Final Project
Conversation in TV show

一、Team Name and Members

隊名：NTU_b04902092_CGSS

組員：

董書博(b04902003)
鐘偉傑(b04902082)
張均銘(b04902092)
許耀文(r06921075)

二、Work Division

董書博(b04902003) Word2Vec改 實作、參數調整
鐘偉傑(b04902082) Word2Vec參數調整、TfIDf實作
張均銘(b04902092) Word2Vec參數調整、Doc2Vec實作
許耀文(r06921075)　Word2Vec 參數調整測試

三、Model Description

- **Word2Vec model**

1. 模型介紹：

透過助教提供的hint還有後續的手把手教學，先將所有句子做word embedding，再去比較問題和選項的vector之間的相似程度。

2. Preprocessing：
   - jieba分詞：使用助教推薦的繁體字典dict.txt.big。
   - word embedding：使用gensim的word2vec。
     調整參數：
     min_count = 5
     sg = 1 (skip gram)

3. Sentence similarity：

比較問題和選項中的每個詞之間的cosine similarity(使用gensim套件中的model.similarity)，全部加總後數值最大的選項當作正確答案。當遇到字典中未出現的詞時(OOV)，將cosine similarity設為0。

- **Doc2Vec model**

1. 模型介紹：

Doc2vec是Cornell大學的論文**[註1]**中提及的一種做法，簡單而言他就是sentence embedding與word embedding結合的用法，因此input sentence除了獲得個別word的向量，也會獲得整句sentence的向量並且算兩句sentence的相似性。predict的模式和原本word2vec的不同在於下一個單詞的vector除了由前面幾個字決定外，句子的vector也會參與運算。相對於word2vec的CBOW,Skipgram，Doc2vec也有兩種模式(在Discussion地方詳述)。

2. Preprocessing：
使用jieba將traindata斷詞後，存成一個split.txt，split.txt裡面每一行代表一句每一句內的詞用空白隔開，故split.txt的內容如下:
關馬 西 在 船 上
祈禱 未來 會 一帆風順
接著使用gensim doc2vec裡API的TaggedLineDocument讀檔並開始model training。

3. Cosine Similarity：**[註2]**
相較於word2vec的model，此model在做句子相似度判斷的時候不用針對每個字來做Similarity，而是可以直接input整個句子的vector直接做cosine similarity但為了處理OOV的問題，因此自己手刻cosine similarity的函式，將每個問句及對應的六個選項算出infer_vector後再拿去做similarity運算。

4. Sentence similarity：
算出每個question與六個選項分別的相似度(cosine similarity)後取絕對值並取最大值的那個選項。

- **TFIDF model**

1. 模型介紹：
tf-idf 是一種用於文字挖掘的常用加權技術，用於評估一個詞彙對於語料庫中，其中一份檔案的重要程度。字詞的重要性隨著他在檔案中的頻率增高而增高，但同時會隨著他在整個語料庫中的頻率增高而降低。是許多word_to_vector方法中的其中一種。

2. Preprocessing:
先使用jieba將training data的句子斷詞，再將每個句子的斷詞用空白隔開。
例如：絕不可以跟他合作談生意 -> 絕不 可以 跟 他 合作 談 生意

3. Similarity:
由於sklearn的tfidf沒有內建相關的函式，因此使用手刻的cosine similarity來查看問題與回答之間的相似度。

- **Word2vec model 改**

1. 模型介紹：
在經過1/19的報告分享後，我們採納了一些高手們的作法後自己整理出來的模型。

2. Preprocessing：
   - HanziConv：繁體字轉簡體字
   - jieba 分詞：使用jieba預設的字典(簡體)
   - 忽略stop words
   - word embedding：使用gensim的Word2vec

