[資料前處理]

```
train = pd.DataFrame(columns = ['mood','sentence'])
trainfile = open('training_label.txt','r')
line = trainfile.readline()
i = 0
while line:
    if line != '\n':
        i+=1
        temp = line.split("+++$+++", 1)
        new = pd.DataFrame({'mood':temp[0],'sentence':temp[1]},index=[1])
        train = train.append(new,ignore_index=True)
    if i == 100000 : break
    line = trainfile.readline()
trainfile.close()
```

- 切割字串，存成dataframe

以#####或+++$+++切割字串，前面儲存到mood column，後面儲存到sentence column。
test取全部90筆資料，train則取10000筆資料。

- 停用字

有嘗試直接使用nltk的停用字，但透過觀察停用字發現他會刪掉像no ,not ,don’t等可能對於負面情緒有意義的字眼。因此我透過countvectorizer觀察vocabulary的詞頻來刪減重複次數多且較無意義的字，以下是我刪除的停用字：

```
stop_words = [',','.','..','...','"','`',':','1','2','3','4','5','6','7','8','9','0',
              '00','000','0000','000pv','day','so','all','up','got','today','from','one',
              'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s',
              't','u','v','w','x','y','z','im','you','your','u','ur','they','an','him',
              'we','he','she','me','my','his','her','it','is','am','are','was','were',
              'btw','thing','be','have','has','had','do','does','did','ll','re','ve',
              'this','that','there','say','says','said','guys','to','for','in','at',
              'when','the','on','its','and','in','of','some','someone','before','after',
              'with','been','being','which','them','their','our','us','left','10','these',
              '30','site','online','12','da','room','20','sometimes','11','sat','15','google'
              '24','info','sit','web','website','09','17','18','33','50','session','16','21',
              '25','b4','pm','13','333','45','70','2009','month','yr','yrs','23',
              'txt','06','22','26','29','31','37','80','07','19','28',
              '32','34','35','36','38','48','52','69','79','89','95','97']
```

- 將文字轉成數字list

```
token = Tokenizer(num_words=4000)
token.fit_on_texts(train_x)
token.word_index
x_train_seq = token.texts_to_sequences(train_x)
x_test_seq = token.texts_to_sequences(test_x)
train_fit = sequence.pad_sequences(x_train_seq, maxlen=400)
test_fit = sequence.pad_sequences(x_test_seq, maxlen=400)
```

使用Tokenizer模組建立token，建立一個4000字的字典，會把訓練資料中出現次數最多的前4000字放進字典中。由於每則推特內容的長度不一，因此使用sequence.pad_sequences會把每則都變成固定長度(大於400的截去前面的數字，小於400的前面數字補0)。

- 建RNN模型

```
#RNN
modelRNN = Sequential()
modelRNN.add(Embedding(output_dim=32,input_dim=4000,
                       input_length=400))
modelRNN.add(Dropout(0.2))
modelRNN.add(SimpleRNN(units=16)) #16個神經元的RNN層
modelRNN.add(Dense(units=128,activation='relu')) #128個神經元的隱藏層
modelRNN.add(Dropout(0.3))
modelRNN.add(layers.Dense(1, activation='sigmoid')) #1個神經元的輸出層
modelRNN.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
RNN_history = modelRNN.fit(train_fit,train_y, epochs=20,
                           batch_size=1000, verbose=2, validation_split=0.2)
```

神經網路架構：
1. embedding層會把數字list轉成向量list，這裡會轉成32維的向量
2. dropout(0.2)：放棄20%的神經元 [有測試不dropout]
3. 16個神經元的RNN層
4. 128個神經元的隱藏層(使用ReLU作為activation function)
5. dropout(0.3)：放棄30%的神經元 [有測試不dropout]
6. 輸出層(使用sigmoid作為activation function)

