HW4 - Probabilistic Latent Semantic Analysis

- **使用套件**

  使用數學相關套件進行邏輯計算。使用 random 分配隨機數。

```
import numpy as np
import math
import random
import time
```

- **資料前處理 - 計算每個字出現的數量 （word count）**

  由於 PLSA 可以處理同義詞和一詞多義的狀況，因此要考量所有字彙。

  ☆ 策略：為了減少記憶體使用量，我將數量小於 50 和長度小於 2 的字刪去，最後保留 9099 個字。

- **資料前處理 - 計算 tf**

  計算此 word 在這篇 document 出現幾次，將資料儲存至 Dictionary，利用此特性將時間複雜度縮減成 O(C)，增加處理效能。

- **資料前處理 - 計算 Background Word**

  將字與字數從過濾完成的 word_count 中取出，並除以全部文章字數的總長做正規化。

- **初始化參數 Topic, Topic_word, document_Topic**

  T_w[topic, doc_len] 表示機率 $p(T_k|d_i)$，d_T[word_len, topic] 表示機率 $p(w_j|T_k)$。
  利用 random 隨機產出此維度的數字，並除以總合做 normalize 保證和為 1。

- **實作 Expectation-Maximization Algorithm**

  由於 PLSA 增加了 Topic 這個隱含變數，因此需要利用 EM 演算法尋找參數最大似然估計，將迭代過程分為 Estep 和 Mstep，在未收斂前重複執行迭代過程，判斷收斂的方法可以利用 log likelihood，若似然率的變化小於自設的閥值，則停止迭代，避免 overfitting。

  ✓ Estep：計算參數 e_step： $p(T_k|w_i, d_j)$ -> 未知變量的期望估計。

  ✓ Mstep：根據 e_step 參數更新 T_w, d_T，給出當前參數的最大似然估計。

  ✓ 參數設定：*iteration = 20, threshold = 100*

- **實作 PLSA model**

  核心公式：

$$P(q|d_j) = \prod_{i=1}^{|q|} \left[\alpha \cdot P(w_i|d_j) + \beta \cdot \left(\sum_{k=1}^{K} P(w_i|T_k) P(T_k|d_j) \right) + (1 - \alpha - \beta) \cdot P_{BG}(w_i) \right]$$

  1. 參數設定：$\alpha = 0.8$，$\beta = 0.1$
  2. 利用前處理得到的 tf、BG_word，以及 EM 演算法得出的 T_w, d_T 進行計算。
  3. 將此 query 中所有的 word 的分數用 PLSA 公式計算出來，再進行連乘，得到 score。

  ☆ 當 topic = 32, $\alpha = 0.6$，$\beta = 0.4$ 時分數為 0.54，調整參數 $\alpha = 0.8$，$\beta = 0.1$，分數上升至 0.562。

- **實作遇到的困難、心得**

  由於這次資料量過大，若我沒有過濾 word 的數量，設定 e_step 初始值時居然會出現 MemoryError！而 topic 數量的設定會導致計算速度線性成長，維度是 [14955, 9099, 8] 就花了我八個小時，這次作業空間和時間的複雜度都要慎重考慮，也許當維度過大時使用 C++來撰寫會比較適合…。