Information Retrieval

Assignment - 1

Abdullah Atakan Guney
2018700069

March 17, 2019

Contents

1 Description

In this assignment, I have implemented a document retrieval system for simple boolean and wildcard queries by using the bigram indexing schema. I have used Reuters-21578 data set, which is provided.

2 Pre-processing

In pre-processing part, I have followed following steps:

- Extract text from Reuters-21578 data and get dictionary of id: text. Id is the "NEWID" attribute of <REUTERS> tag as described in project description, and texts are combination of contents of <TITLE> and <BODY> tag. I have concatenated contents of those tags by a space.

  {

  1: "text 1"

  2: "text 2"

  ⋮

  n: "text n"

  }

- Replace each punctuation by a space in texts
- Split by any white space to get tokens from texts
- Apply case-folding to any token in documents
- Remove stop-words from tokens for each document

At the end, I got a dictionary which consists of {id: tokens}.
{
1: [token1, token2, ...]

2: [token2, token3, ...]

⋮

n: [token343, token311, ...]

}

2.1 Answers to Some Questions

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

"Number of tokens before stopword removal: 2902785"

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

"Number of tokens after stopword removal: 2227436"

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

"Number of terms before stopword removal and casefolding: 61462"

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

"Number of terms after stopword removal and casefolding: 45346"

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

Top 20 terms before stopword removal and casefolding:

Term: the, Frequency: 120018

Term: of, Frequency: 72267

Term: to, Frequency: 68785

Term: and, Frequency: 53476

Term: said, Frequency: 52894

Term: in, Frequency: 50021

Term: a, Frequency: 48440

Term: 3, Frequency: 26814

Term: for, Frequency: 25595

Term: mln, Frequency: 25591

Term: The, Frequency: 24278

Term: s, Frequency: 20526

Term: dlrs, Frequency: 20306

Term: it, Frequency: 18098

Term: on, Frequency: 18032

Term: pct, Frequency: 17130

Term: is, Frequency: 16739

Term: lt, Frequency: 16666

Term: 1, Frequency: 15895

Term: that, Frequency: 15193

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

Top 20 terms after stopword removal and casefolding:

Term: to, Frequency: 73074

Term: said, Frequency: 53096

Term: s, Frequency: 32590

Term: 3, Frequency: 26814

Term: mln, Frequency: 26732

Term: dlrs, Frequency: 21273

Term: reuter, Frequency: 18964

Term: pct, Frequency: 18046

Term: lt, Frequency: 16680

Term: 1, Frequency: 15895

Term: from, Frequency: 15277

Term: vs, Frequency: 14836

Term: at, Frequency: 14517

Term: 000, Frequency: 13448

Term: year, Frequency: 13109

Term: u, Frequency: 11326

Term: billion, Frequency: 10726

Term: has, Frequency: 10185

Term: 2, Frequency: 9996

Term: company, Frequency: 9699

3 Conjunctive Queries

For conjunctive queries, first I have split the query string by " AND " to get each keyword. Then I applied pre-processing which is only case-folding, to each keyword. Then, I searched each keyword in inverted index, and return intersection of each postings of matched index. Screen shot of a running conjunctive query is:

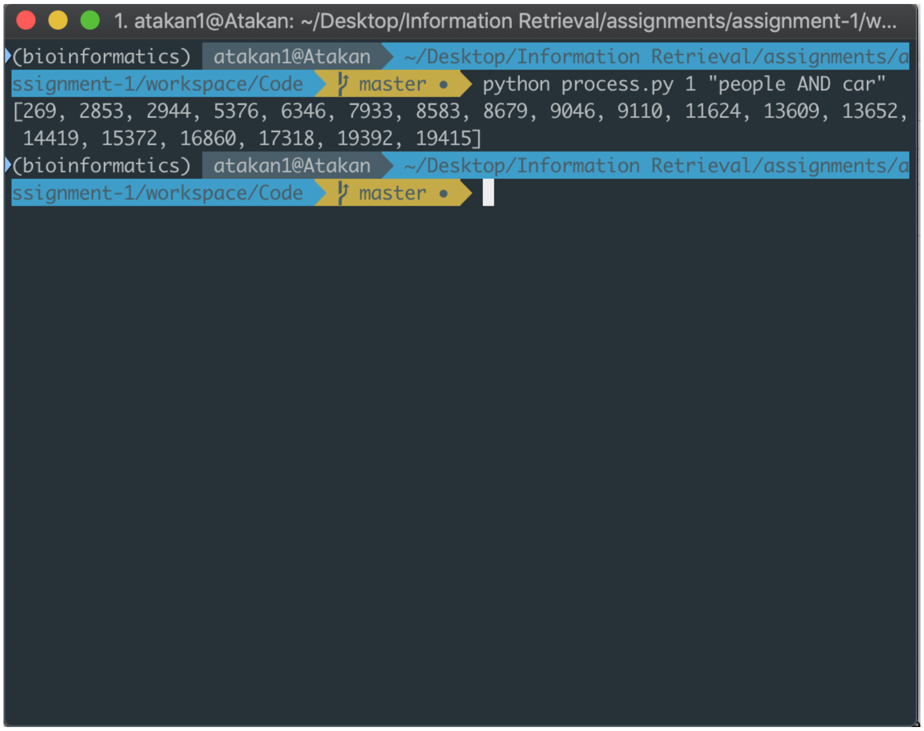


Figure 1: A Conjunctive Query Example

4 Disjunctive Queries

For disjunctive queries, first I have split the query string by " OR " to get each keyword. Then I applied pre-processing which is only case-folding, to each keyword. Then, I searched each keyword in inverted index, and return union of each postings of matched index. Screen shot of a running conjunctive query is:

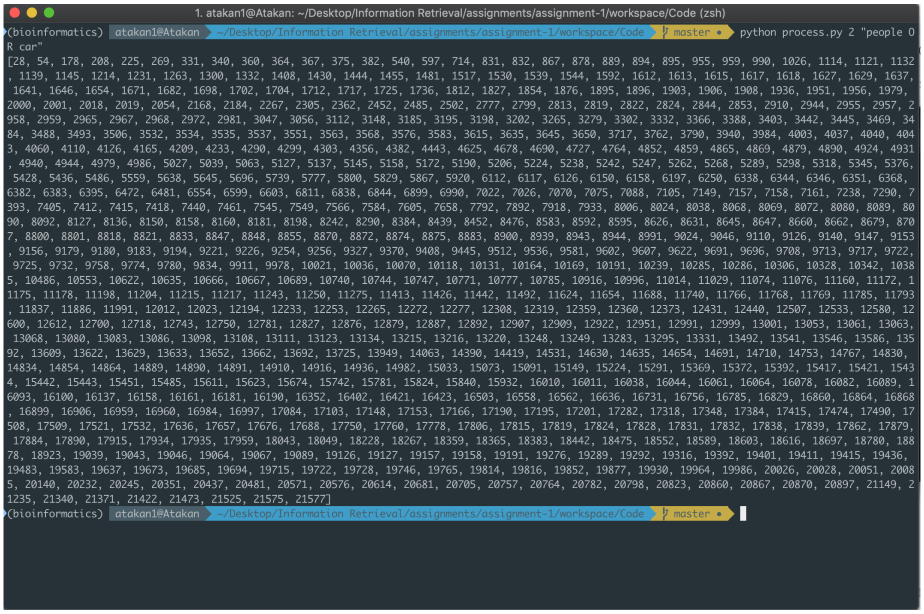


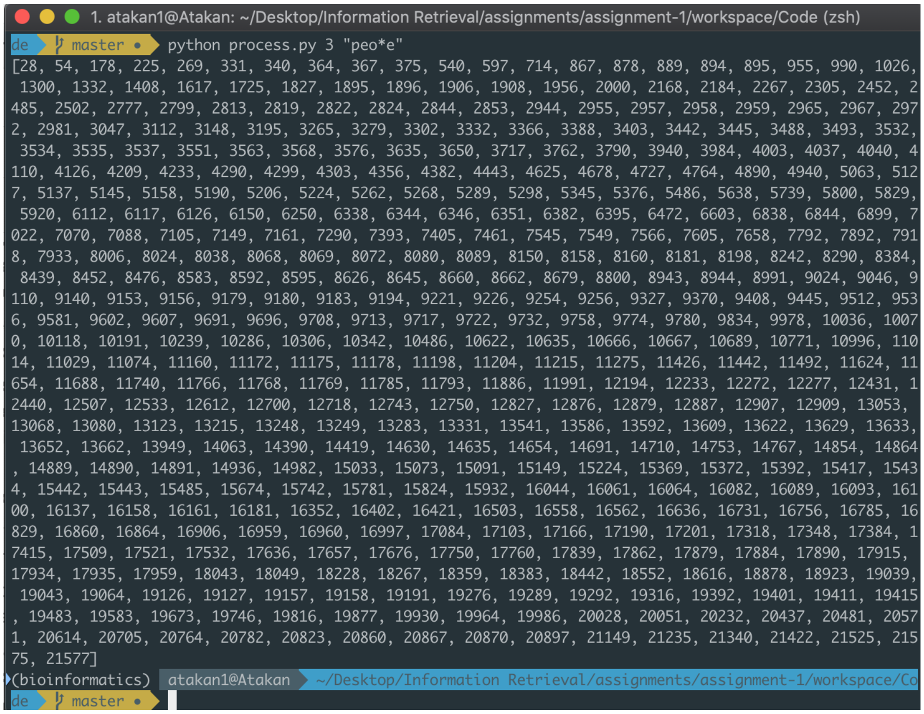
Figure 2: A Disjunctive Query Example

5 Wildcard Queries

For wildcard queries, first I split the query by "*" and get bi-grams for 2 parts of the query, if "*" appears in the query. Then, I have extracted bi-grams in the following way.

