣子的模型所產生的字句更是生硬且結構錯亂。但隨著深度學習與演算法模型的突破，新的訓練方式已能更好的處理以上所提的問題。

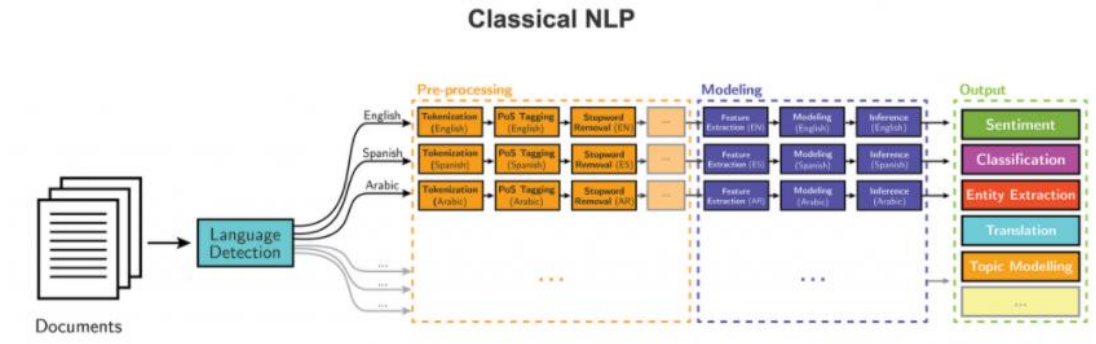

圖 1、傳統 NLP 模型示意圖

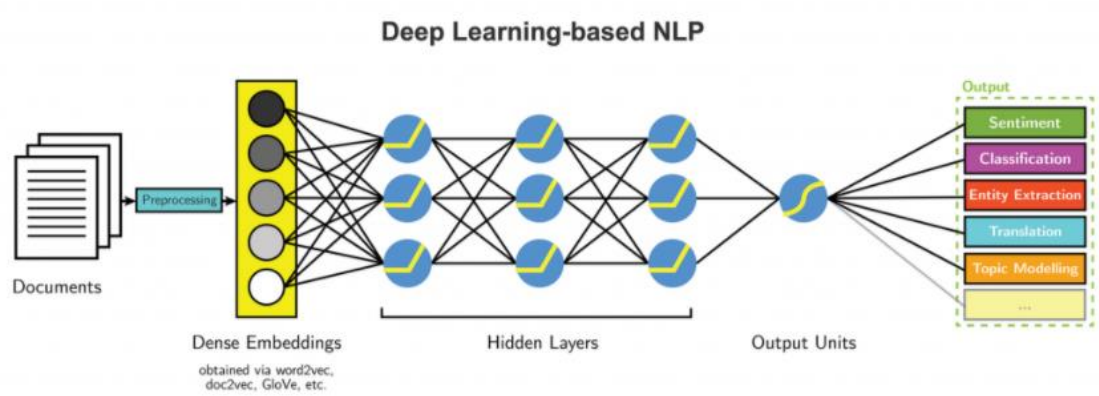

圖 2、深度學習 NLP 模型示意圖

2.2. 遞歸神經網路（Recurrent Neural Network , RNN）

遞歸神經網路常應用在處理時間、空間序列上有強關聯的訊息，因此常被應用在 NLP（Natural Language Processing，自然語言處理）領域上。由於我們在閱讀文章時，是根據上下文來理解文章意義，RNN 的概念在於將狀態在自身網絡中循環傳遞，因此可以接受更廣泛的時間序列結構輸入，允許訊息持續存在。

RNN 的缺點為無法捕捉長期時間（當序列的距離太大時）之間的關聯，因此簡單的 RNN 結構無法處理隨著遞歸權重指數級爆炸或消失的問題（Vanishing gradient problem）。

