Final Project Report

Department of Management Information System master 2
7107029022 邱靖詒

一、 概述 Abstract

類型 ： 文本分類
目標 ： 創建一個模型來預測每個評論的惡意類型(type of toxicity)
資料集 ：Toxic comment dataset
https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data
輸入 ： 評論
輸出 ： 評論的惡意類型
模型 ： 長短期記憶網路 (Long Short Term Memory Network, LSTM)

二、 資料集描述 Data Descriptions

資料集為來自 Wikipedia 的大量評論(Wikipedia's talk page)，每個評論有 6 個輸出標籤，如下表一惡性類別所示。評論可以屬於這些類別，也可以屬於這些類別的子集，這使其成為多標籤分類問題。

toxic	惡意
severe_toxic	極度惡意
obscene	猥褻
threat	恐嚇
insult	侮辱
identity_hate	種族歧視

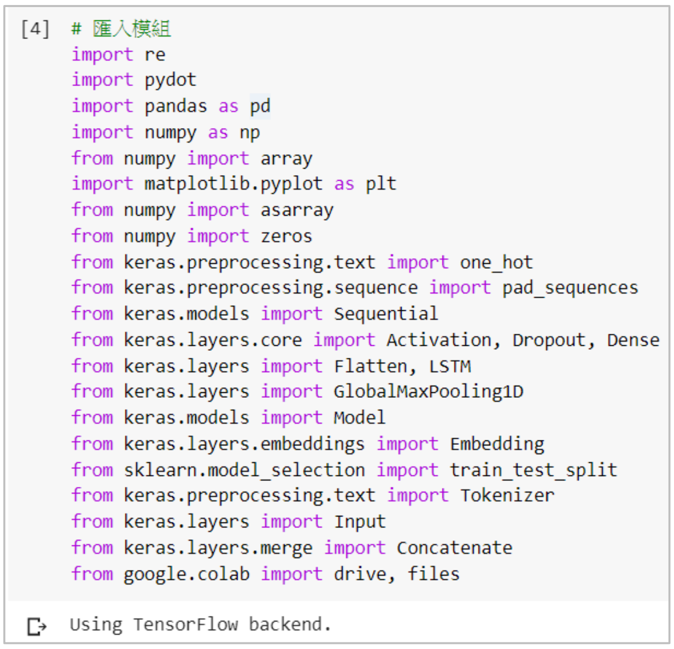
可以從下方 Kaggle 連結下載本文的數據集，只使用包含 160,000 筆評論的 train.csv 檔案。
https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

三、 檔案描述 File Descriptions

- **train.csv** - 訓練集，包含二進制標籤的評論。
- **test.csv** – 測試集，預測這些評論的惡意類別的機率，為了不使用手工標記，測試集包含一些未包含在評分中的評論。
- **sample_submission.csv** - 格式正確的樣本提交文件。
- **test_labels.csv** - 測試數據標籤，值為-1 表示未用於評分。

四、 步驟說明 Step Descriptions

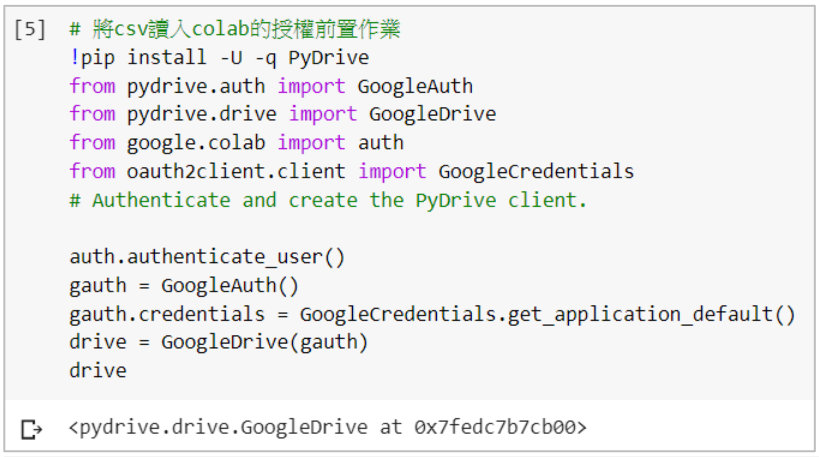
(1) 匯入模組



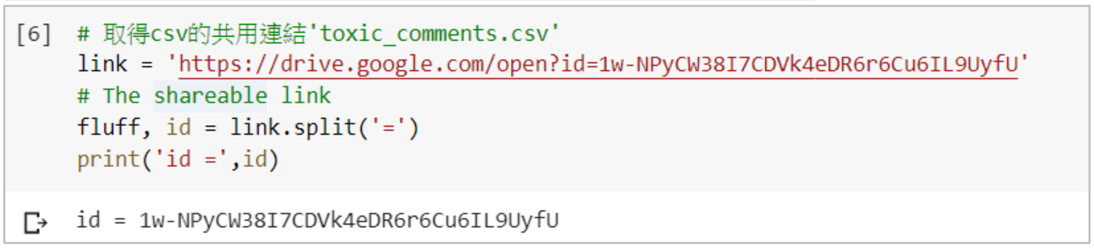
```
[4] # 匯入模組
    import re
    import pydot
    import pandas as pd
    import numpy as np
    from numpy import array
    import matplotlib.pyplot as plt
    from numpy import asarray
    from numpy import zeros
    from keras.preprocessing.text import one_hot
    from keras.preprocessing.sequence import pad_sequences
    from keras.models import Sequential
    from keras.layers.core import Activation, Dropout, Dense
    from keras.layers import Flatten, LSTM
    from keras.layers import GlobalMaxPooling1D
    from keras.models import Model
    from keras.layers.embeddings import Embedding
    from sklearn.model_selection import train_test_split
    from keras.preprocessing.text import Tokenizer
    from keras.layers import Input
    from keras.layers.merge import Concatenate
    from google.colab import drive, files

    Using TensorFlow backend.
```

