데이터 마이닝

검색 엔진 구현

최종 보고서

소프트웨어 공학과

20153235
조정근

1) 각 문서에 대해 Doc. ID 부여

```
# file read

terms = []          #set of terms   #word vector
term_file_cnt = []  # cnt of file which has terms(index)  #단어가 나온 파일의 수
term_freq = []      #freq of terms in each files
file_cnt = 0

for txt in f_list:
    try:
        file = open(path + txt, 'r', encoding='cp949')
    except UnicodeDecodeError:
        print(path + txt)
    file_cnt += 1
    s = set()
    file_terms_cnt = 0      #cnt of terms in a file #파일에 있는 단어의 수
    term_cnt = {}          #freq of terms in a file #
    tmp = {}
    while(True):
        try:
            line = file.readline()
        except UnicodeDecodeError:
            print(path + txt[:-1])
            continue

        if not line:
            file.close()
            break

        if( len(line[:-1])<=10):
            continue
        try:
            morphs = kkma.morphs(line[:-1])
        except :
            print(line[:-1] + "line end")

        for m in morphs:
            file_terms_cnt += 1
            if( m not in s):
                s.add(m)
            if( m not in terms):
                terms.append(m)
                term_file_cnt.append(0)

            term_cnt[terms.index(m)] = term_cnt.get(terms.index(m),0) + 1


    for k,v in term_cnt.items():
        tmp[k] = v/file_terms_cnt
    term_freq.append(tmp)

    l = list(s)
    for m in l:
        term_file_cnt[terms.index(m)] += 1
```

f_list에 문서 목록이 있고, 각 문서의 형태소를 분석하여 문서 벡터(terms), 각 단어가 나온 문서의 수(term_file_cnt), 각 문서에 나온 전체 단어의 수(file_terms_cnt), 문서에서 나온 각각의 단어의 빈도(term_freq)를 채운다.

```
# word ID
f = open('word_ID.txt','w')
for t in terms:
    f.write(t + '₩n')
f.close()
```

Terms를 word_ID.txt 파일에 저장한다. (Doc ID)

2) forward indexing, backward indexing

```
#forward index, backward indexing
backward_index = [{} for i in range(len(terms))]

# tf * idf
f = open('tf-idf.txt','w')
for i in range(len(term_freq)):
    for w_id,v in term_freq[i].items():
        tf_idf = v*math.log10(file_cnt/term_file_cnt[k])
        f.write('{}:{} '.format(w_id, tf_idf) )
        backward_index[w_id][i]=(tf_idf)
    f.write('₩n')
f.close()
```

각각의 문서-단어에 대해서 tf-idf로 가중치를 계산한다. (forward_index) forward index를 계산하고 tf-idf.txt파일에 저장하면서 backward_index값을 계산한다

```
print(backward_index[0])  # 단어에 대한 문서-가중치
# 0번 단어는 0~10번, 17번 21번 문서에 존재
```

```
{0: 0.008128341805277857, 1: 0.012192512707916785, 2: 0.012216005025273079, 3: 0.006215790792271302, 4: 0.006618065352940216, 5: 0.
00949117755706097, 6: 0.03336898214798278, 7: 0.012334837758981962, 8: 0.004002592555629248, 9: 0.005513136180971068, 10: 0.0044151
16022365409, 17: 0.012334837758981962, 21: 0.006143514155151868, 23: 0.004255105106118609, 28: 0.004408975388120117, 30: 0.00540964
```

0번 단어는 0~10, 17, 21...번 문서에 존재하는 것을 볼 수 있다.

