# Report

109550031 李旻融

## ➢ Preprocessing Method

在進行 preprocessing 的過程，會依序執行下面四個步驟：

1. **Remove stopwords :** 根據 nltk 所提供的 stopwords 列表移除 stopwords
2. **Remove punctuation :** 利用 string 所提供的 punctuation 來檢查每個 char 是否為標點符號並移除標點符號，除此之外，並利用 replace 的函式將 <br /><br /> 用一個空格代替
3. **Lemmatization :** 利用 nltk 提供的 wordnet 列表將名詞、動詞、形容詞都進行字詞還原
4. **Remove digits :** 利用 string 所提供的 digits 來檢查每個 char 是否為數字並移除數字

下方為 preprocessing 的過程：

Initial sentence **->** I went to the elementary school when I was 6 years old.
Remove stopwords **->** went elementary school 6 years old .
Remove punctuation **->** went elementary school 6 years old
Lemmatization **->** go elementary school 6 year old
Remove digits **->** go elementary school year old

## ➢ Perplexity

在計算 Perplexity 的時候，我大致用到了以下的概念：

1. 特定詞彙後面所要接的詞語的條件機率計算如右： $P(B|A) = \frac{\text{bi-gram(AB)}}{\text{uni-gram(A)}}$
2. 計算 entropy 的公式如下：

$$\text{entropy } = -\frac{1}{m} * \sum_{x=1}^{m} \log_2 P(x)$$

3. Perplexity = $2^{entropy}$

✧ 我們可以發現在計算 Perplexity 的時候，會用到 P(x)，但因為 P(x)要取 log 值，所以 P(x)=0 是不被允許的，於是在計算 P(x)前，我會先檢查 P(x)是否為零

- **feature_num selection**

因為並不會選擇所有的 feature，所以我將原先的 feature list 進行排序，依照 feature 在訓練時的個數來進行排序，由大到小，再依照 feature_num 選到一定的個數。

在下表可以看到，我選擇 300, 500, 700 作為 feature_num 來做比較，無論是 bi-gram 還是 DistilBert 都有一個特點，那就是 non-preprocessing 的表現都較優秀，但如果我只用另外三個函式互相排列組合進行 preprocessing(不包含 remove_stopwords)，表現則會和 non-preprocessing 差不多，在下個部分有更詳細的比較。

**bi-gram (non-preprocessing)**

| Feature_num | Perplexity | F1 score | Precision | Recall |
|---|---|---|---|---|
| 300 | 93.988 | 0.6807 | 0.6854 | 0.6821 |
| 500 | 93.988 | 0.7057 | 0.7088 | 0.7065 |
| 700 | 93.988 | 0.7173 | 0.7211 | 0.7182 |

**bi-gram (preprocessing)**

| Feature_num | Perplexity | F1 score | Precision | Recall |
|---|---|---|---|---|
| 300 | 225.908 | 0.6849 | 0.6936 | 0.6874 |
| 500 | 225.908 | 0.6988 | 0.7053 | 0.7005 |
| 700 | 225.908 | 0.7141 | 0.7218 | 0.7160 |

**DistilBert (non-preprocessing)**

| Feature_num | F1 score | Precision | Recall | Loss |
|---|---|---|---|---|
| 300 | 0.9318 | 0.9319 | 0.9318 | 0.2289 |
| 500 | 0.9344 | 0.9347 | 0.9344 | 0.2286 |
| 700 | 0.9320 | 0.9321 | 0.9320 | 0.2289 |

**DistilBert (preprocessing)**

| Feature_num | F1 score | Precision | Recall | Loss |
|---|---|---|---|---|
| 300 | 0.9006 | 0.9023 | 0.9007 | 0.2941 |
| 500 | 0.8922 | 0.8967 | 0.8925 | 0.2930 |
| 700 | 0.8980 | 0.9007 | 0.8982 | 0.2945 |

## ➢ **Compare**

✓ 比較不同的 preprocessing 之間的表現：

在這個階段，我利用 bi-gram 的數據來呈現 preprocessing 與否對 Perplexity 和 F1-score 的影響 (都是在 feature_num = 500 的情況下進行)
◎ 代表有使用該函式來處理

根據下表可以發現只使用 lemmativation 和 remove digit 的時候，有最低的 Perplexity，但其他的數字(紅色為表現較差的數據、藍色為表現較好的數據)顯示 Perplexity 和 F1-score 似乎沒有明顯的關係，而且有移除 stopwords 的 Perplexity 都特別高，這應該是因為 Perplexity 指的是句子的通暢程度，移除 stopwords 會使得句子十分不流暢。

| Remove stopwords | Remove punctuation | lemmativation | Remove digit | Perplexity | F1-score |
|---|---|---|---|---|---|
| | | | | 93.988 | 0.7057 |
| ◎ | | | | 155.858 | 0.6769 |
| | ◎ | | | 123.515 | 0.7150 |
| | | ◎ | | 93.771 | 0.7084 |
| | | | ◎ | 94.005 | 0.7055 |
| ◎ | ◎ | | | 205.397 | 0.6895 |
| ◎ | | ◎ | | 161.054 | 0.6902 |
| ◎ | | | ◎ | 156.413 | 0.6764 |
| | ◎ | ◎ | | 124.497 | 0.7162 |
| | ◎ | | ◎ | 123.246 | 0.7155 |
| | | ◎ | ◎ | 93.729 | 0.7084 |
| ◎ | ◎ | ◎ | | 224.528 | 0.6998 |
| ◎ | ◎ | | ◎ | 207.190 | 0.6900 |
| ◎ | | ◎ | ◎ | 161.386 | 0.6880 |
| | ◎ | ◎ | ◎ | 124.205 | 0.7159 |
| ◎ | ◎ | ◎ | ◎ | 225.908 | 0.6988 |

