Conversations in TV shows

Team name: NTU_r06921037_陳冠宇打球_場你站_你當球

Members: r06921037 許哲瑋　r06942063 楊惟喻　r06942010 蘇建翰

1. Work division (1)

Main model: 許哲瑋

Second model: 楊惟喻

Thrid model: 蘇建翰

Report: 許哲瑋、楊惟喻、蘇建翰

2. Preprocessing/Feature Engineering (3)

A. Dictionary Making

原始一共有五個 training data，將這五個 training data 利用 jieba 分詞後，使用 gensim 的 word2vec 套件做好 word embedding。接著利用 collection 裡面的 Counter 將最常出現的(10000~25000)字給取出來當作字典儲存。因此我們便擁有了 {word:embedding}的 dictionary 可以做轉換。

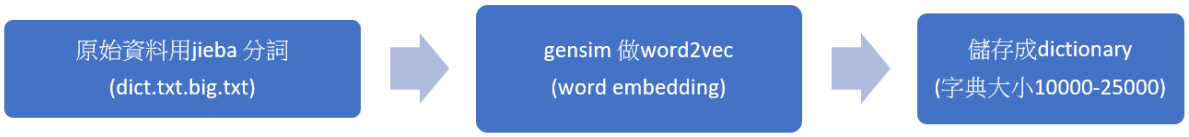


Fig.1 Dictionary Making Flow

B. Stopword Making

利用 https://goo.gl/mWGzzf 這個 github 上面所提供的 stopword 做參考修改，將一些出現頻率較高的字(你們、他們、我們、的、也)或是標點符號(@ $ * % = -)做成另外一個參照的字典，在做 sentence embedding 的時候我們便可以去掉這些詞，對於字句的判斷相關與否會較有幫助。

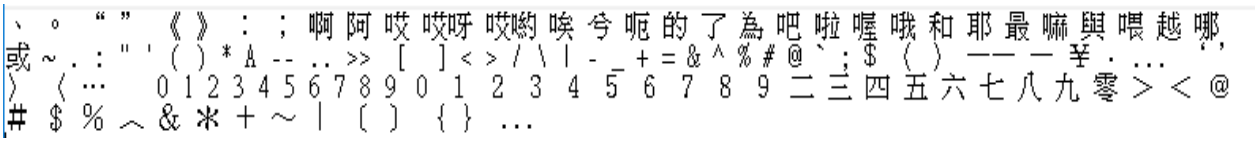


Fig.2 Stopwords

C. Sentence Combinition (Model 2)

先將 training data 的檔案 1~5 合併起來，經過結巴分詞和 gensim.Word2Vec。因為對話中句子有些比較短，所以將每 2~4 句合併成一句(model 2 是兩句合成一句，model 3 的細節在下一段)。將前後兩個句子做為一組，標示 label 為 1。另外，label 0 的 data 是用 random 從原本的 data 隨機選一個句子，然後從那個句子的後三個句子當 input 2。如下表格:

Input 1	Input 2	Label
Sentence 1	Sentence 2	1
Sentence 2	Sentence 3	1
…	…	…
Sentence i	Sentence i+3	0
Sentence k	Sentence k+3	0
…	…	…

Table 1

原本 label 1 和 label 0 的數量比例是 1:5，是依照 testing data 選擇題 6 選 1，後來發現不同的 label 0 的資料量可以增加正確率，改成 1:2 的成績比 1:5 和 1:1 都好。最後再亂數排列 training data 的順序，讓 label 1 和 label 0 的資料交錯。

D. Sentence Combinition (Model 3)

處理資料的過程中，由於 testing data 僅可確認選項為一句句子，題目由幾句句子組成則未知，因此先將 provided data 依兩句一組、三句一組、四句一組以及五句一組各製作出一組 training data，其中每一組除最後一句以外作為題目，最後一句作為答案。

