정보검색과
데이터마이닝

hw4 - 문서유사도 보고서

20142772 최승호

1. 이전 과제 forward_indexing_table과 df_table 설명은 생략하고 코드만 스크린샷으로 첨부하겠다.

fnames 파일 읽어서 list 형태로 가지고 있자

```
import subprocess
import os

f = open("fnames.txt", 'r')
f_list = []
while True:
    line = f.readline()
    if not line: break
    f_list.append(line[:-1])
f.close()
# f_list
```

1-1) index2018.exe를 이용하여 각 텍스트 파일에 대한 인덱싱 파일을 만들자(색인어 추출)

- 추가로 각 파일당 unique한 색인어만 들어간 all_unique_indexing.txt파일 만들기

```
# all_unique_indexing파일 존재하면 삭제
if os.path.exists("all_unique_indexing.txt"):
    os.remove("all_unique_indexing.txt")
f = open("all_unique_indexing.txt", 'a')

# 색인어파일 디렉토리 있으면 삭제
if os.path.exists("색인어파일"):
    subprocess.call("rm -rf 색인어파일")
subprocess.call("mkdir 색인어파일")

for i in range(len(f_list)):
    input_f_name = "../../textfiles/" + f_list[i]
    output_f_name = "./색인어파일/" + str(i) + ".txt"

    # 각 파일 indexing file 만듬
    cmd_exe = " ".join(["index2018.exe", input_f_name, output_f_name])
    subprocess.call(cmd_exe)

    # 각 파일에 unique한 색인어 목록 all_unique_indexing.txt파일에 써주기
    cmd_exe = " ".join(["wordcount.exe -new -uniq -i", output_f_name, "temp_unique_indexing.txt"])
    subprocess.call(cmd_exe)
    f_read = open("temp_unique_indexing.txt", 'r', encoding = "ANSI")
    lines = f_read.readlines()
    for line in lines:
        try:
            f.write(line)
        except:
            print("encoding error")
    f_read.close()

f.close()
```

1-2) 만든 인덱싱 파일과 wordcount.exe를 이용해서 모든 파일에 docid : (빈도수, 단어), (빈도수, 단어) 형태로 저장하자

```
forward_indexing_table = dict()

for i in range(len(f_list)):

    max_tf = 0

    # word_count실행
    f_name = "./색인어파일/" + str(i) + ".txt"
    cmd_exe = " ".join(["wordcount.exe -new -i", f_name, f_name])
    subprocess.call(cmd_exe)

    #파일 열어서 forward_indexing_table만들기
    f = open(f_name, "r", encoding = "ANSI")
    word_list = []
    while True:
        try:
            line = f.readline()
            if not line:
                break
            temp_list = line.split()
            word_list.append(temp_list)
            # 각 문서의 max_tf구하는 과정 후에 문서유사도에서 tf계산할 때 쓰임
            if max_tf < int(temp_list[0]):
                max_tf = int(temp_list[0])
        except:
            print("encoding - error")
    # key는 docid = i 고 value는 해당파일 word_list를 가진 dictionary로 만든다.
    word_list.append(max_tf)
    forward_indexing_table[i] = word_list.copy()

print(forward_indexing_table[2])

f.close()
```

1-3) wordcount.exe를 all_unique_indexing.txt(각 파일당 unique indexing list)에 이용하여 DF계산하자

```
output_f_name = "word_df.txt"
cmd_exe = " ".join(["wordcount.exe -new -i","all_unique_indexing.txt", output_f_name])
subprocess.call(cmd_exe)

# word_df 파일을 wordcount하여서 각 색인어당 df계산 [색인어, df]
f = open("word_df.txt", 'r')
lines = f.readlines()
df_table = dict()
for line in lines:
    try:
        a = line.split()
        df_table[a[1]] = a[0]
    except:
        pass
print(df_table)
```

1-4) <TID, DF> table 구성 (그냥 색인어 대신 TID를 넣어주면 된다.)

```
tid_df_table = []
index = 0
for df in df_table.values():
    tid_df_table.append((index, df))
    index = index + 1
# print(tid_df_table)
```

변경된 것은 tf계산을 위해 각 문서마다 max_tf를 forward_indexing_table 마지막에 추가하였다. 나머진 기존 과제와 동일하고 이것들을 이용해서 문서간의 유사도를 계산할 것이다.

2. 각 term의 weight구하여 저장해주기!