3. Sentence similarity：
1. 將問題和選項的所有詞向量分別加總起來再取平均，最後比較兩者之間的cosine similarity，相似度較高的選項即是正確答案。

2. 根據[註6]中提到的，賦予每個詞向量一個加權，所以一句話的向量並不單純只是句子中所有詞向量的平均。最後同樣用cosine similarity比較相似度。

四、Experiments and Discussion

- **Word2vec model**

1.簡介：

這個模型的概念很簡單，word embedding也是我們在前幾次作業就有利用到的技術，所以這裡會比較不同的參數微調對最終結果的影響。

2.參數調整：

(1)threshold

在計算cosine similarity的時候，要決定相似度大於多少的時候才將這個值考慮進去，相似度低的值我們選擇忽略。

準確率比較：

thershold	kaggle public score
>0.15	0.44347
>0.3	0.44822
>0.6	0.38972
>0.3 and <1	0.44031

因為上傳次數有限所以沒有取得太多數據，但仍能從上表發現threshold太高或太低都會使結果變差。另一點值得注意的是上表的最後一列，cosine similarity = 1的意思是問題和選項中出現完全相同的詞彙，之所以要將完全相同的詞彙避開的原因是因為有些題目如下：

題目：

A:這個流氓 三番兩次找我麻煩

A:而且還威脅我 如果不還他五千萬 就會來我家潑油漆

選項：

A:真不曉得該拿他怎麼辦　B:錯字也太多了吧　B:可以買一支新手機嗎

B:最近流感盛行 小心不要感冒　A:附近最近流氓很多 大家要小心一點

B:家裡粉刷了粉紅色的油漆 好漂亮

依照回答合理性來說，應該要選擇第一個選項：A:真不曉得該拿他怎麼辦，但因為這個word2vec模型是比較詞之間的相似度，所以當他發現倒數兩個選項分別出現流氓和油漆時，cosine similarity=1，最後的加總就容易選到錯誤的答案。

然而從測試結果來看，成績並沒有往上提升，反而稍微往下降。我們推測是因為除了像上述這種例子是因為關鍵字：流氓、油漆相同才導致的錯誤，一般常見的名詞(你我他)、連接詞(而且、然後)...等等，這些詞彙對準確率的影響是不能忽略的，當我們忽略題目和選項同時出現的這些詞彙時，反而會使正確率下降。

(2)字體轉換

一開始看助教的投影片有發現到jieba對於繁體字的分詞能力較差，所以才需要特別改用繁體字典dict.txt.big，於是我們決定反過來想，先把所有的資料都轉成簡體字再用jieba預設的簡體分詞字典，看能否獲得更好的結果。

hanziconv簡繁轉換：使用HanziConv.toSimplified()將繁體字轉成簡體字。

準確率比較：

模型	kaggle public score
繁體分詞	0.44703
簡體分詞	0.45810

(3)vector dimension

改變word2vec生成出的向量維度來比較對準確率的影響。

因為每次改變維度時其他的變數也會有所變動，所以沒有辦法純粹比較不同維度之間的差異。但從我們的實驗結果來看，維度的影響並沒有很顯著，從64維到8192維都有嘗試過，但影響準確率最多的往往是其他的變數。

(4)window size

word2vec中的window參數代表一個詞最遠能夠影響到的詞的距離，改變這個參數來比較準確率的變化。

準確率比較：

window size	kaggle public score
3	0.44308
5	0.45810
7	0.45375

最初一直使用預設的window=5，後來分別試了較小以及較大的window size發現結果並沒有比較好於是便停止測試了。

(5)OOV handling

我們最初的模型在遇到OOV時會選擇忽略(cosine similarity = 0)，為了看出OOV對最後結果的影響有多大，我們嘗試給予OOV負數的權重(penalty)以表達OOV的不重要性。

準確率比較：

OOV cosine similarity	kaggle public score
0	0.45810
-1	0.43754

我們認為造成這樣的結果可能的原因是：

1. 權重過大
2. <min_count 的詞被當作OOV

這個模型的min_count是設定為5，所以就代表training data中出現次數小於五次的會被忽略，如果這些詞再次出現在testing data的話，就會被判斷成