3) term table과 posting file로 분리하여 저장

```
term_table = []
posting_file = []
loc_of_files = 0

#backward_index
f = open('backward_index.txt','w')
for word in backward_index:
    n_of_files = 0
    for file,v in word.items():
        f.write('{}:{} '.format(file, v))

        term_table
        posting_file.append([file, v])
        n_of_files += 1

    term_table.append([loc_of_files, n_of_files])
    loc_of_files += n_of_files

    f.write('₩n')
f.close()

f = open('posting_file.txt','w')
for doc_weight in posting_file:
    f.write(doc_weight)
    f.write('₩n')
f.close()
```

backward_index를 backward_index.txt 파일에 저장하면서 term_table과 posting_file을 만든다.

posting_file을 postingfile.txt에 저장한다.

```
print(term_table[:10])
# 0번 단어는 0부터 322개  # 1번 단어는 322 부터  245개
#상위 10개만 출력
```

```
[[0, 322], [322, 245], [567, 431], [998, 25], [1023, 49], [1072, 1], [1073, 4], [1077, 8], [1085, 991], [2076, 47]]
```

상위 10개의 단어에 대해 term table을 출력해 보았다.

0부터 332개는 0번 단어, 322번째부터 245개는 1번 단어에 대한 정보를 가진다.

```
print(posting_file[320:330])
# 0번 단어에 대한 문서-가중치 : 133 까지  / 1번 단어에 대한 문서-가중치 : 134 부터
#경계 값을 보기위해 134 근처 출력
```

```
[[982, 0.002691046947417966], [986, 0.0042954651816509], [0, 0.012192512707916785], [22, 0.0034307936191107836], [24, 0.00254827436
01755336], [26, 0.001697030676690773], [28, 0.0022044876940600583], [30, 0.0027048236382750546], [34, 0.009645192608696848], [43,
0.0022675631645624923]]
```

0번 단어의 문서 번호와 가중치는 0부터 331까지 이고, 1번 단어의 문서 번호와 가중치는 332부터 245개 이다. 따라서 출력 화면에서 2번째(331)의 문서 번호는 매우 크고, 3번째(332)의 문서 번호는 다시 작아진 것을 볼 수 있다.

4) 키워드 검색

```
#keyword 검색

keyword = list(map(str, input()[:].split(' ') ) )
#keyword = kkma.morphs(str(input()))
print(keyword)
```

```
삼성 노트북
['삼성', '노트북']
```

키워드 검색을 위한 입력을 받는다. 형태소 분석을 한 결과와 띄어쓰기로 나눈 결과 2가지로 나눌 수 있지만, 이번에는 띄어쓰기로 나눈 결과를 사용하였다.

```
#keyword 에 해당하는 word id
keyword_index = [terms.index(k) for k in keyword]
print(keyword_index)
```

```
[1059, 2171]
```

각각의 keyword에 해당하는 word_id를 출력한다.

```
#termtable 검색한 결과
result_term_table = [term_table[i] for i in keyword_index]
print(result_term_table)
```

```
[[112230, 63], [155784, 34]]
```

