CMPE 493
Information Retrieval
Assignment-1 Report

Uğurcan Arıkan
2014400111
Boğaziçi University Computer Engineering

CMPE 493
Assignment Report

Introduction

In this project, we were supposed to process the Reuters-21578 dataset, perform tokenizing operations such as case folding and punctuation removal, remove stop words given, then construct inverted-index file and bigrams file to be used in conjunction, disjunction and wildcard query operations without using any 3rd party libraries.

Implementation Details

Project source code consists of 2 python scripts: preprocess.py and process.py. Preprocess.py handles the preprocessing of the corpus: It removes punctuation marks, case folds and removes stopwords.

Process.py handles the processing of queries.

Preprocess.py and Preprocessing

For the preprocessing step of the corpus, preprocess.py is used. In order for it to work as expected, file structure must be the same as the workspace given in the .zip file in moodle. prepocess.py should be under the "Code" folder and reuters dataset must be in the "reuters21578" folder which is in the same directory as the "Code" folder and the file names should also be in the same format as given in the moodle. Also, punctuations.txt and stopwords.txt should be on the parent directory of the preprocess.py script.

Preprocessing first begins by reading the stopwords.txt and punctuations.txt and putting them in different lists in order to store those and be able to use them while removing them.

Then the reuters dataset begins to be read. Files are opened with latin-1 encoding and until a line whose first characters are “<TEXT" is read, reading continues.

Once the “<TEXT" line is read, in_text variable that determines if the cursor is inside the text tag turns true and all the text inside is appended to the text variable. This process continues until “</TEXT>\n” in encountered. To put it more simply, text is read until “<TEXT" is encountered, once it is encountered texts start to be recorded until “</TEXT>\n" is encountered this time.

Once the end of text tag “</TEXT>\n” is encountered, counter that keeps the document id being read is incremented and all the text recorded and stored under the “text” variable is processed.

First all the HTML tags are removed such as text, title and body tag. At that point, “text” variable is the news of the document being read. This is appended to “corpus” variable to process number of tokens and terms afterwards.

Then punctuations are replaced with space, text is lower-folded and it is appended to “corpus_before_stopword_removal” variable this time.

After that step, stop words are removed from the text and text is appended to the “corpus_after_stopword_removal” variable. At this stage, news part of the document being read is fully preprocessed and ready for being used for the inverted_index construction at the next step.

At this step, each word is added to the inverted_index with the current document id. Data structure used for the inverted index was dictionary.

After the construction of inverted_index, bigrams are constructed with the use of inverted_index and both are written to the files as described in the project decription.

At that stage, corpus, corpus_before_stopword_removal and corpus_after_stopword_removal are also kept. Those variables are used to answer questions in the project description. corpus keeps the dataset before removal of punctuation marks, stopwords and case-folding. corpus_before_stopword_removal keeps corpus after removal of punctuation marks and case-folding. corpus_after_stopword_removal keeps corpus after removal of stopwords. Number of words before stop word removal are counted by turning corpus_before_stopword_removal into list and getting its length. Same is done to corpus_after_stopword_removal to find number of words after stop word removal. Number of terms before case-folding and stopword removal are found by turning corpus into list, then

set to remove duplicates and finally get its length. Same is done to corpus_after_stopword_removal to find the number of terms after case-folding and stopword removal. Most frequent words before stopword removal and case-folding are found by using collections.Counter on corpus. Same is done to corpus_after_stopword_removal to find most frequent words before stopword removal and case-folding.

(a) How many tokens does the corpus contain before stopword removal?

2980809

(b) How many tokens does the corpus contain after stopword removal?

2353548

(c) How many terms (unique tokens) are there before stopword removal and case-folding?

134903

(d) How many terms (unique tokens) are there after stopword removal and case-folding?

47712

(e) List the top 20 most frequent terms before stopword removal and case-folding?