OOV，但這些詞彙可能是對句子的表達有幫助的，如果又以負的權重表示的話反而會造成反效果。

- **Doc2vec model**

1.簡介：

Doc2vec是除了word embedding外也將整個句子下去做training得到整個句子的vector，就如同在model description內講得一樣。

2.種類：**[註3]**

(1)dm(distributed Memory):

將一段文章或句子裡的詞的vector與整個sentence的vector做平均後預測下一個字的vector。

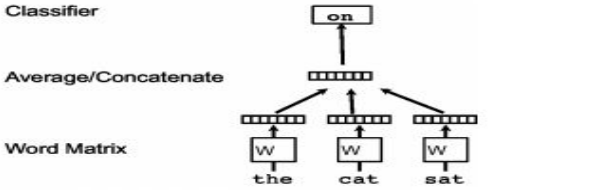
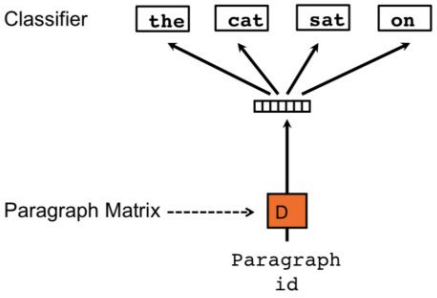


Figure 1. A framework for learning word vectors. Context o three words ("the," "cat," and "sat") is used to predict the fourtl word ("on"). The input words are mapped to columns of the ma trix W to predict the output word.

(2)dbow(distributed bag of word):

train一個段落或句子來預測句子內部分的一段文字裡面的詞，一段文字的大小取決於window的設定。



3.準確率：

模型	kaggle public score
DM	0.20711
DBOW	0.22173

4.優點：

(1)OOV handling

這個model最大的特色在於他能處理OOV的問題，Doc2vec的API裡有一個function叫 infer_vector，當輸入一句話並沒出現在training data中infer_vector可以用現存的model來計算出此OOV的vector，再根據這個vector值去做similarity，因此OOV發生時並不會因此就跳過算該詞的similarity。

(2)sentence similarity

助教的sample code再跑testing data時會有一個問題，因為畢竟是針對每個詞作similarity，因此判斷的是問題與選項的詞的相似度，並沒有考慮整個句子的意義和句子中詞的順序，因此常常造成model會偏向選與問句重複字詞多的選項，在Doc2Vec的作法裡，有部分是以整個句子來train，因此較不會因為上述問題造成判斷錯誤。

5.缺點：

(1)training data不足、與testing data有些差異

在拿到Training data時其實想了很久要怎麼用，因為他就是句子，沒有label就這個題目而言，似乎也沒有辦法label，因此training方面只能用gensim這種淺層的NN來做word embedding，似乎很難套上RNN，自己覺得用RNN train效果應該會不錯，另外就是testing data看起來跟training data差異蠻大。自己在測的時候發現OOV問題有點太嚴重，很多詞是training data沒有的。即使用infer_vector處理也是會造成一定的誤差。

