CSC 575 Intelligent Information Retrieval

Siravich Khongrod (In-class)

CSC 575 HW#4

http://condor.depaul.edu/ntomuro/courses/575/assign/HW4.html (http://condor.depaul.edu/ntomuro/courses/575/assign/HW4.html)
https://nbviewer.jupyter.org/url/condor.depaul.edu/ntomuro/courses/575/assign/575hw4.ipynb
(https://nbviewer.jupyter.org/url/condor.depaul.edu/ntomuro/courses/575/assign/575hw4.ipynb)

Overview:

Implement the 'Inverted Index Retrieval Algorithm' (in lecture note (#6) (http://condor.depaul.edu/ntomuro/courses/575/notes/VS-Retrieval.pptx)) and the evaluation metric Mean Average Precision (MAP) (in lecture note (#7) (http://condor.depaul.edu/ntomuro/courses/575/notes/Evaluation.pptx)), and apply to a corpus called Medline collection (http://ir.dcs.gla.ac.uk/resources/test_collections/).

The Medline collection is one of the Information Retrieval (IR) standard test collections, which have been used by many researchers as benchmark to evaluate IR systems. It contains 1033 documents (abstracts of papers published on Medline), 30 queries and relevance judgments of all query-document pairs.

Programming: Vector-space Retrieval & Evaluation -- Partially filled code

(1) Step 1: Load Inverted Index (H) and compute DocLen (DL).

In [3]:
```python
import csv
import math

tindexfile = 'medline_term_index.csv'
invindexfile = 'medline_inverted_index.csv'
dindexfile = 'medline_doc_index.csv'

# Number of documents in the corpus (hard-coded for this corpus)
N = 1033

# Major data structures
H_invindex = {} # inverted index; term -> (idf, L:hashmap of (docID . tf))
DL_doclen = {}  # document lengths; docID -> len

## (1) Read the term index file and populate the invindex first
tid2term_map = {} # temporary storage to hold mappings of termID -> term

fin = open(tindexfile, 'r', encoding='utf-8')
reader = csv.reader(fin, delimiter='\t')
for line in reader:
    term = line[0]    # term string
    termID = line[1]  # termID
    df = int(line[2]) # document frequency
    idf = math.log10(N/df) # idf
    # record term -> (idf, emptyL) in H
    H_invindex[term] = (idf, dict())
    # record termID -> term
    tid2term_map[termID] = term
fin.close()

## (2) Read the inverted index file and add postings lists in H.
## Also compute document lengths too, incrementally -- and record in DL.
fin = open(invindexfile, 'r')
reader = csv.reader(fin, delimiter='\t')
for line in reader:
    termID = line[0]
    idx = 1
    while idx < (len(line)-1):
        docID = line[idx]
        tf = int(line[idx+1]) # raw tf of the term in this document
        # Record docID -> tf in term's L
        L = (H_invindex[tid2term_map[termID]])[1]
        L[docID] = tf  # docID -> raw term frequency

        # Accumulate the component vector length for the document
        tfidf = tf * (H_invindex[tid2term_map[termID]])[0] # tf * idf
        tfidfsq = math.pow(tfidf, 2.0)
        if docID in DL_doclen:
            DL_doclen[docID] += tfidfsq
        else:
            DL_doclen[docID] = tfidfsq
        #
        idx += 2
fin.close()

# Fix the DL entries by applying sqrt to make vector length.
for docID in DL_doclen.keys():
    val = DL_doclen[docID]
    DL_doclen[docID] = math.sqrt(val)


print ('Total # terms: %d' % len(H_invindex))
for term in ['pentobarbit', 'defici', 'treatment']:
    print (' - Entry for \'%s\': df=%s, idf=%s' % (term, len(H_invindex[term][1]), H_invindex[term][0]))

print ('\nTotal # documents: %d' % len(DL_doclen))
for docID in ['59', '1033']:
    print (' - Vector len for Doc %s = %s' % (docID, DL_doclen[docID]))
```

```
Total # terms: 11463
 - Entry for 'pentobarbit': df=4, idf=2.412040330191658
 - Entry for 'defici': df=39, idf=1.4230357144931214
 - Entry for 'treatment': df=172, idf=0.7785718746120717

Total # documents: 1033
 - Vector len for Doc 59 = 13.811725366348801
 - Vector len for Doc 1033 = 31.163653356034512
```

