HW4 Sentiment Analysis

0813458 簡辰穎

一.Import Library

Import Library

```
import pandas as pd
import numpy as np

#去除停頓詞
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
import nltk
import spacy  #要先下載好model
from nltk import word_tokenize

#文字轉向量
from gensim.utils import simple_preprocess
from gensim import models
from gensim.models.word2vec import Word2Vec
import gensim
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

#Split train test set
from sklearn.model_selection import train_test_split

#Create model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv1D, Conv2D, MaxPool2D, MaxPooling1D, Flatten
from keras.layers import ZeroPadding2D, Activation
from keras.layers.embeddings import Embedding
from keras.layers import LSTM
from tensorflow import keras

#plot
import matplotlib.pyplot as plt
```

二.資料前處理

按照之前處理作業二的方法進行資料前處理

a. 只保留 text, stars

a. 讀取資料僅保留 'text' 與 'star' 兩個欄位

```
In [5]: df = df[['stars', 'text']]
        df.head()
```

Out[5]:

	stars	text
0	5	My wife took me here on my birthday for breakf...
1	5	I have no idea why some people give bad review...
2	4	love the gyro plate. Rice is so good and I als...
3	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...
4	5	General Manager Scott Petello is a good egg!!!...

b. Stars>=4 轉成 1 / <4 轉成 0 -> 1 代表 positive, 0 代表 negative

```
In [6]: df.stars[df.stars<4] = 0
        df.stars[df.stars>=4] = 1
        df.head(20)
```

Out[6]:

	stars	text
0	1	My wife took me here on my birthday for breakf...
1	1	I have no idea why some people give bad review...
2	1	love the gyro plate. Rice is so good and I als...
3	1	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...
4	1	General Manager Scott Petello is a good egg!!!...
5	1	Quiessence is, simply put, beautiful. Full wi...
6	1	Drop what you're doing and drive here. After I...
7	1	Luckily, I didn't have to travel far to make m...
8	1	Definitely come for Happy hour! Prices are ama...
9	1	Nobuo shows his unique talents with everything...
10	1	The oldish man who owns the store is as sweet ...

c. 去除停頓詞(用 spacy)

b. 去除停頓詞stop words

```
In [7]: #用 Python NLP 中的 spacy 進行stopwords 處理
        #先看一下spacy中有哪些停頓詞
        nlp = spacy.load('en_core_web_sm')
        spacy_stopwords = spacy.lang.en.stop_words.STOP_WORDS
        print('spaCy has {} stop words'.format(len(spacy_stopwords)))
        print('The first twenty stop words are {}'.format(list(spacy_stopwords)[:20]))

        spaCy has 326 stop words
        The first twenty stop words are ['moreover', 'somehow', 'therein', 'nine', 'in', 'make', 'keep', 'anyway', 'even', 'be', 'serio
        us', 'fifty', 'next', 'myself', 'you', 'thereupon', 'via', 'hereby', 'hereupon', 'yourself']
```

```
In [8]: """
        1. imdb_df['text'].apply將每個row的text值分別取出
        2. apply(lambda x: rmsw_function(x, spacy_stopwords)) 這邊可以拆解成
            - 將imdb_df['text'] 取出，當作lambda function的 x
            - 然後丟進 rmsw_function 得到ruturn的值
            - 然後放到 imdb['spacy_rmsw_text_fancy']之中
        """
        #nltk.download('punkt')
        #def rmsw_function(text, stopword_list):
        #    return ' '.join([word for word in word_tokenize(text) if word not in stopword_list])
        #df1['rmsw_text'] = df1['text'].apply(lambda x: rmsw_function(x, spacy_stopwords))
        df['rmsw_text'] =  df['text'].apply(lambda x:' '.join([word for word in x.split() if word not in spacy_stopwords]))
```

```
In [9]: df['rmsw_text'][0]

Out[9]: 'My wife took birthday breakfast excellent. The weather perfect sitting outside overlooking grounds absolute pleasure. Our wait
        ress excellent food arrived quickly semi-busy Saturday morning. It looked like place fills pretty quickly earlier better. Do fa
        vor Bloody Mary. It phenomenal simply best I\'ve had. I\'m pretty sure use ingredients garden blend fresh order it. It amazing.
        While EVERYTHING menu looks excellent, I white truffle scrambled eggs vegetable skillet tasty delicious. It came 2 pieces gridd
        led bread amazing absolutely meal complete. It best "toast" I\'ve had. Anyway, I can\'t wait back!'
```

```
In [10]: df['text'][0]

Out[10]: 'My wife took me here on my birthday for breakfast and it was excellent.  The weather was perfect which made sitting outside ov
         erlooking their grounds an absolute pleasure.  Our waitress was excellent and our food arrived quickly on the semi-busy Saturda
         y morning.  It looked like the place fills up pretty quickly so the earlier you get here the better.\n\nDo yourself a favor and
         get their Bloody Mary.  It was phenomenal and simply the best I\'ve ever had.  I\'m pretty sure they only use ingredients from
         their garden and blend them fresh when you order it.  It was amazing.\n\nWhile EVERYTHING on the menu looks excellent, I had th
         e white truffle scrambled eggs vegetable skillet and it was tasty and delicious.  It came with 2 pieces of their griddled bread
         with was amazing and it absolutely made the meal complete.  It was the best "toast" I\'ve ever had.\n\nAnyway, I can\'t wait to
         go back!'
```

