

# PD2 Homework 5



Generated by DALL.E



## Announcement:

- Deadline: 2024/6/5 23:59pm



5/21 Update : 本次作業計算 IDF 時，如果分母為 0 (包含詞彙 query\_word 的文檔數為 0) ，請將 IDF 定義為 0 作計算

## Search with TF-IDF

### Introduction

作業四中，我們學會了 TF-IDF 的計算方式，接下來要實際應用到搜尋文本，本作業要求同學基於作業四實作一個簡單的搜尋引擎。

本作業將會分為兩個部分，第一部分是序列化搜尋索引：讀取指定的文本並建立相對應的搜尋索引，最後將搜尋索引儲存在硬碟上。第二部分是搜尋引擎：我們將給定  $k$  的查詢字串，同學需依據查詢內容從硬碟讀取搜尋索引（反序列化）並且每個查詢字串需輸出最符合的  $n$  筆結果

### 序列化搜尋索引 homework requirement

給定一個文本的檔案路徑 (e.g. /home/share/hw5/corpus0.txt) 依據此文本建立搜尋索引，並儲存至硬碟。

文本內容大致如下： (此範例僅供參考不代表作業文本內容)

```
1 Local MSMEs have leveraged on their technical and scientific expertise to become part of several exciting on land, at sea, in the air, and beyond – in space. 2 "Investing in American innovation, in industries that will define the future, and that China's government is intent on dominating." 3 So, some Commission bills ....
```

- 每一行的格式為 `sentence_ID + \t + sentence`
- 每 5 個句子依序組成一份文本，文本 ID 從 0 開始。例：第1句到第5句 → 文本 ID=0，第6句到第10句 → 文本 ID=1
- 文本檔案大小的範圍為 10MB~50MB

## Parsing rules

處理字彙 parsing 時，請依照以下步驟處理

1. 將所有**非英文字元**以**空白**代替
2. 將所有英文大寫轉換成小寫
3. 以**空白**進行**詞彙的segmentation**

建議同學參考以下 code 實作 parsing

```
String input = "Local MSMEs have leveraged on their technical and scientific" input = input.replaceAll("[^a-zA-Z]+", " "); // Step 1  
input = input.toLowerCase(); // Step 2 // Step 3 ....
```

## Serialization

請使用 Java 提供的 `java.io.Serializable` 介面實現序列化，以下是範例程式碼，詳細說明可以參考上課講義或官方文件

```
public class Indexer implements Serializable { private static final long serialVersionUID = 1L; private String name; private transient int counter; // .... }
```

Indexer.java

```
Indexer idx = new Indexer(); try { FileOutputStream fos = new FileOutputStream("example.ser") ObjectOutputStream oos = new ObjectOutputStream(fos); oos.writeObject(idx); oos.close(); fos.close(); } catch (IOException e) { e.printStackTrace(); }
```

BuildIndex.java

```
try { FileInputStream fis = new FileInputStream("example.ser"); ObjectInputStream ois = new ObjectInputStream(fis); Indexer deserializedIdx = (Indexer) ois.readObject(); ois.close(); fis.close(); } catch (IOException e) { e.printStackTrace(); } catch (ClassNotFoundException c) { c.printStackTrace(); }
```

TFIDFSearch.java

## Command line argument

```
$ java BuildIndex corpusfile
```

- **corpusfile:** 文本檔案路徑
  - 助教在進行測試時，**corpusfile** 參數必為絕對路徑，請確保程式能夠讀取絕對路徑的檔案
  - 助教會拿多個不同的 **corpusfile** 進行測試，每次測試時 **corpusfile** 必為不同的檔案名稱（簡單來說我們不會拿一樣名稱但不同內容的檔案進行測試）
  - **corpusfile** 的檔案名稱格式為 **corpus{number}.txt** (e.g. **corpus0.txt**, **corpus1.txt** ...)

