CAB431 – Text Analysis and Web Search
Assignment 1

Specification

Question 1. Representation of Documents & Queries

The motivation for Question 1 is to design your own text pre-processing model for representing documents and queries. So, please don't use python packages that we didn't use in the workshop.

Task 1.1: Define a document parsing function *Parse_Rcv1Doc*(*stop_words, inputpath*) to parse a data collection (e.g., *RCV1v3* dataset), where parameter *stop_words* is a list of common English words (you may use the file 'common-english-words.txt' to find all stop words), and parameter *inputpath* is the folder that stores a set of XML files.

The following are the major steps in the document parsing function:

Step 1) The function reads XML files from *inputpath* (e.g., *RCV1v3*). For each file, it finds the *docID* (document ID) and index terms, and then represents it in a *RnewsDoc* Object.

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

- *RnewsDoc* needs a *docID* variable (attribute) which is simply assigned by the value of '*itemid*' in <newsitem …>.
- In this task, *RnewsDoc* can be initialized with three attributes: ***docID*** attribute, an empty dictionary (the attribute 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, *get_term_list*() to get a sorted list of all terms occurring in the document, etc.

Step 2) It then builds up **a collection** of *RnewsDoc* objects for the given dataset, this collection can be a dictionary structure (as we used in the workshop), a linked list, or a class *Rcv1Coll* for storing a collection of *RnewsDoc* objects. Please note the rest descriptions are based on the dictionary structure with *docID* as key and *RnewsDoc* object as value.

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

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

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

<u>Tokenizing</u> –

- You are required to provide **definitions of words and terms** and decide whether some special tokens (e.g., “&quot”) are terms and describe the definitions as comments in your Python solution
- You need to tokenize at least the ‘<text>…</text>’ part of document, exclude all tags, and/or discard punctuations and/or numbers based on your definition of terms.
- Define method *add_term*() for class *RnewsDoc* to add new term or increase term frequency when the term occur again.

<u>Stopping words removal and stemming of terms</u> –

- Use the given stopping words list (“common-english-words.txt”) to ignore/remove all stopping words (you could add new stopping words into the list). 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 *RnewsDoc*’s *terms*.

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

For example, let *query0* =

'CANADA: Sherritt to buy Dynatec, spin off unit, canada.'

this function will return a dictionary:

{'canada': 2, 'sherritt': 1, 'buy': 1, 'dynatec': 1, 'spin': 1, 'unit': 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 functions *Parse_Rcv1Doc*( ) and *Parse_Q*( ). The main function uses the provided dataset, calls function *Parse_Rcv1Doc*( ) to get a collection of *RnewsDoc* objects. For each document in the collection, firstly print out its *docID*, the number of terms and the total number of words in the document (*doc_len*). It then sorts 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").

Output Example for file "783803newsML.xml"

```
Document 783803 contains 200 indexing terms and have total 490 words
pari : 10
reuter : 8
french : 8
franc : 8
advertis : 6
news : 6
amp : 5
product : 5
media : 4
group : 4
made : 4
more : 4
percent : 4
europ : 4
market : 3
internet : 3
brand : 3
peopl : 3
televis : 3
canal : 3
plus : 3
publish : 3
amauri : 3
channel : 3
tribun : 3
network : 3
conserv : 3
name : 2
award : 2
public : 2
purchas : 2
pay : 2
newspap : 2
compani : 2
prepar : 2
immedi : 2
…
```

```
Query: CANADA: Sherritt to buy Dynatec, spin off unit, canada
The parsed query:
{'canada': 2, 'sherritt': 1, 'buy': 1, 'dynatec': 1, 'spin': 1, 'unit': 1}
```

Question 2. IR Model 1 (TF*IDF-based method)

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 *e*. You may review lecture notes to get the meaning of each variable in the equation.

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

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

Task 2.1 outputs example:

```
There are 15 documents in this data set and contains 1068 terms.
The following are the terms' document-frequency:

quot : 10
corp : 9
one : 8
top : 8
car : 7
over : 7
market : 7
percent : 7
year : 7
posit : 7
compani : 7
group : 7
inc : 7
drive : 6
four : 6
power : 6
buy : 6
design : 6
ltd : 6
```

```
report : 6
sale : 6
sport : 6
three : 6
two : 6
unit : 6
day : 6
follow : 5
manag : 5
share : 5
time : 5
activ : 5
chang : 5
reuter : 5
consum : 5
honda : 5
manual : 5
model : 5
motor : 5
onli : 5
rav : 5
statement : 5
suzuki : 5
toyota : 5
…
```

Task 2.2: Use Eq. (1) to define a function *m1_tfidf*(*doc, df, ndocs*) to calculate TF*IDF value (weight) of every term in a *RnewsDoc* object, where *doc* is a *RnewsDoc* object or a dictionary of {term:freq,…}, *df* is a {*term*:*df*, …} dictionary, and *ndocs* is the number of documents in a given *RnewsDoc* collection. The function returns a {*term*:*tfidf_weight* , …} dictionary for the given document *doc*.