d. 文字轉向量(word2vec, tokenizer.....)

- Word2vec

- 使用word2vec

```
In [11]: tokenized_tweet = df['rmsw_text'].apply(lambda x: x.split()) # tokenizing

         model_w2v = gensim.models.Word2Vec(
                     tokenized_tweet,
                     vector_size=200, # desired no. of features/independent variables
                     window=5, # context window size
                     min_count=2, # Ignores all words with total frequency lower than 2.
                     sg = 1, # 1 for skip-gram model
                     hs = 0,
                     negative = 10, # for negative sampling
                     workers= 32, # no.of cores
                     seed = 34
         )
```

```
In [12]: model_w2v.build_vocab(tokenized_tweet, progress_per = 1000)
         model_w2v.train(tokenized_tweet, total_examples =model_w2v.corpus_count, epochs = model_w2v.epochs)
         model_w2v.save("./w2v_model")
```

```
In [13]: model_w2v.wv.most_similar('bad')

Out[13]: [('ruined', 0.621534526348114),
          ('terrible', 0.6081273555755615),
          ('either.', 0.6075522303581238),
          ('poor', 0.5982681512832642),
          ('compelled', 0.5860457420349121),
          ('holidays,', 0.5842126607894897),
          ('saving', 0.5840566158294678),
          ('ATM', 0.5801668763160706),
          ('bad.', 0.5750031471252441),
          ('deserve', 0.5711111426353455)]
```

```
In [14]: def word_vector(tokens, size):
             vec = np.zeros(size).reshape((1, size))
             count = 0
             for word in tokens:
                 try:
                     vec += model_w2v.wv[word].reshape((1, size))
                     count += 1.
                 except KeyError:  # handling the case where the token is not in vocabulary
                     continue
             if count != 0:
                 vec /= count
             return vec
```

```
In [15]: #轉成array的形式等等丟model
         wordvec_arrays = np.zeros((len(tokenized_tweet), 200))
         for i in range(len(tokenized_tweet)):
             wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 200)
         wordvec_df = pd.DataFrame(wordvec_arrays)
         # wordvec_df.shape
         print(len(wordvec_arrays))

         10000
```