(2) Step 2: Queries as Vectors

In [4]:
```
import re
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

queryfile = 'medline.query'

# A list of queries. Each Query is a tuple, (qID, Q:term->tf map)
Queries_list = []

fin = open(queryfile, 'r', encoding='utf-8')#'iso-8859-1')
porter = nltk.PorterStemmer()

for line in fin:
    matchObj = re.match(r'^(\d+)\s+(.*)', line)
    if not matchObj:
        print ("ERROR with line -- %s" % line)
    else:
        queryID = matchObj.group(1) # queryID
        text = matchObj.group(2)    # query string (ignoring sentences)

        # process text string -- same processing as one applied to documents.
        tokens = word_tokenize(text.lower())
        terms = [porter.stem(w) for w in tokens if w not in stopwords.words('english') and len(w) > 1] # (**)
        # term frequencies of the terms in this query are obtained by NLTK's FreqDist
        fdist = nltk.FreqDist(terms)
        # append the Query in the list
        Queries_list.append((queryID, dict(fdist)))
fin.close()

print ('Total # queries: %d' % len(Queries_list))
for qid in [1, 21]:
    print (' - Query %s: %s' % (Queries_list[qid][0], Queries_list[qid][1]))
```

```
Total # queries: 30
 - Query 2: {'relationship': 1, 'blood': 1, 'cerebrospin': 1, 'fluid': 1, 'oxygen': 1, 'concentr': 1, 'partial': 1, 'pr
essur': 1, 'method': 1, 'interest': 1, 'polarographi': 1}
 - Query 22: {'mycoplasma': 1, 'infect': 1, 'presenc': 1, 'embryo': 1, 'fetu': 1, 'newborn': 1, 'infant': 1, 'anim': 1,
'pregnanc': 1, 'gynecolog': 1, 'diseas': 1, 'relat': 1, 'chromosom': 2, 'abnorm': 1}
```

In [5]:
```
print ('Total # terms: %d' % len(H_invindex))
for term in ['pentobarbit']:
    print (' - Entry for \'%s\': df=%s, idf=%s' % (term, len(H_invindex[term][1]), H_invindex[term][0]))
    print(H_invindex[term])
    print()

print ('\nTotal # documents: %d' % len(DL_doclen))
for docID in ['59', '1033']:
    print (' - Vector len for Doc %s = %s' % (docID, DL_doclen[docID]))
```

```
Total # terms: 11463
 - Entry for 'pentobarbit': df=4, idf=2.412040330191658
(2.412040330191658, {'187': 1, '301': 1, '416': 1, '419': 1})


Total # documents: 1033
 - Vector len for Doc 59 = 13.811725366348801
 - Vector len for Doc 1033 = 31.163653356034512
```

(3) Retrieval and Ranking -- Step 3,4,5 for each Query

- For each query, follow Step 3,4,5 of the Vector Space Retrieval Algorithm and obtain a ranked list of retrieved documents (sorted by the cosine measure) -- 'R' in the algorithm.
- Then save the ranked lists in a list (in the same order of the query) -- to be used in the next step/Evaluation.
- (**) Also, WRITE the ranked lists to an output file. See the homework page for details.

Write the ranked list of documents (sorted by the descending order of cosine) to an output file.

- Name the file "rankedlist.txt".
- For each query, write the query ID number, followed by the document ID numbers in the rank order, in one line, separated by a comma. For example the beginning part of the first three lines should look like:
  1, 72, 500, 965, 360, 171, 15, 166, 181, 513, 511, ..
  2, 258, 712, 289, 162, 299, 187, 237, 96, 291, 192, ..
  3, 70, 62, 160, 286, 71, 230, 234, 407, 276, 59, ..