## Output

此程式會儲存序列化的搜尋索引至硬碟，儲存的檔案需符合以下規範

- 請將檔案處存至當前工作目錄下，助教在測試時會確保你的程式有權限於當前工作目錄建立檔案
- 我們沒有規定一個文本只能建立一個搜尋索引檔案，你可以一個文本建立多個檔案。但請確保建立的檔案不會覆蓋掉其他已經存在的檔案
- 請確保你建立的搜尋索引檔案能夠透過文本檔案名稱查詢，例如：

```
$ java BuildIndex /home/share/hw5/corpus1.txt $ ls  
BuildIndex.java corpus1.ser
```

建立一個檔案名稱叫 corpus1.ser，日後給定 corpus1 可以知道其對應的搜尋索引檔案為 corpus1.ser

建議同學也可以採用像範例的方式命名

## 搜尋引擎 homework requirement

給定一個搜尋字串  $q$  回傳  $n$  個文檔 ID。步驟為以下

1. 使用者由字串  $q$  的 AND 以及 OR 規則得到  $k$  個文檔 (AND/OR 詳細規則請看 [User query](#))
2. 將  $k$  個文檔依照計算出的 TF-IDF 值由大到小排序輸出  $n$  個文檔 ID，當 TF-IDF 一樣時由小到大排序文檔 ID (TD-IDF 計算方式請看 [TF-IDF 計算方式](#))
3. 若  $k < n$  則  $k + 1 \dots n$  用 -1 取代

## User query

$q$  為使用者的查詢字串，格式為 `query_word1 [AND/OR] query_word2 [AND/OR] query_word3...`

例：

the AND quick AND brown AND fox jumps OR over OR the OR lazy OR dog  
 hello apple AND apple AND apple people OR people

- `the AND quick` 代表取包含 `the` 以及包含 `quick` 文檔的交集(intersection)
- `jumps OR ever` 代表取包含 `jumps` 以及包含 `ever` 文檔的聯集(union)
- `hello` 不含 AND 或 OR 代表取所有包含 `hello` 的文檔。當  $q$  不含 AND 或 OR 時， $q$  只會有一個 query\_word
- query\_word 必為小寫，AND/OR 必為大寫
- query\_word 不會是空字串或空白字元
- query\_word、AND/OR 之間會有空白字元隔開
- 每一行 query 中 AND、OR 不會同時出現
- query\_word 有可能沒有在文本中出現過

## TF-IDF 計算方式

如果要計算  $q$  對於文檔  $d_j$  的 TF-IDF 需計算  $q$  中的每個 query\_word  $t_i$  對應文檔  $d_j$  的 TF-IDF 並加總

$$\sum_i TF - IDF(t_i, d_j, D)$$

範例：

$q$  為 `hello AND world` 包含 `hello` 的文檔 ID 為 `1, 2, 4, 5` 包含 `world` 的文檔 ID 為 `2, 3, 4` 取交集後文檔 ID 為 `2, 4`。 $q$  對於文檔 ID 2 的 TF-IDF 為 `tf_idf("hello", 2, docs) + tf_idf("world", 2, docs)`， $q$  對於文檔 ID 4 的 TF-IDF 為 `tf_idf("hello", 4, docs) + tf_idf("world", 4, docs)`



計算 IDF 時，如果分母為 0 (包含詞彙 query\_word 的文檔數為 0)，請將 IDF 定義為 0 作計算

## Command line argument

command line argument 格式為以下：

```
$ java TFIDSearch testFileName testcase
```

- testFileName: 讀取 corpus 的檔案名稱
  - 請注意 `testFileName` 不是檔案路徑而是檔案名稱，助教在測試程式時會確保該 corpus 檔案有先呼叫 BuildIndex 程式建立索引檔案（但我們不會幫你檢查你的 BuildIndex 程式有沒有成功建立檔案，如果你的 BuildIndex 程式失敗會影響後續測試）
  - 測試範例：

```
$ javac Indexer.java BuildIndex.java TFIDFSearch.java
```

```
$ java BuildIndex /home/share/hw5/corpus1.txt #-->建立了一個名稱為  
corpus1.data 的檔案
```

```
$ java TFIDFSearch corpus1 tc3.txt # 查詢目標為之前建立的  
corpus1.data
```