Word_id로 term_table을 검색하여 '삼성'은 112230부터 63개, 노트북은 155784부터 34개의 정보를 갖는 것을 확인하였다.

```
#posting_file 검색 결과
result_posting_file = [ posting_file[idx:idx+cnt] for idx, cnt in result_term_table ]
print(result_posting_file)
#calc_weight = [  for idx, cnt in result_term_table for file_idx, w in posting_file[idx:idx+cnt] ]
#term_cnt[terms.index(m)] = term_cnt.get(terms.index(m),0) + 1
```

```
[[[10, 0.004415116022365409], [16, 0.009040075962143623], [17, 0.009251128319236471], [38, 0.007713025070701616], [119, 0.0019400571016269057], [120, 0.0015196803950423604], [124, 0.002515915320681241], [138, 0.0037649089121833306], [140, 0.002213724374342433], [160, 0.002637315560780669], [189, 0.001272092016074785], [213, 0.007023751781517794], [215, 0.01031470706309663], [216, 0.022048284185877734], [217, 0.007571783369566156], [239, 0.0019496022780186742], [308, 0.0026155555314012905], [314, 0.0018176911147123648], [318, 0.003828566792341019], [333, 0.002766189619597176 6], [341, 0.0013903742561659491], [347, 0.006563257358298891], [349, 0.002130412166706 89], [358, 0.007264100146788185], [363, 0.0020268883018275986], [369, 0.0020320854513194643], [371, 0.002254660955944782 4], [380, 0.002099373049045274], [390, 0.002453601628528146], [393, 0.0037426839481208545], [396, 0.021712693863413 45], [407, 0.0018345215879967383], [436, 0.0003184702937511917], [437, 0.0027856355923184216], [438, 0.002760168310020343], [459, 0.0007007191211446427], [463, 0.003590094342081953], [472, 0.004567800149940005], [482, 0.0140267845312317], [488, 0.005630645300281286], [518, 0.0050159071266746265], [539, 0.0035419589989478923], [548, 0.0019520032660457908], [559, 0.0034645391301184308], [564, 0.006436656454940841], [565, 0.0014014382422892855], [592, 0.003884869245169564], [595, 0.002277337143720089], [602, 0.003351007720991928 5], [605, 0.0024161991646786314], [675, 0.002307171254773191], [692, 0.010063661282724965], [712, 0.006618065352940216], [713, 0.005096548720351067], [824, 0.006228002562000715], [836, 0.03316946655965403], [895, 0.002488267899574854], [909, 0.002019142231884308], [960, 0.0012220714356431627], [979, 0.003406859466729424], [991, 0.004354468824255995], [998, 0.002637315560780669], [999, 0.003208555975767575]], [[22, 0.024015555333775487], [33, 0.0009629566537236828], [50, 0.013319551697724218], [93, 0.002581476631969352], [106, 0.0031771135 56405933], [179, 0.024976177547126506], [180, 0.04081613696212486], [183, 0.012938993077789241], [195, 0.01065206083352945], [196, 0.006215790792271302], [217, 0.0025239277898553856], [224, 0.002387088331369 25], [257, 0.008929727617065815], [285, 0.010800863046195448], [307, 0.004307137641383647], [320, 0.0016666946919339453], [329, 0.006179441138515329], [335, 0.010874968453030409], [366, 0.0019027930996748882], [394, 0.007703653229789462], [431, 0.008386384402270804], [450, 0.003894414378450078 7], [514, 0.021892633315320 19], [527, 0.007529817824366661], [536, 0.0011394871689641855], [537, 0.0032782350610737997], [603, 0.009412272280458326], [640, 0.028115772098078615], [652, 0.0010775164187825845], [910, 0.0020993730490452 74]]]
```

Term table을 통해 나온 결과를 가지고 posting file을 검색하여 97(63+34)개의 문서 번호와 각 가중치를 검색하였다.

```
calc_weight = {}
for idx, cnt in result_term_table:
    for file_idx, w in posting_file[idx:idx+cnt]:
        calc_weight[file_idx] = calc_weight.get(file_idx, 0) + w
print( calc_weight )
```

```
{10: 0.004415116022365409, 16: 0.009040075962143623, 17: 0.009251128319236471, 38: 0.007713025070701616, 119: 0.0019400571016269057, 120: 0.0015196803950423604, 124: 0.002515915320681241, 138: 0.0037649089121833306, 140: 0.002213724374342433, 160: 0.002637315560780669, 189: 0.001272092016074785, 213: 0.007023751781517794, 215: 0.01031470706309663, 216: 0.022048284185877734, 217: 0.010095711159421542, 239: 0.0019496022780186742, 308: 0.0026155555314012905, 314: 0.0018176911147123648, 318: 0.003828566792341019, 333: 0.002766189619597176 6, 341: 0.0013903742561659491, 347: 0.006563257358298891, 349: 0.002130412166706589, 358: 0.007264100146788185, 363: 0.0020268883018275986, 369: 0.0020320854513194643, 371: 0.002254660955944782 4, 380: 0.002099373049045274, 390: 0.002453601628528146, 393: 0.0037426839481208545, 396: 0.021712693863413 45, 407: 0.0018345215879967383, 436: 0.0003184702937511917, 437: 0.002785635592318421 6, 438: 0.002760168310020343, 459: 0.0007007191211446427, 463: 0.003590094342081953, 472: 0.004567800149940005, 482: 0.0140267845312317, 488: 0.005630645300281286, 518: 0.0050159071266746265, 539: 0.0035419589989478923, 548: 0.0019520032660457908, 559: 0.0034645391301184308, 564: 0.006436656454940841, 565: 0.0014014382422892855, 592: 0.003884869245169564, 595: 0.002277337143720089, 602: 0.0033510077209919285, 605: 0.0024161991646786314, 675: 0.002307171254773191, 692: 0.010063661282724965, 712: 0.006618065352940216, 713: 0.005096548720351067, 824: 0.006228002562000715, 836: 0.03316946655965403, 895: 0.002488267899574854, 909: 0.002019142231884308, 960: 0.0012220714356431627, 979: 0.003406859466729424, 991: 0.004354468824255995, 998: 0.002637315560780669, 999: 0.003208555975767575, 22: 0.024015555333775487, 33: 0.0009629566537236828, 50: 0.013319551697724218, 93: 0.002581476631969352, 106: 0.0018757711858333516, 109: 0.004803111066755097, 129: 0.002071930264090434, 178: 0.03177113556405933, 179: 0.024976177547126506, 180: 0.04081613696212486, 183: 0.012938993077789241, 195: 0.01065206083352945, 196: 0.006215790792271302, 224: 0.002387088331369 25, 257: 0.008929727617065815, 285: 0.010800863046195448, 307: 0.004307137641383647, 320: 0.0016666946919339453, 329: 0.006179441138515329, 335: 0.010874968453030409, 366: 0.001902793099674888 2, 394: 0.007703653229789462, 431: 0.008386384402270804, 450: 0.003894414378450078 7, 514: 0.021892633315320 19, 527: 0.007529817824366661, 536: 0.0011394871689641855, 537: 0.0032782350610737997, 603: 0.009412272280458326, 640: 0.028115772098078615, 652: 0.0010775164187825845, 910: 0.002099373049045274}
```

