HW2 - Best Match Model

- **使用套件**

  只利用數學相關套件進行邏輯計算。

```
import numpy as np
import math
```

- **資料前處理**

  將檔案讀取後，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。

- **計算 Term Frequency （tf）**

  tf 定義：此 term 在這篇 document 中總共出現了多少次。(local)

  ☆ **tf 公式改良是參考 sklearn 中 sublinear_tf 參數的 source code。**

```
Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).
```

- **計算 Inverse Document Frequency （idf）**

  idf 定義：在所有 document 中此 word 出現的頻率。(global)

  1. **將所有 document 中出現過的 word 存至 all_word_list，利用 set 過濾重複出現的字。**
  2. **再逐一搜尋每篇 document 是否有這個 word，計算頻率儲存至 df_dict。**
  3. **利用 Smooth 公式計算出 idf 值，將其各 + 1 防止分子或分母為 0。**

  ☆ **Smooth 公式中，math 的 log 預設為 e 為底，將騎改以 10 為底，訓練分數有顯著的上升。**

  ☆ **Smooth 公式是參考 sklearn 中 smooth_idf 參數的 source code。**

```
zero divisions: idf(t) = log [ (1 + n) / (1 + df(t)) ] + 1.
```

- **實作 Best Match Model**

  BM model 是基於機率模型提出來的演算法，BM25 結合了 BM15 的 term-frequency 和 BM11 將文章長度正規化的概念，再結合一些參數進行優化。

  1. **參數設定：**
     - ✓ **K1 = 3.5，將範圍設定為[3, 4]，發現值為 3.5 時分數最好。**
     - ✓ **b = 0.75，根據研究 b 為 0.75 時效果最佳。**
  2. **公式只計算 Document Term Weight 和 Discriminative Power，由於 query 中的字幾乎沒有重複，因此 query 的 tf 為 1，此時 Query Term Weight 約分後也為 1，因此省略不計算。**
  3. **加總 document weight 時，只取與 query 有關的 term 的 tf 進行運算，可將許多為 0 的字濾除掉，提升效率以及準確率。**

  ☆ **建構模型之策略：我推測 word 在 document 出現的頻率對於模型建構來說較為重要，因此我將 idf weight 平方以提升 idf 的權重，確實改良後分數有顯著的上升，從 0.713 躍升為 0.725。**

- **實作遇到的困難、心得**

  這次作業可以沿用上次建構 vector space model 時的 tf 和 idf，需要做的只有改良模型公式，和對其參數進行修改，我覺得最困難與困惑的地方是，有時修改了參數但並不知道為甚麼這樣的數值對於模型是最佳的，因此只能不斷的進行猜測，也許這就是 ML 和 DL 最深奧的地方吧。