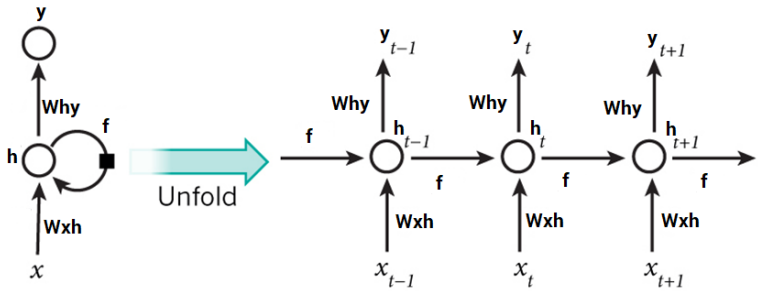

圖 3、RNN 模型示意圖

2.3. 長短期記憶模型（Long Short-Term Memory , LSTM）

長短期記憶模型是一種特殊的 RNN 模型，其特色是能夠學習長距離的依賴關係（Long-Term Dependencies），它不同於 RNN 只有單一的神經網路層（tanh），而是有四個層，以特別的方式進行溝通，因此 LSTM 可以用來解決 RNN 無法捕捉長期時間之間關聯的限制。

LSTM 的核心關鍵在於 Cell State（單元狀態）沿着整個鏈運行，可以增加或者刪除單元狀態中的訊息，而這些訊息經過”gate”的結構來處理，分別為遺忘門（Forget gate）、輸入門（Input gate）、輸出門（Output gate），以選擇性地控制訊息通過。

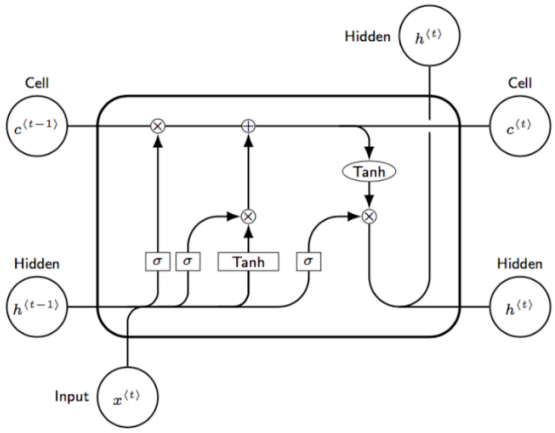

圖 4、LSTM 模型示意圖

2.4. 雙向長短期記憶模型（Bidirectional Long Short-Term Memory, Bi-LSTM）

雙向 LSTM 是傳統 LSTM 的擴充套件，可以提高序列分類問題的模型效能。在輸入序列為時間問題的分類資料上，雙向 LSTM 在輸入序列上所訓練的模型為兩個 LSTM 而非一個。輸入序列中的第一個為原始樣本，而第二個為輸入序列的反向樣本，這種作法可為模型提供額外的上下文，並且能更快、更全面地學習該問題。

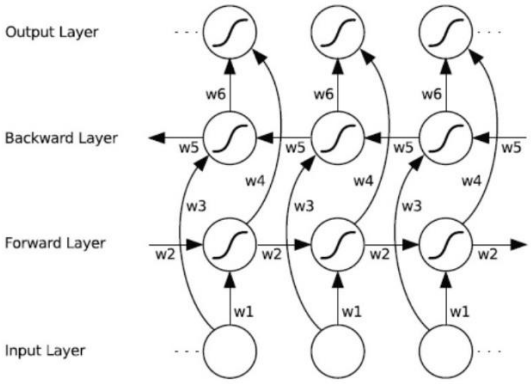

圖 5、Bidirectional LSTM 模型示意圖

三、資料前處理

3.1. 資料來源

本研究由 UCI 上的公開資料集中取得郵件資料集作為模型的資料來源。資料集大小為（5572,2），共計 5572 筆訊息資料，並且包含 2 個資料欄位，分別為郵件類別（label）以及訊息內容（message）。

（資料來源：https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection）

3.2. 資料處理

(1) 導入函式庫與資料集

```
import tensorflow as tf
print(tf.__version__)
2.2
```

```
# import libraries for reading data, exploring and plotting
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
%matplotlib inline

# library for train test split
from sklearn.model_selection import train_test_split

# deep learning libraries for text pre-processing
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Modeling
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense, Dropout, LSTM, Bidirectional
```

圖 6、導入函式庫之 python 程式碼

利用 pandas 中的函式 pd.read_csv()來匯入資料，並將資料欄位命名為 label 及 message，匯入資料後可以看出共有 5572 筆郵件資料，其中 label 欄位的 ham 代表有用的郵件，而 spam 則代表垃圾郵件。