In [6]:
```
i = 0
for key in H_invindex:
    print(key+' '+str(H_invindex[key]))
    if(i==5):break
    i+=1
```

```
'' (1.4230357144931214, {'497': 1, '545': 1, '556': 1, '560': 1, '565': 1, '592': 2, '602': 2, '604': 4, '606': 1, '60
8': 6, '611': 1, '612': 2, '613': 1, '615': 2, '624': 1, '625': 1, '630': 2, '633': 2, '640': 1, '647': 1, '660': 1, '6
78': 1, '691': 1, '699': 1, '709': 1, '713': 1, '715': 2, '720': 2, '738': 1, '740': 3, '743': 1, '750': 1, '752': 1,
'789': 1, '798': 1, '801': 1, '807': 1, '808': 3, '817': 2})
'a (2.7130703258556395, {'421': 1, '609': 1})
'achondroplast (3.0141003215196207, {'576': 1})
'adequ (3.0141003215196207, {'421': 1})
'agnos (3.0141003215196207, {'358': 2})
'air (3.0141003215196207, {'70': 1})
```

```
In [223]: # RETRIEVE IDF IN QUERY
from collections import defaultdict
from scipy.spatial import distance
import numpy as np
import pandas as pd
docs_tfidf = defaultdict(list)
i=0
f=open('rankedlist.txt','w')

############### For each token, T, in Q: ###############
for q in Queries_list:
#     doc_w_terms=defaultdict(list)
#     doc_w_terms=[]
    q_tfidf=[]
    doc_score={}
########################################################################################################################
#
### Let Ibe the IDF of T, and Kbe the frequency of T in Q;
### Set the weight of T in Q: W= K*I; (tf*idffor T in Q)
### If a term in a query was not in the vocabulary (the dictionary structure made from the documents), ignore/skip it.
    for token in (q[1].keys()): # ITERATE OVER QUERY TERMS
        if(token not in H_invindex.keys()):
            continue
#         print("QUERY TERM: "+token+' '+str(H_invindex[token][0])+'x'+str(q[1][token]))
        qt_tfidf=H_invindex[token][0]*q[1][token]
        q_tfidf.append(qt_tfidf)
#         print("DOCS: ",end="")
#         print(H_invindex[token][1])
########################################################################################################################
#
###     Let L be the posting list of T from H;
###     For each HashMap, O, in L:
        for docid in H_invindex[token][1].keys():
#             print('\t'+str(docid)+': '+str(H_invindex[token][1][docid])+' x '+str(H_invindex[token][0])+' = '
#                   +str(H_invindex[token][1][docid]*H_invindex[token][0]))
########################################################################################################################
#
###         Increment D’s score by W* C * I; (tf*idffor T in Q x tf*idffor T in D)
            if (docid not in doc_score.keys()):
                doc_score[docid]=0
            doc_score[docid]+=qt_tfidf*H_invindex[token][1][docid]*H_invindex[token][0]
########################################################################################################################
#
### Compute the length, L, of the vector Q (square-root of the sum of the squares of its weights).
    q_len=sum(np.array(q_tfidf)**2)
### For each retrieved document D in R:
###     Let S be the current accumulated score of D;
###     (S is the dot-product of D and Q)
###     Let Y be the length of D in DL;
###     Let D’s final score to be S/(L * Y); (the cosine)
    for docid in doc_score.keys():
        doc_score[docid] = doc_score[docid] / (q_len*DL_doclen[docID])