[('the', 119852), ('of', 72350), ('to', 68639), ('and', 53431), ('in', 49993), ('a', 48314), ('said', 35789), ('for', 25206), ('mln', 24926), ('The', 22866), ('&#2;', 21578), ('&#3;', 21578), ('on', 17754), ('it', 17525), ('is', 16691), ('said.', 15817), ('dlrs', 15563), ('from', 14968), ('that', 14866), ('its', 14750)]

(f) List the top 20 most frequent terms after stopword removal and case-folding?

[('to', 73332), ('said', 53200), ('s', 32781), ('2', 32381), ('3', 30256), ('mln', 26883), ('dlrs', 21465), ('on', 19464), ('reuter', 19159), ('pct', 18190), ('1', 17450), ('lt', 16741), ('that', 15558), ('from', 15337), ('by', 15180), ('vs', 14988), ('at', 14623), ('000', 13613), ('year', 13181), ('u', 11445)]

Queries

All the queries were processed successfully in my trials, using process.py script as given in the project description example. It successfully processed conjunctive and disjunctive operations, as well as wildcard queries with one wildcard only.

```
UA-Mac:Code ugurcan$ python3 process.py 1 "arab"
[236, 242, 247, 248, 288, 349, 668, 723, 1093, 1899, 1938, 1956, 1990, 2121, 2187, 2324, 3452, 3479, 3571,
4067, 4101, 4171, 4742, 5193, 5237, 5272, 5400, 5588, 5796, 6137, 6177, 6348, 6444, 6576, 6618, 7203, 7350,
8088, 8142, 8493, 8796, 9170, 9208, 10617, 10618, 10628, 10632, 10633, 10638, 10647, 10691, 10727, 10759,
10782, 10934, 10947, 10948, 11149, 11780, 11838, 13236, 13527, 14664, 14917, 16120, 16366, 16658, 16761, 17
196, 17220, 17247, 17310, 17436, 17584, 17780, 17924, 17925, 17926, 17928, 18329, 18477, 18521, 18620, 1916
4, 19491, 19499, 19529, 19594, 19819, 19979, 20522, 21006, 21131, 21574]
UA-Mac:Code ugurcan$ python3 process.py 1 "arab AND arabic"
[]
UA-Mac:Code ugurcan$ python3 process.py 1 "arab AND arabs"
[17220]
UA-Mac:Code ugurcan$ python3 process.py 1 "arab AND arabs AND arabia"
[17220]
UA-Mac:Code ugurcan$ python3 process.py 1 "car AND people AND city"
[14419]
UA-Mac:Code ugurcan$ python3 process.py 1 "car AND people"
[269, 2853, 2944, 5376, 6346, 7933, 8583, 8679, 9046, 9110, 11624, 13609, 13652, 14419, 15372, 16860, 17318
, 19392, 19415]
UA-Mac:Code ugurcan$ python3 process.py 1 "car AND people AND new"
[269, 5376, 6346, 7933, 9046, 11624, 14419, 15372, 16860]
UA-Mac:Code ugurcan$ python3 process.py 1 "car AND people AND new AND york"
[]
```

```
UA-Mac:Code ugurcan$ python3 process.py 2 "arabes"
[288, 6962]
UA-Mac:Code ugurcan$ python3 process.py 2 "arabes OR arabes"
[288, 288, 6962, 6962]
UA-Mac:Code ugurcan$ python3 process.py 2 "arabes OR arabes OR araps"
[288, 288, 6962, 6962]
UA-Mac:Code ugurcan$ python3 process.py 2 "arabes OR arabes OR arabia"
[247, 248, 254, 273, 288, 288, 288, 349, 352, 668, 1067, 1306, 1387, 1956, 1990, 2121, 2172
, 2383, 2522, 3452, 3455, 3499, 4067, 4101, 4138, 4246, 4593, 4689, 5123, 5125, 5167, 5170,
 5171, 5182, 5184, 5193, 5244, 5273, 5281, 5599, 5851, 5953, 6060, 6137, 6576, 6962, 6962,