訓練模型：
loss function用的是cross entropy，並使用adam作為優化器加速收斂

評估模型：
使用80%作為訓練集，20%作為驗證集，訓練20個epoch，每個批次訓練1000筆資料
最後會使用test data進行測試並計算準確率

[左邊是有dropout兩次的模型表現，右邊是都沒進行dropout過的模型表現]

```
Epoch 1/20
 - 26s - loss: 0.6921 - accuracy: 0.5155 - val_loss: 0.6901 - val_accuracy: 0.5295
Epoch 2/20
 - 26s - loss: 0.6839 - accuracy: 0.5669 - val_loss: 0.6840 - val_accuracy: 0.5500
Epoch 3/20
 - 26s - loss: 0.6677 - accuracy: 0.6072 - val_loss: 0.6676 - val_accuracy: 0.5970
Epoch 4/20
 - 25s - loss: 0.6276 - accuracy: 0.6961 - val_loss: 0.6292 - val_accuracy: 0.6710
Epoch 5/20
 - 26s - loss: 0.5583 - accuracy: 0.7635 - val_loss: 0.5887 - val_accuracy: 0.7045
Epoch 6/20
 - 26s - loss: 0.4844 - accuracy: 0.7887 - val_loss: 0.5691 - val_accuracy: 0.7225
Epoch 7/20
 - 26s - loss: 0.4180 - accuracy: 0.8226 - val_loss: 0.5807 - val_accuracy: 0.7170
Epoch 8/20
 - 26s - loss: 0.3712 - accuracy: 0.8462 - val_loss: 0.5933 - val_accuracy: 0.7240
Epoch 9/20
 - 25s - loss: 0.3316 - accuracy: 0.8673 - val_loss: 0.6253 - val_accuracy: 0.7205
Epoch 10/20
 - 25s - loss: 0.2927 - accuracy: 0.8876 - val_loss: 0.6507 - val_accuracy: 0.7170
Epoch 11/20
 - 26s - loss: 0.2655 - accuracy: 0.9007 - val_loss: 0.6936 - val_accuracy: 0.7160
Epoch 12/20
 - 25s - loss: 0.2398 - accuracy: 0.9105 - val_loss: 0.7326 - val_accuracy: 0.7120
Epoch 13/20
 - 26s - loss: 0.2145 - accuracy: 0.9229 - val_loss: 0.7760 - val_accuracy: 0.7135
Epoch 14/20
 - 25s - loss: 0.1948 - accuracy: 0.9295 - val_loss: 0.8219 - val_accuracy: 0.7015
Epoch 15/20
 - 25s - loss: 0.1780 - accuracy: 0.9374 - val_loss: 0.8686 - val_accuracy: 0.6960
Epoch 16/20
 - 25s - loss: 0.1619 - accuracy: 0.9436 - val_loss: 0.9100 - val_accuracy: 0.6980
Epoch 17/20
 - 25s - loss: 0.1449 - accuracy: 0.9499 - val_loss: 0.9561 - val_accuracy: 0.7020
Epoch 18/20
 - 26s - loss: 0.1347 - accuracy: 0.9513 - val_loss: 0.9927 - val_accuracy: 0.6955
Epoch 19/20
 - 26s - loss: 0.1212 - accuracy: 0.9579 - val_loss: 1.0356 - val_accuracy: 0.6960
Epoch 20/20
 - 25s - loss: 0.1148 - accuracy: 0.9597 - val_loss: 1.0836 - val_accuracy: 0.6915
```

```
In [43]: scores = modelRNN.evaluate(test_fit,test_y,verbose=1)
    ...: scores[1]
