# CAB431 – Search Engine Technology
# **Assignment 1:** Portfolio
# **(Sem 1, 2022)**

Required to be submitted:

1. Please put your outputs into text files specified in the question descriptions and put all .py files, data and a "readme.txt" in a folder (e.g., Your Surname_code), where "readme.txt" contains a short user manual to help your tutor run your Python code. Then zip all .txt files and the folder into a zip file named as "your student ID_Surname_Asm1.zip".
2. Submit your zip file for this assignment in BB before **11.59pm on 6 May 2022**.
3. Answer all three questions (9 tasks).
4. See the marking guide for more details on the distribution of marks and marking criteria.

**Individual working:** You should work on this assignment individually.
**Due date:** Friday week 8 (**6 May 2022**)
**Weighting:** 20% of the assessment for CAB431.

**Dataset (*Rnews_t120* document collection)**

- You will be working with a sample dataset which is a small subset of XML documents (TREC RCV1 data collection), which is a pre-tokenized version of (for convenience, and for copyright reasons). The dataset can be downloaded from Blackboard.

You are asked to design Python code for three questions (9 tasks). You can add new variables, functions, methods, or update function parameters. However, you should provide comments to clearly describe why you are doing this.

## Question 1. Document & query parsing

The motivation for Question 1 is to design your own document and query parsers. So please don't use python packages that we didn't use in the workshop.

**Task 1.1:** Define a document parsing function *parse_rcv_coll*(*inputpath*, *stop_words*) to parse a data collection (e.g., *Rnews_t120* dataset), where parameter *inputpath* is the folder that stores a set of XML files, and parameter *stop_words* is a list of common English words (you may use

the file 'common-english-words.txt' to find all stop words). The following are the major steps in the document parsing function:

Step 1) The function reads XML files from *inputpath* (e.g., *Rnews_t120*). For each file, it finds the *docID* (document ID) and index terms, and then represents it in a *BowDoc* Object.

You need to define a *BowDoc* class by using Bag-of-Words to represent a document:

- *BowDoc* needs a *docID* variable which is simply assigned by the value of '*itemid*' in <newsitem …>.
- In this task, *BowDoc* can be initialled with an attribute *docID*; an empty dictionary (the variable name is *terms*) of key-value pair of (*String* term: *int* frequency); and *doc_len* (the document length) attribute.
- You may define your own methods, e.g., *getDocId*() to get the document ID.

Step 2) It then builds up **a collection** of *BowDoc* objects for the given dataset, this collection can be a dictionary structure (as we used in the workshop), a linked list, or a class *BowColl* for storing a collection of *BowDoc* objects. Please note the rest descriptions are based on the dictionary structure with *docID* as key and *BowDoc* object as value.

Step 3) At last, it returns the collection of *BowDoc* objects.

You also need to follow the following requirements to define this parsing function:

Please use the basic text pre-processing steps, such as tokenizing, stopping words removal and stemming of terms.

Tokenizing – (please provide a definition of a **word**, and describe it in a Python comment)

- You need to tokenize at least the '<text>…</text>' part of document, exclude all tags, and discard punctuations and/or numbers based on your definition of words.
- Define method *addTerm*() for class *BowDoc* to add new term or increase term frequency when the term occur again.

Stopping words removal and stemming of terms –

- Use the given stopping words list ("common-english-words.txt") to ignore/remove all stopping words. Open and read the given file of stop-words and store them into a

list *stopwordList*. When adding a term, please check whether the term exists in the *stopwordList*, and ignore it if it is in the *stopwordList*.

- Please use porter2 stemming algorithm to update *BowDoc*'s *terms*.

**Task 1.2:** Define a query parsing function *parse_query*(*query0*, *stop_words*), where we assume the original query is a simple sentence or a title in a String format (*query0*), and *stop_words* is a list of stop words that you can get from 'common-english-words.txt'.
For example, let *query0* =

' CHINA: China to establish 60 mln tonne coal mine.'

the function will return a dictionary

{'china': 2, 'establish': 1, 'mln': 1, 'tonn': 1, 'coal': 1, 'mine': 1}

Please note you should use the same text transformation technique as the document, i.e., tokenizing steps for queries **must be identical** to steps for documents.

**Task 1.3:** Define a main function to test function *parse_rcv_coll*( ). The main function uses the provided dataset, calls function *parse_rcv_coll*() to get a collection of *BowDoc* objects. For each document in the collection, firstly print out its *docID*, the number of index terms and the total number of works in the document (*doc_len*). It then sorts index terms (by frequency) and prints out a *term*:*freq* list. At last, it saves the output into a text file (file name is "your full name_Q1.txt").