- Term Frequency (*tf*)
  - Measure of *how well that term describes the document* contents

$$f_{ij} = \frac{freq_{ij}}{\max_l freq_{lj}} \quad (freq_{ij} : \text{Raw frequency of term } k_i \text{ in the document } d_j)$$

- Inverse Document Frequency (*idf*)
  - *Terms which appear in many documents are not very useful for distinguishing* a relevant document from a non-relevant one

$$idf_i = \log \frac{N}{n_i}$$

n_i : Number of documents in which the index term k_i appears

N : Total number of documents

문서 유사도 계산을 위해 docid : [(term, weight) , (term, weight) ...] 형태로 저장

weight = tf * idf

tf = 문서에서 단어 나온 횟수 / 문서에서 제일 많이 나온 단어횟수

idf = log(전체문서개수 / df(단어가 문서전체에 나온 횟수))

```
import math

doc_vector = dict()

# forward_indexing_table을 이용하여 docid와 term과 빈도수가 담긴 튜플들을 가져와 weight를 계산하여 다시 저장한다.
for docid, tid_tf_tuples in forward_indexing_table.items():

    word_list = []

    # 마지막은 max_tf니까 마지막 전까지 루프를 돌린다
    for tuple in tid_tf_tuples[:-1]:

        # tuple[0]에는 빈도수, tid_tf_tuples[-1]는 max_tf
        tf = int(tuple[0]) / tid_tf_tuples[-1]

        # int(df_table[tuple[1]])는 해당 단어의 df
        idf = math.log(623 / int(df_table[tuple[1]]))

        weight = tf * idf
        word_list.append([tuple[1], weight])

    doc_vector[docid] = word_list.copy()

#docid가 2인 문서의 벡터를 보여주겠다.
print(doc_vector[2])
```

forward_indexing_table 문서를 loop를 돌려 각 term의 weight를 구하여[(term, weight), (term, weight) ...]처럼 리스트를 만들어doc_vector에 추가한다.

print(doc_vector[2]) 2번 인덱스의 문서 벡터 출력 (중간은 생략하였다.)
출력화면에서 보다시피 term과 weight로 이루어진 튜플의 리스트로 구성된 것을 볼 수 있다.

[['08', 0.16405320001107243], ['120Gb', 0.2681061049494772], ['1600', 0.19344946039830824], ['1Gb', 0.2681061049494772], ['20', 0.19781453185584795], ['2003', 0.16797549258257127], ['20일', 0.1393126693928807], ['21', 0.15672246893224268], ['23', 0.2007662356710253], ['23일', 0.18146270737948403], ['25', 0.17422842474448078], ['25일', 0.14542114748420887], ['32', 0.36725486966651216], ['32비트', 0.60313857579416911], ['37', 0.15527067990355178], ['3D', 0.2953478967409407], ['3D게', 0.23922497242614615], ['3D게임', 0.23922497242614615], ['46', 0.14328392688472757], ['64', 4.036651245989082], ['64비트', 2.9034033180717445], ['64비트급', 0.2681061049494772], ['AMD', 2.0960317957182046], ['CPU', 0.9154894491369181], ['CPU개발계획', 0.2681061049494772], ['DDR', 0.16456832787497721], ['HDD', 0.15527067990355178], ['HP', 0.21189385172306888], ['IBM', 0.19095098269966004], ['IBM도', 0.20104619193138967], ['PC', 1.5145655929593282], ['PC기종', 0.2681061049494772], ['PC수요', 0.2681061049494772], ['PC시장', 0.3363876038324235], ['PC업체', 0.42985178065418281], ['PC용', 0.60313857579416911], ['XP', 0.126389547380220731], ['bailh', 0.17216505940805862], ['bailh@etnews.co.kr', 0.17216505940805862], ['co', 0.0037777535275467585], ['etnews', 0.004367733886031537], ['kr', 0.0034858667530297347], ['가능성', 0.08061403535238283], ['감지', 0.17216505940805862], ['강조', 0.09049444340608906], ['개발', 0.12179400735520868], ['개발자', 0.11440279436139653], ['개발중', 0.161233215055246511], ['개선', 0.08877005939930346], ['거', 0.10189843634263243], ['것', 0.006536571507630166], ['게', 0.21084383990281508], ['게임', 0.19095098269966004], ['게임전문가', 0.2681061049494772], ['게재', 0.005192843356705692], ['결정', 0.10428003691929696], ['결정키', 0.2681061049494772], ['계획', 0.07250217595539665], ['고', 0.09863745264302695], ['고객', 0.06968219099957071], ['고성능', 0.13235208253191544], ['고성능PC', 0.23922497242614615], ['고속', 0.168193801916211751], ['고위', 0.161233215055246511], ['공동', 0.06862728233355862], ['관계자', 0.3103974169743107], ['관련', 0.031717635483998356], ['관심', 0.06692636589521467], ['관심도', 0.168193801916211751], ['구도', 0.161233215055246511], ['국내', 0.07060815509226697], ['규격', 0.13

3. doc_vector를 활용하여 cosine_similarity를 사용해서 doc_similarity(문서간 유사도) 구하기!