- testcase: 測資檔案路徑

範例：

```
$ java TFIDFSearch corpus1 tc1.txt $ java TFIDFSearch courpus2  
/home/share/testcase/tc2.txt
```

## Input

第一行代表每個 query 輸出 n 筆結果 ( $1 \leq n \leq 1024$ ) ，接下來每一行為一筆 query  
範例：

```
5 the AND to AND of AND in van OR drop OR front OR this visit
```

## Output

- 針對每一筆 query 依序輸出文檔 ID
- 輸出答案至 `output.txt` 檔案，`output.txt` 位置應為當前工作目錄下

範例：

n=5

```
4351 14402 11863 7061 19832 16720 12094 1259 5042 18216 10936 12299  
846 8552 10713
```

output.txt

## Filename requirement

請同學實作三個 class `BuildIndex` 、 `TFIDFSearc` 、 `Indexer` 並將檔案名稱命名為 `BuildIndex.java` 、 `TFIDFSearc.java` 、 `Indexer.java`

- `BuildIndex` 處理作業第一部分的序列化搜尋索引
- `TFIDFSearc` 處理作業第二部分的搜尋引擎
- `Indexer` 處理序列化，該 Class 應該 implement `Serializable`

## Challenge point

執行 testcase 0 ~ 4 的平均時間少於 `30` 秒即可獲得challenge point

## Homework ranking



切記檔案要放在~/hw5 底下，避免錯誤的作業繳交情況！

- 在deadline前，一共會提供5個test cases給大家，分別為 `tc0.txt~tc4.txt`，同時也將提供對應的答案 `ans0~ans4`。
- 在繳交deadline之後，我們一共會用10個test cases，也就是除了提供的測資以外，還有5個隱藏測資。
- 除了有隱藏的test case之外，我們在 `tc5.txt~tc9.txt` 會加入不一樣的文本進行評分。

- output的java files會透過diff程式比較，所以output格式是嚴格被要求的，沒有例外。請與測資的golden output確認是一致的。
- 每通過1個test case，你可以獲得1 pt，總分10分，若含challenge point，則是總分11分。

## Homework validation

- Deadline過後我們會 copy ~/hw5 下的 `BuildIndex.java` 、 `Indexer.java` 以及 `TFIDFSearch.java` 檔案，請確認 `BuildIndex.java` 、 `Indexer.java` 以及 `TFIDFSearch.java` 位於該路徑下。
- 於測資的地方也會提供 `validate_hw5.py` (`/home/share/hw5/validate_hw5.py`) 程式協助同學檢查自己程式於測試資料上執行的正確性。
- `validate_hw5.py` 的使用方法 `python3 validate_hw5.py [--build_index | --no-build_index] [--testcase TESTCASE]`
  - `--build_index` 執行 `BuildIndex` (預設執行)
  - `--no-build_index` 不執行 `BuildIndex`
  - `--testcase TESTCASE` 執行指定 testcase (預設執行所有 testcase)
- List

## Homework validation process

Deadline過後我們會按照以下步驟測試同學的程式碼

1. copy `BuildIndex.java` 、 `Indexer.java` 以及 `TFIDFSearch.java` 到一個空的 directory
2. 依序編譯 `Indexer.java` 、 `BuildIndex.java` 以及 `TFIDFSearch.java`
3. 對每個 corpus file 執行 `java BuildIndex corpusFile`
4. 測試 10 個 testcase `java TFIDFSearch corpusName testcase`

## Step-By-Step Example

編譯 `Indexer.java` 、 `BuildIndex.java` 以及 `TFIDFSearch.java`

```
$ javac Indexer.java BuildIndex.java TFIDFSearch.java
```