(2) 讀取 csv

- colab 的授權前置作業，出現提示時，點擊連結進行身分認證，允許訪問你的 Google Drive，許可後將驗證碼貼製 colab 的驗證框中。完成驗證後，到 Google Drive 中的 CSV 文件，右鍵選擇「取得檔案共用連結」，該連結將被複製到剪貼板中，並貼到 Colab 中的 link 變數。

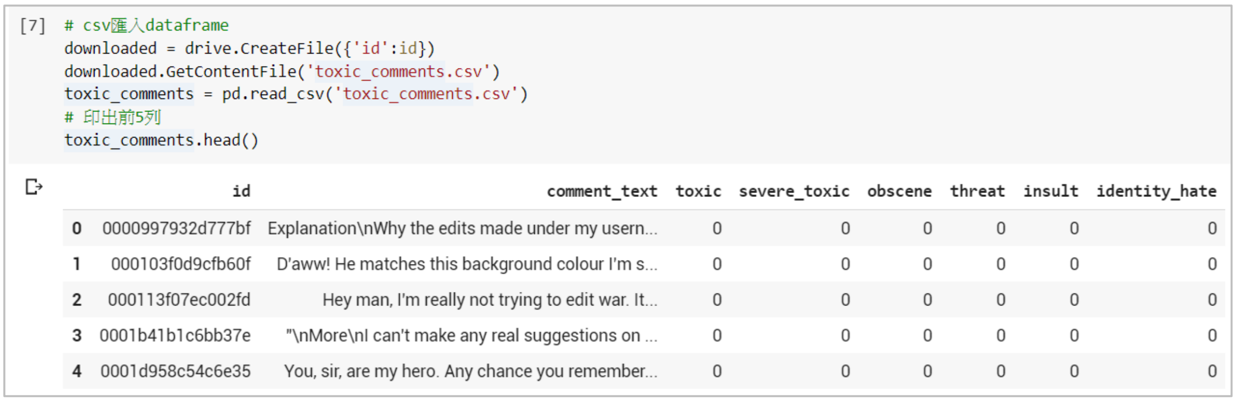


```
[5] # 將csv讀入colab的授權前置作業
    !pip install -U -q PyDrive
    from pydrive.auth import GoogleAuth
    from pydrive.drive import GoogleDrive
    from google.colab import auth
    from oauth2client.client import GoogleCredentials
    # Authenticate and create the PyDrive client.

    auth.authenticate_user()
    gauth = GoogleAuth()
    gauth.credentials = GoogleCredentials.get_application_default()
    drive = GoogleDrive(gauth)
    drive

    <pydrive.drive.GoogleDrive at 0x7fedc7b7cb00>
```



```
[6] # 取得csv的共用連結'toxic_comments.csv'
    link = 'https://drive.google.com/open?id=1w-NPyCW38I7CDVk4eDR6r6Cu6IL9UyfU'
    # The shareable link
    fluff, id = link.split('=')
    print('id =',id)

    id = 1w-NPyCW38I7CDVk4eDR6r6Cu6IL9UyfU
```

- 透過 id 來匯入 csv，並建立 DataFrame (toxic_comments)，印出前 5 行

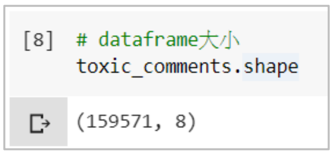


```
[7] # csv匯入dataframe
    downloaded = drive.CreateFile({'id':id})
    downloaded.GetContentFile('toxic_comments.csv')
    toxic_comments = pd.read_csv('toxic_comments.csv')
    # 印出前5列
    toxic_comments.head()
```

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

(3) 檢查 DataFrame

- 查看大小

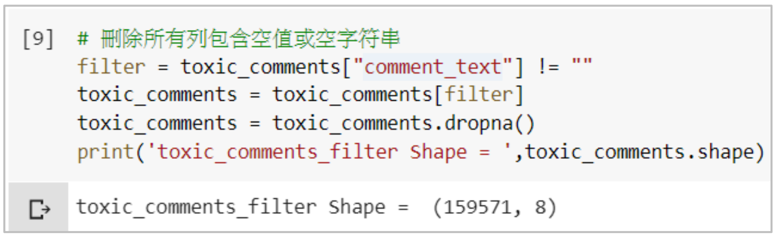


```
[8]  # dataframe大小
     toxic_comments.shape
```

```
(159571, 8)
```