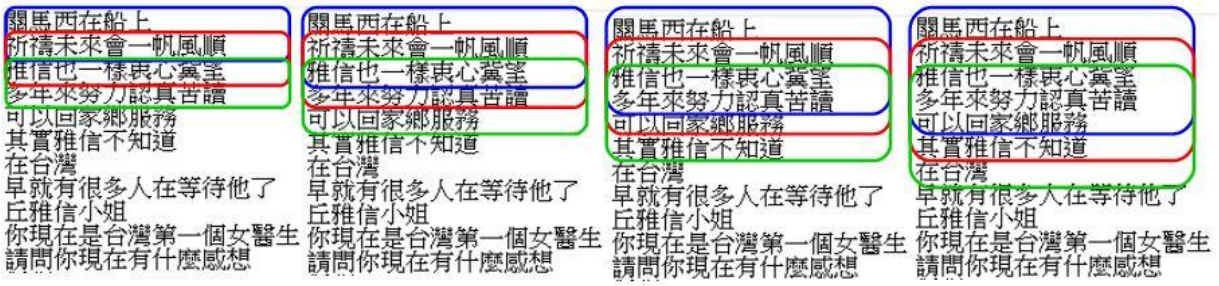


Fig.3 Sentence processing

將所得的所有題目與答案中的每一句都接上句首符號 BOS 以及句尾符號 EOS，以四句一組(前三題目的四答案)為例，

問題: 祈禱未來會一帆風順 雅信也一樣衷心冀望 今年來努力認真苦讀

→BOS 祈禱未來會一帆風順 EOSBOS 雅信也一樣衷心冀望 EOSBOS 今年來努力認真苦讀 EOS

答案: 可以回鄉服務

→BOS 可以回鄉服務 EOS

將所得到的句組經過 jieba.cut 切成 n 個詞組以及經由 size 為 dim 的 word2vec 轉換成 shape 為(n,dim)的 array 後，給定最大的長度 L，對於 n 小於 L 的 array 皆 padding 上內容為 0 的 vector 使其長度等於 L。

3. Model Description (At least two different models) (7)

A. main model: word to vector : (Kaggle Max 0.49525)

a. import module

pickle, csv, jieba, pandas, multiprocessing,
gensim.models.word2vec,
numpy.linalg.norm,
collections.Counter