Task 2.3: Define a main function to call *m1_tfidf*() and **print out top 15 terms** (with its value of tf*idf weight) for each document in *RCV1v3* if it has more than 15 terms and save the output into a text file (file name is "your full name_Q2.txt").

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

$$R(Q,D)=\sum_i g_i(Q)f_i(D) \tag{2}$$

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

<u>Task outputs example:</u>

```
Document 783803 contains 200 terms

pari : 0.1997033518453283
french : 0.1862101294497417
amp : 0.15778957479425215
media : 0.14429635239866556
more : 0.14429635239866556
europ : 0.14429635239866556
franc : 0.13854815251620575
internet : 0.12690056318605591
brand : 0.12690056318605591
canal : 0.12690056318605591
plus : 0.12690056318605591
publish : 0.12690056318605591
amauri : 0.12690056318605591
tribun : 0.12690056318605591
conserv : 0.12690056318605591
…

The Ranking Result for query: FRANCE: Reuters French Advertising &amp; Media Digest - Aug 6.

783803 : 0.8918095069433722
741299 : 0.07683113194878158
783802 : 0.05553462627320665
809481 : 0.05257096103548241
807606 : 0.052325609213195914
780723 : 0.05114268800335972
80483 : 0.040506858052742636
80484 : 0.040506858052742636
80884 : 0.040417020231357854
807600 : 0.03040520368217731
741309 : 0
86961 : 0
…
```

Question 3. IR Model 2 (BM25-based method)

BM25 is a popular IR model with an 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.

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

You can use the *RnewsDoc* collection to calculate these variables, such as N and n_i (you may assume $R = r_i = 0$).

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

- In the *RnewsDoc* class, for the variable (attribute) *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 *RnewsDoc* object when creating the *RnewsDoc* object. At the same time, sum up every *RnewsDoc*'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 *m2_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_Q*() that you defined for Question 1). For the given query *q*, the function returns a dictionary of {*docID*: *bm25_score*, … } for all documents in collection *coll*.

Task 3.3: Define a main function to implement the IR model 2 (the BM25-based method) to rank documents in the given document collection *RCV1v3* using your functions.

- You are required to test the following four queries:
    - This British fashion
    - All fashion awards
    - The stock markets
    - The British-Fashion Awards

- The IR model 2 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”).

PS: you may get negative BM25 scores because *N* is not large enough and n_i can be close to *N*. You can fix this by increasing *N*.

Task outputs example:

```
Average document length for this collection is: ...

The query is: The British-fashion Awards
The following are the BM25 score for each document:

Document ID: 741299, Doc Length: 199 -- BM25 Score: 0.0
Document ID: 780723, Doc Length: 124 -- BM25 Score: 0.0
Document ID: 741309, Doc Length: 104 -- BM25 Score: 0.0
Document ID: 86961, Doc Length: 443 -- BM25 Score: 0.0
Document ID: 780718, Doc Length: 107 -- BM25 Score: 0.0
Document ID: 783803, Doc Length: 490 -- BM25 Score: 3.0444718793585714
Document ID: 80483, Doc Length: 610 -- BM25 Score: 0.0
Document ID: 809481, Doc Length: 151 -- BM25 Score: 0.0
Document ID: 809495, Doc Length: 703 -- BM25 Score: 0.0
Document ID: 80484, Doc Length: 610 -- BM25 Score: 0.0
Document ID: 783802, Doc Length: 120 -- BM25 Score: 4.656534695407324
Document ID: 807600, Doc Length: 538 -- BM25 Score: 0.0
Document ID: 80884, Doc Length: 610 -- BM25 Score: 0.0
…

For query " The British-fashion Awards", the top-5 possible relevant documents are:

783802 4.656534695407324
783803 3.0444718793585714
741299 0.0
780723 0.0
741309 0.0

…
```

Requirements

- Your Python solution should be a few .py files and be able to be tested in Pycharm (lab version).

- You can add more methods, variables or functions. For any new one, you should provide comments to understand its definition and usage.
- Your output does not need to match the example output exactly.
- 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, missing required files or comments, or not following the specification and requirements.

THE END OF ASSIGNMENT 1