→ 數據集包含 159571 條評論和 8 個欄位。

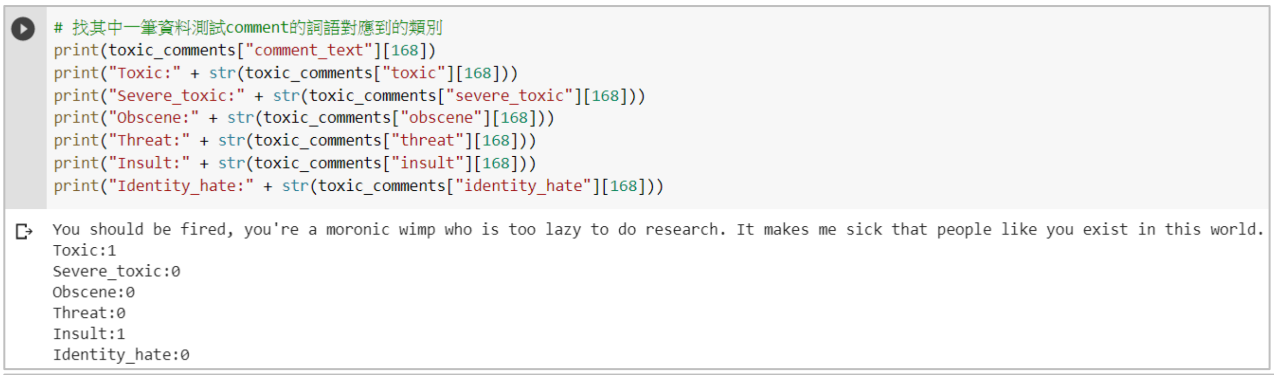
- DataFrame 整理，刪除所有列包含空值或空字符串



```
[9]  # 刪除所有列包含空值或空字符串
     filter = toxic_comments["comment_text"] != ""
     toxic_comments = toxic_comments[filter]
     toxic_comments = toxic_comments.dropna()
     print('toxic_comments_filter Shape = ',toxic_comments.shape)
```

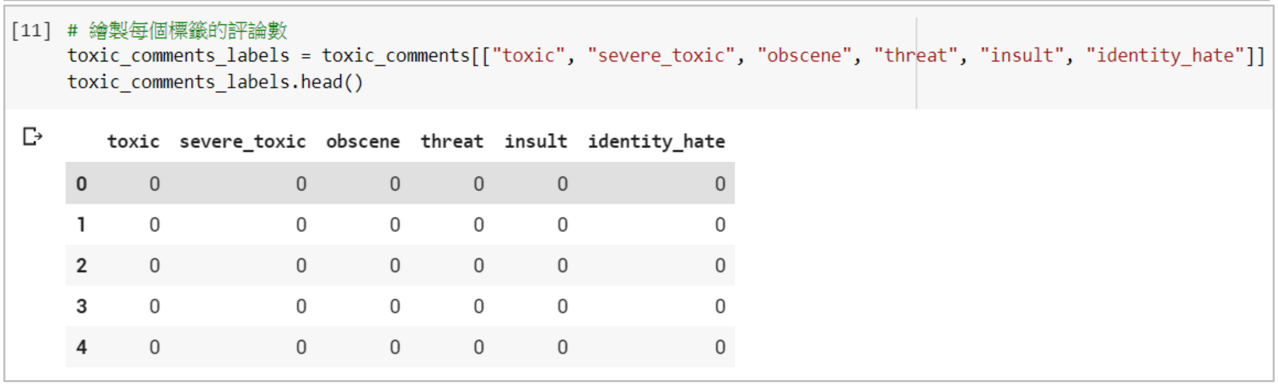
```
toxic_comments_filter Shape =  (159571, 8)
```

- 找其中一筆 comment 測試並查看對應到的惡意類別 (以第 168 筆評論為例)
  這顯然是惡意評論，查看與此評論相關的標籤，並為每個標籤繪製評論個數。



```
# 找其中一筆資料測試comment的詞語對應到的類別
print(toxic_comments["comment_text"][168])
print("Toxic:" + str(toxic_comments["toxic"][168]))
print("Severe_toxic:" + str(toxic_comments["severe_toxic"][168]))
print("Obscene:" + str(toxic_comments["obscene"][168]))
print("Threat:" + str(toxic_comments["threat"][168]))
print("Insult:" + str(toxic_comments["insult"][168]))
print("Identity_hate:" + str(toxic_comments["identity_hate"][168]))
```

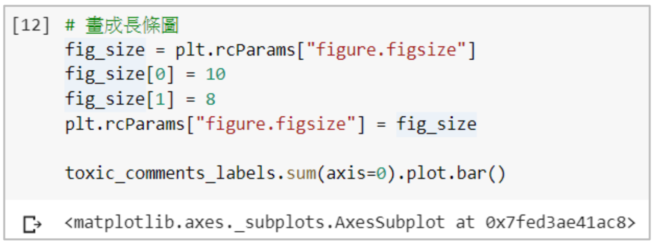
```
You should be fired, you're a moronic wimp who is too lazy to do research. It makes me sick that people like you exist in this world.
Toxic:1
Severe_toxic:0
Obscene:0
Threat:0
Insult:1
Identity_hate:0
```



```
[11]  # 繪製每個標籤的評論數
      toxic_comments_labels = toxic_comments[["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]]
      toxic_comments_labels.head()
```

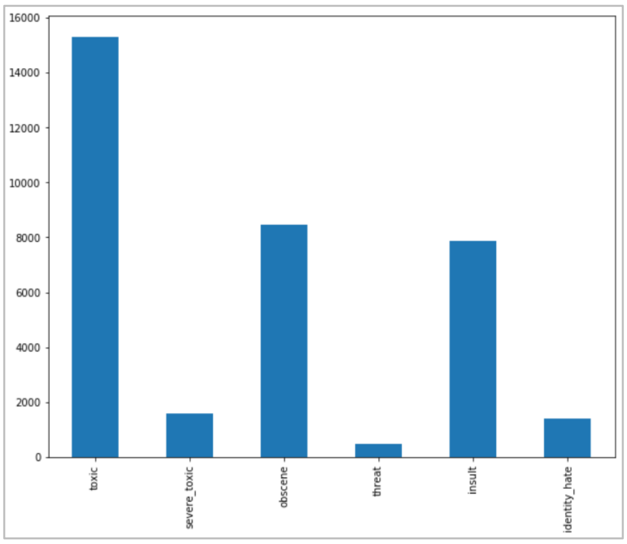
	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0