✓ 比較 uni-gram, bi-gram, tri-gram, DistilBert 之間的表現：

我另外實作了 uni-gram 和 tri-gram，下圖為四個 model 在未經 preprocessing 的表現：

(都是在 feature_num = 500 的情況下進行)
根據下方圖表可以發現，在所有 n-gram 裡頭，uni-gram 的表現最好，tri-gram 的表現最差，我認為應該是這次作業要判斷的是情緒字詞，和前後文是什麼字沒有很大的關

聯，所以 uni-gram 一個字一個字分開，反而有更好的表現，但 uni-gram 的 Perplexity 就極大(大約 1500)。

| | F1-score |
|---|---|
| Uni-gram | 0.7776 |
| Bi-gram | 0.7057 |
| Tri-gram | 0.6652 |
| DistilBert | 0.9286 |

## ➢ **Discussion**

1. Why bi-gram can't outperform DistilBert?

   bi-gram 只有考慮到相鄰的兩個字之間的關係，但 DistilBert 就像人一樣，能根據周圍的所有單詞學習單詞上下文，自動偵測所有與答案相關的字詞，判斷哪些是相關那些則是沒有關係，這種機制稱為 self-attention。

2. Can bi-gram consider long-term dependencies?

   答案是不能，因為 bi-gram 只有考慮到相鄰的兩個字之間的關係，上下文的語句對 bi-gram 來說並不重要，bi-gram 並不能正確的判斷語義。

3. Would the preprocessing methods improve the performance of the bi-gram model?

   根據 Compare 那邊的圖表顯示，preprocessing 對於改善 bi-gram 是有用的，因為 preprocessing 可以精簡化內容，除去一些不必要的東西，但並不是所有的 preprocessing method 都有幫助，像是移除 stopwords 就相對較沒有用。

4. If you convert all words that appeared less than 10 times as [UNK], would it in general increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of the words that appeared just once as [UNK]?

   將出現次數少於十次的字詞轉成 [UNK] 和只出現一次的詞彙轉為 [UNK] 進行比較，可以發現如果我們將次數少於十次的字詞都轉成 [UNK]，模型不認識的詞彙就會變少，這樣就更好進行計算，機率會提高，entropy 的值降低也會讓 perplexity 降低。

## ➢ **Problems**

1. 在寫 preprocess 的時候，原先我想用 stemming 的方法，但那時候我試寫了一些句子之後，發現有些字後面有 ed 便會直接被刪去，即便它不是過去式，舉例來說，像是 indeed 經過 stemming 後，會變成 inde，但這會影響句子判斷，所以後來我找到另一個方法—字形還原(lemmatization)，比起 stemming，lemmatization 能更加精準的判斷原型字詞。

2. 我在網路上看了計算 Perplexity 的文章，但都有兩個說法，我一直不知道要用哪一個，一個是 $\frac{1}{\mathrm{M}} * \sum_{x=1}^{M} \log_2 \mathrm{P(x)}$，另一個是 $\sum_{x=1}^{M} P(x) * \log_2 \mathrm{P(x)}$，後來查了很多資料才決定使用前者。

3. 一開始很猶豫要選哪幾個 feature 出來判斷，因為不知道怎麼樣才會讓 model 更加精準，後來想到可以將 feature 出現的 frequency 從大的開始挑，這樣便不會因為挑到很少出現的 feature，就讓很多值都是 0，造成無法準確判斷。

4. 當我好不容易打完 code，要在 cmd 上執行的時候，他一直出現以下的錯誤訊息，ModuleNotFoundError: No module named 'pandas'，但我已經確定過我有安裝 pandas，也上網找了很多資料，試了很多方法，但都還是沒有用，弄這個弄了五個多小時，最後想說來更新所有的 packages 好了($ pip update --all)，結果就突然可以正常執行了。

5. 一開始我不知道要利用 GPU 或是在 colab 上面執行程式，我是用 CPU 在跑 run.sh，跑了 30 個小時，正當我在想要不要跑第二次的時候，我看到助教在討論區說可以利用 GPU 或是在 colab 上面執行程式，我就決定去安裝一些 package，讓我的程式可以利用 GPU 來執行，雖然過程很繁瑣，要安裝很多東西，但為了不想再跑 30 個小時，我就去安裝了一些需要的東西，沒想到安裝了兩個小時，好不容易執行完每個步驟，等到要跑程式的時候，它卻說我沒有安裝，這時的我差不多已經束手無策了，只剩下最後一個辦法—colab，沒想到 colab 很容易就上手，最後大概也才花兩小時就跑完了，這次作業的另一個收穫就是學會如何自己從頭開始使用 colab。

6. 因為本次作業要測量好幾次，再加上前面有很多次是失敗的，所以我遇到了 GPU 的上限，而且甚至是在跑到一半的時候，還好我有三個 Google 帳號，才得以讓我在 deadline 前完成作業並成功繳交。