Posting file에서 나온 결과를 문서 번호를 기준으로 가중치를 더하였다.

키워드의 수에 대한 처리를 하지 않았다.

```
result_calc_weight = [[w, file_idx] for file_idx, w in calc_weight.items() if w >= 0.01 ]
result_calc_weight.sort(reverse = True)
print(result_calc_weight[:])
```

```
[[0.04081613696212486, 180], [0.03316946655965403, 836], [0.03177113556405933, 178], [0.028115772098078615, 640], [0.024976177547126506, 179], [0.024015555333775487, 22], [0.022137243743424 33, 140], [0.022048284185877 34, 216], [0.021892633315320 19, 514], [0.021712693863413 45, 396], [0.014026784531231 7, 482], [0.013319551697724218, 50], [0.012938993077789241, 183], [0.010874968453030409, 335], [0.010800863046195448, 285], [0.01065206083352945, 195], [0.01031470706309663, 215], [0.010095711159421542, 217], [0.010063661282724965, 692]]
```

가중치로 정렬을 하여 가중치 값이 임계보다 큰 것만 출력해 보았다.

```
search_result = [ f_list[i[1]] for i in result_calc_weight]
print(search_result[:])
```

```
['IT-notebookUS.txt', '뜨거운 우정 차가운 승부.txt', 'IT-notebook.txt', 'news-WSnotebook.txt', 'IT-notebookProtect.txt', 'IT-20HPnotebook.txt', 'IT-LGIBM.txt', 'IT-samsungHDD.txt', 'news-noteb.txt', 'news-eWork.txt', 'news-LCD.txt', 'IT-dualcoreNB.txt', 'IT-oneNote.txt', 'news-Cebit.txt', 'IT-windowVista.txt', 'IT-PC.txt', 'IT-samsung.txt', 'IT-samsungLCD.txt', '[음주단속] 소주1병 마시면 8시간 지나야 안심.txt']
```

키워드 검색을 통해 나온 결과이다.

