Airi Chow (#40003396)
COMP 479
Fall '20

Project #2

Using Python3
Yellow Highlight: Issue with running.
Green Highlight: Given names for outputs/inputs.

Preprocessing:

Step #1 – Reading the Reuter's collection (block-1-reader.py)

- **python3 ./block-1-reader.py --path reuters21578**
- **python3 ./block-1-reader.py --path reuters21578 -o collection.json**

==========
Input: reuters21578 (Corpus' path)
Output: Creates a JSON file of the entire collection of SGM 000 to 021 data as a string.
==========

- Changing directory to /reuters21578 to scan for .sgm files.
- open(file_name,'r', errors='ignore'): Opening files but set errors parameter to ignore due to unicode issue in sgm17.
Error:
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xfc in position 1519554: invalid start byte

Step #2 – Extracting the raw text of each article from the corpus (block-2-document-segmenter.py)

- **python3 ./block-1-reader.py --path reuters21578 | python3 ./block-2-document-segmenter.py -o documents.json**
- **python3 ./block-2-document-segmenter.py -i collection.json -o documents.json**

==========
- Input: collection.json contains all the documents or pipelined stdin data from
- Output: Creates a JSON file with the documents from the entire collection.
==========

- **BeautifulSoup must be installed to parse the documents.

- **JSON is needed to parse the pipelined data
- replace("reuters", "REUTERS") is used to satisfied the assert <REUTERS> since html parser lower-cases the tags.
- sys.stdin.isatty(): Check if stdin filestream is being used.
- sys.stdin.read(): Read the pipelined data
- str(document): Change the "tag" object to "string"
- enumerate(documents): Used to iterate but keep an index variable

Step #3 – Extraction
(block-3-extractor.py)

Tested Commands

- **python3 ./block-3-extractor.py -i documents.json -o articles.json**
- **python3 ./block-1-reader.py --path reuters21578 | python3 ./block-2-document-segmenter.py | python3 ./block-3-extractor.py -o articles.json**

==========

- Input: documents.json contains all the documents of the collection. (This is the list of tokens"
- Output: list of dictionaries(List F) with "ID and Text" (Term-document pairs)

==========

- **BeautifulSoup must be installed to parse the documents.**
- document_id: Consists of ID with whitespace and escape characters
- document_id_arr: Consists of an array with one element (the ID #). Python's regex library is used to parse the document_id without the escaped characters and whitespace.
- document.body.contents[0]: Used to get the body tag's text
- str(text): Cast the iterable string to a normal string.
- re.sub(r"[\\]+","",text): Remove escaped backslash lines

Step #4 – Tokenisation
(block-4-tokeniser.py)

Tested Commands

- **python3 block-4-tokenizer.py -i articles.json -o tokens.json**
- ***python3 ./block-1-reader.py --path reuters21578 | python3 ./block-2-document-segmenter.py | python3 ./block-3-extractor.py | python3 ./block-4-tokenizer.py -o tokens.json***

==========

- Input: articles.json contains lists of dictionaries with ID and Text (Term-Document ID pairs)
- Output: Lists of Tokens [ID, Token]

==========

- **NLTK must be installed to tokenise the texts.
- **JSON is needed to parse the pipelined data
- word_tokenize(full_text): Use NLTK to tokenise the text.
- Since the pipelined document needs to be properly formatted into JSON, some replacements are needed.
    - data.replace("}","},")
    - data= re.sub(r"{","[{",data, count=1) #For start of index
    - data= re.sub(r"},$","}]",data, count=1)

Step #5 – Sorting & Building an Index
(block-5-sorter.py)

Tested Commands

- **python3 ./block-5-sorter.py -i tokens.json**

==========
- Input: Lists of Tokens [ID, Token]
- Output: A sorted dictionary with unique terms as the key and strings of docIDs as the value. Writes to a file known as "index.json"
==========

- (set(map(tuple,data)): Convert the list to tuple then remove the duplicate tuples. If you do set(data), it won't work because the data is a list.
- key=lambda token: (token[1], float(token[0]): Part of the sorted() function. Sort first by the second parameter which is the token, then by the first value which is the document ID. Note that a conversion is needed because the ID is seen as a string and it is sorting alphabetically rather than as a number.

