

# Final Project Report

Group 16

r09922176 鄒咏霖

r09922095 陳昕璘

d09922015 黃翔偉

## A. Approach

本次的Final project, 我們有採用四種方法進行處理, 分別是BM25、Doc2Vec、TF-IDF, Word2Vec相關方法的資料前處理、模型建置及預測方法說明如下：

### 1. BM25 (鄒咏霖)

#### 1.1 資料前處理

首先, 第一步對dataset的處裡就是把那些xml的tag去除掉, 只留下內容, 而這個部分使用pandas套件中的htmlparser就可以很輕鬆地完成。

第二步, 因為BM25是用計算frequency去做進一步的matching的, 所以要將每個queries、documents都處理成list of words(string)。

再來, 因為BM25是用frequency來做matching的, 所以必須要把query中的一些common word過濾掉(e.g. the, a, that, were, is, etc.), 不然大部分matching score高的document都只是因為有很多common words而已。另外, 在BM25中, document的長度也會被用來做normalization(document越長, matching的分數會越低), 所以documents中的common words也要過濾掉, 不然含有同樣多的key word的兩個document, 可能因為某個document長度較長(i.e. 用較多的common words), 導致matching score較低。所以我們有自訂了一個要過濾掉的word list(在DataLoader.py中的excludes這個list), 在資料前處理的時候一併將每個documents、queries中的common word用python內建函式filter()過濾掉。

#### 1.2 模型建置

在模型的建置方面, 我們選擇了直接使用gensim套件所提供的BM25這個class, 使用方式為將document collection以list of list of string的資料結構傳進BM25的constructor去做初始化(term frequency、IDF的計算)。

#### 1.3 預測

而得到各個document與query之間的matching score的方法為去call BM25.getscore()這個method並將單一query以list of string的型態當作parameter傳入此method這樣就會回傳各個document與該query之間的matching score(以list of float

的型態)。最後就是將得到的結果依照值得大小對index做排序(使用numpy.argsort())，後取前50個index相對應的file name當作最後對該query的document retrieval ranking。

## 1.4 效能

效能的部分，在經過多次的filter word set修改後，最後還是只能在MAP@50 metric上得到1x%左右的成績，勉強通過public及private的weak baseline。但我認為在 data preprocessing的部分，如果花更多時間去濾掉更多無用的word和特殊符號的話，BM25的表現或許能再提高一些。

# 2. Doc2Vec(黃翔偉)

## 2.1 資料前處理

- a. 首先使用套件bs4的html parser移除所有XML檔案內的tag。
- b. 針對萃取出來的文字，先進行詞型還原，將所有詞彙還原回原始型態，例如將 going還原回go，以統一所有詞彙。
- c. 然後再針對這些詞彙消除stopword、數字、符號及空白等資訊。
- d. 最後將處理後的結果儲存成CSV檔案，以便後續匯入模型訓練。

## 2.2 模型建置

模型建置的部分，這邊採用gensim套件的doc2vec模組，將所有文章進行token轉換之後，帶入doc2vec模組進行模型訓練，而模型採用的參數分別為vector\_size=256, window=10, min\_count=15, workers=4, epochs=40，並且採用PV-DM(Distributed Memory Model of paragraphvectors)方式進行訓練，完成訓練後，便將模型儲存以便後續使用。

模型使用的相關參數說明如下：

- a. vector\_size: 是特徵向量的維度，這邊設定256。
- b. window: 預測上下文詞之間的最大距離，這邊設定10。
- c. min\_count: 忽略次數小於此數量的詞，這邊設定15。
- d. workers: 使用多少現成訓練模型，這邊設定4。
- e. epochs: 模型訓練的迭代次數，這邊設定40。

## 2.3 預測

- a. 先導入預先訓練好的Doc2Vec模型。
- b. 將doc跟test\_query資料夾內的檔案逐一進行cos的相似度計算。
- c. 取得所有cos相似度結果後，再將結果逐一排序，並取前50名的文章作為預測結果。

### 3. TF-IDF(黃翔偉)

此處主要以是助教於deadline以後提供的sample code 進行說明

#### 3.1 資料前處理

- a. 首先使用套件bs4的html parser移除所有XML檔案內的tag。
- b. 將文章內容去除空白及換行等符號，並且移除stopword。
- c. 將所有處理過的資料另存成新檔，等待後續處理。

#### 3.2 模型建置

模型建置的部分採用sklearn套件的TfidfVectorizer模組，此處設定的參數主要有3個，分別是max\_df、min\_df、sublinear\_tf，經多次調整後，最後採用的參數數值為max\_df=0.75、min\_df=0、sublinear\_tf=True，相關參數說明如下。

- a. max\_df: 如果詞彙頻率超過這個值，就忽略這個詞彙。
- b. min\_df: 如果詞彙頻率低於這個值，就忽略這個詞彙。
- c. sublinear\_tf: 應用線性縮放TF。

#### 3.3 預測

- a. 將doc與test\_query經過前處理後的檔案導入模型，個別計算出TF-IDF值。
- b. 將前面計算得到的TF-IDF值進行cos運算計算其相似度。
- c. 針對所有cos值進行排序並取前50名作為預測結果。

### 4. Word2Vec(陳昕璘)

#### 4.1 資料前處理

1. 將所有document中的xml tag移除，使用BeautifulSoup4套件中的xml parser，可以取得raw text。
2. 接著使用字元替換，將特殊符號(如句號，逗號，引號)以空白代替，以免特殊符號被判定進單詞裡。
3. 使用nltk套件中的stopword list，挑掉文件中特定的詞，提升預測效能。