5) 필터링 기능

4) 키워드 검색 부분부터 코드를 다소 수정하여 '-'를 입력하고 주는 keyword에 대해서 필터링을 해주는 기능을 추가하였다.

```
#keyword 에 해당하는 word id
keyword_index = [terms.index(k) for k in keyword if k[0] != '-']
print(keyword_index)

removal_keyword_index = [terms.index(k[1:]) for k in keyword if k[0]=='-'] # 검색 제외할 문서

#termtable 검색한 결과
result_term_table = [term_table[i] for i in keyword_index]
print(result_term_table)

result_term_table_removal = [term_table[i] for i in removal_keyword_index] # 제외할 문서->tremtable 검색

#posting_file 검색 결과
# 제외 할 문서->posting file 에서 검색
result_posting_file_removal = [posting_file[idx:idx+cnt] for idx, cnt in result_term_table_removal ]
result_removal_file = [idx for word in result_posting_file_removal for idx,weight in word] # 제외 할 doc id 목록

result_posting_file = [ posting_file[idx:idx+cnt] for idx, cnt in result_term_table ]
print(result_posting_file)
#calc_weight = [  for idx, cnt in result_term_table for file_idx, w in posting_file[idx:idx+cnt] ]
#term_cnt[terms.index(m)] = term_cnt.get(terms.index(m),0) + 1


calc_weight = {}
for idx, cnt in result_term_table:
    for file_idx, w in posting_file[idx:idx+cnt]:
        if(file_idx in result_removal_file):    # 제외 keyword 제외
            continue
        calc_weight[file_idx] = calc_weight.get(file_idx, 0) + w
print( calc_weight )


result_calc_weight = [[w, file_idx] for file_idx, w in calc_weight.items() if w >= 0.01 ]
result_calc_weight.sort(reverse = True)
print(result_calc_weight[:])


search_result = [ f_list[i[1]] for i in result_calc_weight]
print(search_result[:])
```

기존의 코드에서 '삼성'으로 검색한 최종 결과는

```
#keyword 검색

keyword = list(map(str, input()[:].split(' ') ) )
#keyword = kkma.morphs(str(input()))
print(keyword)
```

삼성
['삼성']

```
search_result = [ f_list[i[1]] for i in result_calc_weight]
print(search_result[:])
```

['뜨거운 우정 차가운 승부.txt', 'IT-samsungHDD.txt', 'news-eWork.txt', 'news-LCD.txt', 'IT-samsung.txt', '[음주단속] 소주1병 마시면 8시간 지나야 안심.txt']

이었지만, '삼성 -야구'로 검색해 '야구'에 대해서 필터링 한 결과는 '뜨거운 우정 차가운 승부.txt'가 필터링 되어 보이지 않는다.

```
#keyword 검색

keyword = list(map(str, input()[:].split(' ') ) )
#keyword = kkma.morphs(str(input()))
print(keyword)
```

삼성 -야구
['삼성', '-야구']

```
search_result = [ f_list[i[1]] for i in result_calc_weight]
print(search_result[:])
```

['IT-samsungHDD.txt', 'news-eWork.txt', 'news-LCD.txt', 'IT-samsung.txt', '[음주단속] 소주1병 마시면 8시간 지나야 안심.txt']

6) 문서 보여주기

검색을 마친 후 읽어 올 문서의 번호를 입력 받아 입력한 번호의 문서를 보여준다.

```
while(True) :
    i=input('i : 읽을 문서 번호, enter : stop')
    if( i== '' or i == 'enter'):
        break
    try :
        i = int(i) - 1
    except:
        print('enter number or enter')
        continue
    f = open(path+search_result[i], 'r')
    while(True):
        line = f.readline()
        if not line:
            f.close()
            break
        if( len(line)<10):
            continue
        print(line)
```

i : 읽을 문서 번호, enter : stop

i : 읽을 문서 번호, enter : stop1
뜨거운 우정 차가운 승부

2002 프로야구 한국시리즈 패권을 놓고 1990년에 이어 12년 만에 맞붙은 삼성과 LG의 처지가 얄궂다. 우선 팀만 놓고 보면 LG는 90년, 94년 두 차례 한국시리즈 정상을 차지한 반면, 삼성은 창단이래 7차례나 한국시리즈에 올랐지만, 번번이 눈물만 뿌렸다.

하지만 양 팀 사령탑의 운명은 판이하다. 삼성 김응용 감독은 9차례나 우승을 차지한 '우승제조기'. 이와반대로 LG 김성근 감독은 감독 데뷔 20년 만에 처음으로 한국시리즈에 올랐을 만큼 한국시리즈와는 인연이 없다.