(2)斷詞問題

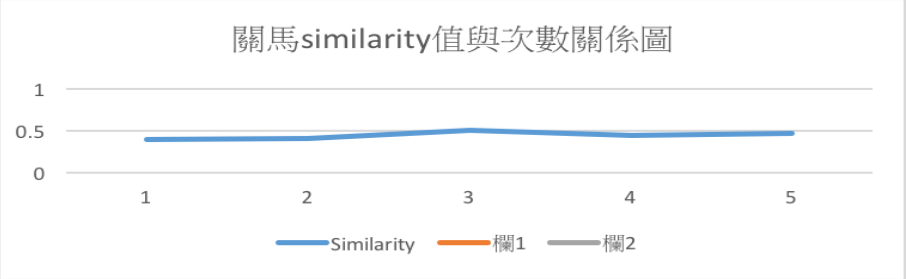
因為Doc2vec在網路上文件幾乎都是以英文單字為主拿來做情感分析，雖然有看到有些文件用中文字來train，而且在similarity上似乎也有不錯的情況但還要再研究一下怎麼辦到的，而我認為的問題就是中文詞的訓練的問題，一來如結巴斷詞遇到人名幾乎都會有錯誤的斷詞，如['關馬','西']這種，這會造成在training很大上的差異，二來語法上中文與英文的詞的性質差蠻多所以詞之間的關係可能在使用上會出問題。

6.實驗發現：

(1)Doc2vec的infer_vector

infer_vector這個函式設計理念是因為句子相似度的判斷很常有OOV的現象但是再測試infer_vector的效能時發現，對同一個字使用infer_vector每次的vector都會有不同的值，導致光是對同樣的字做similarity的判斷每次出來結果都不一樣，網路上論壇說infer_vector和word embedding這些vector計算方式都有random的因素，因此都會使每次vector產生的值不一樣。

以 "關馬"這個詞為例(training data中的第一個詞)取原本model train出來的vector與呼叫不同次數的infer_vector('關馬')產生的vector的similarity的圖表



(2)DM效果比DBOW差

雖然兩者在結果上表現不理想，但DM得準確率比DBOW低2%，然而在論文中提及DM準確率照理來說要比DBOW高**[註4]**，因DBOW只有在剛開始給text一個initialize vector，接著就給text內的word一個vector，過程中沒有牽扯到神經網路、text的vector也沒有update，而DM有，所以照理來說DM應該是要比較高的。解釋就是運氣問題吧，剛好initialize vector蠻符合text的。

- **TFIDF model**

1. 遇到的困難

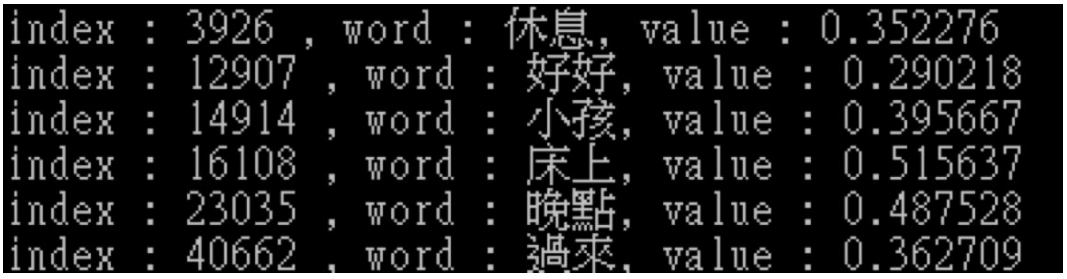
利用tfidf，我們能成功的將中文詞彙數據化成vector，但使用cosine similarity時卻無法從問句與答句之間得到任何關聯。之前不知道為什麼會這樣，直到看見一個網頁**[註5]**，才意識到這個model的不足。

2. 可行性不足:

由於tfidf僅僅是經由查看指定詞彙的存在與否來進行數據化，所以當問句和答句之間若無相關詞彙，便無法經由cosine similarity 來獲得任何可用的數據，而就算剛好有相關詞彙，也無法因為這樣來辨別他們是適當的問句與答句。