b. code flow

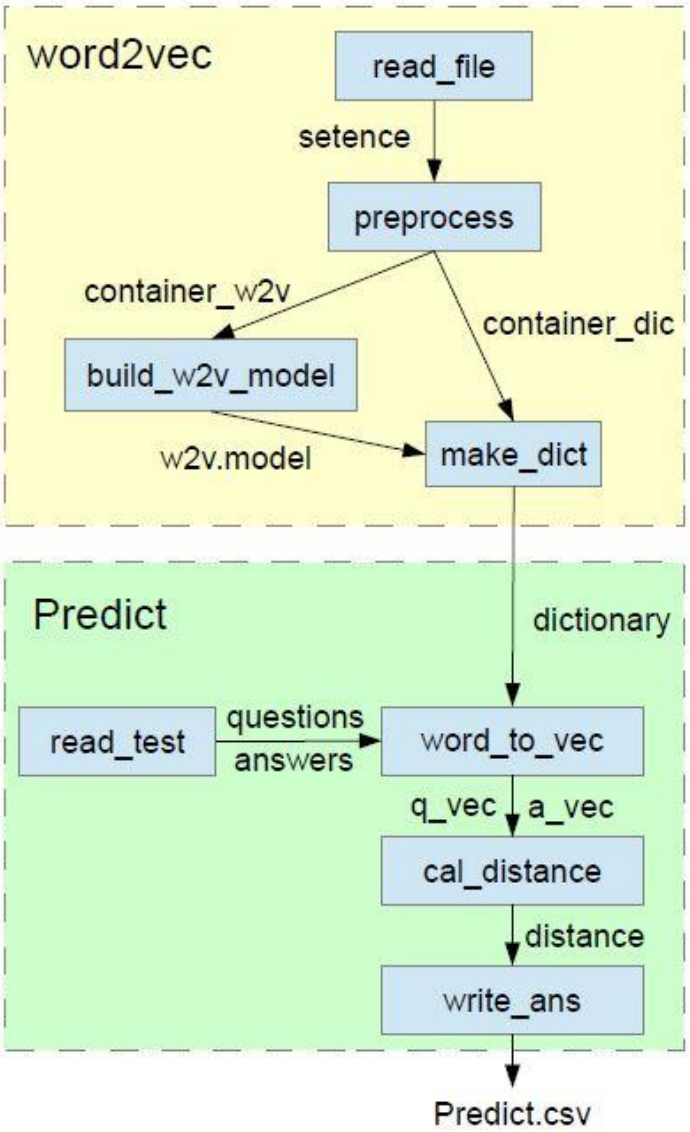


Fig.4 Code flow

c. jieba

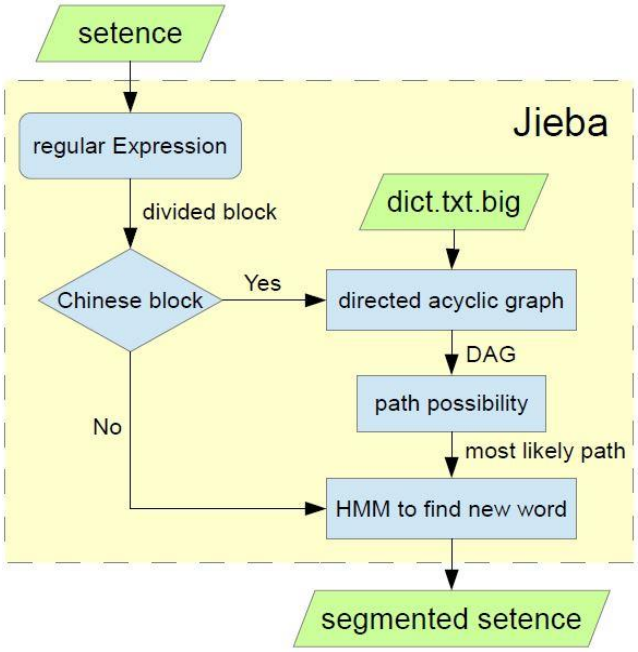


Fig.5 jeiba

(1) 粗分:

透過 regular expression 先將句組粗分成很多個區塊，將單純包含中文的區塊拆分出來

(2) 建構可能的詞組 map

透過輸入含詞以及詞頻的字典，對輸入的中文區塊製作 directed acyclic graph，key 為單字的 index，value 為該單字延伸可以成詞的第幾個 index 例如:

「農曆新年有什麼計畫嗎」

→{0: [0,1,3], 1:[1], 2: [2,3], 3:[3], 4:[4], 5:[5,6], 6:[6], 7:[7,8], 8:[8], 9:[9]}

這個方法相當依賴輸入的字典的適用與否，因此字典選擇相當重要。

(3) 計算路徑機率:

將得到的 DAG 搭配字典中的詞頻，計算所有可能路徑出現的機率進而找出最有可能的組合

(4)發現新詞:

利用 HMM，在所得到擁有最大機率的路徑中尋找有無新詞出現的可能性。

d. similarity

將 question 與 options 先經由 jieba.cut 轉成詞組之後透過先前製作的 dictionary 轉成大小為(詞組數量, embedding_size)的 vector array，接著對

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$$

vector array 取平均以得到大小同為 embedding_size 的 average vector。比對相似度的方式是將每一題問題的 average vector 以及六個選項中每個選項的 average 去計算 cosine similarity，選擇具有最大的 similarity 的選項作為答案。

B. second model: Retrieval model

a. Retrieval model

將兩個 input sentence 分別經過兩個 embedding layer，之後再經過兩個 bidirectional GRU，拆開兩個句子分別 train 的原因是參考報告組別的做法。之後將兩組句子 dot 合起來，最後用 softmax output 一個值介於 0~1。Model 中的參數由於時間關係，沒有做太多調整，只有比較兩個部分。第一個是 Embedding size，比較 64 和 512，發現 64 比較好；另外是 GRU layer，一開始是用 300，後來改成 100 後準確率有改善。Model 結構如下圖：