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

圖 7、資料集示意圖

```
# 匯入資料集
url = 'https://raw.githubusercontent.com/ShresthaSudip/SMS_Spam_Detection_DNN_LSTM_BiLSTM/master/SMSSpamCollection'
message = pd.read_csv(url, sep ='\t',names=["label", "message"])
message
```

圖 8、匯入資料集之 python 程式碼

(2) 檢查資料是否包含空值

利用函式 message.info()，來檢查資料，從結果可以看到資料的兩個欄位 label 以及 message 皆不包含空值，因此不須再做空值處理。

```
# Check data information
message.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   label    5572 non-null   object
 1   message  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

圖 9、檢查空值之 python 程式碼與結果

(3) 字詞分析視覺化

利用函式 WorldCloud()，視覺化垃圾郵件的字詞分析，由圖中可以看出垃圾郵件最常出現字詞的大致上為 Free、call、text…等。

圖 10、垃圾郵件字詞分析

```
# wordcloud of spam messages
spam_msg_cloud = WordCloud(width =520, height =260,
                           stopwords=STOPWORDS, max_font_size=50, background_color ="black", colormap='coolwarm').generate(spam_msg_text)
plt.figure(figsize=(16, 10))
plt.imshow(spam_msg_cloud, interpolation='bilinear')
plt.axis('off') # turn off axis
plt.show()
```

圖 11、垃圾郵件字詞分析之程式碼

(4) 降取樣（Downsampling）

利用函式 discribe()，觀察 label 欄位的資料。由下圖可以看出 ham messages 有 4825 筆，而 spam messages 卻只有 747 筆，兩者有明顯的數量差距，spam messages 大約為 ham messages 的 15%，為極度不平衡的資料。

```
#資料描述
message.groupby('label').describe().T
```

	label	ham	spam
message	count	4825	747
	unique	4516	653
	top	Sorry, I'll call later	Please call our customer service representativ...
	freq	30	4

圖 12、資料描述之 python 程式碼與結果

本研究選用降取樣的方法，隨機的刪減 ham messages 的資料，使得 ham messages 和 spam messages 的資料達平衡。由下圖可以看出，經過降取樣後，兩類資料的數量相同，各為 747 筆資料。

```
#Downsample the ham msg
ham_msg_df = ham_msg.sample(n = len(spam_msg), random_state = 44)
spam_msg_df = spam_msg
print(ham_msg_df.shape, spam_msg_df.shape)
```

```
(747, 2) (747, 2)
```

圖 13、降取樣之 python 程式碼

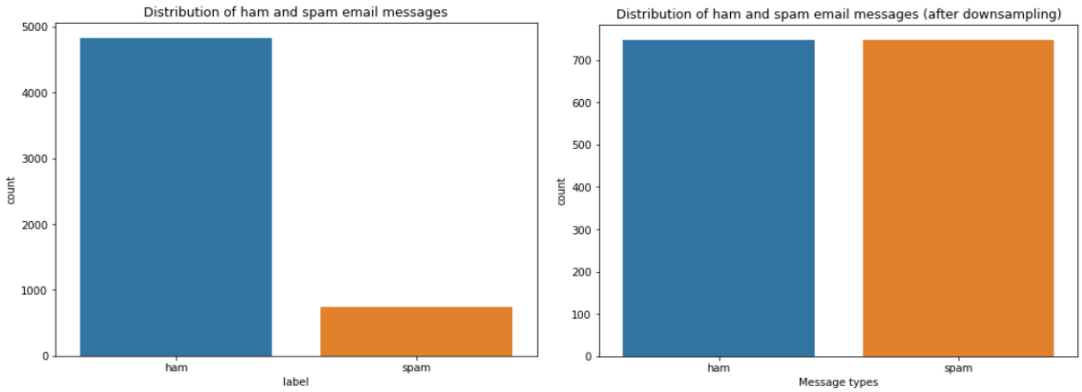

圖 14、降取樣前後之資料筆數比較圖

(5) 資料標籤化

針對資料欄位 label 進行標籤化，label 資料僅有兩種種類，使用 map 的方法來做資料標籤化，其中將 ham messages 標示為 0，而 spam messages 則標示為 1。

	label	message	msg_type
0	ham	Height of recycling: Read twice- People spend ...	0
1	ham	Yup song bro. No creative. Neva test quality. ...	0
2	ham	Feb <#> is "I LOVE U" day. Send dis to...	0
3	ham	Don't forget who owns you and who's private pr...	0
4	ham	Lol no. I just need to cash in my nitros. Hurr...	0
...	...	...	...
1489	spam	Want explicit SEX in 30 secs? Ring 02073162414...	1
1490	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...	1
1491	spam	Had your contract mobile 11 Mnths? Latest Moto...	1
1492	spam	REMINDER FROM O2: To get 2.50 pounds free call...	1
1493	spam	This is the 2nd time we have tried 2 contact u...	1

1494 rows × 3 columns

圖 15、資料標籤化之示意圖