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

```
doc_index = 0

similarity_matrix = []

# 623개를 돌려야한다
for a_index, weight_tuples in doc_vector.items():

    # 오래걸려서 몇번 문서까지 진행되는지 확인하는 프린트문
    print(a_index)

    # 기준이 되는 문서의 weight의 합을 구하자! : 첫번째 분모 = a_denominator
    a_denominator = 0
    for tuple in weight_tuples:
        a_denominator += math.pow(tuple[1], 2)

    # doc_index를 증가시키며 upper triangle형태로 만든다
    temp_similarity = []
    for b_index in range(doc_index, 623):
        b_denominator = 0
        # 비교할 문서의 weight의 합을 구하자! : 두번째 분모 b_denominator
        for tuple in doc_vector[b_index]:
            b_denominator += math.pow(tuple[1], 2)
        # a * b 로 분모 구성 완료
        denominator = math.sqrt(a_denominator) * math.sqrt(b_denominator)

        # 각 term의 weight곱의 합을 구해야한다!
        numerator = 0
        for a_tuple in weight_tuples:
            for b_tuple in doc_vector[b_index]:
                if a_tuple[0] == b_tuple[0]:
                    numerator += a_tuple[1] * b_tuple[1]
#   |                 print(a_tuple, b_tuple)
                    break
        similarity = numerator / denominator
        temp_similarity.append(similarity)

#     doc_index += 1
    similarity_matrix.append(temp_similarity.copy())

print(similarity_matrix[2])
```

doc_vector를 623개 다 돌려서 623 x 623의 similarity_matrix를 만들어 준다.
tuple[0]에는 term이 들어가 있고, tuple[1]에는 weight가 들어가 있다.
기준이 되는 문서(a_vector)와 비교할 문서(b_vector)를 받아와 cosine 유사도를 구해준다.
doc_index주석을 풀어주면 upper triangle의 형태로 마지막 vector는 자기 자신과의 유사도만을 가지게 된다.
print(similarity_matix[2]) index 2번 문서의 유사도 결과

```
[0.005876797470635268, 0.060260285834644 2, 1.0000000000000002, 0.005293992941346082, 0.003464489140433201, 0.003897878994536316, 0.0085
385544708767 53, 0.003962144989739963, 0.022944590131189 45, 0.004631464659215 5815, 0.010294363914396495, 0.006284358489671 39, 0.005531 58
9587219683, 0.013381239067877967, 0.001238908077981 4496, 0.008709711490983813, 0.008432519759540034, 0.006133089314112343, 0.0081401053
77934755, 0.004970235250243711, 0.015583814128961615, 0.011098660954714764, 0.005601111510363249, 0.002662488101547168, 0.0068720748448
04966, 0.002535221495717811, 0.004385140355336219, 0.012429436550931 68, 0.008486726524485 3153, 0.048016240063892 32, 0.0039750836454666 0
2, 0.006258382903079 2995, 0.003070668472475 4398, 0.005768542063601767, 0.005966112028183 2535, 0.003471748620859 6634, 0.005012239830612 1
346, 0.015789257601153 71, 0.002127434308400 8874, 0.003020225662909 6647, 0.001954351673516 8283, 0.008335770617823 283, 0.017737090999 0897
44, 0.004967819923744139, 0.008189048141643644, 0.010670966071849001, 0.002554351516580534, 0.006038200104163073, 0.012039503080739132,
0.011120370373825409, 0.006492111079030428, 0.001211848496243561, 0.008467773791032664, 0.008050134770701515, 0.004588737997270008, 0.0
0356535711810238, 0.014353611600427996, 0.010939392117681421, 0.006092213844711669, 0.011368688369812356, 0.005436537542086 57, 0.007603
525992666761, 0.010085338816522299, 0.004937397877155002, 0.002697209253436418, 0.010596199506733841, 0.006246205844858382, 0.008820088
981007355, 0.009325016393074 5144, 0.002649781328730 2127, 0.007179264255802787, 0.006245563099070483, 0.024271019429411466, 0.0132371003
```

보면 2번 인덱스의 값이 1이다. 그리고 623개의 similarity가 나오는데 중간 생략하였다.