- For indexing building, I'm using a dictionary(index).
- **index[current_token] = str(token[0]):** If the token doesn't exist, the token will be added to the dictionary. The value is the string version of the docID.
- **current_listing = index[current_token]**
  index[current_token] = current_listing + ", " + str(token[0]): If the token does exist, the value must be updated by appending the docID to the current docIDs.

- **json.dump(index, file):** Dumps dictionary in final "index.json" file.

Step #6: Querying
(query.py)

Tested Commands:

- **python3 query.py -i index.json -q here -s stopwords-150.json**

- **python3 query.py -i index.json -q ! -s stopwords-150.json**
- **python3 query.py -i index.json -q man -s stopwords-150.json**

==========
- Input: Term-Document Dictionary in JSON format, query term, stopwords list, output file name is optional.
- Output: Writes a query in JSON to a file(default: sampleQueries.json). Print docID associated with query term. Prints error if term is not in the index.
==========

except KeyError as err: In the case that the term is not found in the index, an error will be raised and it will print "not found".

Test Queries:

$ python3 query.py -i index.json -q man -s stopwords-150.json

Result:
28 269 1685 1895 2910 3864 4003 5030 5206 6343 6397 6471 8074 8089 8133 8246 8354 9126 9692 10486 12268 13249 14030 14635 15372 16789 17395 17397 17915 18064 18126 18132 18344 19247 19340 19353 21498

$ python3 query.py -i index.json -q ! -s stopwords-150.json

Result:
9519 12025 12797 12834 17483

$ python3 query.py -i index.json -q war -s stopwords-150.json

Result:
107 318 496 714 875 1093 1354 1410 1441 1634 1895 1899 1938 2034 3280 3374 3455 3854 3866 3869 4030 4267 4603 5160 5229 5285 5288 5298 5391 5656 5752 5786 5891 5895 5954 6058 6157 6404 6618 7176 7350 7753 7961 8088 8093 8421 8440 8483 8516 8675 8749 8991 9348 9537 10080 10395 10546 10618 10642 10643 10647 10665 10720 10779 10781 10861 10996 11175 11198 11215 11260 11287 11357 11460 11461 11487 11498 11545 11555 11780 12025 12039 12047 12457 12473 12603 12815 12848 12936 12968 13034 13046 13056 13074 13247 13253 13273 13320 13365 13420 13527 13529 13542 13576 13666 13749 13847 13993 14419 14538 14571 14611 14698 14838 14891 14939 15650 15658 15802 16072 16113 16139 16162 16220 16242 16256 16366 16423 16759 16794 16926 16935 16957 16958 17018 17023 17166 17198 17201 17245 17269 17274 17291 17325 17372 17419 17462 17780 17888 17925 18001 18231 18271 18357 18369 18418 18448 18568 18938 19285 19557 19559 19560 19684 19819 19882 20333 20464 20500 20522 20632 20741 20881 20890 20909 20991 21002 21006 21013 21131 21216 21369 21486 21506 21508 21539 21563 21568

$ python3 query.py -i index.json -q heist

Result:

('heist',) heist is not in the index.

Challenge Queries Results:
Unfiltered index (Comment out ***create_table_non_positional_index(args.input_file, args.query, args.stopwords***))

Final compressed index (***create_table_non_positional_index(args.input_file, args.query, args.stopwords)***) must be uncommented)

1. pineapple

Unfiltered index
$ python3 query.py -i index.json -q pineapple

Result:
4630

Final compressed index
$ python3 query.py -i index.json -q pineapple -s stopwords-150.json

Compressed (Non-Positional Index) Query Result:
['4630']

2. Phillippines

Unfiltered index
$ python3 query.py -i index.json -q Phillippines

Result:
('Phillippines',) Phillippines is not in the index.

Final compressed index
$ python3 query.py -i index.json -q Phillippines -s stopwords-150.json

Compressed (Non-Positional Index) Query Result:
('Phillippines',)
Phillippines is not in the index.

