R E P O R T

텍스트 문서로부터 feature vector 구성 및 similarity matrix 구성

국민대학교	
학 부	컴퓨터공학부
과 목	빅데이터 최신기술
담당교수	강승식 교수님
제 출 일	2016년 12월 28일
학 번	20143390
이 름	김선영

1. 텍스트 문서로부터 feature vector구성

```
# -*- coding: utf-8 -*-
import os
import glob
import math
import numpy as np
```

1) Term 추출

```
path = os.getcwd() + "₩₩docs₩₩"
flist=glob.glob(path+"*.txt")
if not flist: # flist가 비어있다면 비어있다면 flist == false
    file_copy()

path = os.getcwd() + "₩₩result₩₩"
flist=glob.glob(path+"*.txt")
if not flist:
    each_morpheme()

path = os.getcwd() + "₩₩output₩₩"
flist=glob.glob(path+"*.txt")
if not flist:
    wordcount_df()
```

함수의 결과로 생성될 txt파일이 저장될 폴더를 보고 이미 연산이 끝났다면 다시 연산을 수행하지 않고 넘어간다.

```
def file_copy():
    i=1
    path = os.getcwd() + "₩₩ITnews₩₩"
    flist=glob.glob(path+"*.txt") # 파일 이름 포함 전체 경로
    os.system("mkdir docs")
    for fname in flist:
        # 모든 파일을 docs/i.txt로 복사
        os.system("copy " + fname + " "+ os.getcwd() +"₩₩docs₩₩" + "%d" %i + ".txt")
        i = i+1
```

사용할 뉴스txt파일을 1.txt, 2.txt, … 와 같은 파일명으로 복사해온다.

```
def each_morpheme():
    path = os.getcwd() +"₩₩docs₩₩"
    flist = os.listdir(path)
    os.system("mkdir result")
    move_path = os.getcwd() + "₩₩result₩₩"

    num = 1
    for f in flist:
        os.system(os.getcwd() +"KLT2010-TestVersion-2017₩₩EXE₩₩index.exe -sw " + path + f + " TLIST_"
                  + str(num) + ".txt")
        os.system('move' + " TLIST_" + str(num) + ".txt " + move_path)
        num+=1
```

복사한 txt파일을 index.exe를 이용하여 형태소 분석

형태소 분석으로 term들만 추출한 각 파일들에 대해 wordcount.exe를 이용하여 unique term들만 저장한다.

2) Term table 작성

```
def wordcount_df():
    path = os.getcwd() + "₩₩result₩₩"
    flist = os.listdir(path)
    os.system("mkdir output")

    for fname in flist:
        os.system("wordcount.exe -new -1 " + path + fname +
                " " + os.getcwd() + "₩₩output₩₩" + fname)
# TermTable 구축
try:
    data = open("termtable.txt", "r")
except IOError:
    os.system("copy " + os.getcwd() + "₩₩output₩₩*.txt all.txt")
    os.system('wordcount.exe -new all.txt termtable.txt')
finally:
    data = open("termtable.txt", "r")
```

unique term들만 저장한 txt파일을 하나의 파일로 합친 후 wordcount.exe로 빈도를 계산한다(DF계산)

3) IDF table 작성

```
# mapping 및 IDF계산
lines = data.read().split('₩n')
lines = lines[:-1] # remove last null line
id_count = 1
id_list = []

num_f = len(flist) # 총 문서 개수 = 418

for i in lines:
    new_list = i.split('₩t')

    # TERM ID, TERM, DF, IDF
    id_list.append((id_count, new_list[1], int(new_list[0]), math.log10(num_f / int(new_list[0]))))
    id_count += 1
data.close()
```

id_list를 만들어서 Term id, Term, DF, IDF순으로 저장한다.