4. doc_similarity(문서간 유사도) CSV파일로 출력하고 가장 유사도 높은 문서 검색 프로그램 만들기

결과를 csv 파일에 쓰기

623 X 623 매트릭스를 print하기 힘들어 csv파일에 써주겠다.

```
import csv

f = open('output.csv', 'w', newline='')
csv_writer = csv.writer(f)

zero_count = 0
for similarity_vector in similarity_matrix:
    # 0번인덱스부터 채웠기 때문에 upper triangle로 만들어줄려면 앞에 0으로 채워줘야한다.
    row = ([0] * zero_count) + (similarity_vector[zero_count:])
#     row = similarity_vector
    csv_writer.writerow(row)
    zero_count += 1
f.close()
```

처음에는 매트릭스로만 출력을 하였다.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	0.005433	0.005877	0.017805	0.096693	0.013086	0.015359	0.005795	0.052837	0.036604	0.006148	0.006812	0.012133
2	0.005433	1	0.06026	0.004626	0.002105	0.00694	0.006062	0.007252	0.021649	0.006153	0.005663	0.006942	0.016282
3	0.005877	0.06026	1	0.005294	0.003464	0.003898	0.008539	0.003962	0.022945	0.004631	0.010294	0.006284	0.005532
4	0.017805	0.004626	0.005294	1	0.028014	0.023914	0.005678	0.016349	0.029457	0.042855	0.101033	0.004424	0.010438
5	0.096693	0.002105	0.003464	0.028014	1	0.005182	0.00447	0.022644	0.091417	0.079304	0.004824	0.00327	0.009113
6	0.013086	0.00694	0.003898	0.023914	0.005182	1	0.005597	0.035036	0.007751	0.005375	0.029426	0.008646	0.01463
7	0.015359	0.006062	0.008539	0.005678	0.00447	0.005597	1	0.009264	0.012195	0.040283	0.002961	0.005386	0.007958
8	0.005795	0.007252	0.003962	0.016349	0.022644	0.035036	0.009264	1	0.01157	0.019487	0.014093	0.003898	0.01254
9	0.052837	0.021649	0.022945	0.029457	0.091417	0.007751	0.012195	0.01157	1	0.547236	0.014648	0.001432	0.02702
10	0.036604	0.006153	0.004631	0.042855	0.079304	0.005375	0.040283	0.019487	0.547236	1	0.057641	0.002511	0.017608
11	0.006148	0.005663	0.010294	0.101033	0.004824	0.029426	0.002961	0.014093	0.014648	0.057641	1	0.001958	0.013325
12	0.006812	0.006942	0.006284	0.004424	0.00327	0.008646	0.005386	0.003898	0.001432	0.002511	0.001958	1	0.004075
13	0.012133	0.016282	0.005532	0.010438	0.009113	0.01463	0.007958	0.01254	0.02702	0.017608	0.013325	0.004075	1
14	0.006406	0.009853	0.013381	0.018707	0.0147	0.018989	0.006901	0.018517	0.008234	0.014703	0.015004	0.006763	0.008574
15	0.022153	0.006352	0.001239	0.013135	0.028587	0.005258	0.000896	0.015888	0.024505	0.013753	0.010186	0.001774	0.00365
16	0.009916	0.014093	0.00871	0.011625	0.003151	0.016228	0.010596	0.010059	0.01983	0.002452	0.007262	0.005501	0.022315

결과를 보다시피 symmetric한 것을 볼수 있다.