90/90 [==============================] - 0s 3ms/step
Out[43]: 0.7555555701255798
```

```
Epoch 1/20
 - 28s - loss: 0.6930 - accuracy: 0.5160 - val_loss: 0.6919 - val_accuracy: 0.5450
Epoch 2/20
 - 28s - loss: 0.6851 - accuracy: 0.6280 - val_loss: 0.6856 - val_accuracy: 0.6090
Epoch 3/20
 - 28s - loss: 0.6677 - accuracy: 0.6999 - val_loss: 0.6705 - val_accuracy: 0.6505
Epoch 4/20
 - 28s - loss: 0.6320 - accuracy: 0.7695 - val_loss: 0.6417 - val_accuracy: 0.6900
Epoch 5/20
 - 28s - loss: 0.5859 - accuracy: 0.7550 - val_loss: 0.6099 - val_accuracy: 0.7100
Epoch 6/20
 - 28s - loss: 0.5194 - accuracy: 0.8267 - val_loss: 0.5752 - val_accuracy: 0.7225
Epoch 7/20
 - 28s - loss: 0.4522 - accuracy: 0.8403 - val_loss: 0.5513 - val_accuracy: 0.7225
Epoch 8/20
 - 28s - loss: 0.3868 - accuracy: 0.8602 - val_loss: 0.5481 - val_accuracy: 0.7270
Epoch 9/20
 - 28s - loss: 0.3311 - accuracy: 0.8742 - val_loss: 0.5617 - val_accuracy: 0.7255
Epoch 10/20
 - 28s - loss: 0.2856 - accuracy: 0.8954 - val_loss: 0.5838 - val_accuracy: 0.7215
Epoch 11/20
 - 28s - loss: 0.2426 - accuracy: 0.9149 - val_loss: 0.6270 - val_accuracy: 0.7165
Epoch 12/20
 - 28s - loss: 0.2071 - accuracy: 0.9300 - val_loss: 0.6548 - val_accuracy: 0.7155
Epoch 13/20
 - 28s - loss: 0.1770 - accuracy: 0.9446 - val_loss: 0.7044 - val_accuracy: 0.7105
Epoch 14/20
 - 28s - loss: 0.1553 - accuracy: 0.9534 - val_loss: 0.7125 - val_accuracy: 0.7040
Epoch 15/20
 - 28s - loss: 0.1344 - accuracy: 0.9619 - val_loss: 0.7611 - val_accuracy: 0.7010
Epoch 16/20
 - 28s - loss: 0.1138 - accuracy: 0.9710 - val_loss: 0.8256 - val_accuracy: 0.7030
Epoch 17/20
 - 28s - loss: 0.0961 - accuracy: 0.9740 - val_loss: 0.8860 - val_accuracy: 0.6965
Epoch 18/20
 - 28s - loss: 0.0832 - accuracy: 0.9774 - val_loss: 0.9387 - val_accuracy: 0.6950
Epoch 19/20
 - 28s - loss: 0.0715 - accuracy: 0.9810 - val_loss: 0.9942 - val_accuracy: 0.6940
Epoch 20/20
 - 28s - loss: 0.0620 - accuracy: 0.9830 - val_loss: 1.0529 - val_accuracy: 0.6940
```

```
In [46]: scores = modelRNN.evaluate(test_fit,test_y,verbose=1)
    ...: scores[1]
90/90 [==============================] - 0s 3ms/step
Out[46]: 0.7555555701255798
```

