Kaggle Competition Report

一、 前言

在拿到這份作業的時候，覺得非常不安，原因是以前很少做過 AI 相關的作業，在修了這門課後也只做了一次作業，而在這之前相當於就只有讀過書上的定義，而沒有實際操作，突然就碰到實戰，著實令人覺得不安但又興奮，興奮地點在於，用 ai 處理數據是我一職以來感興趣的領域，然而一直沒機會，那藉此次作業，我也算是終於初次的入門了這個領域。

二、 資料準備

首先我們拿到的三個檔案裏面，我先把資料欄位依據 id 來把 identification 和 emotion 結合，接著透過找資料了解如何取出，json 內的 id 欄位還有 text 欄位，一樣把他依據 id 做合併，例如:

	tweet_id	text	identification	emotion
0	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation
1	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness
2	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear
3	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	train	joy
4	0x2c91a8	Still waiting on those supplies Liscus. <LH>	train	anticipation

接著再根據 identification 欄位把 train test 分開，這就完成了我們的資料準備。

三、 資料預處理

首先，要讓 AI 能讀懂我們的標籤，將 8 種 label 一一對應到 0~7 號，這就完成了 label 的處理，再來則是對 text 的處理，首先我觀察到了，推文裡面，有許多無意義的詞語可能是#AAJDAD 或者是@HSFJFSKL，再不然還有網址還有各種奇奇怪怪的東西，總而言之大概率不是我們要的 fearture，但這邊值得一提的是，有些文章會希望拿掉 emoji 然而我覺得表情符號應該會是一種表達情感的重要信息，因此我沒有把它去除，除此之外我去除的內容有這些：

```python
def sentence_remove(tweet):

    r = "[_.!+-=——,$%^，。？、~@#￥%……&*《》<>「」“”‘’（）{}【】□‡()//\\\[\]\"]"

    tweet = re.sub(r'[http|https]*://[a-zA-Z0-9.?/&=:]*', ' ', tweet)  # 刪除網址
    tweet = re.sub(r'#\S+', ' ', tweet)                                # 刪除 hash tag
    tweet = re.sub(r'@\S+', ' ', tweet)                                # 刪除 @用户
    tweet = re.sub(r'\\S+', ' ', tweet)                                # 刪除空格換行
    tweet = re.sub(r, ' ', tweet)                                      # 刪除特殊符号
    return tweet
```

刪除完畢之後我們會得到一個乾淨的句子，這邊我還在做了一點處理，詞性還原，畢竟同一個詞可能會被不同型態分散太多權重，不過我認為一個字例如 player playing 代表的意思不同，所以我傾向於照著詞性來還原，而不用較為暴力的 stem，最後大致上像是：

```python
def get_text(ans_train):

    texts=ans_train[['text']]
    texts= texts['text'].apply(sentence_remove).tolist()

    datas=[]
    wnl = WordNetLemmatizer()
    for text in tqdm(texts):

        text = "".join([c for c in text.lower() if c not in punctuation]) #去除標點

        tmp = [item for item in text.split(" ") if len(item)>=3 and item!='lh']#去除停用詞


        #詞性還原

        tagged_sent = pos_tag(tmp)
        lemmas_sent = []

        for tag in tagged_sent:
            wordnet_pos = get_wordnet_pos(tag[1]) or wordnet.NOUN
            temp_word=wnl.lemmatize(tag[0], pos=wordnet_pos)
            lemmas_sent.append(temp_word)

        #print(lemmas_sent)
        tmp=" ".join(lemmas_sent)
        datas.append(tmp)

    return datas
```

至此，文本前處理就完成了。

四、 特徵選取

把文本處理完後，切成訓練:測試=4:1，然後開始提取詞頻，爾後我選擇使用最高頻率出現~mean/2 出現次數的詞，約莫有 12775 個，這就是我們的 feature 數量，然後對他們使用 tfidf 取得詞語權重。

五、 模型選擇

1. Regression:

作為我的 baseline 以及我的入門，選擇了這個分類器來做測試，因為它的速度快，邏輯簡單好理解，然而結果也不慎理想，經過約莫 10 次的調整以及改善後，最終只有做到 0.35854，因此我決定放棄這個，轉往下一個目標！

2. SVM:

做為第二種模型，因為時間不構我只嘗試了一次，結果也只有 0.31701 我覺得實在太差，也可能是我沒有選到適合他的參數，但我就決定放棄了，因為找到更多的資料，顯示 XGboost 的優秀，因此讓我迫不及待地想試試看看。

3. XGboost:

作為本次我重點嘗試的模型，一開始我先用最基本的:

```
reg = xgb.XGBClassifier(**params)
model_t=reg.fit(X_train, y_train,
        eval_set=[(X_valid, y_valid)], eval_metric='merror',
        verbose=False)
```

經過多次調整 feature 後我得到了 0.4100 是一個大躍進，也讓我感到很興奮，但還不夠。
後來我發現，有許多能夠調整的參數，因此我先去網路上查詢，其他人得到好結果時調整的參數是多少，再次把它調整，套用了這樣的參數：

```
params_2={'max_depth': 9, 'n_estimators': 1500, 'gamma': 0.009804179085702403, 'colsample_bytree': 0.2297635947368334}
```

也有些微的提升，最後得到了 0.42 左右，就當我以為這是瓶頸了之後，突然想到以前 survey 過的 paper 中使用了 optuna 的方式挑整參數，然而一到 xgboost 開始模型的訓練時間一次就來到了 2hr，因此我只能設定調整參數的回數為 10 回，然後得到了:

```
#optune 的 參數

params={'max_depth': 7,
        'n_estimators': 1500,
        'gamma': 2.7783540662654385e-06,
        'colsample_bytree': 0.8169210168405894,
        'objective':'softmax',
      #'tree_method': 'gpu_hist'
       }
```

然後得到了 0.43 左右的成績，也作為我第一階段嘗試的告一段落，因為同一時間我也嘗試了 bert 但對我的 1650S 來說已經是無法負荷的模型了，所以也原本打算就停在這裡結束。

4. LSTM:

過了幾天之後由於時間有限，我又利用課餘時間，閱讀一些資料後，覺得好像可以使用 neuro network 的方法但又不能用到 bert 等級，所以就找了 LSTM 來嘗試，因為設備問題，所以使用的結構不是很複雜：

```
model.add(Embedding(53592, 100, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(8, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

然而，就這樣一個簡單的模型，再經過 23hr 只 train 了 2 個 epoch 的情況，就得到了 0.44638 的結果，直接推翻了我前面的結果，令我非常的訝異，也因此這個成為了我最好的結果，後面還有再次的修改，並且調整 epoch 到 7，則花費了 3 天 train，但無奈時間已經超時，但我還會繼續把它嘗試完。

六、後記

在這次的作業中，雖然說很大一部分的知識都需要靠自己來取得，但我覺得這算是研究生的本分，再靠自己找到答案後，心情也是豁然開朗，而且看著分數一步步的上升，喜悅也是不盡言語表達，唯一覺得可惜的是，沒有辦法把 bert 實做出來，以及 LSTM 也沒辦法嘗試太多次，也是第一次體會到以前大家說的，跑深度學習，需要的硬體設備要有一定程度的意思，一個模型要 train3 天甚至更多的情況下，deadline 的壓力真的是非常巨大，期望下次的時候我能找到好的設備替代方案。