그래서 upper triangle로 밑 부분은 다 0으로 만들어주어 출력을 하였다.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	0.00543	0.00588	0.01781	0.09669	0.01309	0.01536	0.00579	0.05284	0.0366	0.00615	0.00681	0.01213
2	0	1	0.06026	0.00463	0.00211	0.00694	0.00606	0.00725	0.02165	0.00615	0.00566	0.00694	0.01628
3	0	0	1	0.00529	0.00346	0.0039	0.00854	0.00396	0.02294	0.00463	0.01029	0.00628	0.00553
4	0	0	0	1	0.02801	0.02391	0.00568	0.01635	0.02946	0.04286	0.10103	0.00442	0.01044
5	0	0	0	0	1	0.00518	0.00447	0.02264	0.09142	0.0793	0.00482	0.00327	0.00911
6	0	0	0	0	0	1	0.0056	0.03504	0.00775	0.00537	0.02943	0.00865	0.01463
7	0	0	0	0	0	0	1	0.00926	0.01219	0.04028	0.00296	0.00539	0.00796
8	0	0	0	0	0	0	0	1	0.01157	0.01949	0.01409	0.0039	0.01254
9	0	0	0	0	0	0	0	0	1	0.54724	0.01465	0.00143	0.02702
10	0	0	0	0	0	0	0	0	0	1	0.05764	0.00251	0.01761
11	0	0	0	0	0	0	0	0	0	0	1	0.00196	0.01333
12	0	0	0	0	0	0	0	0	0	0	0	1	0.00408
13	0	0	0	0	0	0	0	0	0	0	0	0	1

zoom-out 출력

숫자를 입력하면 가장 유사도가 높은 문서를 찾는 프로그램을 간편하게 만들었다.

임의의 문서 x 와 제일 유사도가 높은 문서는?!

```
x = int(input('임의의 문서 x 숫자를 입력해주세요. (0 ~ 622)'))

# x를 제외하고 max를 찾는 이유는 x index에는 자기 자신과의 유사도라 무조건 max이기 때문이다.
similarity_value = max(similarity_matrix[x][:x] + similarity_matrix[x][x+1:])

# similarity_value로 index를 찾아 doc_number를 찾는다
doc_number = similarity_matrix[x].index(similarity_value)

print('가장 유사도가 높은 문서번호 : ', doc_number ,', 유사도 : ', similarity_value)

# 처음에 만들었는 file name list를 활용하여 예쁘게 출력해본다.
print(f_list[x], '와 제일 유사도가 높은 문서는 ', f_list[doc_index], '이다.')
```

```
임의의 문서 x 숫자를 입력해주세요. (0 ~ 622)8
가장 유사도가 높은 문서번호 :  9 , 유사도 :  0.5472360332114795
IT-baidu.txt 와 제일 유사도가 높은 문서는  IT-2007newWebIR.txt 이다.
```

이것의 맹점은 623 x 623 매트릭스가 차있을 경우 잘 작동한다.
upper triangle일 경우는 x번째 row와 x번째 col을 가져와서 max를 계산하여야 한다.

5. 개선해야 할 점 + 느낀점

과제를 통하여 간단하게 문서유사도 계산을 만들었다. 분석하는 재미가 있었다. 하지만 실제로 유사도 계산 프로그램을 만들면 이렇게 만들면 안될 것 같다.
개선해야 할 점들을 정리해보겠다.

개선해야할 점

1. term말고 tid를 활용하여 저장공간과 효율적인 접근 가능하게 하기

2. tf의 정의가 여러가지 있음.
 1) 그냥 빈도수
 2) 빈도수 / 문서의 전체 빈도수의 합
 3) 빈도수 / 문서에서 가장 큰 빈도수
 4) 로그변환 등등
 이거에 따라 유사도 값이 달라지는 것을 알아야 할듯...

3. 623개의 문서뿐만 아니라 어떤 데이터 셋이 txt파일로 와도 될 수 있게 바꾸기
ex) 623 -> len(f_list)등등

4. loop횟수 줄이기 - 효율성, 알고리즘적 측면
 weight를 구하는 부분에서 너무오래걸린다.
 딕셔너리를 ordered되게해서 접근이 쉽게 만들기
 서로 공통된 단어들을 갖고있는 벡터를 만들어 weight계산을 빠르게 만들어주기
 각 문서의 weight들의 합 즉, 분모를 구성하는 걸 전에 loop돌릴 때 계산해서 값을 미리 저장해 놓는 것!
 등등 시간복잡도를 개선해야 할 부분들이 많이 보였다.

5. 623개의 문서만 하여 ram이 부족하진 않았지만, 실제로 큰 데이터 셋을 마주하게 되면 file i/o를 통하여 프로그램을 만들어야 할 것 같다.
그래서 csv파일 출력을 시도해 보았다.

6. 교수님께서 제공해주신 wordcount, index 실행파일도 좋지만, scikit learn 알고리즘이나 konlpy등 공개된 라이브러리를 이용하여 stemming, stopword처리부분을 활용하면 내가 원하는 결과를 얻을 수 있을 것 같다.

나중에 텍스트 분석이나 검색엔진 구현작업을 하게 되면 도움이 될 수 있는 유용한 과제였다.