```
#資料標籤化 (ham=0 & spam=1)
msg_df['msg_type']= msg_df['label'].map({'ham': 0, 'spam': 1})
msg_label = msg_df['msg_type'].values
msg_df
```

圖 16、資料標籤化之 python 程式碼

(6) 資料切分

將處理完的資料切分為 80%的訓練集（Training data）以及 20%的測試集（Testing data），以供後續的模型訓練使用。

```
# 將資料切分為 training and testing
train_msg, test_msg, train_labels, test_labels = train_test_split(msg_df['message'], msg_label, test_size=0.2, random_state=434)
```

圖 17、切分資料集之 python 程式碼

(7) 分詞（Tokenization）

利用 Tokenizer()函式，將訓練集資料內容切成單詞，並將每個單詞編號，由結果可看出訓練集內共有 4169 個不同的單詞。

```
# 定義 pre-processing-Tokenizer 的超參數
max_len = 50
trunc_type = "post"
padding_type = "post"
oov_tok = "<OOV>"
vocab_size = 500

tokenizer = Tokenizer(num_words = vocab_size, char_level=False, oov_token = oov_tok)
tokenizer.fit_on_texts(train_msg)

# Get the word_index
word_index = tokenizer.word_index
print(word_index)

# check how many words
tot_words = len(word_index)
print('There are %s unique tokens in training data. ' % tot_words)
```

圖 18、Tokenization 之 python 程式碼

```
{'<OOV>': 1, 'to': 2, 'you': 3, 'a': 4, 'i': 5, 'call': 6, 'the': 7, 'u': 8, 'your': 9, 'for': 10,
There are 4169 unique tokens in training data.
```

圖 19、單詞編號結果示意圖

(8) 排序（Sequencing）與填充（Padding）

利用 texts_to_sequences()函式，將訓練集與測試集的每個句子用數字序列表示，接著再使用 pad_sequences()函式使每個序列擁有相同的長度。

```
# Sequencing and padding on training and testing
training_sequences = tokenizer.texts_to_sequences(train_msg)
training_padded = pad_sequences (training_sequences, maxlen = max_len, padding = padding_type, truncating = trunc_type )
testing_sequences = tokenizer.texts_to_sequences(test_msg)
testing_padded = pad_sequences(testing_sequences, maxlen = max_len,
padding = padding_type, truncating = trunc_type)
```

圖 20、Sequencing & Padding 之 python 程式碼

四、模型架構

本研究共建構三種類神經網路模型來尋找最適合用在垃圾郵件識別的模型，分別是 dense model、長短期記憶模型以及雙向長短期記憶模型。利用這三種模型來進行訓練，並進行超參數的整及優化，以找到最佳化的模型。而在模型訓練的過程中，使用 EarlyStopping()函式，監控測試集的損失變化，來判斷是否要提早停止模型訓練，以避免模型過擬合的情形發生。

4.1. Dense Model

此 Dense 模型架構為一嵌入層（Embedding）、一池化層（Pooling1D）、一密集層（Dense）以及一輸出層（dense），並且再加入一層 dropout。其中嵌入層將每個單詞對應到一個 N 維向量，embedding_dim 是該向量的大小，通常有相似含義的兩個單詞有非常接近的向量。而池化層則有助於減少模型中的參數數量，可用來避免過擬合的情況發生。