文本檔案名稱為 `example.txt`，執行 `java BuildIndex`  
`/home/share/hw5/example.txt`

1 Between subtle shading and the absence of light lies the nuance of iqlusion. 2 It was totally invisible How's that possible? They used the Earths magnetic field X 3 The information was gathered and transmitted undergruund to an unknown location X 4 Does Langley know about this? They should Its buried out there somewhere X 5 Who knows the exact location? Only WW This was his last message X 6 Thirty eight degrees fifty seven minutes six point five seconds north 7 Seventy seven degrees eight minutes forty four seconds west ID by rows 8 Slowly, desparatly slowly, the remains of passage debris 9 that encumbered the lower part of the doorway was removed 10 with trembling hands. 11 I made a tiny breach in the upper left-hand corner 12 and then widening the hole a little, I inserted the candle and peered in. 13 The hot air escaping from the chamber caused the flame to flicker 14 but presently details of the room within emerged from the mist X 15 Can you see anything Q?

example.txt



因為 notion 顯示問題，複製以上文本時要注意格式對不對

對文本處理過後內容大致如下

0 between subtle shading and the absence of light lies the nuance of iqlusion it was totally invisible how s that possible they used the earths magnetic field x the information was gathered and transmitted undergruund to an unknown location x does langley know about this they should its buried out there somewhere x who knows the exact location only ww this was his last message x 1 thirty eight degrees fifty seven minutes six point five seconds north seventy seven degrees eight minutes forty four seconds west id by rows slowly desparatly slowly the remains of passage debris that encumbered the lower part of the doorway was removed with trembling hands 2 i made a tiny breach in the upper left hand corner and then widening the hole a little i inserted the candle and peered in the hot air escaping from the chamber caused the flame to flicker but presently details of the room within emerged from the mist x can you see anything q



以上是為了範例才輸出文本，同學不必將處理過後的文本輸出

接著建立索引並儲存序列化的索引

執行 `java TFIDFSearch example exampleTC.txt`

3 the AND x and OR was

exampleTC.txt

首先來看 query `the AND x`

1. `the` 出現在文檔 `0, 1, 2`，`x` 出現在文檔 `0, 2`。兩個取交集為 `0, 2`
2. 計算 `the AND x` 對於文檔 `0` 的 TF-IDF
  - a. `tf_idf("the", 0)` → 0
  - b. `tf_idf("x", 0)` → 0.12475849480251212
  - c. 加總為：0.12475849480251212
3. 計算 `the AND x` 對於文檔 `2` 的 TF-IDF
  - a. `tf_idf("the", 2)` → 0
  - b. `tf_idf("x", 2)` → 0.08109302162163289
  - c. 加總為：0.08109302162163289
4. 根據加總的 TF-IDF 值排序文檔（由大到小）→ `[0, 2]`
5. `n` 為 3 所以用 -1 取代第三個文檔 → `[0, 2, -1]`

接著看 query `and OR was`

1. `and` 出現在文檔 `0, 2`，`was` 出現在文檔 `0, 1`。兩個取聯集為 `0, 1, 2`
2. 計算 `and OR was` 對於文檔 `0` 的 TF-IDF
  - a. `tf_idf("and", 0)` → 0.06237924740125606
  - b. `tf_idf("was", 0)` → 0.0935688711018841
  - c. 加總為：0.15594811850314017

3. 計算 `and OR was` 對於文檔 1 的 TF-IDF

- a. `tf_idf("and", 1)` → 0
- b. `tf_idf("was", 1)` → 0.13515503603605478
- c. 加總為：0.13515503603605478

4. 計算 `and OR was` 對於文檔 2 的 TF-IDF

- a. `tf_idf("and", 2)` → 0.16218604324326577
- b. `tf_idf("was", 2)` → 0
- c. 加總為：0.16218604324326577

5. 根據加總的 TF-IDF 值排序文檔（由大到小）→ `[2, 0, 1]`