可以看到沒dropout的模型在訓練集和驗證集的預測表現上普遍都優於有dropout的，但在最後預測測試集時卻是一樣的準確率，因為只使用了10000筆資料來訓練，所以推測如果在更大的input上還是沒有使用dropout的話，可能測試的準確率會降的比有使用的還低。

- 建LSTM模型

```
#LSTM
modelLSTM = Sequential()
modelLSTM.add(Embedding(output_dim=32,input_dim=4000,
                        input_length=400))
modelLSTM.add(Dropout(0.2))
modelLSTM.add(LSTM(16)) #16個神經元的LSTM層
modelLSTM.add(Dense(units=128,activation='relu')) #128個神經元的隱藏層
modelLSTM.add(Dropout(0.3))
modelLSTM.add(layers.Dense(1, activation='sigmoid')) #1個神經元的輸出層
modelLSTM.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
LSTM_history = modelLSTM.fit(train_fit,train_y, epochs=20,
                             batch_size=1000, verbose=2, validation_split=0.2)
```

神經網路架構：

1. embedding層會把數字list轉成向量list，這裡會轉成32維的向量
2. dropout(0.2)：放棄20%的神經元 [有測試不dropout]
3. 16個神經元的RNN層
4. 128個神經元的隱藏層(使用ReLU作為activation function)
5. dropout(0.3)：放棄30%的神經元 [有測試不dropout]
6. 輸出層(使用sigmoid作為activation function)

訓練模型：

loss function用的是cross entropy，並使用adam作為優化器加速收斂

評估模型：

使用80%作為訓練集，20%作為驗證集，訓練20個epoch，每個批次訓練1000筆資料

最後會使用test data進行測試並計算準確率

[左邊是有dropout兩次的模型表現，右邊是都沒進行dropout過的模型表現]

```
Epoch 1/20
 - 105s - loss: 0.6925 - accuracy: 0.5337 - val_loss: 0.6918 - val_accuracy: 0.5370
Epoch 2/20
 - 104s - loss: 0.6896 - accuracy: 0.5683 - val_loss: 0.6884 - val_accuracy: 0.5815
Epoch 3/20
 - 104s - loss: 0.6826 - accuracy: 0.6255 - val_loss: 0.6809 - val_accuracy: 0.6220
Epoch 4/20
 - 104s - loss: 0.6680 - accuracy: 0.6802 - val_loss: 0.6657 - val_accuracy: 0.6510
Epoch 5/20
 - 104s - loss: 0.6390 - accuracy: 0.7221 - val_loss: 0.6391 - val_accuracy: 0.6845
Epoch 6/20
 - 104s - loss: 0.5927 - accuracy: 0.7577 - val_loss: 0.6012 - val_accuracy: 0.7210
Epoch 7/20
 - 104s - loss: 0.5296 - accuracy: 0.7950 - val_loss: 0.5652 - val_accuracy: 0.7280
Epoch 8/20
 - 104s - loss: 0.4698 - accuracy: 0.8104 - val_loss: 0.5438 - val_accuracy: 0.7355
Epoch 9/20
 - 104s - loss: 0.4213 - accuracy: 0.8306 - val_loss: 0.5401 - val_accuracy: 0.7310
Epoch 10/20
 - 104s - loss: 0.3861 - accuracy: 0.8468 - val_loss: 0.5485 - val_accuracy: 0.7325
Epoch 11/20
 - 104s - loss: 0.3580 - accuracy: 0.8581 - val_loss: 0.5682 - val_accuracy: 0.7345
Epoch 12/20
 - 104s - loss: 0.3327 - accuracy: 0.8681 - val_loss: 0.5914 - val_accuracy: 0.7315
Epoch 13/20
 - 103s - loss: 0.3146 - accuracy: 0.8795 - val_loss: 0.6152 - val_accuracy: 0.7315
Epoch 14/20
 - 103s - loss: 0.2973 - accuracy: 0.8866 - val_loss: 0.6395 - val_accuracy: 0.7270
Epoch 15/20
 - 104s - loss: 0.2848 - accuracy: 0.8951 - val_loss: 0.6677 - val_accuracy: 0.7200
Epoch 16/20
 - 103s - loss: 0.2742 - accuracy: 0.8967 - val_loss: 0.6993 - val_accuracy: 0.7150
Epoch 17/20
 - 104s - loss: 0.2639 - accuracy: 0.9018 - val_loss: 0.7296 - val_accuracy: 0.7150
Epoch 18/20
 - 103s - loss: 0.2515 - accuracy: 0.9060 - val_loss: 0.7548 - val_accuracy: 0.7090
Epoch 19/20
 - 104s - loss: 0.2423 - accuracy: 0.9112 - val_loss: 0.7924 - val_accuracy: 0.7090
Epoch 20/20
 - 104s - loss: 0.2365 - accuracy: 0.9140 - val_loss: 0.8274 - val_accuracy: 0.7080
```

```
In [41]: scores = modelLSTM.evaluate(test_fit,test_y,verbose=1)
    ...: scores[1]
90/90 [==============================] - 1s 16ms/step
Out[41]: 0.7444444298744202
```