- Tokenizer

- 使用keras tokenizer

```
In [18]: maxlen = 200
         max_words = 10000
         tokenizer = Tokenizer(num_words=1000)    #只考慮1000個最常用的詞
         tokenizer.fit_on_texts(df['text'])
         sequences = tokenizer.texts_to_sequences(df['text'])
         tokenizer.word_index

Out[18]: {'the': 1,
          'and': 2,
          'i': 3,
          'a': 4,
          'to': 5,
          'of': 6,
          'was': 7,
          'is': 8,
          'it': 9,
          'for': 10,
          'in': 11,
          'that': 12,
          'my': 13,
          'with': 14,
          'but': 15,
          'you': 16,
          'this': 17,
          'on': 18,
          'they': 19,
```

建立 1000 字的字典
讀取所有評論，依照每個字在資料中出現的次數進行排序，前 1000 名的英文單字會加入字典中

透過 texts_to_sequences 將文字轉成向量

padding 因為進行深度學習模型訓練時長度必須固定

```
In [19]: # use pad_sequence to make traning samples the same size, fill with zeros
         #長度小於200的 前面數字補0
         #長度大於200的 截去前面的數字
         data_input = pad_sequences(sequences, maxlen = maxlen)

In [20]: data_input

Out[20]: array([[  0,   0,   0, ...,   5,  50,  65],
                [275,   2,   3, ..., 179,  21,  56],
                [  0,   0,   0, ...,  69,  45, 245],
                ...,
                [368,  78,  56, ...,   6,   1, 450],
                [  0,   0,   0, ...,   4,   4, 104],
                [  0,   0,   0, ...,  18,   1, 522]])

In [21]: data_input.shape

Out[21]: (10000, 200)
```

每一篇評論的長度通常不太一樣，但後續深度學習模型訓練時長度必須固定，所以進行 padding

長度小於 200，前面的數字補 0
長度大於 200，前面的數字截掉

e. 切割訓練集與測試集

以 0.8:0.2 的方式切割訓練與測試集

- 使用Tokenizer轉向量的方法

```
y = df['stars']
x = data_input
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=42,test_size=0.2)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8000, 200)
(2000, 200)
(8000,)
(2000,)
```

三.建立模型

(雖然在將文字轉向量有使用 word2vec 以及 tokenizer 兩種方法，但由於在嘗試多次以後發現 Tokenizer 的方法成效好像稍微好一點，，因此這裡以 Keras tokenizer 處理的文字向量進行訓練比較)

a. LSTM (With Dropout Layer)

使用LSTM (有Dropout Layer)

```
max_words = 1000
maxlen = 200
model_lstm = Sequential()
model_lstm.add(Embedding(output_dim = 64,input_dim = max_words,input_length = maxlen))
model_lstm.add(Dropout(0.7))
model_lstm.add(LSTM(10,dropout=0.7))
model_lstm.add(Dense(10,activation = 'relu'))
model_lstm.add(Dropout(0.7))
model_lstm.add(Dense(1, activation='sigmoid'))
model_lstm.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 64)           64000
_________________________________________________________________
dropout (Dropout)            (None, 200, 64)           0
_________________________________________________________________
lstm (LSTM)                  (None, 10)                3000
_________________________________________________________________
dense (Dense)                (None, 10)                110
_________________________________________________________________
dropout_1 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 67,121
Trainable params: 67,121
Non-trainable params: 0
```