    if(i<3):print(q[0], end=" ")
    f.write(q[0]+' ')
    for w in sorted(doc_score, key=doc_score.get, reverse=True):
        if(i<3):print(w,end=' ')
#             print(str(w) + '('+str(doc_score[w])+')', end=" ")
        f.write(w+' ')
    f.write('\n')
    if(i<3):print('\n')

    i+=1
#     if(i==3):break
f.close()
```

```
1 500 965 72 15 212 166 513 499 511 360 181 171 182 637 168 509 206 142 172 512 401 13 14 79 164 165 167 169 170 183 18
4 727 138 180 175 336 549 58 65 570 41 403 542 185 186 211 504 506 510 99 177 640 758 838 259 645 11 173 174 209 256 54
0 541 642 700 763 876 899 986 112 913 125 127 231 469 213 501 502 503 505 507 508 9 78 130 158 163 178 215 333 404 445
567 578 584 589 590 593 603 652 654 658 660 730 734 816 869 870 873 880 900 905 4 24 35 75 86 100 114 156 160 162 195 1
96 208 214 220 229 234 253 262 267 300 322 326 342 372 374 383 421 442 454 464 480 485 496 619 627 650 655 685 715 719
726 732 743 744 769 797 799 811 826 831 835 849 863 886 888 896 907 950 981 999 1001 1029 42 67 70 82 83 87 95 108 176
188 189 207 248 282 299 367 375 408 466 486 516 521 557 569 573 576 577 580 581 585 604 606 608 631 632 643 648 664 800
809 812 872 875 909 910 927 984 1020

2 712 289 258 162 291 187 299 313 243 292 713 418 192 296 293 80 237 297 708 760 715 157 302 420 3 862 301 118 236 974
880 96 669 294 295 692 83 137 417 310 457 321 256 723 235 413 938 272 90 290 64 988 410 465 315 824 332 288 143 312 422
259 75 960 414 411 697 6 41 128 109 399 742 658 848 1024 566 1029 881 992 679 637 121 304 425 975 199 836 409 209 124 1
02 684 379 189 5 563 930 251 386 85 557 655 691 322 356 965 127 287 421 995 388 412 548 964 208 76 2 12 324 508 536 678
113 201 488 640 654 798 805 88 255 78 401 502 352 883 979 699 4 10 326 435 681 733 107 507 393 139 430 517 194 982 533
195 374 549 589 624 707 722 752 777 972 1009 601 70 275 161 215 358 406 546 976 981 249 261 380 29 389 193 269 662 52 9
2 436 544 728 732 38 133 135 167 504 530 581 698 864 868 874 907 111 344 345 54 207 273 499 579 592 775 854 860 877 931
946 954 980 993 1000 32 82 387 542 15 286 403 442 638 716 912 942 84 327 150 357 385 449 535 643 718 756 789 18 26 51 8
6 101 117 190 198 202 230 350 371 376 378 438 496 497 564 596 627 743 755 769 818 852 897 902 922 927 300 307 381 600 7
36 978 115 116 200 260 314 316 423 702 724 839 182 191 270 391 416 424 431 437 455 458 547 573 635 648 738 758 825 983
1031 57 155 170 188 206 266 283 298 325 336 341 348 478 487 510 511 525 580 584 585 594 653 700 748 771 845 861 863 917
944 945 947 952 1020 13 72 129 130 145 151 154 248 253 254 305 306 317 334 354 366 370 383 462 477 480 519 534 543 565
610 618 630 652 657 734 750 767 778 779 788 795 796 816 832 888 900 915 935 959 971 989 990 997 1001 1003 1025 44 56 60
68 108 120 123 126 240 246 263 264 265 319 323 330 384 390 426 432 554 567 571 593 595 623 633 660 701 729 735 753 782
804 835 879 977 987 1017 1022 1033

3 70 71 407 230 282 160 286 234 275 276 62 78 277 266 856 906 394 172 287 341 784 59 1017 11 186 403 408 632 208 209 27
8 344 38 141 163 258 259 285 399 523 774 288 73 93 207 248 281 400 405 411 464 468 524 736 982 1003 69 143 466 473 508
619 748 913 23 41 51 64 67 94 173 176 177 178 179 210 231 232 233 245 274 283 297 342 402 404 406 422 458 459 460 467 5
25 638 640 720 905
```