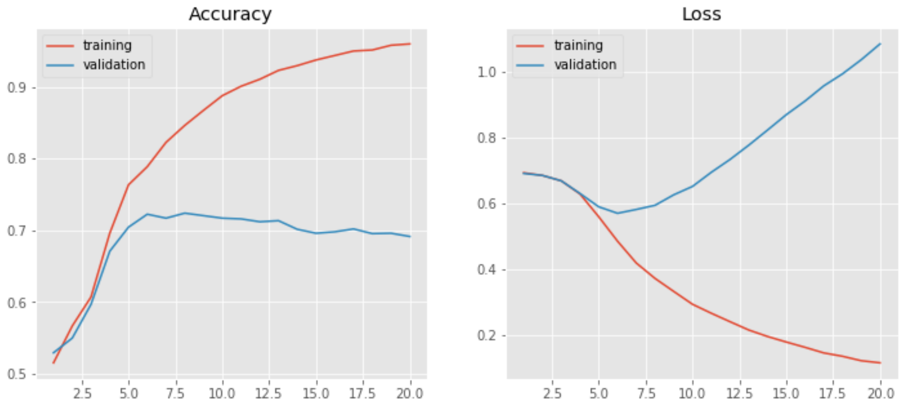
```
Epoch 1/20
 - 111s - loss: 0.6928 - accuracy: 0.5409 - val_loss: 0.6918 - val_accuracy: 0.5525
Epoch 2/20
 - 109s - loss: 0.6891 - accuracy: 0.5972 - val_loss: 0.6875 - val_accuracy: 0.6070
Epoch 3/20
 - 110s - loss: 0.6806 - accuracy: 0.6699 - val_loss: 0.6786 - val_accuracy: 0.6395
Epoch 4/20
 - 110s - loss: 0.6635 - accuracy: 0.7026 - val_loss: 0.6612 - val_accuracy: 0.6605
Epoch 5/20
 - 110s - loss: 0.6306 - accuracy: 0.7431 - val_loss: 0.6313 - val_accuracy: 0.7010
Epoch 6/20
 - 111s - loss: 0.5760 - accuracy: 0.7724 - val_loss: 0.5884 - val_accuracy: 0.7140
Epoch 7/20
 - 111s - loss: 0.5073 - accuracy: 0.7900 - val_loss: 0.5576 - val_accuracy: 0.7280
Epoch 8/20
 - 110s - loss: 0.4456 - accuracy: 0.8175 - val_loss: 0.5452 - val_accuracy: 0.7350
Epoch 9/20
 - 110s - loss: 0.3970 - accuracy: 0.8394 - val_loss: 0.5466 - val_accuracy: 0.7335
Epoch 10/20
 - 111s - loss: 0.3636 - accuracy: 0.8519 - val_loss: 0.5830 - val_accuracy: 0.7230
Epoch 11/20
 - 111s - loss: 0.3379 - accuracy: 0.8630 - val_loss: 0.5908 - val_accuracy: 0.7245
Epoch 12/20
 - 111s - loss: 0.3227 - accuracy: 0.8714 - val_loss: 0.6197 - val_accuracy: 0.7200
Epoch 13/20
 - 111s - loss: 0.3016 - accuracy: 0.8831 - val_loss: 0.6337 - val_accuracy: 0.7150
Epoch 14/20
 - 110s - loss: 0.2852 - accuracy: 0.8915 - val_loss: 0.6500 - val_accuracy: 0.7145
Epoch 15/20
 - 110s - loss: 0.2702 - accuracy: 0.8979 - val_loss: 0.6792 - val_accuracy: 0.7105
Epoch 16/20
 - 110s - loss: 0.2582 - accuracy: 0.9003 - val_loss: 0.7090 - val_accuracy: 0.7175
Epoch 17/20
 - 110s - loss: 0.2458 - accuracy: 0.9057 - val_loss: 0.7462 - val_accuracy: 0.7125
Epoch 18/20
 - 110s - loss: 0.2365 - accuracy: 0.9105 - val_loss: 0.7848 - val_accuracy: 0.7050
Epoch 19/20
 - 111s - loss: 0.2265 - accuracy: 0.9156 - val_loss: 0.8136 - val_accuracy: 0.7080
Epoch 20/20
 - 110s - loss: 0.2184 - accuracy: 0.9209 - val_loss: 0.8501 - val_accuracy: 0.7000
```

```
In [44]: scores = modelLSTM.evaluate(test_fit,test_y,verbose=1)
    ...: scores[1]
90/90 [==============================] - 1s 13ms/step
Out[44]: 0.7444444298744202
```