```
# Dense model hyperparameters
max_len = 50
vocab_size = 500 # As defined earlier
embeding_dim = 16

# Dense model
model = Sequential()
model.add(Embedding(vocab_size, embeding_dim ,input_length=50))
model.add(GlobalAveragePooling1D())
model.add(Dense(24, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_absolute_error',optimizer='adam' ,metrics=['accuracy'])

# fitting a dense spam detector model
early_stop = EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(training_padded, train_labels, epochs=30,
                    validation_data=(testing_padded, test_labels),callbacks =[early_stop], verbose=2)
```

圖 21、Dense Model 之 python 程式碼

Embedding	(500,16,50)
dense	24,relu
dropout	0.2
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1283
accuracy	0.9431

Embedding	(500,32,50)
dense	24,relu
dropout	0.2
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1288
accuracy	0.9331

Embedding	(500,16,50)
dense	64,relu
dropout	0.2
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1246
accuracy	0.9465

Embedding	(500,16,50)
dense	24,sigmoid
dropout	0.2
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1420
accuracy	0.9431

Embedding	(500,16,50)
dense	24,relu
dropout	0.3
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.0631
accuracy	0.9465

Embedding	(500,16,50)
dense	24,relu
dropout	0.2
dense	1,sigmoid
loss_function	mean_absolute_error
optimizer	adam
epochs	30
loss	0.0592
accuracy	0.9532

Embedding	(500,16,50)
dense	24,relu
dropout	0.2
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	sgd
epochs	30
loss	0.6512
accuracy	0.7659

Embedding	(500,16,50)
dense	24,relu
dropout	0.2
dense	1,softmax
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1126
accuracy	0.4716

圖 22、Dense Model 之超參數調整過程

上圖為調整 Dense 模型超參數的部分過程，在嘗試多種超參數的組合後，找到一組最佳的組合，為 Embedding（500,16,50）、Dense（24,relu）、dropout＝0.2、Dense（1,sigmoid）、loss_function＝‘mean_absolute_error’、optimizer＝‘adam’，在此參數組合下，模型測試集的損失為 0.0592，而準確率為 0.9532。

接著我們進一步繪製出每一個 epochs 所對應的訓練集與測試集的的損失及準確率圖。從此兩圖形可以看出訓練集與測試集有相同的變化趨勢，其損失隨著 epochs 數增加有顯著的下降；而準確率也有明顯的提升。

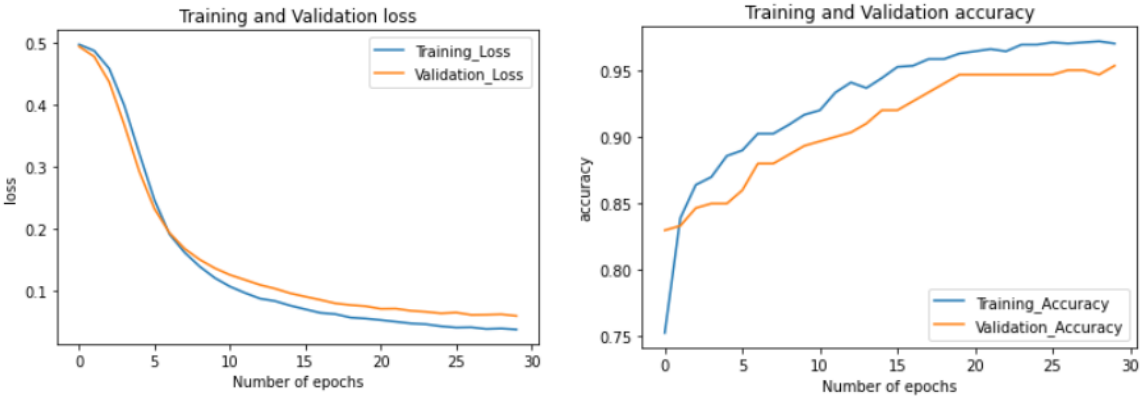

圖 23、Dense Model 之損失與準確率圖

4.2. 長短期記憶模型架構（LSTM Model）

此 LSTM 模型架構為一嵌入層（Embedding）、一輸出層（Dense）以及兩層 keras 中的所定義的 LSTM 層。