- 使用 toxic_comments_labelsData Frame 繪製柱狀圖，顯示不同標籤的總評論個數。



```
[12]  # 畫成長條圖
      fig_size = plt.rcParams["figure.figsize"]
      fig_size[0] = 10
      fig_size[1] = 8
      plt.rcParams["figure.figsize"] = fig_size

      toxic_comments_labels.sum(axis=0).plot.bar()
```

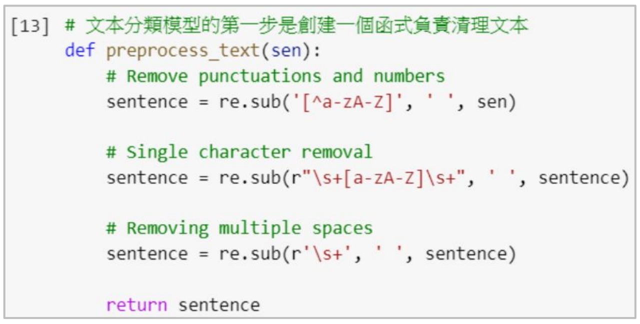
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed3ae41ac8>
```



→ 可看出"toxic"評論的出現頻率最高，其次分別是"obscene"和"insult"。
已經成功分析了數據集，在下一部分，將使用該數據集創建多標籤分類模型。

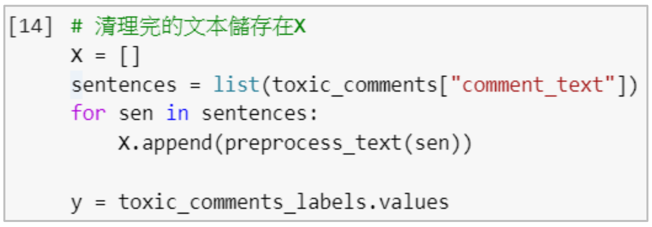
(4) 文本預處理

- 建立文本清理的函式
  包含移除標點符號和數字、移除多個空格、移除單一字母。



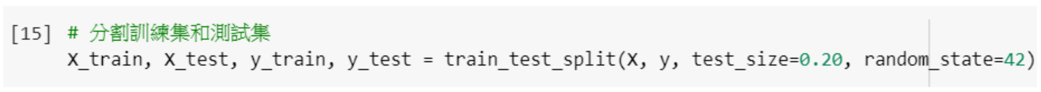
```
[13] # 文本分類模型的第一步是創建一個函式負責清理文本
     def preprocess_text(sen):
         # Remove punctuations and numbers
         sentence = re.sub('[^a-zA-Z]', ' ', sen)

         # Single character removal
         sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

         # Removing multiple spaces
         sentence = re.sub(r'\s+', ' ', sentence)

         return sentence
```

創建輸入集和輸出集，輸入是 DataFrame 中 comment_text 欄位的評論，把預處理過的評論儲存在 X 變量中。標籤或輸出儲存在 DataFrame 的 toxic_comments_label 中，使用該 DataFrame 的值將輸出儲存在 y 變量中，輸出的標籤已經是 one-hot encoded 向量的形式。



```
[14] # 清理完的文本儲存在X
     X = []
     sentences = list(toxic_comments["comment_text"])
     for sen in sentences:
         X.append(preprocess_text(sen))

     y = toxic_comments_labels.values
```

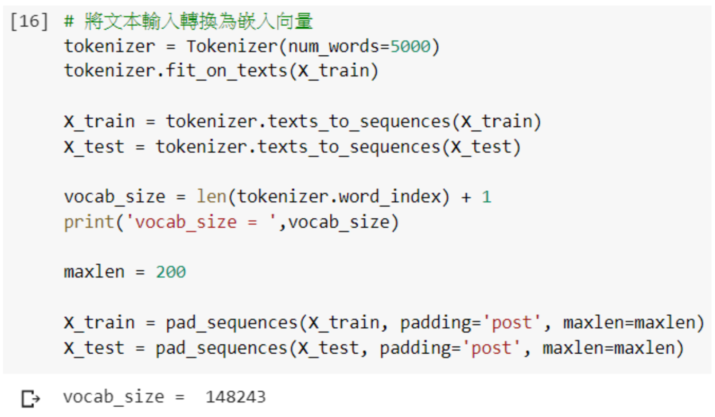
(5) 分割訓練集和測試集



