Vector Space Model

資管碩一 M10909112 石家安

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…。
4. 使用 numpy 做陣列操作。而 math、operator 做資料運算 ex:math.log。
5. 使用 sklearn.metrics.pairwise 的 cosine_similarity 計算餘弦定理。

TFIDF 參數設定-Documents

```
for wc in doc_dict:
    # IDF
    self.docidf[wc] = self.docidf.get(wc, 0.0)+1.0
    # TF
    doc_dict[wc] = 1+math.log(doc_dict.get(wc, 0.0),2)
```

```
if w in dicTemp:
    docTFIDF.append(
        dicTemp[w]*math.log10((1+4191)/(self.docidf[w]+1)))
else:
    docTFIDF.append(0)
```

1. TF 計算：
   計算該文章每個字出現的次數，並運用 Log Normalization (1+log(tf))概念調整 TF 參數，以完成該 doc 個文字的 TF 計算。
2. IDF 計算：
   self.docidf 為文字出現的次數，計算方式為:設 apple 在 a 文章出現 2 次，b 文章出現 1 次，則 IDF[apple]為 2 而非 3，即該字出現在幾篇文章中。
3. TFIDF 計算：
   為避免分母可能為 0 的現象，因此在 IDF 計算使用 log(N+1/IDF+1)方法。

TFIDF 參數設定-Queries

```
for wc in query_dict:
    query_dict[wc] = 0.8 + query_dict.get(wc, 0.0)*0.2/maxquery
```

1. TF 計算：
   計算該 query 各文字出現的次數，並將「(上述計算結果*0.2 /該 query 中出現最多次的文字次數) + 0.8」。0.2 及 0.8 參數，是以 Double normalization σ 的概念去進行參數設定，經由實驗測出 σ=0.8 的準確較高。
2. TFIDF 計算：Query TFIDF 的計算方式與 Document 的計算概念相同。

心得

調整參數其實蠻耗費心力的，因為每次跑都需要花費一些時間，但在做第二份作業時，發現因為我在計算相似度時，一直重複計算 tfidf，使時間複雜度太高，這時才了解在設計模型的時候，除提升準確率外，更要思考如何快速跑出結果。