```
# #LSTM hyperparameters
vocab_size = 500 # As defined earlier
embeding_dim = 36

#LSTM model
model1 = Sequential()
model1.add(Embedding(vocab_size, embeding_dim, input_length=max_len))
model1.add(LSTM(32, dropout=0.2, return_sequences=True))
model1.add(LSTM(32, dropout=0.2, return_sequences=True))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])

# fitting a LSTM model
early_stop = EarlyStopping(monitor='val_loss', patience=2)
history = model1.fit(training_padded, train_labels, epochs=30,
                     validation_data=(testing_padded, test_labels),callbacks =[early_stop], verbose=2)
```

圖 24、LSTM Model 之 python 程式碼

Embedding	(500,16,50)
LSTM(n,dropout)	(20,0.2)
LSTM(n,dropout)	(20,0.2)
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.2359
accuracy	0.9243

Embedding	(500,32,50)
LSTM(n,dropout)	(20,0.2)
LSTM(n,dropout)	(20,0.2)
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.2259
accuracy	0.9321

Embedding	(500,36,50)
LSTM(n,dropout)	(20,0.2)
LSTM(n,dropout)	(20,0.2)
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.216
accuracy	0.9328

Embedding	(500,36,50)
LSTM(n,dropout)	(24,0.2)
LSTM(n,dropout)	(24,0.2)
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.2449
accuracy	0.9252

Embedding	(500,36,50)
LSTM(n,dropout)	(32,0.2)
LSTM(n,dropout)	(32,0.2)
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1987
accuracy	0.9357

Embedding	(500,36,50)
LSTM(n,dropout)	(32,0.25)
LSTM(n,dropout)	(32,0.25)
dense	1,sigmoid
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.2192
accuracy	0.9312

Embedding	(500,36,50)
LSTM(n,dropout)	(32,0.2)
LSTM(n,dropout)	(32,0.2)
dense	1,sigmoid
loss_function	mean_squared_error
optimizer	adam
epochs	30
loss	0.063
accuracy	0.9256

Embedding	(500,36,50)
LSTM(n,dropout)	(32,0.2)
LSTM(n,dropout)	(32,0.2)
dense	1,tanh
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.221
accuracy	0.9097

圖 25、LSTM Model 之超參數調整過程

上圖為調整 LSTM 模型超參數的部分過程，在嘗試多種超參數的組合後，找到一組最佳的組合，為 Embedding（500,36,50），LSTM（32, dropout = 0.2），LSTM（32, dropout = 0.2），Dense（1,sigmoid），loss_function = 'binary_crossentropy'，optimizer = 'adam'，在此參數組合下，模型測試集的損失為 0.1987，而準確率為 0.9357。

接著我們進一步繪製出每一個 epochs 所對應的訓練集與測試集的損失及準確率圖。由此兩圖形可以看出訓練集與測試集有大致相同的變化趨勢，其損失隨著 epochs 數增加有顯著的下降；而準確率也有明顯的提升。

圖 26、LSTM Model 之損失與準確率圖

4.3. 雙向長短期記憶模型（Bidirectional LSTM Model）

此雙向的 LSTM 模型架構為一嵌入層（Embedding）、一輸出層（Dense）以及一雙向的 LSTM 層。

```
# Bidirectional LSTM hyperparameters
vocab_size = 500 # As defined earlier
embeding_dim = 32

# Bidirectional LSTM Spam detection architecture
model2 = Sequential()
model2.add(Embedding(vocab_size, embeding_dim, input_length=max_len))
model2.add(Bidirectional(LSTM(36, dropout=0.2, return_sequences=True)))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss = 'mean_absolute_error', optimizer = 'adam', metrics=['accuracy'])

