運用工具 & 資料前處理說明

1. 透過 nltk 的 word.tokenize function，將 Documents 與 Queries 進行斷詞。
2. 運用 snowball_stemmer function 將詞性還原。
3. 使用 nlt.corpus 中 stopwords 篩選掉不帶有資訊的字詞，ex: the, a, and…。

心得

由於這次作業的難度真的是爆炸性上升，光是聽懂老師上課在說什麼就很吃力了。連一開始的 kmeans 分類我就遇到很大的困難了，自己也嘗試做過於 kmeans_collection.py，雖然最後其實沒有用到…，但我有嘗試去做且透過此次作業稍微了解 kmeans。而於 EM 的部分，一開始真的很茫然，到處問資工同學甚至連師大的學生都被我叨擾了，但大家起初也不太理解怎麼做，不過還是很感謝他們的幫助。當然，我有上網自己研究也嘗試自己修改，但由於資料量太大，若使用三維的結構會導致 memoryerror，無論我怎麼修改或上網搜尋都還是會報錯，故這份作業我沒有用到 EM algo，只用最基本 P(w|D)P(w|BG)，以下會述說參數的比例及作法。但到今天我才發現原來有人在 Kaggle 上說使用稀疏矩陣就可解決 memoryerror…，故我會再找時間研究並且嘗試做出來。

P(w|D) P(w|BG) 參數設定- background.py

```
for w in list_of_words:
    w = snowball_stemmer.stem(w)  # 詞性還原
    if w not in stop_words:
        lengthofdoc += 1.0
        doc_dict[w] = doc_dict.get(w, 0.) + 1.0
        document[w] = document.get(w, 0.) + 1.0
for wc in document.keys():
    collection+=wc+":"+str(document[wc]/lengthofdoc)+" "
testfile =  open("collection_docname.txt", "a")
testfile.write(collection+"\n")
```

collection_docname.txt：作法為「每篇文章的字/該文章的長度」，即是 P(w|D)

```
for w in doc_dict.keys():
    # bgm[w] = doc_dict[w]/sum(doc_dict.values())
    bgm[w] = doc_dict[w]/len(doc_dict.keys())
with open('BGLM.txt', 'w') as f:
    for key, values in bgm.items():
        text = str(key)+" "+str(values)
        f.write(str(text)+"\n")
```

BGLM.txt：作法為「該字出現在所有文章次數/corpus length」，即 P(w|BG)，而本模型之 corpus length 為 unique，經過測試該作法所獲之正確率較高。

相似度計算-PLSA_final.py

本作業使用方法為 alpha* P(w|D)+beta* P(w|BG)，其中 alpha 經過測試設為 0.988，beta 為 0.012，由這兩參數所求知準確率會較高，而 P(w|D)和 P(w|BG) 參數由上述預處理的 collection_docname、BGLM.txt 所獲得。