3. Brierley
Unfiltered index
$ python3 query.py -i index.json -q Brierley

Result:

125 907 946 3933 4064 7596 9210 9277 9779 10361 12610 13881 14208 16685 16872 17777 19984 21319

Final compressed index
$ python3 query.py -i index.json -q Brierley -s stopwords-150.json

Compressed (Non-Positional Index) Query Result:
('Brierley',)
Brierley is not in the index.

4. Chrysler
Unfiltered index
$ python3 query.py -i index.json -q Chrysler

Result:
627 678 1114 1121 1214 1408 1455 1539 1599 1682 1733 1736 1817 1854 2198 2235 3056 3100 3139 3152 3198 3241 3271 3319 3369 3373 3396 3645 3781 3795 3861 3866 3916 4043 4540 4582 4850 4869 4931 4979 4986 5027 5039 5131 5605 5631 6360 6380 6554 7418 7659 7796 8021 8258 8774 8848 8855 9046 9147 9325 10164 10187 11367 11413 11653 12095 12360 13001 13098 13216 13492 13884 13938 14102 14184 14352 14633 14830 15255 16038 16423 16643 16734 16840 16842 17086 17828 19127 19181 19395 19415 19685 19694 19722 19728 19770 20411 20452 20742 20889 21004 21180 21338 21341

Final compressed index
$ python3 query.py -i index.json -q Chrysler -s stopwords-150.json

Compressed (Non-Positional Index) Query Result:
('Chrysler',)
Chrysler is not in the index.

Size of	Word Types (Terms)			Non-Positional Postings		
	Dictionary			*Non-Positional Index*		
	Size	Δ	cml	Size	Δ	cml
Unfiltered	76841	0	0	1629687	0	0
No Numbers	52034	32	32	1510328	7	7
Case Folding	45262	13	45	1464951	3	10
30 stopwords	45233	0.06	45.06	1263599	13	23
150 stopwords	45113	0.3	45.36	1131430	10.5	33.5

Word Types (Terms) & Non-Positional Postings
No Numbers:

re.match(r'\d', no_number_terms): Checks if the first character of the string is a digit, if it's a digit, ignore this term.

no_number_terms[0] == "-": Checks if the first character starts with – which would not be considered a digit. This is also includes the vocabulary such as "-human". These terms will be ignored.

count_doc_id = count_doc_id + len(no_number_doc_id.split(" ")): Since the posting list is separated by spaces, it needs to parse. Len counts the # of elements in the list.

Case Folding:

```
if (casefold_terms.casefold() not in index3):
        index3[casefold_terms.casefold()] = casefold_doc_id
    else:
        index3[casefold_terms.casefold(
        )] = index3[casefold_terms.casefold()] + " " + casefold_doc_id
```

- If the element hasn't been added in the new index, add the casefolded key.
- Else, get the current key and append the docid.

for remove_duplicate_terms, remove_duplicate_doc_id in index3.items():
- Since we appended the doc_ids, there are duplicates in the list.

```
remove_duplicate_doc_id = remove_duplicate_doc_id.split(" ")
    remove_duplicate_doc_id = [float(i) for i in remove_duplicate_doc_id]
    remove_duplicate_doc_id = sorted(remove_duplicate_doc_id)  # Sorted but as float instead of string
```

- We parse the list of doc_ids and convert them to float so the sorting is done properly. If not, it would sort as a string (alphabetically) instead of numerically. But after this we still have the duplicates but just in order.

for index, elements in enumerate(remove_duplicate_doc_id):
remove_duplicate_doc_id[index] = str(elements)
remove_duplicate_doc_id = sorted(set(remove_duplicate_doc_id))
- We convert the list of doc_ids back into a string so we can remove the duplicate doc_ids via set() and then we sort back what's left.

for index, float_value in enumerate(remove_duplicate_doc_id):
remove_duplicate_doc_id[index] = str(int(float(float_value)))
- We have to actually convert it inside of the list instead of just using it temporary. We need to make sure that this float() is actually a float before using int() or this issue arises: *ValueError: invalid literal for int() with base 10: ''.*

for final_casefold_terms, final_casefold_doc_id in index3.items():
- Finally we get a list of casefolded terms with its doc_ids with no duplicates.