6989, 6996, 7350, 8088, 8095, 8796, 9170, 9293, 10078, 10175, 10632, 10633, 10934, 11885, 1
2196, 12770, 13096, 13130, 13256, 13281, 13340, 13497, 13795, 14230, 14689, 14716, 14850, 1
4863, 15975, 16193, 16292, 16362, 16762, 17196, 17220, 17291, 17359, 17365, 17419, 17519, 1
7584, 17767, 17769, 17928, 17929, 17963, 18263, 18621, 19110, 19285, 19397, 19403, 19491, 1
9499, 19505, 19559, 20891, 21058, 21131, 21169, 21293, 21370, 21486]
```

```
UA-Mac:Code ugurcan$ python3 process.py 3 "ark*ansas"
[2481, 2486, 4348, 4371, 6625, 6777, 6784, 7679, 8273, 8705, 9653, 10487, 11093, 11466, 11715, 13555, 145
31, 17830, 20284]
UA-Mac:Code ugurcan$ python3 process.py 3 "ark*as"
[2481, 2486, 4348, 4371, 6625, 6777, 6784, 7679, 8273, 8705, 9653, 10487, 11093, 11466, 11715, 13555, 145
31, 17830, 20284]
UA-Mac:Code ugurcan$ python3 process.py 3 "*nsas"
[97, 178, 365, 386, 674, 710, 1497, 1908, 2481, 2486, 3110, 3270, 3918, 4348, 4371, 4522, 5003, 5044, 506
5, 5322, 5345, 5539, 6006, 6021, 6079, 6209, 6296, 6588, 6625, 6656, 6777, 6784, 6793, 7151, 7326, 7679,
7792, 7973, 8209, 8265, 8273, 8705, 8739, 8943, 9328, 9609, 9626, 9637, 9646, 9653, 10319, 10487, 11056,
11075, 11093, 11111, 11343, 11380, 11399, 11403, 11466, 11516, 11715, 12497, 12581, 12773, 13549, 13555,
13655, 13848, 13852, 13968, 13978, 13986, 14011, 14014, 14046, 14078, 14121, 14133, 14134, 14138, 14157,
14159, 14167, 14192, 14241, 14243, 14269, 14306, 14347, 14368, 14385, 14392, 14408, 14474, 14518, 14531,
14578, 14581, 14636, 14675, 14692, 14694, 14702, 14808, 15282, 15511, 15806, 16271, 16335, 16887, 17083,
17137, 17332, 17830, 17981, 18064, 18417, 18984, 19940, 20284, 20403, 20435, 20611, 21426]
UA-Mac:Code ugurcan$ python3 process.py 3 "k*nsas"
[97, 178, 365, 386, 674, 710, 1497, 1908, 3110, 3270, 3918, 4522, 5003, 5044, 5065, 5322, 5345, 5539, 600
6, 6021, 6079, 6209, 6296, 6588, 6656, 6793, 7151, 7326, 7792, 7973, 8209, 8265, 8705, 8739, 8943, 9328,
9609, 9626, 9637, 9646, 10319, 11056, 11075, 11111, 11343, 11380, 11399, 11403, 11516, 12497, 12581, 1277
3, 13549, 13555, 13655, 13848, 13852, 13968, 13978, 13986, 14011, 14014, 14046, 14078, 14121, 14133, 1413
4, 14138, 14157, 14159, 14167, 14192, 14241, 14243, 14269, 14306, 14347, 14368, 14385, 14392, 14408, 1447
4, 14518, 14531, 14578, 14581, 14636, 14675, 14692, 14694, 14702, 14808, 15282, 15511, 15806, 16271, 1633
5, 16887, 17083, 17137, 17332, 17981, 18064, 18417, 18984, 19940, 20403, 20435, 20611, 21426]
UA-Mac:Code ugurcan$ python3 process.py 3 "k*nsaasdasxs"
[]
```

How to run

Project was written with Python 3.6.1 in MacOS Mojave 10.14.3. It does not use any 3rd party library.

To start the preprocessing and create index and bigrams files, run

python3 preprocess.py

To run queries, run the exact same command in the project report:

python3 process.py *query*

i.e python3 process.py 1 “car AND people”.