```
[15] # 分割訓練集和測試集
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

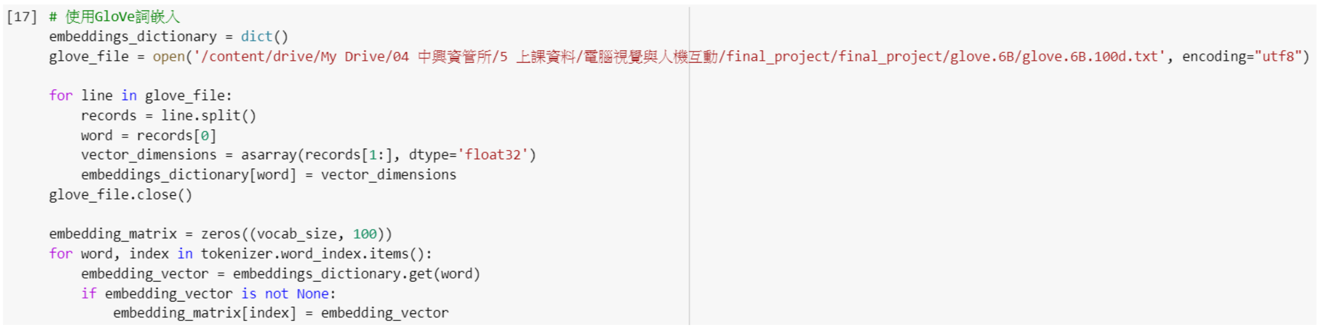
(6) 嵌入向量

將文本輸入轉換為嵌入式向量。



```
[16] # 將文本輸入轉換為嵌入向量
     tokenizer = Tokenizer(num_words=5000)
     tokenizer.fit_on_texts(X_train)

     X_train = tokenizer.texts_to_sequences(X_train)
     X_test = tokenizer.texts_to_sequences(X_test)

     vocab_size = len(tokenizer.word_index) + 1
     print('vocab_size = ',vocab_size)

     maxlen = 200

     X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
     X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

     vocab_size =  148243
```

使用 GloVe 預訓練模型做 word embedding，把文本輸入轉換為向量輸入。
GloVe 預訓練模型下載：https://www.kaggle.com/terenceliu4444/glove6b100dtxt
http://nlp.stanford.edu/data/glove.6B.zip



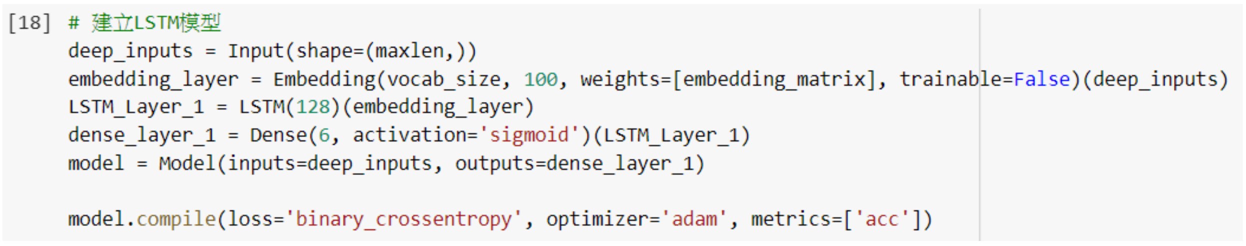
```
[17] # 使用GloVe詞嵌入
     embeddings_dictionary = dict()
     glove_file = open('/content/drive/My Drive/04 中興資管所/5 上課資料/電腦視覺與人機互動/final_project/final_project/glove.6B/glove.6B.100d.txt', encoding="utf8")

     for line in glove_file:
         records = line.split()
         word = records[0]
         vector_dimensions = asarray(records[1:], dtype='float32')
         embeddings_dictionary[word] = vector_dimensions
     glove_file.close()

     embedding_matrix = zeros((vocab_size, 100))
     for word, index in tokenizer.word_index.items():
         embedding_vector = embeddings_dictionary.get(word)
         if embedding_vector is not None:
             embedding_matrix[index] = embedding_vector
```

(7) 創建多標籤文本分類模型

分成兩種方法，使用單個 Dense 輸出層和多個 Dense 輸出層。在第一種方法中，使用具有 6 個輸出的單個 Dense 層，並具有 sigmoid 激活函數和二進制交叉熵損失函數。Dense 輸出層中的每個神經元代表 6 類輸出標籤的其中之一。Sigmoid 激活函數將為每個神經元回傳 0 到 1 之間的值，如果任何神經元的輸出值大於 0.5，則假定評論屬於特定一類。在第二種方法中，將每個標籤創建一個 Dense 輸出層，在輸出中總共包含 6 個 Dense 層，每一層都有自己的 sigmoid 激活函數。

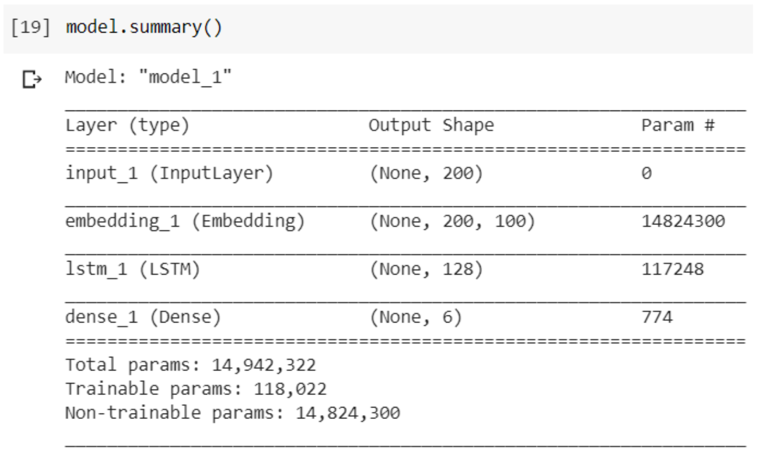
- **Model 1：Multi-lable Text Classification Model with Single Output Layers**