# Training
early_stop = EarlyStopping(monitor='val_loss', patience=2)
history = model2.fit(training_padded, train_labels, epochs=30,
                     validation_data=(testing_padded, test_labels),callbacks =[early_stop], verbose=2)
```

圖 27、Bi-LSTM Model 之 python 程式碼

Embedding	(500,16,50)
Bi-LSTM (n,dropout)	(20,0.2)
dense	(1,sigmoid)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1871
accuracy	0.9443

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(20,0.2)
dense	(1,sigmoid)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1633
accuracy	0.9564

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(24,0.2)
dense	(1,sigmoid)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1791
accuracy	0.9507

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(36,0.2)
dense	(1,sigmoid)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1319
accuracy	0.9636

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(36,0.25)
dense	(1,sigmoid)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1958
accuracy	0.9492

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(36,0.2)
dense	(1,sigmoid)
loss_function	mean_absolute_error
optimizer	adam
epochs	30
loss	0.0533
accuracy	0.9569

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(36,0.2)
dense	(1,tanh)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1492
accuracy	0.9558

Embedding	(500,32,50)
Bi-LSTM (n,dropout)	(36,0.2)
dense	(1,relu)
loss_function	binary_crossentropy
optimizer	adam
epochs	30
loss	0.1871
accuracy	0.9404

圖 28、Bi-LSTM Model 之超參數調整過程

上圖為調整模型超參數的部分過程，在嘗試多種超參數的組合後，找到一組最佳的組合，為 Embedding（500,32,50），Bi-LSTM（36, dropout = 0.2），Dense（1,sigmoid），loss_function = 'binary_crossentropy'，optimizer = 'adam'，在此參數組合下，模型測試集的損失為 0.1519，而準確率為 0.9636。

接著我們進一步繪製出每一個 epochs 所對應的訓練集與測試集的損失及準確率圖。由此兩圖形可以看出訓練集與測試集有大致相同的變化趨勢，其損失隨著 epochs 數增加有顯著的下降；而準確率也有明顯的提升。

圖 29、Bi-LSTM Model 之損失與準確率圖

4.4. 模型比較

在訓練完 Dense 模型、LSTM 模型以及 Bi-LSTM 模型後，比較這三種模型在資料前處理後所設定的超參數組合下的模型準確度，以及經過超參數調整後的最佳化超參數組合下的模型準確度。由下表可以看出，此三種模型經過超參數的調整後，準確率都有些微的提升，而其中又以 Bidirectional-LSTM 模型的準確度為最高。

表 2、三種類神經網路模型之準確率比較表

模型	Training Process	資料前處理	最佳化模型	Improvement
Dense	▪ Embedding ▪ Dense ▪ Dropout ▪ Loss_function ▪ Optimizer	0.9431	0.9532	+1.01%
LSTM	▪ Embedding ▪ LSTM ▪ Dense ▪ Loss_function ▪ Optimizer	0.9243	0.9357	+1.14%
Bi-LSTM	▪ Embedding ▪ Bi-LSTM ▪ Dense ▪ Loss_function ▪ Optimizer	0.9443	0.9636	+1.93%

五、結論與未來展望

5.1. 結論

本次研究主題為垃圾郵件的識別，透過建構三種類神經網路模型，來分類有效郵件與垃圾郵件。由模型訓練結果比較可以看出，雙向的長短期記憶模型有最高的準確率，達到 96.36%，推論原因為此模型考慮到訊息上下文的連接，因此其準確率會比長短期記憶模型更高。

透過此模型的應用，我們可以有效的區分所接收到的郵件是否為垃圾郵件，並自動將其放入垃圾桶內，以減少人工刪除所耗費的時間以及垃圾郵件所佔用的空間。

5.2. 未來展望

由於本研究使用的資料集為不平衡資料，因此在資料前處理的過程中進行有效郵件的降取樣（downsampling），使得資料筆數減少，可能在模型訓練上會有影響，因此模型仍然有改善的空間，未來可考慮增加垃圾郵件的資料筆數，再進行模型訓練。

而本研究所建構的類神經網路模型不僅可用來區別垃圾郵件，還有許多更深入的應用，例如：文本分類、情感分析、文本生成、語言翻譯等，因此若未來將模型建構的更加完善或嘗試其他不同模型，將可以解決更多實務上的問題。

參考資料

1. http://203.64.157.175/ER/file.axd?file=2014%2F9%2F17263581-201212-201301300045-201301300045-259-289.pdf
2. https://oosga.com/nlp/
3. https://ithelp.ithome.com.tw/articles/10209418
4. https://www.itread01.com/content/1545275535.html