64000 = 64*1000

3000 =(10*10+10*64+10)*4

110 = 10*10+10

11 = 10+1

```
opt = keras.optimizers.RMSprop(learning_rate=0.01)
model_lstm.compile(optimizer='adam',loss='binary_crossentropy',metrics=['acc'])
```

```
history_lstm = model_lstm.fit(x_train, y_train,
                    epochs=18,
                    batch_size=110,
                    validation_split=0.2)
```

```
Epoch 1/18
59/59 [==============================] - 9s 153ms/step - loss: 0.6752 - acc: 0.6264 - val_loss: 0.6225 - val_acc: 0.6931
Epoch 2/18
59/59 [==============================] - 7s 127ms/step - loss: 0.6514 - acc: 0.6802 - val_loss: 0.6190 - val_acc: 0.6931
Epoch 3/18
59/59 [==============================] - 7s 127ms/step - loss: 0.6338 - acc: 0.6825 - val_loss: 0.5694 - val_acc: 0.6975
Epoch 4/18
59/59 [==============================] - 7s 125ms/step - loss: 0.6031 - acc: 0.7197 - val_loss: 0.5075 - val_acc: 0.7706
Epoch 5/18
59/59 [==============================] - 7s 127ms/step - loss: 0.5695 - acc: 0.7588 - val_loss: 0.5072 - val_acc: 0.7694
Epoch 6/18
59/59 [==============================] - 7s 124ms/step - loss: 0.5391 - acc: 0.7795 - val_loss: 0.4654 - val_acc: 0.7981
Epoch 7/18
59/59 [==============================] - 8s 128ms/step - loss: 0.5224 - acc: 0.7959 - val_loss: 0.4554 - val_acc: 0.8062
Epoch 8/18
59/59 [==============================] - 7s 126ms/step - loss: 0.5218 - acc: 0.7978 - val_loss: 0.4627 - val_acc: 0.8081
Epoch 9/18
59/59 [==============================] - 7s 119ms/step - loss: 0.5076 - acc: 0.8016 - val_loss: 0.4334 - val_acc: 0.8094
Epoch 10/18
59/59 [==============================] - 8s 132ms/step - loss: 0.5010 - acc: 0.8067 - val_loss: 0.4329 - val_acc: 0.8081
Epoch 11/18
59/59 [==============================] - 7s 117ms/step - loss: 0.4959 - acc: 0.8078 - val_loss: 0.4526 - val_acc: 0.8100
Epoch 12/18
59/59 [==============================] - 7s 121ms/step - loss: 0.4787 - acc: 0.8163 - val_loss: 0.4277 - val_acc: 0.8175
Epoch 13/18
59/59 [==============================] - 7s 115ms/step - loss: 0.4881 - acc: 0.8123 - val_loss: 0.4198 - val_acc: 0.8144
```

b. LSTM (Without Dropout Layer)

使用LSTM (沒有Dropout Layer)

```
max_words = 1000
maxlen = 200
model_lstm_w = Sequential()
model_lstm_w.add(Embedding(max_words, 32,input_length = maxlen))
model_lstm_w.add(LSTM(10))
model_lstm_w.add(Dense(10, activation='relu'))
model_lstm_w.add(Dense(1, activation='sigmoid'))
model_lstm_w.summary()
```

```
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_24 (Embedding)     (None, 200, 32)           320000
_________________________________________________________________
lstm_20 (LSTM)               (None, 10)                1720
_________________________________________________________________
dense_48 (Dense)             (None, 5)                 55
_________________________________________________________________
dense_49 (Dense)             (None, 1)                 6
=================================================================
Total params: 321,781
Trainable params: 321,781
Non-trainable params: 0
_________________________________________________________________
```

32000 = 32*1000

1720 =(10*10+10*32+10)*4

110 = 5*5+5

11 = 10+1

```
from keras.optimizers import SGD
#opt = SGD(lr=0.01, momentum=0.9, clipvalue=1)
#opt = keras.optimizers.RMSprop(learning_rate=0.01)
model_lstm_w.compile(optimizer= 'adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

```
history_lstm_w = model_lstm_w.fit(x_train, y_train,epochs=9,batch_size=100,validation_split=0.2)
```

```
Epoch 1/9
64/64 [==============================] - 4s 67ms/step - loss: 0.6300 - acc: 0.6744 - val_loss: 0.5828 - val_acc: 0.6931
Epoch 2/9
64/64 [==============================] - 3s 54ms/step - loss: 0.5270 - acc: 0.7156 - val_loss: 0.5025 - val_acc: 0.7606
Epoch 3/9
64/64 [==============================] - 3s 52ms/step - loss: 0.4371 - acc: 0.8070 - val_loss: 0.4570 - val_acc: 0.7850
Epoch 4/9
64/64 [==============================] - 4s 64ms/step - loss: 0.3819 - acc: 0.8367 - val_loss: 0.4377 - val_acc: 0.7819
Epoch 5/9
64/64 [==============================] - 4s 55ms/step - loss: 0.3544 - acc: 0.8514 - val_loss: 0.4382 - val_acc: 0.7825
Epoch 6/9
64/64 [==============================] - 3s 54ms/step - loss: 0.3385 - acc: 0.8586 - val_loss: 0.4403 - val_acc: 0.7969
Epoch 7/9
64/64 [==============================] - 4s 57ms/step - loss: 0.3214 - acc: 0.8658 - val_loss: 0.4476 - val_acc: 0.7969
Epoch 8/9
```

c. CNN (With Dropout Layer)

使用CNN (有Dropout Layer)