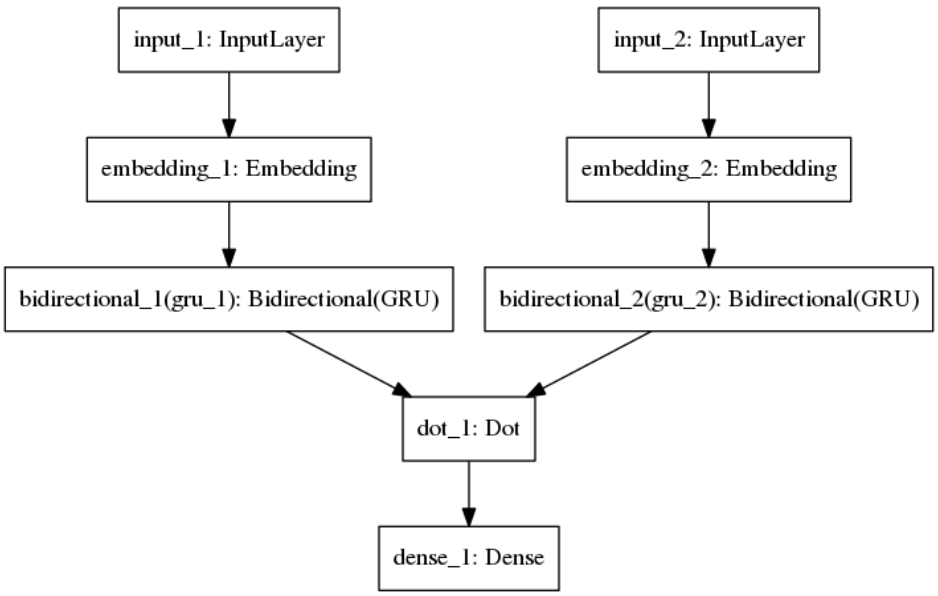


Fig.6 Retrieval model

b. testing

testing 的部分，去掉前面對話人物(A: B: …等)，將抽出對話的句子合併成一句做為 input 1，將選項的句子一個一個做為 input 2 放進 model 中預測，挑選 output 最大的選項。另外參考報告組別的做法，如果發現選出來的句子在選項中另外有相同的句子，則避免選取與最後一個說話者相同的人物的選項。例如:

題目編號	問題	選項
1210	A:媽 我真沒想到 妳讓我過得這麼痛苦 是為彌補自己的遺憾 A:好 我去找博仁說清楚 告訴她 我已經是有志的人了	A:兩千塊 B:兩千塊 A:中午 B:中午 A:妳敢 B:妳敢

選項中的文字兩兩相同，如果答案是"妳敢"，根據內建排序的功能會選出比較前面的選項，也就是"A:妳敢"，但如果避免與最後說話者同一人的話，則會選到"B:妳敢"這個選項，看起來比較合理。

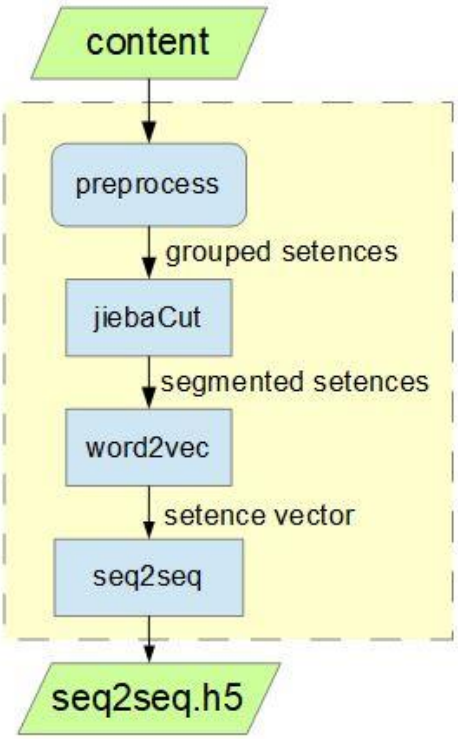
C. third model: sequence to sequence

a. import module:

os, sys, sklearn, jieba,
gensim.models.word2vec
seq2seq.models.Seq2Seq
keras

b. code flow

Fig.7 seq2seq model code flow



c. Seq2seq

Seq2seq 的部分採用的是 farizrahman4u 建立的模組，連結如下：
https://github.com/farizrahman4u/seq2seq

4. Experiments and Discussion (8)

三個 model 在 kaggle 上的 public score 分別是：

	Model 1	Model 2	Model 3
Kaggle Score	**0.49525**	**0.33754**	**0.37944**

其中 model 1 的是 ensemble 之後的結果。

A. Stopword 的效果:

在這個 project 裡我們嘗試過把原本連結上的 stopwords 全部拿進來使用，卻發現一味的使用所謂"公認的"stopwords 似乎對於結果反而有害無益。因此我們改用 gensim 裡的 word2vec，試著從原始資料裡面 train 出我們所想要的 stopwords(將 training 過程中出現頻率最高的那些字詞抓取出來)，但是調整了幾次仍然發現結果不如預期，kaggle 上面的分數反而下降了 2~3 分。在多次地微調後，把 stopword 的內容刪減到現在的樣子，分數大概可以進步 0.2~0.5 分。

原句	原本 stop words	新 stop words
關馬西在船上	'關', '馬西', '船上'	'關', '馬西', **'在'**, '船上'
祈禱未來會一帆風順	'祈禱', '未來', '一帆風順'	'祈禱', '未來', **'會'**, '一帆風順'
雅信也一樣衷心冀望	'雅信', '衷心', '冀望'	'雅信', **'也'**, **'一樣'**, '衷心', '冀望'
可以回家鄉服務	'回家', '鄉', '服務'	**'可以'**, '回家', '鄉', '服務'
其實雅信不知道	'雅信', '知道'	**'其實'**, '雅信', **'不'**, '知道'