Sample Example of output for file "78091.xml"

Document 78091 contains 45 indexing terms and have total 90 words
mine : 3
johannesburg : 3
three : 2
buffelsfontein : 2
gold : 2
southwest : 2
compani : 2
ltd : 2
south : 1
african : 1
miner : 1
kill : 1
overnight : 1

renew : 1
ethnic : 1
clash : 1
take : 1
death : 1
toll : 1
sinc : 1
weekend : 1
polic : 1
thursday : 1
anoth : 1
peopl : 1
injur : 1
fight : 1
between : 1
xhosa : 1
sotho : 1
worker : 1
…

## Question 2. Tf*idf based IR model

Tf*idf is a popular term weighting method, which uses the following Eq. (1) to calculate a weight for term *k* in a document *i*, where the base of *log* is 10. You may review lecture notes to get the meaning of each variable in the equation.

$$d_{ik} = \frac{(\log(f_{ik}) + 1) \cdot \log(N/n_k)}{\sqrt{\sum_{x=1}^{t}[(\log(f_{ix}) + 1) \cdot \log(N/n_x)]^2}} \quad (1)$$

**Task 2.1:** Define a function *calc_df*(*coll*) to calculate document-frequency (*df*) for a given *BowDoc* collection *coll* and return a {*term*:*df*, …} dictionary.

Example of output for this task

There are 12 documents in this data set and contains 870 terms
mine : 12
compani : 9
ltd : 9
one : 9
between : 8
gold : 8
peopl : 8
quot : 8

south : 8
three : 8
buffelsfontein : 7
death : 7
ethnic : 7
fight : 7
johannesburg : 7
kill : 7
polic : 7
clash : 7
hostel : 6
miner : 6
month : 6
more : 6
presid : 6
sotho : 6
sunday : 6
told : 6
two : 6
use : 6
violenc : 6
worker : 6
sinc : 6
african : 6
amp : 6
last : 6
africa : 5
appoint : 5
area : 5
attack : 5
earli : 5
explos : 5
field : 5
four : 5
here : 5
industri : 5
injur : 5
involv : 5
labour : 5
mandela : 5
nelson : 5
now : 5
reuter : 5
…

**Task 2.2:** Use Eq (1) to define a function *tfidf*(*doc, df, ndocs*) to calculate tf*idf value (weight) of every term in a *BowDoc* object, where *doc* is a *BowDoc* object or a dictionary of {term:freq,…}, *df* is a {*term*:*df*, …} dictionary, and *ndocs* is the number of documents in a

given *BowDoc* collection. The function returns a {*term*:*tfidf_weight* , …} dictionary for the given document *doc*.

**Task 2.3:** Define a main function to **print out all terms** (with its value of tf*idf weight in descending order) for each document in *Rnews_t120* and save the output into a text file (file name is "your full name_Q2.txt").

You also need to implement a tf*idf based IR model. You can assume titles of XML documents (the <title>…</title> part) are the original queries, and test at least three titles. You need to use function *parse_query()* that you defined for Question 1 to parse original queries. For each query, please use the abstract model of ranking (Eq. (2)) to calculate a score for each document.

$$R(Q,D) = \sum_i g_i(Q) f_i(D) \qquad (2)$$

At last, append the output (in descending order) into the text file ("your full name_Q2.txt").

Example of output for this task

Document 78091 contains 45 terms
newsroom : 0.35131343097356754
southwest : 0.26298067729993324
renew : 0.25331702769099573
critic : 0.25331702769099573
affect : 0.25331702769099573
take : 0.19599280656514362
toll : 0.19599280656514362
weekend : 0.19599280656514362
thursday : 0.19599280656514362
anoth : 0.19599280656514362
klerksdorp : 0.19599280656514362
johannesburg : 0.1599200189220541
overnight : 0.15532062440842395
xhosa : 0.15532062440842395
near : 0.15532062440842395
mile : 0.15532062440842395
wednesday : 0.15532062440842395
output : 0.15532062440842395
buffelsfontein : 0.12902246432799303
injur : 0.12377282899385192
randgold : 0.12377282899385192
explor : 0.12377282899385192
…

The Ranking Result for query: USA: RESEARCH ALERT - Minnesota Mining cut.

67460 : 0.5593787296822499
81047 : 0.0423818768342092
71759 : 0.024579713959536946
69633 : 0.0
75135 : 0.0
71948 : 0.0
71406 : 0.0
75338 : 0.0
69629 : 0.0
86836 : 0.0
78092 : 0.0
78091 : 0.0
…

## Question 3. BM25-based IR model

BM25 IR model is a popular and effective ranking algorithm, which uses the following Eq. (3) to calculate a document score or ranking for a given query $Q$ and a document $D$, where the base of *log* is 2. You may review lecture notes to get the meaning of each variable in the equation.

$$\sum_{i \in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i} \quad (3)$$

You can use the *BowDoc* collection to work out some variables, such as $N$ and $n_i$ (you may assume $R = r_i = 0$).