  模型 1 具有一個輸入層，一個嵌入層，一個具有 128 個神經元的 LSTM 層和一個具有 6 個神經元的輸出層，輸出有 6 個標籤。



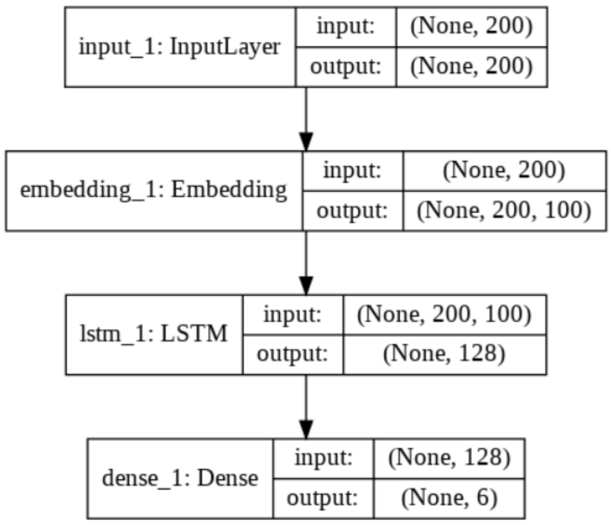
```
[18] # 建立LSTM模型
     deep_inputs = Input(shape=(maxlen,))
     embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False)(deep_inputs)
     LSTM_Layer_1 = LSTM(128)(embedding_layer)
     dense_layer_1 = Dense(6, activation='sigmoid')(LSTM_Layer_1)
     model = Model(inputs=deep_inputs, outputs=dense_layer_1)

     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

模型概要(神經網絡架構)如下：



```
[19] model.summary()

Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 200)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 200, 100)          14824300
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 14,942,322
Trainable params: 118,022
Non-trainable params: 14,824,300
_________________________________________________________________
```

```
[20] from keras.utils import plot_model
     plot_model(model, to_file='model_plot4a.png', show_shapes=True, show_layer_names=True)
```



→ 可看出輸出層僅包含 1 個具有 6 個神經元的 Dense 層。

訓練模型(5 個 epochs)：

```
[21] history = model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1, validation_split=0.2)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add i

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is de

Train on 102124 samples, validate on 25532 samples
Epoch 1/5
102124/102124 [==============================] - 226s 2ms/step - loss: 0.1474 - acc: 0.9634 - val_loss: 0.1402 - val_acc: 0.9632
Epoch 2/5
102124/102124 [==============================] - 223s 2ms/step - loss: 0.1189 - acc: 0.9651 - val_loss: 0.0825 - val_acc: 0.9757
Epoch 3/5
102124/102124 [==============================] - 224s 2ms/step - loss: 0.0651 - acc: 0.9780 - val_loss: 0.0605 - val_acc: 0.9793
Epoch 4/5
102124/102124 [==============================] - 223s 2ms/step - loss: 0.0571 - acc: 0.9800 - val_loss: 0.0573 - val_acc: 0.9799
Epoch 5/5
102124/102124 [==============================] - 223s 2ms/step - loss: 0.0551 - acc: 0.9807 - val_loss: 0.0575 - val_acc: 0.9802
```

評估模型準確率：

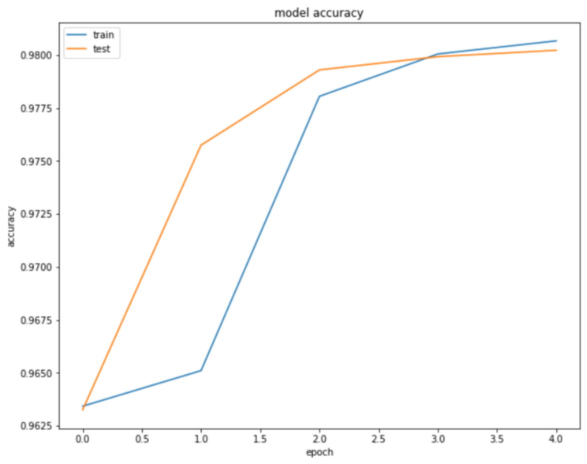
```
[22] score = model.evaluate(X_test, y_test, verbose=1)

     print("Test Score:", score[0])
     print("Test Accuracy:", score[1])
```

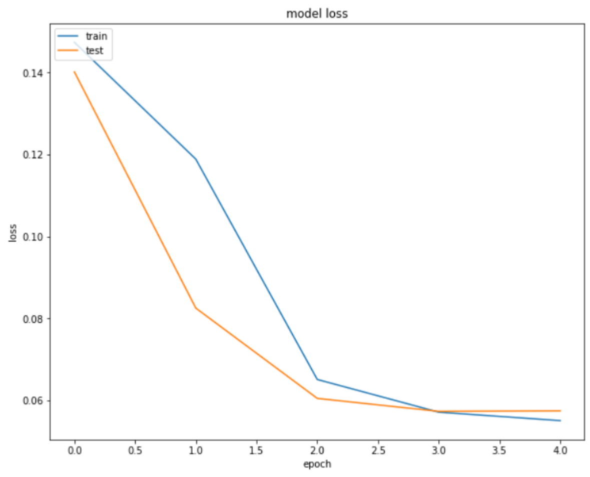
```
31915/31915 [==============================] - 89s 3ms/step
Test Score: 0.05540275421135388
Test Accuracy: 0.9808031641333226
```

→ 模型實現了約 98%的準確率。

繪製訓練和測試集的損失和準確率曲線，以查看模型是否過擬合。

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```





→ 模型在測試集上沒有過度擬合

- **Model 2：Multi-lable Text Classification Model with Multiple Output Layers**

創建一個多標籤文本分類模型，其中每個輸出標籤將具有一個專用的輸出 Dense 層。延續先前定義的預處理函式(步驟 4)，接著模型創建輸入和輸出，輸入是文本評論，輸出是 6 個標籤。

```
[14] # 清理完的文本儲存在X
     X = []
     sentences = list(toxic_comments["comment_text"])
     for sen in sentences:
         X.append(preprocess_text(sen))

     y = toxic_comments[["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]]
```

y 變量包含 6 個標籤的組合輸出，但是，要為每個標籤創建單獨的輸出層，即創建 6 個變量，這些變量儲存來自訓練數據的各個標籤， 6 個變量分別儲存測試數據的各個標籤值。以下程式碼把 train 和 test 各分成 6 個：