```
# define model
model_cnn = Sequential()
model_cnn.add(Embedding(max_words, 64, input_length=maxlen))
#model_cnn.add(Dropout(0.7))
model_cnn.add(Conv1D(filters=64, kernel_size=4, activation='relu'))
model_cnn.add(Dropout(0.7))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dropout(0.7))
model_cnn.add(Dense(32, activation='relu'))
model_cnn.add(Dropout(0.7))
model_cnn.add(Dense(1, activation='sigmoid'))
model_cnn.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 200, 64)           64000
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 197, 64)           16448
_________________________________________________________________
dropout_15 (Dropout)         (None, 197, 64)           0
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 98, 64)            0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_16 (Dropout)         (None, 6272)              0
_________________________________________________________________
dense_18 (Dense)             (None, 32)                200736
_________________________________________________________________
dropout_17 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 33
=================================================================
Total params: 281,217
Trainable params: 281,217
Non-trainable params: 0
```

64000 = 64*1000

16448 = (64*4+1)*64

200736 = 6272*32+32

32+1

```python
#opt = keras.optimizers.RMSprop(learning_rate=0.001)
#from keras.optimizers import SGD
#opt = SGD(lr=0.001, momentum=0.9, clipvalue=1)
model_cnn.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
```

```python
history_cnn = model_cnn.fit(x_train, y_train,
                    epochs=14,
                    batch_size=100,
                    validation_split=0.2)
```

```
Epoch 1/14
64/64 [==============================] - 4s 69ms/step - loss: 0.6397 - acc: 0.6767 - val_loss: 0.6318 - val_acc: 0.6931
Epoch 2/14
64/64 [==============================] - 4s 65ms/step - loss: 0.6278 - acc: 0.6800 - val_loss: 0.6050 - val_acc: 0.6931
Epoch 3/14
64/64 [==============================] - 4s 65ms/step - loss: 0.6040 - acc: 0.6803 - val_loss: 0.5622 - val_acc: 0.6931
Epoch 4/14
64/64 [==============================] - 4s 65ms/step - loss: 0.5330 - acc: 0.6998 - val_loss: 0.4820 - val_acc: 0.7513
Epoch 5/14
64/64 [==============================] - 5s 76ms/step - loss: 0.4830 - acc: 0.7663 - val_loss: 0.4470 - val_acc: 0.7975
Epoch 6/14
64/64 [==============================] - 5s 77ms/step - loss: 0.4431 - acc: 0.8000 - val_loss: 0.4327 - val_acc: 0.8081
Epoch 7/14
64/64 [==============================] - 5s 76ms/step - loss: 0.4264 - acc: 0.8131 - val_loss: 0.4267 - val_acc: 0.8075
Epoch 8/14
64/64 [==============================] - 5s 76ms/step - loss: 0.4116 - acc: 0.8228 - val_loss: 0.4571 - val_acc: 0.7944
Epoch 9/14
64/64 [==============================] - 5s 76ms/step - loss: 0.3990 - acc: 0.8280 - val_loss: 0.4288 - val_acc: 0.8050
Epoch 10/14
64/64 [==============================] - 5s 79ms/step - loss: 0.3883 - acc: 0.8358 - val_loss: 0.4200 - val_acc: 0.8131
Epoch 11/14
```

d. CNN (Without Dropout Layer)

使用CNN (沒有Dropout Layer)