(4) Evaluation -- Compute MAP

Print the MAP score (to the terminal).

- Read the relevancy answers from the file "medline.rel".
- Compare the ranked lists with the anwers, and compute the MAP score.
- (**) Also print the MAP score (to the terminal).

In [229]:
```
import requests
r=requests.get("http://condor.depaul.edu/ntomuro/courses/575/assign/medline.rel")
r.content
rel_docs={}
for line in r.text.split('\n'):
#     print(line)
    line=(line.strip().split(' '))
    if(line[0]!=''):
        rel_docs[line[0]]=line[1:]
#     print(rel_docs[line[0]])
for docid in rel_docs.keys():
    print(docid,end=' ')
    print(rel_docs[docid])
```

```
1 ['13', '14', '15', '72', '79', '138', '142', '164', '165', '166', '167', '168', '169', '170', '171', '172', '180', '1
81', '182', '183', '184', '185', '186', '211', '212', '499', '500', '501', '502', '503', '504', '506', '507', '508', '5
10', '511', '513']
2 ['80', '90', '162', '187', '236', '237', '258', '289', '290', '292', '293', '294', '296', '300', '301', '303']
3 ['59', '62', '67', '69', '70', '71', '73', '78', '81', '160', '163', '230', '231', '232', '233', '234', '276', '277',
'279', '282', '283', '287']
4 ['93', '94', '96', '141', '173', '174', '175', '176', '177', '178', '207', '208', '209', '210', '259', '396', '397',
'399', '400', '404', '405', '406', '408']
5 ['1', '2', '4', '5', '6', '7', '8', '9', '10', '11', '12', '158', '159', '188', '304', '305', '306', '307', '325', '3
26', '327', '329', '330', '331', '332', '333']
6 ['112', '115', '116', '118', '122', '238', '239', '242', '260', '309', '320', '321', '323']
7 ['92', '121', '189', '247', '261', '382', '385', '386', '387', '388', '389', '390', '391', '392', '393']
8 ['52', '60', '61', '123', '190', '251', '262', '263', '264', '265', '266']
9 ['30', '31', '53', '56', '57', '64', '83', '84', '89', '124', '125', '126', '192', '252', '253', '267', '268', '269',
'270', '271', '272', '273', '409', '412', '415', '420', '421', '422']
10 ['54', '55', '58', '152', '153', '154', '155', '254', '255', '256', '257', '529', '531', '532', '533', '534', '535',
'537', '538', '539', '540', '541', '542', '543']
11 ['32', '63', '66', '148', '150', '225', '226', '228', '229', '440', '441', '444', '445', '446', '447', '448', '451',
'452']
12 ['16', '17', '19', '20', '193', '364', '365', '366', '367']
13 ['21', '22', '143', '144', '145', '146', '194', '195', '196', '197', '198', '199', '470', '471', '474', '475', '47
7', '478', '479', '481', '483']
14 ['23', '24', '25', '26', '28', '29', '454', '455', '456', '457', '459', '461', '463', '466', '467', '468']
15 ['33', '34', '101', '102', '104', '105', '107', '109', '110', '140', '215', '216', '218', '219', '220', '222', '34
9', '350', '351', '352', '353', '355', '356', '357', '358', '359', '361', '362', '363']
16 ['35', '36', '98', '99', '202', '205', '484', '487', '488', '490', '492', '493', '495']
17 ['37', '38', '39', '41', '42', '127', '129', '130', '131', '132', '133', '334', '335', '337', '338', '339', '340',
'341', '342', '346', '348']
18 ['43', '514', '515', '516', '517', '518', '519', '521', '522', '523', '524', '525', '526', '527', '528']
19 ['544', '545', '549', '550', '555', '560', '562', '563', '564', '565', '566', '844', '845', '846', '847', '854', '85
6', '857', '858', '859', '860', '861', '862', '864', '865', '866', '867']
20 ['567', '570', '571', '573', '574', '575', '576', '577', '578', '580', '581', '584', '585', '588', '589', '590', '59
3', '594', '595', '596', '597', '598', '599', '601', '602', '848', '869', '870', '871', '872', '873', '874', '875', '87
6', '877', '878', '879', '880', '883']
21 ['604', '605', '608', '610', '612', '613', '615', '616', '618', '619', '620', '622', '626', '630', '631', '884', '88
5', '886', '888', '890', '891', '892', '893', '894', '895', '896', '898']
22 ['633', '635', '636', '638', '640', '641', '643', '644', '645', '647', '648', '651', '652', '654', '655', '656', '65
7', '658', '659', '660', '899', '900', '901', '904', '907']
23 ['797', '798', '799', '800', '801', '802', '803', '804', '805', '806', '807', '808', '809', '810', '811', '812', '81
3', '814', '815', '816', '817', '818', '819', '820', '821', '822', '849', '914', '915', '916', '917', '918', '919', '92
1', '922', '923', '924', '927', '928']
24 ['663', '666', '667', '668', '670', '674', '682', '684', '686', '850', '851', '852', '929', '930', '932', '935', '93
6', '938', '940', '941', '942', '943']
25 ['687', '688', '689', '690', '691', '692', '693', '694', '695', '697', '698', '699', '944', '945', '947', '948', '94
9', '951', '952', '953', '954', '955', '956', '958']
26 ['708', '712', '713', '714', '715', '716', '717', '719', '721', '722', '723', '724', '725', '726', '959', '960', '96
1', '962', '963', '964', '965', '966', '967', '968', '969', '970', '972', '973']
27 ['727', '728', '729', '730', '731', '732', '733', '734', '735', '736', '737', '738', '739', '769', '975', '977', '98
0', '984']
28 ['770', '771', '772', '773', '774', '775', '776', '777', '778', '779', '780', '782', '783', '784', '785', '786', '78
7', '788', '789', '790', '791', '792', '793', '795', '796', '989', '990', '991', '992', '993', '994', '995', '996', '99
7', '999', '1000', '1001', '1002', '1003']
29 ['740', '741', '742', '743', '744', '745', '746', '747', '748', '749', '750', '751', '752', '754', '757', '759', '76
0', '761', '762', '763', '764', '765', '766', '767', '768', '853', '1004', '1006', '1007', '1008', '1009', '1010', '101
2', '1013', '1015', '1016', '1017']
30 ['823', '825', '827', '831', '843', '1019', '1020', '1021', '1022', '1024', '1026', '1027', '1032', '1033']
```