#### 4.2 模型

我們使用gensim內建的Word2Vec Model做訓練，由於參數和前面的Doc2Vec大同小異，在此就省略介紹。過程中，為了加速訓練，有採用negative sampling。訓練環境是在系統上的工作站，用了24個workers(threads)，單次訓練時間約需7~8小時。

## 4.3 預測

在前述模型被訓練完成後，我們比對所有單一document和單一query中所有單詞組的相似度，並且取平均值，對所有document做相似度的比序，計出最相似的前50筆。然而，此方法預測出來的準確度卻相當差，而且比對所有單詞組的相似度過於耗時。推測可能是preprocessing依然做得不盡理想導致，應該嘗試加入如smoothing的方式調整機率分佈，後來在神經模型這一塊我們就用Doc2Vec作為主要方法。

## B. Comparison of different models

### 1 資料前處理

其實在資料前處理的部分，三個模型使用的方法基本上是一樣的流程，都是先去掉tag後再filter out一些stop words，所以這邊應該不是造成效能有差異的地方。

### 2 參數調整

#### 2.1 TF-IDF

在TF-IDF的部分，主要可以調整的參數有max\_df、min\_df、sublinear\_tf，如果sublinear\_tf=False，普遍預測結果較低，而min\_df如果設定0.4上，則預測結果為0，在相同的前處理及環境下，表現最好的參數組合為max\_df=0.75, min\_df=0, sublinear\_tf=True

#### 2.2 Doc2Vec

在Doc2Vec部分，可以調整的參數相當多，我們主要控制的是vector\_size, min\_count, alpha。標準的vector\_size大小是100，我們有試過調為256，但訓練時間相當昂貴，為了加速，後來也嘗試壓低維度，調整到約50~75。min\_count為最小單詞出現之次數，調大數值可以加速訓練時間，但也有篩掉關鍵字（特定冷門單詞）的風險。alpha則是learning rate，但在我們的模型中，調整alpha並沒有顯著的影響，表現都差不多。整體而言，Doc2Vec的表現並不盡理想，我們認為可能是在前處理還有訓練參數有待改進的地方。

基本上在BM25中可以調整的參數不多，只有K和B兩個可以調，所以可變性不大。

### 3 效能

我們試出來最好的效能是由BM25產出的，這部分有點意外，原本以為現在不管什麼task都應該是由NN的model可以得到更好的結果。最後，總結兩個BM25比NN approach效果更好的原因：第一，可能是因為我們對NN方法需要的前處理方法不熟悉，所以導致embedding的效果很差。第二：可能在這個task上，太多醫療專業術語了，embedding model很學習出words之間的相關性，因此NN的方法效果比BM25還差。

## C. The difficulties encountered

### 1. BM25(鄒咏霖)

最大的困難應該還是在於dataset size蠻大的，所以每次改preprocess的方法後都要等一段時間才能看到結果，所以比較好的方法是在改動code後先用一小部分的data去測試看看code的正確性，再跑在整個dataset上。

### 2. Doc2Vec及TF-IDF(黃翔偉)

實作的過程主要碰到以下幾個困難的問題，分別說明如下：

#### a. 龐大的資料量

本次的dataset為10萬個xml文件檔案，以往並沒有處理過這麼大量資料的經驗，所以對於處理大量資料的效能及時間沒辦法正確評估何種情況為正常或異常。在本次實作的過程中，在處理資料的時間花了太多的時間，平均每次都要花上至少將近兩天的時間，但此時還不確定是否資料量過大所造成的問題。直到實作助教提供的sample檔，才發現處理時間異常的問題。但即便處理資料的時間縮短，也還是需要耗費數小時的時間，才能得到處理後的結果。

#### b. 硬體設備限制

本次硬體設備有使用到32G記憶體的筆記型電腦及桌上型電腦及64G記憶體的桌上型電腦，在初始測試資料處理的過程中，都曾因為資料量過大，導致電腦後續運作異常，其中32G的桌上型電腦不知何原因，執行到一半便自動關機並無法再正常開機。另外，可能因為硬體資源不足，執行過程也會顯示記憶體不足的提示。

#### c. 套件熟悉度及支援度

本次原本希望採用nltk套件的language model模組，但因有許多其他配合的套件不熟悉，花很多時間測試及整合，最後發現部分套件只能支援舊版python(3.6)，最終只能放棄，改採Doc2Vec模型的方式處理，同時研讀相關文件，壓縮到最後的實作及測試時間。

### 3. Word2Vec(陳昕璘)

主要的困難點在於，資料量過於龐大，處理時間非常久。光是預處理就要3~4個小時跑不掉，雖然一開始想說預處理只需要做個一兩次，之後就專注在訓練模型上就好，但後來發現光是預處理就有許多要反覆調整的地方，再來就是在這次任務中神經網路的模型似乎表現並不佳，我認為除了強化參數的fine-tuning之外，在這種特定的domain下做information retrieval，應該可以加入domain-specific的資訊，提升整體表現。

## D. Member's responsibility

如每個模型後括號內的名子所示，我們分工的方法為不同的組員各自負責自己較有興趣的模型。

## E. Source Code

### 1. Enviroment

The description of the environment and how to run the code is in readme.md at the root directory of the github below.

### 2. Source code

<https://github.com/TonyTTTTT/IRIE-final-project>