```python
# define model
model_cnn_w = Sequential()
model_cnn_w.add(Embedding(max_words, 64, input_length=maxlen))
model_cnn_w.add(Conv1D(filters=32, kernel_size=8, activation='relu'))
model_cnn_w.add(MaxPooling1D(pool_size=2))
model_cnn_w.add(Flatten())
model_cnn_w.add(Dense(5, activation='relu'))
model_cnn_w.add(Dense(1, activation='sigmoid'))
model_cnn_w.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 200, 64)           64000
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 193, 32)           16416
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 96, 32)            0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_14 (Dense)             (None, 10)                30730
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 11
=================================================================
Total params: 111,157
Trainable params: 111,157
Non-trainable params: 0
```

64000 = 64*1000

16416 = (32*8+1)*64

30730 = 3072*10+10

11 = 10+1

```
model_cnn_w.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
```

```
history_cnn_w = model_cnn_w.fit(x_train, y_train,
                    epochs=7,
                    batch_size=100,
                    validation_split=0.2)
```

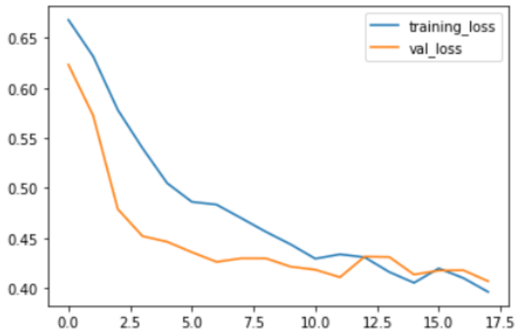
```
Epoch 1/7
64/64 [==============================] - 3s 53ms/step - loss: 0.6153 - acc: 0.6730 - val_loss: 0.5720 - val_acc: 0.6931
Epoch 2/7
64/64 [==============================] - 3s 50ms/step - loss: 0.5050 - acc: 0.7445 - val_loss: 0.4635 - val_acc: 0.7919
Epoch 3/7
64/64 [==============================] - 3s 51ms/step - loss: 0.4325 - acc: 0.8202 - val_loss: 0.4445 - val_acc: 0.8019
Epoch 4/7
64/64 [==============================] - 3s 50ms/step - loss: 0.4040 - acc: 0.8452 - val_loss: 0.4370 - val_acc: 0.8075
Epoch 5/7
64/64 [==============================] - 3s 50ms/step - loss: 0.3831 - acc: 0.8581 - val_loss: 0.4479 - val_acc: 0.8150
Epoch 6/7
64/64 [==============================] - 3s 53ms/step - loss: 0.3663 - acc: 0.8658 - val_loss: 0.4382 - val_acc: 0.8200
Epoch 7/7
64/64 [==============================] - 3s 52ms/step - loss: 0.3477 - acc: 0.8745 - val_loss: 0.4376 - val_acc: 0.8125
```

四、評估模型

LSTM (With Dropout Layer)

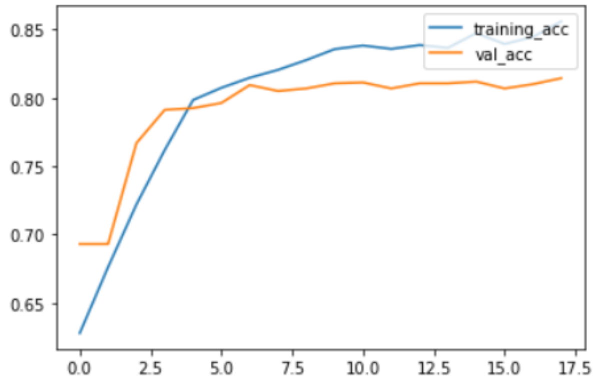
- loss and validation loss

```
#plot model loss
plt.plot(history_lstm.history['loss'], label = 'training_loss')
plt.plot(history_lstm.history['val_loss'], label = 'val_loss')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```