• Consider rank position of each relevantdoc
– K1, K2, … KR
• Compute Precision@Kfor each K1, K2, … KR
• Average precision = average of P@K
• Ex: has AvgPrecof
• MAP is the average of the average precision value for a set of queries.

In [225]:
```
# Read the file from snippet above
retrived_docs={}
f=open('rankedlist.txt','r')
lines=f.readlines()
for line in lines:
#     print(line)
    line=line.strip().split(' ')
    retrived_docs[line[0]]=line[1:]
```

In [226]:
```
print(rel_docs['1'],end='\n\n')
print(retrived_docs['1'])
```

```
['13', '14', '15', '72', '79', '138', '142', '164', '165', '166', '167', '168', '169', '170', '171', '172', '180', '18
1', '182', '183', '184', '185', '186', '211', '212', '499', '500', '501', '502', '503', '504', '506', '507', '508', '51
0', '511', '513']

['500', '965', '72', '15', '212', '166', '513', '499', '511', '360', '181', '171', '182', '637', '168', '509', '206',
'142', '172', '512', '401', '13', '14', '79', '164', '165', '167', '169', '170', '183', '184', '727', '138', '180', '17
5', '336', '549', '58', '65', '570', '41', '403', '542', '185', '186', '211', '504', '506', '510', '99', '177', '640',
'758', '838', '259', '645', '11', '173', '174', '209', '256', '540', '541', '642', '700', '763', '876', '899', '986',
'112', '913', '125', '127', '231', '469', '213', '501', '502', '503', '505', '507', '508', '9', '78', '130', '158', '16
3', '178', '215', '333', '404', '445', '567', '578', '584', '589', '590', '593', '603', '652', '654', '658', '660', '73
0', '734', '816', '869', '870', '873', '880', '900', '905', '4', '24', '35', '75', '86', '100', '114', '156', '160', '1
62', '195', '196', '208', '214', '220', '229', '234', '253', '262', '267', '300', '322', '326', '342', '372', '374', '3
83', '421', '442', '454', '464', '480', '485', '496', '619', '627', '650', '655', '685', '715', '719', '726', '732', '7
43', '744', '769', '797', '799', '811', '826', '831', '835', '849', '863', '886', '888', '896', '907', '950', '981', '9
99', '1001', '1029', '42', '67', '70', '82', '83', '87', '95', '108', '176', '188', '189', '207', '248', '282', '299',
'367', '375', '408', '466', '486', '516', '521', '557', '569', '573', '576', '577', '580', '581', '585', '604', '606',
'608', '631', '632', '643', '648', '664', '800', '809', '812', '872', '875', '909', '910', '927', '984', '1020']
```