```
[16] # y變量包含6個標籤的組合輸出
     # First output
     y1_train = y_train[["toxic"]].values
     y1_test =  y_test[["toxic"]].values

     # Second output
     y2_train = y_train[["severe_toxic"]].values
     y2_test =  y_test[["severe_toxic"]].values

     # Third output
     y3_train = y_train[["obscene"]].values
     y3_test =  y_test[["obscene"]].values

     # Fourth output
     y4_train = y_train[["threat"]].values
     y4_test =  y_test[["threat"]].values

     # Fifth output
     y5_train = y_train[["insult"]].values
     y5_test =  y_test[["insult"]].values

     # Sixth output
     y6_train = y_train[["identity_hate"]].values
     y6_test =  y_test[["identity_hate"]].values
```

延續先前步驟 6，將文本轉換成嵌入向量，並搭配 GloVe word embeddings 模型。
接著創建模型，模型具有一層輸入層，一層嵌入層，然後一層具有 128 個神經元的 LSTM 層。LSTM 層的輸出將作為 6 個 Dense 輸出層的輸入，每個輸出層具有 1 個具有 Sigmoid 激活函數的神經元，每個輸出會預測 0 到 1 之間的整數值用來對應標籤。

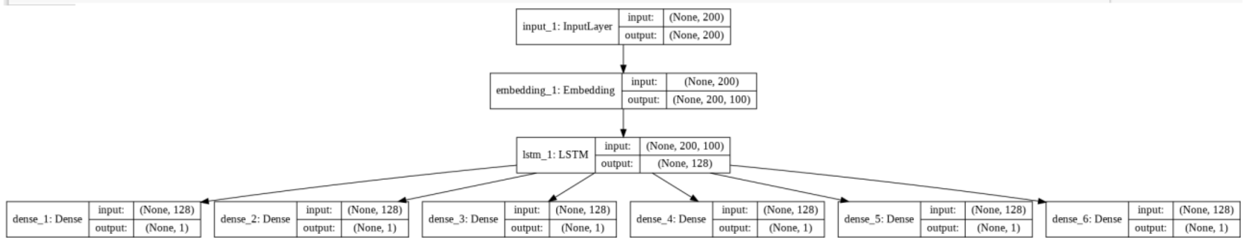
```
[19] # 建立LSTM模型
     input_1 = Input(shape=(maxlen,))
     embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False)(input_1)
     LSTM_Layer1 = LSTM(128)(embedding_layer)

     output1 = Dense(1, activation='sigmoid')(LSTM_Layer1)
     output2 = Dense(1, activation='sigmoid')(LSTM_Layer1)
     output3 = Dense(1, activation='sigmoid')(LSTM_Layer1)
     output4 = Dense(1, activation='sigmoid')(LSTM_Layer1)
     output5 = Dense(1, activation='sigmoid')(LSTM_Layer1)
     output6 = Dense(1, activation='sigmoid')(LSTM_Layer1)

     model = Model(inputs=input_1, outputs=[output1, output2, output3, output4, output5, output6])
     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

模型概要(神經網路架構)如下：

```
[20] model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 200)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 200, 100)     14824300    input_1[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 128)          117248      embedding_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            129         lstm_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            129         lstm_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1)            129         lstm_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)            129         lstm_1[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 1)            129         lstm_1[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 1)            129         lstm_1[0][0]
==================================================================================================
Total params: 14,942,322
Trainable params: 118,022
Non-trainable params: 14,824,300
__________________________________________________________________________________________________
```

```
from keras.utils import plot_model
plot_model(model, to_file='model_plot4b.png', show_shapes=True, show_layer_names=True)
```


input_1: InputLayer | input: (None, 200) | output: (None, 200)
embedding_1: Embedding | input: (None, 200) | output: (None, 200, 100)
lstm_1: LSTM | input: (None, 200, 100) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 1)
dense_2: Dense | input: (None, 128) | output: (None, 1)
dense_3: Dense | input: (None, 128) | output: (None, 1)
dense_4: Dense | input: (None, 128) | output: (None, 1)
dense_5: Dense | input: (None, 128) | output: (None, 1)
dense_6: Dense | input: (None, 128) | output: (None, 1)