4) 각 문서에 대한 자질 벡터 구성

```
# 각 문서 i에 대한 자질벡터 구성
# tf(ti) = freq(ti) / maxTF --> freq(ti)사용
# 각 문서마다 freq 계산
path = os.getcwd() + '₩₩freq₩₩'
flist = glob.glob(path+"*.txt")
if not flist:
    wordcount_freq()
def wordcount_freq():
    os.system("mkdir freq")
    path = os.getcwd() +"₩₩docs₩₩"
    flist = os.listdir(path)

    for fname in flist:
        os.system("wordcount.exe -0 " + path + fname +
                " " + os.getcwd() + "₩₩freq₩₩" + fname)
```

각 파일마다 wordcount.exe를 이용하여 term의 빈도를 계산한다.

```
def idf(word, id_list):
    for term in id_list:
        if term[1] == word:
            return float(term[3])
```

idf는 미리 계산해 놓은 것을 사용한다.

2. Similarity matrix 구성

1) 문서 Di, Dj에 대한 유사도 계산 - Cosine similarity

코사인 유사도(Cosine similarity) 는 내적공간의 두 벡터간 각도의 코사인값을 이용하여 측정된 벡터간의 유사한 정도를 의미한다. 정보 검색 및 텍스트 마이닝 분야에서, 단어 하나 하나는 각각의 차원을 구성하고 문서는 각 단어가 문서에 나타나는 회수로 표현되는 벡터값을 가진다. 이러한 다차원 공간에서 코사인 유사도는 두 문서의 유사를 측정하는 매우 유용한 방법이다.

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

```
def similarity_di_dj(num1, num2, id_list):
    print = "calculate similarity " + "%d" %int(num1) + ".txt" + " %d"%int(num2) + ".txt"
    di_file = open(os.getcwd() + "₩₩freq₩₩" + "%d" % int(num1) + ".txt", "r")
    dj_file = open(os.getcwd() + "₩₩freq₩₩" + "%d" % int(num2) + ".txt", "r")

    lines = di_file.read().split('₩n')
    lines = lines[:-1] # remove last null line

    di_list = []
    for i in id_list:
        count_line = 0
        for j in lines:
            new_list = j.split('₩t')
            if i[1] == new_list[1]:
                w = idf(new_list[1], id_list)
                if w == None:
                    w = 0
                di_list.append((new_list[1], int(new_list[0]) * w))
                break
            count_line += 1

        if count_line == len(lines):
            di_list.append((i[1], 0))
    di_file.close()
```

```
    lines = dj_file.read().split('₩n')
    lines = lines[:-1] # remove last null line

    dj_list = []
    for i in id_list:
        count_line = 0
        for j in lines:
            new_list = j.split('₩t')
            if i[1] == new_list[1]:
                w = idf(new_list[1], id_list)
                if w == None:
                    w = 0
                dj_list.append((new_list[1], int(new_list[0]) * w))
                break
            count_line += 1

        if count_line == len(lines):
            dj_list.append((i[1], 0))
    dj_file.close()
```

```
    # 벡터 내적 / di 크기 / dj 크기
    v = 0
    di = 0
    dj = 0
    for  i in range(len(id_list)):
        v = v + di_list[i][1]*dj_list[i][1]
        di = di + math.pow(di_list[i][1], 2)
        dj = dj + math.pow(dj_list[i][1], 2)
    di = math.sqrt(di)
    dj = math.sqrt(dj)

    print("v = " + str(v))
    print("di = " + str(di))
    print("dj = " + str(dj))
    print("similarity = " + str(v / (di * dj)))
```

```
calculate similarity 1.txt 207.txt
v =  587.358222896
di =  33.3820805807
dj =  91.421551479
similarity =  0.192460258497
```

비교하려는 두 문서를 2개의 <tid, weight> 리스트 형태로 만든 뒤 코사인 유사도 공식(두 벡터의 내적한 값을 각 벡터 크기 곱으로 나눈다)을 사용하여 유사도를 비교한다.

2) Upper triangular matrix

```
def make_upper(num_f, id_list):
    m = [[0 for col in range(num_f)] for row in range(num_f)]
    for i in range(num_f):
        for j in range(i, num_f):
            if i == j :
                m[i][j] = 1
            else:
                similarity_di_dj(i+1, j+1, id_list)

    m = np.triu(m, k = 0)
```

배열을 만들어서 배열의 i, j원소에 i.txt파일과 j.txt파일의 유사도를 계산한다.