例如：

問句：你躺在床上好好休息 晚點護士會抱小孩過來 ， 使用tfidf轉化成vector後的相關數據為



答句1 : 給你餵奶 ， 使用tfidf轉化後的相關數據為

答句2 : 做晚餐給你吃 ， 轉化為

答句 3 : 帶你出去玩， 轉化後皆為0

答句 4 : 陪你過假日 ， 轉化後

答句 5 : 上網打遊戲 ， 轉化後

答句 6: 出國去旅行 ， 轉化後

這題的答案應為答句1，但卻無法得到任何相關性，以上皆是經由tfidf轉化之後所得到非0的值，應證了上述的說法。

- **Word2vec model 改**

1. 訓練句子的長度：

最初我們在訓練word embedding的模型時，都是直接以training data中的一行當作輸入，但多半的時候相鄰的句子其實是有關係的，可能是同一句話的前後句關係，或是一問一答的關係，所以我們決定將相鄰兩句話併在一起當作是word embedding的輸入，以增加詞彙之間的關係。

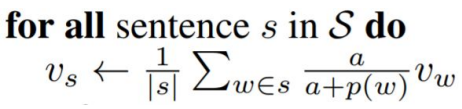
2. Setence embedding：

這個模型比較兩句話的相似度時，是直接將一句話中所有的詞向量平均再比較cosine similarity，這個方法的優點在於不會再像原先的模型偏重在詞之間的關係，所以遇到問題和選項出現相同的詞彙時，比較不會被誤導。

準確率比較：

模型	kaggle public score
word2vec model	0.46284
word2vec model 改	0.50671

另外一個方法也是利用詞向量的平均來表達句子向量，但比較不同的是這個方法會根據每個詞出現的頻率來給予詞向量不同的加權[註6]。右圖中的alpha是一個常數(我們設定為0.01)，p(w)是該詞出現在training data中的頻率。



準確率比較：

sentence embedding	kaggle public score
平均	0.50671
加權後平均	0.52252

3. 更高的準確率：

word2vec model中唯一會有隨機性的部分是訓練詞的向量模型的時候，所以我們不斷微調word2vec的參數來取得更好的準確度，最後以以下的參數訓練出一個較好的模型。

size = 176
min_count = 3
window = 9
sg = 1
iter = 9

準確率比較：

model	kaggle public score
微調前	0.52252
微調後	0.53715

- **遇到的困難與解決方式**

1.中文字相似度的判斷上，常常會遇到不再training data中的字，助教的code是以exception自動忽略這個部分的判定，但有些詞語可能是句子中的關鍵字就因此被忽略。

解法: 使用Doc2Vec的infer_vector，來給予OOV的句子、詞語一個詞向量，在以此向量來做similarity，後來使用numpy的random unifrom vector的方式給OOV一個向量來做similarity

2.word embedding的做法只判斷問句和選項間字詞相似度，沒有對整體句子相似度做判斷。

解法: 改用Doc2Vec這個model來做句子間的similarity判斷

3.問題與選項可能是一組對話，由不同人，但是在做similarity時沒考慮到這點

解法: 紀錄每個選項說話的人，如果選項說話的人和問題問話的人同個人，就把那句話的similarity比重降低(因為通常不太可能自言自語)

4. 繁中字典可能因字詞數較少導致斷句問題

解法:載下簡中字典後使用Opencc將字典轉成繁中字典並與原本繁中字典合併成一個大字典，希望字詞較多能使斷句更精確，但效果似乎沒有比較好，估計是簡中有很多是中國用語，台灣沒有人那樣用，所以導致斷詞反而可能有誤。

5. training data中將一句話拆成許多小句子，因此train起來每個句子其實並沒有完整的中文意思。

解法:將training data 2~3據串再一起在train，如1_train的前幾句:

關馬西在船上
祈禱未來會一帆風順
雅信也一樣衷心冀望

合併成一句話:
關馬西在船上 祈禱未來會一帆風順 雅信也一樣衷心冀望
來train，實際上準確率確實也比較高。

五、Reference

[註1] Doc2vec essay
[註2] Doc2vec cosine similarity
[註3] DBOW and DM
[註4] DM better than DBOW
[註5] Feature of TF-IDF
[註6] A SIMPLE BUT TOUGH-TO-BEAT BASELINE FOR SENTENCE EMBEDDINGS