Stopwords:

for non_stopword_terms, non_stopword_doc_id in index3.items():
if non_stopword_terms not in stopwords:
The code within is the same to create the final index. It checks if the terms in the index are in the stopwords list. If it is not true, it will append that term to the final index.

Tested Queries:
$ python3 query.py -i index.json -q ! -s stopwords-150.json
Compressed (Non-Positional Index) Query Result:
['9519', '12025', '12797', '12834', '17483']

$ python3 query.py -i index.json -q man -s stopwords-150.json
Compressed (Non-Positional Index) Query Result:
('man',) man is not in the index.

$ python3 query.py -i index.json -q war -s stopwords-150.json
Compressed (Non-Positional Index) Query Result:
['107', '209', '243', '288', '318', '496', '714', '875', '1093', '1354', '1410', '1441', '1634', '1895', '1899', '1938', '2034', '3218', '3280', '3374', '3421', '3455', '3854', '3866', '3869', '4030', '4267', '4603', '4875', '5160', '5229', '5285', '5288', '5298', '5391', '5656', '5752', '5786', '5891', '5895', '5954', '6058', '6157', '6404', '6618', '6656', '7176', '7350', '7753', '7961', '8088', '8093', '8136', '8421', '8440', '8483', '8516', '8675', '8749', '8991', '9022', '9208', '9348', '9537', '9751', '9897', '10080', '10395', '10546', '10618', '10642', '10643',

'10647', '10665', '10720', '10779', '10781', '10861', '10996', '11175', '11198', '11215', '11260', '11287', '11357', '11460', '11461', '11487', '11498', '11545', '11555', '11780', '12025', '12039', '12047', '12142', '12457', '12473', '12603', '12650', '12815', '12848', '12887', '12936', '12968', '13034', '13046', '13056', '13074', '13247', '13253', '13273', '13320', '13365', '13420', '13527', '13529', '13542', '13576', '13666', '13749', '13847', '13993', '14419', '14538', '14571', '14611', '14698', '14838', '14852', '14891', '14939', '15650', '15658', '15802', '16072', '16113', '16139', '16162', '16220', '16242', '16256', '16366', '16423', '16759', '16794', '16926', '16935', '16957', '16958', '16970', '17018', '17023', '17166', '17198', '17201', '17245', '17269', '17274', '17291', '17305', '17325', '17372', '17402', '17419', '17462', '17780', '17862', '17888', '17925', '18001', '18231', '18271', '18340', '18357', '18369', '18418', '18448', '18525', '18568', '18938', '19285', '19403', '19557', '19559', '19560', '19588', '19684', '19819', '19882', '19918', '20333', '20464', '20500', '20522', '20614', '20624', '20632', '20741', '20828', '20860', '20881', '20882', '20890', '20899', '20909', '20963', '20991', '21002', '21006', '21013', '21131', '21216', '21369', '21486', '21506', '21508', '21517', '21539', '21563', '21568']

Findings:

- Removing numbers brought more change to the terms rather than the docIDs because those numbers no longer have to be indexed. The docIDs associated with those number keys are discarded too.
- Case folding helps normalised the terms thus duplicate terms can possibility appear. For this reason, after the case folding, we have to sort again the terms and add the additional posting ids those case folded terms and remove the duplicates. It is possible that there were duplicate posting ids in that case, it got removed as well.
- Removing stopwords did not do much for the terms in the dictionary because even if it got remove, it would be at most 30 or 150 terms. However for the docIDs, there's a big change because a stop word like 'a' may have all 21k docIDs so having it removed from the dictionary, reduces the # of docIDs.

As for the query, it would seem that the # of docIDs grew because of the case folding. However, a term like 'man' was part of the stopword list so it got removed and was no longer indexable.

As for the challenged queries, most of them of provide docIDs after querying on the unfiltered index but in the compressed index, only one of the gave the docID because the case folding normalised the term so it is no longer indexable.

What I learnt from this project is how to reuse last project's code to build the index. I learnt how to remove the duplicates and sort the terms as well as how to compress the dictionary and its effect on the querying.