In [219]:
```
import csv

i=0
allMap=[]
for qid in rel_docs.keys():
    print('QUERY: '+qid+' : '+str(len(rel_docs[qid]))+' rel docs')
#     print(rel_docs[qid])
    print('RETRIEVED: '+str(len(retrived_docs[qid]))+' docs')
    MAP=[]
    for i in range(0,len(retrived_docs[qid])):
        retrived=set(retrived_docs[qid][:i+1])
#         print('intersection',end=" ")
#         print(retrived.intersection(rel_docs[qid]))
#         print('last element ',end=" ")
#         print(retrived_docs[qid][i])
#         print(retrived_docs[qid][i] in rel_docs[qid])
        if(retrived_docs[qid][i] in rel_docs[qid]):
#             print(len(retrived.intersection(rel_docs[qid]))/len(retrived))
            MAP.append(len(retrived.intersection(rel_docs[qid]))/len(retrived))
#         print(retrived_docs[qid][:i],end='\n')
#     for doc in retrived_docs[qid]:
#         print(doc,end=" ")
    print('MAP: ',end='')
    print(np.mean(MAP))
    allMap.append(np.mean(MAP))
    print()
print('Final MAP')
print(np.mean(allMap))
```

```
QUERY: 1 : 37 rel docs
RETRIEVED: 223 docs
MAP: 0.7103648073194891

QUERY: 2 : 16 rel docs
RETRIEVED: 434 docs
MAP: 0.44739041748932595

QUERY: 3 : 22 rel docs
RETRIEVED: 97 docs
MAP: 0.5666123941442229

QUERY: 4 : 23 rel docs
RETRIEVED: 244 docs
MAP: 0.26052017108858716

QUERY: 5 : 26 rel docs
RETRIEVED: 391 docs
MAP: 0.7242445103638524

QUERY: 6 : 13 rel docs
RETRIEVED: 302 docs
MAP: 0.5602793855277783

QUERY: 7 : 15 rel docs
RETRIEVED: 672 docs
MAP: 0.6474262940695009

QUERY: 8 : 11 rel docs
RETRIEVED: 639 docs
MAP: 0.346564986986273

QUERY: 9 : 28 rel docs
RETRIEVED: 434 docs
MAP: 0.39111972597336747

QUERY: 10 : 24 rel docs
RETRIEVED: 39 docs
MAP: 0.5967537883479913

QUERY: 11 : 18 rel docs
RETRIEVED: 316 docs
MAP: 0.590986984505543

QUERY: 12 : 9 rel docs
RETRIEVED: 436 docs
MAP: 0.4596016855017869

QUERY: 13 : 21 rel docs
RETRIEVED: 113 docs
MAP: 0.871748599242188

QUERY: 14 : 16 rel docs
RETRIEVED: 570 docs
MAP: 0.6063345565807665

QUERY: 15 : 29 rel docs
RETRIEVED: 433 docs
MAP: 0.5741016977475453

QUERY: 16 : 13 rel docs
RETRIEVED: 411 docs
MAP: 0.6566840733542634

QUERY: 17 : 21 rel docs
RETRIEVED: 741 docs
MAP: 0.26455932700888024

QUERY: 18 : 15 rel docs
RETRIEVED: 121 docs
MAP: 0.4632701664012063

QUERY: 19 : 27 rel docs
RETRIEVED: 377 docs
MAP: 0.43352749568494087

QUERY: 20 : 39 rel docs
RETRIEVED: 753 docs
MAP: 0.2070014279859061

QUERY: 21 : 27 rel docs
```

```
RETRIEVED: 258 docs
MAP: 0.33155930025236735

QUERY: 22 : 25 rel docs
RETRIEVED: 515 docs
MAP: 0.21270850731147442

QUERY: 23 : 39 rel docs
RETRIEVED: 30 docs
MAP: 0.8779170467882377

QUERY: 24 : 22 rel docs
RETRIEVED: 681 docs
MAP: 0.7094776800175664

QUERY: 25 : 24 rel docs
RETRIEVED: 579 docs
MAP: 0.7404115297769844

QUERY: 26 : 28 rel docs
RETRIEVED: 469 docs
MAP: 0.4003222376863044

QUERY: 27 : 18 rel docs
RETRIEVED: 636 docs
MAP: 0.6222548594865671

QUERY: 28 : 39 rel docs
RETRIEVED: 509 docs
MAP: 0.4223579940861998

QUERY: 29 : 37 rel docs
RETRIEVED: 779 docs
MAP: 0.681029317828057

QUERY: 30 : 14 rel docs
RETRIEVED: 460 docs
MAP: 0.5190733137829912

Final MAP
0.5298734760780055
```

Include your overall comments, for instance, how difficult you felt this assignment was, and any particular difficulties you encountered. Write in well-written English prose.

Understanding the data structure is really the fundamental of completing this assignment. Not only a simple hashmap or python dictionary is crucial but also taking time to consider various data structures that would result a good design of such retrieval system.

As for the conceptual ideas behind the algorithm, having a lot of figures to compute might seem like a great challenge, but slowly following each steps from the lecture notes would ease coding and allow the program to be built with good directions