→ 可看出有 6 個不同的輸出層，和 Model 1 不同。

→ Model 1：具有單個輸入層的模型 VS Model 2：具有多個輸出層的模型。

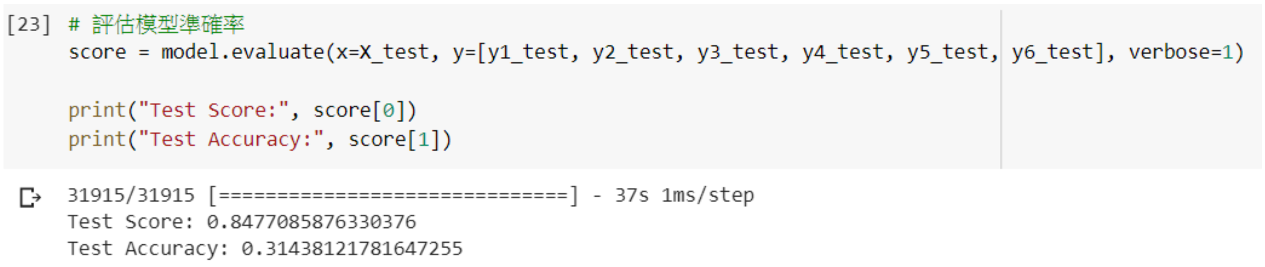
訓練模型(5 個 epochs)：

```
[22] history = model.fit(x=X_train, y=[y1_train, y2_train, y3_train, y4_train, y5_train, y6_train], batch_size=8192, epochs=5, verbose=1, validation_split=0.2)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add inst

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.

Train on 102124 samples, validate on 25532 samples
Epoch 1/5
102124/102124 [==============================] - 85s 835us/step - loss: 3.5806 - dense_1_loss: 0.6208 - dense_2_loss: 0.5916 - dense_3_loss: 0.5932 - dense_4_loss: 0.5811 - dense_5_l
Epoch 2/5
102124/102124 [==============================] - 79s 777us/step - loss: 0.9019 - dense_1_loss: 0.3175 - dense_2_loss: 0.0744 - dense_3_loss: 0.2183 - dense_4_loss: 0.0341 - dense_5_l
Epoch 3/5
102124/102124 [==============================] - 79s 776us/step - loss: 0.8482 - dense_1_loss: 0.3151 - dense_2_loss: 0.0571 - dense_3_loss: 0.2079 - dense_4_loss: 0.0217 - dense_5_l
Epoch 4/5
102124/102124 [==============================] - 78s 768us/step - loss: 0.8442 - dense_1_loss: 0.3146 - dense_2_loss: 0.0574 - dense_3_loss: 0.2062 - dense_4_loss: 0.0214 - dense_5_l
Epoch 5/5
102124/102124 [==============================] - 78s 769us/step - loss: 0.8411 - dense_1_loss: 0.3144 - dense_2_loss: 0.0562 - dense_3_loss: 0.2054 - dense_4_loss: 0.0214 - dense_5_l
```

可以看到在每個 epoch，輸出所有 6 個 Dense 層都有 loss、value loss、accuracy 和 value accuracy。

評估模型準確率：



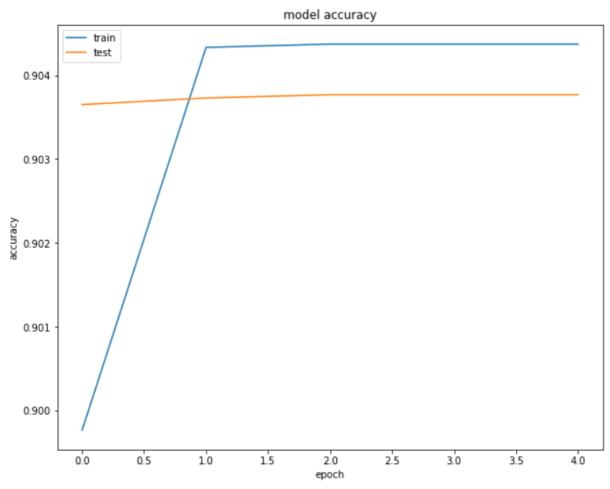
```
[23] # 評估模型準確率
     score = model.evaluate(x=X_test, y=[y1_test, y2_test, y3_test, y4_test, y5_test, y6_test], verbose=1)

     print("Test Score:", score[0])
     print("Test Accuracy:", score[1])
```

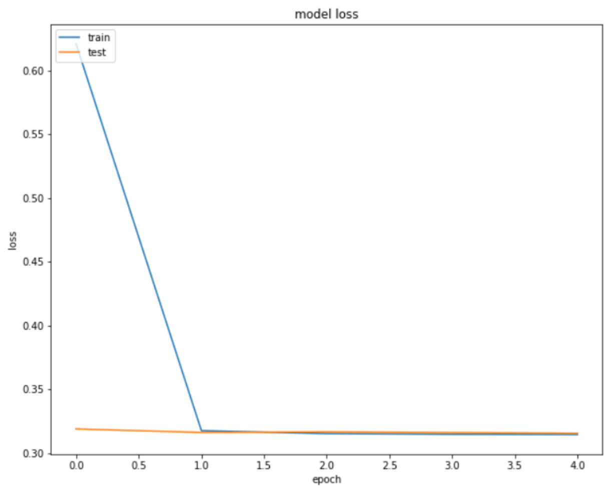
```
31915/31915 [==============================] - 37s 1ms/step
Test Score: 0.8477085876330376
Test Accuracy: 0.31438121781647255
```

嘗試為 5 個 epochs 來訓練模型，但在測試集上過擬合，所以增加了 batch_size，但測試集準確率仍然不是很好。過擬合的原因可能是在這種情況下，每個標籤設置了單獨的輸出層，而增加了模型的複雜性，模型複雜度的增加通常會導致過度擬合。

→ 透過多個輸出層在測試數據集上只能達到 31%的準確率。

繪製訓練和測試集的損失和準確率曲線，以查看模型是否過擬合。





從輸出中可看出，在第一個 epoch 之後，測試集的準確性並未收斂。而且，訓練和測試準確性之間的差異非常小，因此，模型在第一個 epoch 後就開始過擬合，因此得到一個性能很差模型。

五、 結論

多標籤文本分類是最常見的文本分類問題之一。實驗了兩種用於多標籤文本分類的深度學習方法。在第一種方法中，使用具有多個神經元的單個 Dense 輸出層，其中每個神經元代表一個標籤 label。在第二種方法中，每個神經元的標籤都會建立單獨的 Dense 層。實驗結果顯示，具有多個神經元的單個輸出層比多個輸出層的效果更好，可能可以更改激活函數和訓練測試集分割比例，也許可以得到更好的結果。