- accuracy and validation accuracy

```
]: #plot model loss
plt.plot(history_lstm.history['acc'], label = 'training_acc')
plt.plot(history_lstm.history['val_acc'], label = 'val_acc')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```



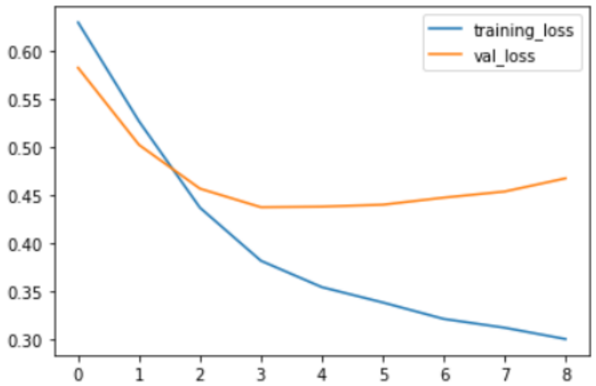

```
model_lstm.evaluate(x_test, y_test)
```

```
63/63 [==============================] - 1s 16ms/step - loss: 0.3975 - acc: 0.8185
[0.3974834084510803, 0.8184999823570251]
```

LSTM (Without Dropout Layer)

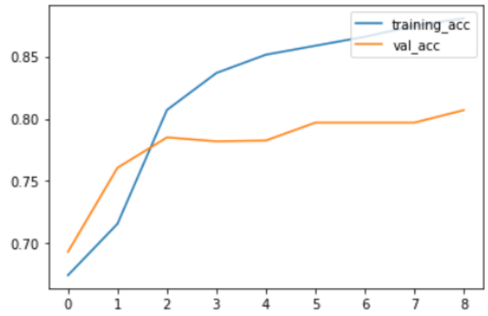
- loss and validation loss

```
#plot model loss
plt.plot(history_lstm_w.history['loss'], label = 'training_loss')
plt.plot(history_lstm_w.history['val_loss'], label = 'val_loss')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```



- accuracy and validation accuracy

```
#plot model loss
plt.plot(history_lstm_w.history['acc'], label = 'training_acc')
plt.plot(history_lstm_w.history['val_acc'], label = 'val_acc')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```




```
model_lstm_w.evaluate(x_test, y_test)
```

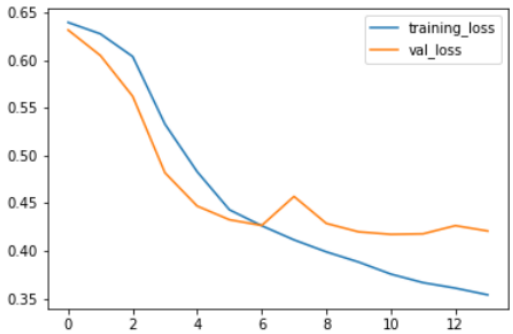
```
63/63 [==============================] - 1s 14ms/step - loss: 0.4647 - acc: 0.8020

[0.46466922760009766, 0.8019999861717224]
```

CNN (With Dropout Layer)

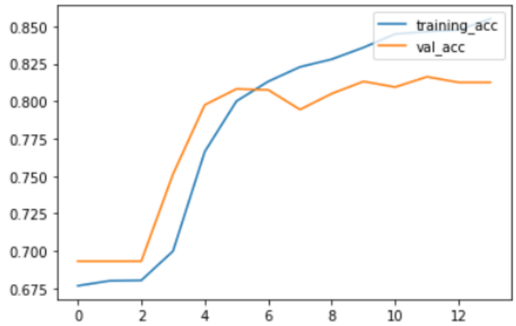
- loss and validation loss

```
#plot model loss
plt.plot(history_cnn.history['loss'], label = 'training_loss')
plt.plot(history_cnn.history['val_loss'], label = 'val_loss')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```



- accuracy and validation accuracy

```
#plot model loss
plt.plot(history_cnn.history['acc'], label = 'training_acc')
plt.plot(history_cnn.history['val_acc'], label = 'val_acc')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```