Table.2 Compare different stopwords dictionary

B. Cosine Similarity V.S. Euclidean Distance

第一個 model 的方式我們是使用 word to vector 的方式把字全部變成高維向量，接著嘗試著使用每一個句子裡面單詞的向量來取得一個最能夠代表整個句子的向量。最後試著把問題的向量和所有答案選項的向量一起考慮，找出一個和原本問句最相似的選項當作答案。求兩個向量相似度的方式可以使用 Euclidean Distance 或是用向量之間做 Cosine Similarity。在嘗試後發現 Cosine Similarity 的表現會比 Euclidean

Distance 好，可能是因為在高維的空間中(我們 embedding 成 1024 維)，計算點與點之間的距離，會發現其實每個點與點之間都"幾乎一樣遠"，效果會比較差。

```
class gensim.models.word2vec.Word2Vec(sentences=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None,
sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, cbow_mean=1, hashfxn=<built-in function hash>, iter=5,
null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False)
```

C. gensim 的參數調整討論

如上圖，gensim 的 word2vec 參數非常的多，但是在這份作業裡有幾個參數一定要調好，否則結果會偏離非常多，以下列出幾個我們調整比較重要的參數：

(1) size: embedding vector 要用幾維來代表一個單詞，這裡使用 1024。我們曾經嘗試了 64，128，256，512 等維度，發現在這個題目中，其實維度開大一點對於詞與詞之間的分離和最後的預測結果相對是比較好的。

(2) window: 看到一個句子裡面的某一個詞，要預測其他詞的最大距離，這裡使用 5。曾經最大使用到 10，最小使用到 3，但是發現太大或太小效果都不好，因此最後決定使用 5。

(3) min_count: 當出現的次數少於 min_count 的時候便直接忽略它。這裡直接使用 1，過濾的部份我們使用 stopword 或是改良的投票權重方式達到。

(4) workers:可以開的多執行緒數目。我們利用 multiprocessing 這個套件抓取電腦最高執行緒。因為我們 embedding 的維度很大，所以如果沒有開多一點執行緒的話會跑非常久。

(5) iter: training 總共要跑幾個 epoch，預設是跑 5 個 epoch。這裡我們設定為 30 個 epoch。

(6) sg: 是否使用 skip-gram，這裡設為 1，skip-gram 效果明顯較好。

假如有個句子: *I love green eggs and ham.*

若是使用 skip-gram(給定字預測上下文)，window size 為 3，(predict, input word) 的組合如下

([I, green], love)
([love, eggs], green)
([green, and], eggs)
…

若是使用 CBOW(給定上下文，預測字)，window size 為 3，(predict, input word) 的組合如下

(love, [I, green])
(green, [love, eggs])
(eggs, [green, and])

…
兩者恰好是互補的作用，但在這個例子裡，使用 skip-gram 的效果會比 CBOW 來的好。

D. 平均真的能夠代表全部?

在主要的模型裡面，我們嘗試用把一個句子所有的字詞分離出來，分別表示成高維向量，最後用這些分離出來的單詞向量來表示整個句子(word-embedding -> sentence embedding)。我們嘗試了兩種方式:

(1)取所有詞向量的平均當作句子向量:

直接把所有詞向量做平均來加總做法非常直觀，但是實驗出來的效果並非最好。句子裡面有太多的冗詞贅字，雖然可以試著使用 stopword 的方式把這些詞給濾掉，但是需要耗費極大的心思去做微調，在我們多次的試驗裡面效果有限，預測的分數很難繼續突破。

(2)將所有詞向量的平均做權重加總當作句子向量:

相較於直接使用平均，將句子中的單詞分別取權重相加的方式分數的進步會來的比較多。將一些較常出現的字詞給予比較低的權重，效果可以做到如同 stopword 的過濾詞功能。這個方式參考"A Simple but Tough-to-Beat Baseline for Sentence Embeddings (Sanjeev Arora, Yingyu Liang, Tengyu Ma, 2017"這篇 Paper 所展示的，公式如下

$$v_s \leftarrow \frac{1}{|s|}\sum_{w\in s}\frac{a}{a+p(w)}v_w$$

其中

|s|: 句子所包含詞的數目

p(w): w 這個詞出現在整個文本裡的機率

v(w): w 這個詞的詞向量

α: 權重調整參數

Paper 建議把α設在$10^{-3}\sim10^{-4}$區間，但實驗做出來大概在 0.005 效果會是比較好的。