**Task 3.1:** Define a Python function *avg_doc_len*(*coll*) to calculate and return the average document length of all documents in the collection *coll*.

- In the *BowDoc* class, for the variable *doc_len* (the document length), add accessor (get) and mutator (set) methods for it.
- You may modify your code defined in Question 1 by calling the mutator method of *doc_len* to save the document length in a *BowDoc* object when creating the *BowDoc* object. At the same time, sum up every *BowDoc*'s *doc_len* as *totalDocLength*, then at the end, calculate the average document length and return it.

**Task 3.2:** Use Eq (3) to define a python function *bm25*(*coll*, *q*, *df*) to calculate documents' BM25 score for a given original query *q*, where *df* is a {*term*:*df*, …} dictionary. Please note you should parse query using the same method as parsing documents (you can call function *parse_query*() that you defined for Question 1). For the given query *q*, the function returns a dictionary of {*docID*: *bm*25_*score*, … } for all documents in collection *coll*.

**Task 3.3:** Define a main function to implement a BM25-based IR model to rank documents in the given document collection *News_v1* using your functions.

- You are required to test all the following queries:
  - Deaths mining accidents
  - Mentioning deaths in mining accidents
  - Statistics on number of mining deaths
  - ethnic clashes and resultant deaths of mine workers near a mine
- The BM25-based IR model needs to print out the ranking result (in descending order) of top-5 possible relevant documents for a given query and append outputs into the text file ("your full name_Q3.txt").

Example of output for this question (Note you may get negative BM25 scores since $N$ is not big enough and $n_i$ can be closed to $N$, e.g., $n_i > N/2$)

Average document length for this collection is: 340.5
The query is: Deaths mining accidents
The following are the BM25 score for each document:
Document ID: 69633, Doc Length: 414 -- BM25 Score: -9.758760770765
Document ID: 75135, Doc Length: 265 -- BM25 Score: -8.39974494308862
Document ID: 71948, Doc Length: 272 -- BM25 Score: -8.733352593385417
Document ID: 71406, Doc Length: 357 -- BM25 Score: -9.695888227954333
Document ID: 75338, Doc Length: 428 -- BM25 Score: -8.653664366565266
Document ID: 71759, Doc Length: 1082 -- BM25 Score: -5.707602980799846
…

The following are possibly relevant documents retrieved -
71759 -5.707602980799846
67460 -7.407032197935156
75135 -8.39974494308862
86836 -8.417937499722314
75338 -8.653664366565266
71948 -8.733352593385417
…

You can solve the negative problem by provide a large *N*, e.g., you may replace *N* using *2N* in Eq (3). Then we may have the following outputs. We accept both outputs for this assignment.

```
Average document length for this collection is: 340.5
The query is: Deaths mining accidents
The following are the BM25 score for each document:
Document ID: 69633, Doc Length: 414 -- BM25 Score: 1.1232065781332516
Document ID: 75135, Doc Length: 265 -- BM25 Score: 2.016725388884778
Document ID: 71948, Doc Length: 272 -- BM25 Score: 0.0
Document ID: 71406, Doc Length: 357 -- BM25 Score: 1.1986310135782883
Document ID: 75338, Doc Length: 428 -- BM25 Score: 1.1061112071306798
Document ID: 71759, Doc Length: 1082 -- BM25 Score: 0.0
Document ID: 81047, Doc Length: 499 -- BM25 Score: 0.0
Document ID: 67460, Doc Length: 30 -- BM25 Score: 0.0
Document ID: 69629, Doc Length: 357 -- BM25 Score: 1.1986310135782883
Document ID: 86836, Doc Length: 129 -- BM25 Score: 0.0
Document ID: 78092, Doc Length: 163 -- BM25 Score: 1.5537357475577047
Document ID: 78091, Doc Length: 90 -- BM25 Score: 1.748675953066234

The following are possibly relevant documents retrieved -
75135 2.016725388884778
78091 1.748675953066234
78092 1.5537357475577047
71406 1.1986310135782883
69629 1.1986310135782883
69633 1.1232065781332516

…
```

## Please Note

- You can add more methods, variables or functions. For any new one, you should provide comments to understand its definition and usage.
- Your programs should be well laid out, easy to read and well commented.
- All items submitted should be clearly labelled with your name or student number.
- Marks will be awarded for programs (correctness, programming style, elegance, commenting) and outputs, according to the marking guide.
- You will lose marks for inaccurate outputs, code problems or errors, or missing required files or comments.

**END OF ASSIGNMENT 1**