```
model_cnn.evaluate(x_test, y_test)
```

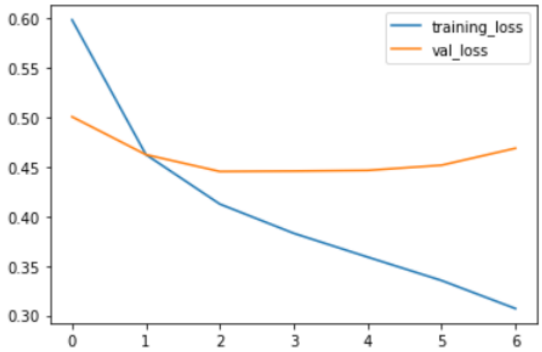
```
63/63 [==============================] - 0s 6ms/step - loss: 0.4290 - acc: 0.8115

[0.4289551079273224, 0.8115000128746033]
```

CNN (Without Dropout Layer)

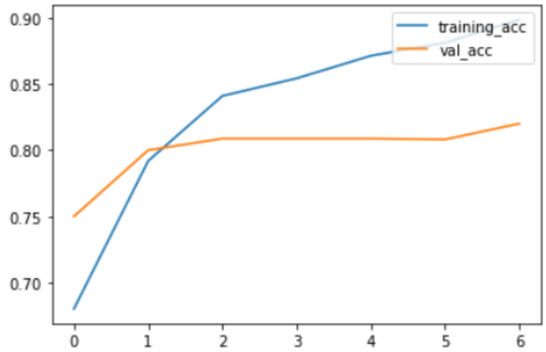
- loss and validation loss

```
#plot model loss
plt.plot(history_cnn_w.history['loss'], label = 'training_loss')
plt.plot(history_cnn_w.history['val_loss'], label = 'val_loss')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```



- accuracy and validation accuracy

```
#plot model loss
plt.plot(history_cnn_w.history['acc'], label = 'training_acc')
plt.plot(history_cnn_w.history['val_acc'], label = 'val_acc')
#圖例
plt.legend(loc = 'upper right')
plt.show()
plt.close()
```




```
model_cnn_w.evaluate(x_test, y_test)
```

```
63/63 [==============================] - 0s 5ms/step - loss: 0.4707 - acc: 0.8050
[0.4706825017929077, 0.8050000071525574]
```

四.統整比較

LSTM (有 Dropout)

```
model_lstm.evaluate(x_test, y_test)
```

```
63/63 [==============================] - 1s 16ms/step - loss: 0.3975 - acc: 0.8185
[0.3974834084510803, 0.8184999823570251]
```

LSTM (沒有 Dropout)

```
model_lstm_w.evaluate(x_test, y_test)
```

```
63/63 [==============================] - 1s 14ms/step - loss: 0.4647 - acc: 0.8020
[0.46466922760009766, 0.8019999861717224]
```

CNN (有 Dropout)

```
model_cnn.evaluate(x_test, y_test)
```

```
63/63 [==============================] - 0s 6ms/step - loss: 0.4290 - acc: 0.8115
[0.4289551079273224, 0.8115000128746033]
```

CNN (沒有 Dropout)

```
model_cnn_w.evaluate(x_test, y_test)
```

```
63/63 [==============================] - 0s 5ms/step - loss: 0.4707 - acc: 0.8050
[0.4706825017929077, 0.8050000071525574]
```

⇨ 有 Dropout 的成效比沒 Dropout 的成效好

因為 Dropout 可以減少 Ovevfitting 的情況，可以使模型泛化性增強，不會太過於依賴某些局部的特徵。

⇨ 如果沒有 Dropout，容易有 overfitting 的問題 (通常 epochs 不能跑太多、模型神經元數不能太多、模型不能建太複雜，不然會有 overfitting 的情形)

⇨ CNN 與 LSTM 做出來的成效差不多，但 LSTM 稍微好一點