- For the first part of query the matched terms must start with that part, so I have constructed bi-grams for "$'first part'"
- For the second part of the query, the matched terms must end with this part, so I have constructed bi-grams for "'second part'$", then I took the intersection of each terms lists of bi-grams
- I applied post-filtering, i.e. I removed the words that are not beginning with the first part and ending with the second part

Here is the example program running on a wildcard query:



```
1. atakan1@Atakan: ~/Desktop/Information Retrieval/assignments/assignment-1/workspace/Code (zsh)
de  master •  python process.py 3 "peo*e"
[28, 54, 178, 225, 269, 331, 340, 364, 367, 375, 540, 597, 714, 867, 878, 889, 894, 895, 955, 990, 1026,
 1300, 1332, 1408, 1617, 1725, 1827, 1895, 1896, 1906, 1908, 1956, 2000, 2168, 2184, 2267, 2305, 2452, 2
485, 2502, 2777, 2799, 2813, 2819, 2822, 2824, 2844, 2853, 2944, 2955, 2957, 2958, 2959, 2965, 2967, 297
2, 2981, 3047, 3112, 3148, 3195, 3265, 3279, 3302, 3332, 3366, 3388, 3403, 3442, 3445, 3488, 3493, 3532,
 3534, 3535, 3537, 3551, 3563, 3568, 3576, 3635, 3650, 3717, 3762, 3790, 3940, 3984, 4003, 4037, 4040, 4
110, 4126, 4209, 4233, 4290, 4299, 4303, 4356, 4382, 4443, 4625, 4678, 4727, 4764, 4890, 4940, 5063, 512
7, 5137, 5145, 5158, 5190, 5206, 5224, 5262, 5268, 5289, 5298, 5345, 5376, 5486, 5638, 5739, 5800, 5829,
 5920, 6112, 6117, 6126, 6150, 6250, 6338, 6344, 6346, 6351, 6382, 6395, 6472, 6603, 6838, 6844, 6899, 7
022, 7070, 7088, 7105, 7149, 7161, 7290, 7393, 7405, 7461, 7545, 7549, 7566, 7605, 7658, 7792, 7892, 791
8, 7933, 8006, 8024, 8038, 8068, 8069, 8072, 8080, 8089, 8150, 8158, 8160, 8181, 8198, 8242, 8290, 8384,
 8439, 8452, 8476, 8583, 8592, 8595, 8626, 8645, 8660, 8662, 8679, 8800, 8943, 8944, 8991, 9024, 9046, 9
110, 9140, 9153, 9156, 9179, 9180, 9183, 9194, 9221, 9226, 9254, 9256, 9327, 9370, 9408, 9445, 9512, 953
6, 9581, 9602, 9607, 9691, 9696, 9708, 9713, 9717, 9722, 9732, 9758, 9774, 9780, 9834, 9978, 10036, 1007
0, 10118, 10191, 10239, 10286, 10306, 10342, 10486, 10622, 10635, 10666, 10667, 10689, 10771, 10996, 110
14, 11029, 11074, 11160, 11172, 11175, 11178, 11198, 11204, 11215, 11275, 11426, 11442, 11492, 11624, 11
654, 11688, 11740, 11766, 11768, 11769, 11785, 11793, 11886, 11991, 12194, 12233, 12272, 12277, 12431, 1
2440, 12507, 12533, 12612, 12700, 12718, 12743, 12750, 12827, 12876, 12879, 12887, 12907, 12909, 13053,
13068, 13080, 13123, 13215, 13248, 13249, 13283, 13331, 13541, 13586, 13592, 13609, 13622, 13629, 13633,
 13652, 13662, 13949, 14063, 14390, 14419, 14630, 14635, 14654, 14691, 14710, 14753, 14767, 14854, 14864
, 14889, 14890, 14891, 14936, 14982, 15033, 15073, 15091, 15149, 15224, 15369, 15372, 15392, 15417, 1543
4, 15442, 15443, 15485, 15674, 15742, 15781, 15824, 15932, 16044, 16061, 16064, 16082, 16089, 16093, 161
00, 16137, 16158, 16161, 16181, 16352, 16402, 16421, 16503, 16558, 16562, 16636, 16731, 16756, 16785, 16
829, 16860, 16864, 16906, 16959, 16960, 16997, 17084, 17103, 17166, 17190, 17201, 17318, 17348, 17384, 1
7415, 17509, 17521, 17532, 17636, 17657, 17676, 17750, 17760, 17839, 17862, 17879, 17884, 17890, 17915,
17934, 17935, 17959, 18043, 18049, 18228, 18267, 18359, 18383, 18442, 18552, 18616, 18878, 18923, 19039,
 19043, 19064, 19126, 19127, 19157, 19158, 19191, 19276, 19289, 19292, 19316, 19392, 19401, 19411, 19415
, 19483, 19583, 19673, 19746, 19816, 19877, 19930, 19964, 19986, 20028, 20051, 20232, 20437, 20481, 2057
1, 20614, 20705, 20764, 20782, 20823, 20860, 20867, 20870, 20897, 21149, 21235, 21340, 21422, 21525, 215
75, 21577]
(bioinformatics)  atakan1@Atakan  ~/Desktop/Information Retrieval/assignments/assignment-1/workspace/Co
de  master •
```

Figure 3: A Wildcard Query Example