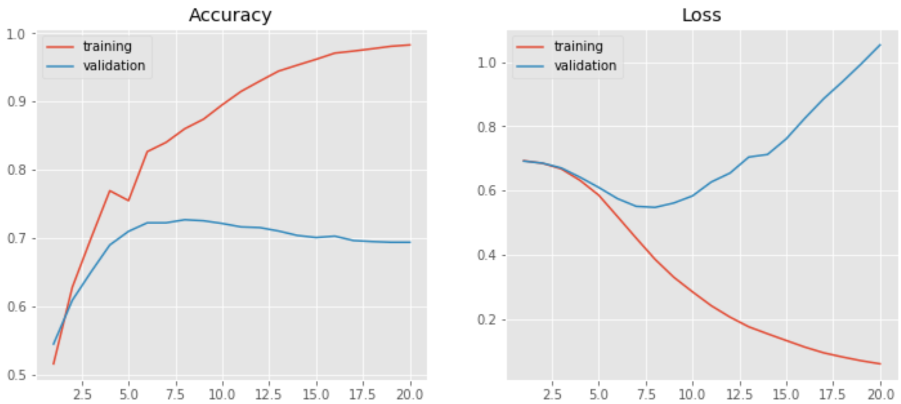
可以看到沒dropout的模型在訓練集和驗證集的預測表現上普遍都優於有dropout的，但在最後預測測試集時卻是一樣的準確率，因為只使用了10000筆資料來訓練，所以推測如果在更大的input上還是沒有使用dropout的話，可能測試的準確率會降的比有使用的還低。

- 訓練過程的accuracy和loss變化圖

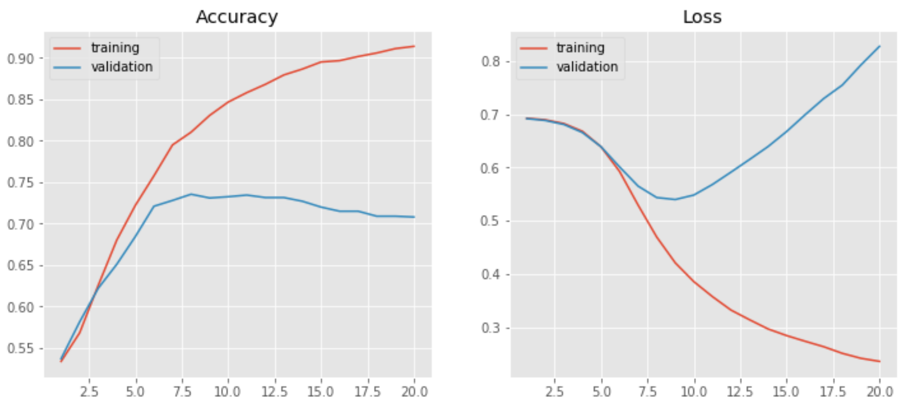
有dropout的RNN model：



沒dropout的RNN model：



有dropout的LSTM model：



沒dropout的LSTM model：