보고서

컴퓨터학과 2014320052 유지은

코드설명

전체로직

1. globalList생성- 60개의 문서(URL)에 대한 단어 사전 생성
2. 제공받은60개의URL을 바탕으로 주제별로 3개의vector를 생성
(golf벡터, entertainment벡터,politics벡터)
3. 테스트데이터(Document폴더)의vector를 생성
4. 테스트데이터와 주제별3개의vector간의similarity를 비교하여similaruty가 가장 높게 나온 벡터가 그 테스트데이터의 토픽을 나타낸다.
5. 단어의 id(단어사전이 globalsSet에서의 단어위치를 나타냄)와 해당 단어의 빈도수를 묶는다. 빈도수를 기준으로 정렬하고, 상위랭크의 단어id를 globalList에서 찾아 해당 상위랭크의 단어(string)을 출력한다.

get_stemmedTokens_fromUrl(url) 함수

url로부터 스테밍된 토큰을 얻는 함수이다.

크롤링(헤드라인,본문,이미지캡션), 토큰화, 스탑워드제거, 스테밍의 단계를 거친 토큰들을 반환한다. 해당 함수를 사용하여 얻은 토큰을 매번(총60번) union하여 globalList에 저장한다. 이로써 globalList를 제공된 문서60개에 대한 단어사전으로 만든다.

1. 크롤링에서 유의했던 점은 첫번째 문단이 크롤링되지 않는 것을 해결하는 것이었다.

```
##BODY 중에서 첫문단 크롤링 - 위의 BODY크롤링에서는 첫문단이 빠지고 크롤링된다.
##모든 기사의 첫 문단마다 반복되는 (CNN)이라는 글자는 크롤링 해오지 않도록 하였다.
if bs.find('cite', class_="el-editorial-source"):
    bs.find('cite', class_="el-editorial-source").decompose()
para1=bs.findAll('div','el__leafmedia el__leafmedia--sourced-paragraph')
first_para = ''
for p in para1:
    first_para += p.getText()
```

본문의 태그를 크롤링하더라도 첫번째 문단은 크롤링이 되지않았기때문에 별도로 첫번째문단의 태그에 대한 크롤링을 진행하였다. 그리고 이때 모든 뉴스기사의 첫문단마다 반복되는 CNN이라는 글자는 문서의 주제별 분류에 큰 영향을 미치지 않을 것으로 판단하여 decompise메소드를 사용하여 크롤링하지 않도록 하였다.

2. 전처리에서 유의했던 점은nltk가 미처 걸러내지 못하는stopword들에 대한 처리이다.느낌표나 마침표,구두점 등nltk가 걸러내지 못하는 부분들에 대하여 따로 처리를 시행하였다. from string import punctuation을 사용하여 일반적인punctuation에 대한 제거를 실시하였고, 그래도 제거되지 않는 불용어는 mystop2라는 별도의 리스트에 담아 텍스트에서 제거되도록 하였다.

```
## Pre-processing: Stopwords(1)
stop = set(stopwords.words('english'))
tokens = [i for i in tokens if i not in stop]

## Pre-processing: Stopwords(2) : NLTK가 Stopwords로 처리하지 못하는 것 따로 처리
mystop = set(punctuation)
mystop2 = ["'",'--','``',"''",'.',',']
tokens = [i for i in tokens if i not in mystop]
tokens = [i for i in tokens if i not in mystop2]
```

또한 스테머는 스노우볼 스테머를 사용하였다.
포터스테머도 존재하였으나 스노우볼 스테머가 포터스테머를 개선한 포터2스테머이기 때문에 기존의 포터스테머보다 성능이 좋을 것으로 판단하였기 때문이다.

```
## Pre-processing: Stemming using SnowballStemmer
snowball_stemmer = SnowballStemmer("english")
stemmedTokens= []
for token in tokens:
    stemmedTokens.append(snowball_stemmer.stem(token))
```

get_stemmedTokens_fromTxt(path) 함수

위의 get_stemmedTokens_fromUrl(url)함수와 같은 역할을 한다. -텍스트에 대한 토큰화, 스탑워드제거, 스테밍- 다만, get_stemmedTokens_fromUrl가 매개변수로 url을 받았다면, get_stemmedTokens_fromTxt 함수는 매개변수로 컴퓨터상에 존재하는 텍스트파일을 받는다. 웹상에서 내용을 가져오는 것이 아니기때문에 크롤링하는 부분이 빠져있고 대신에 파일을 여는 소스코드가 추가되어 있다.

```
def get_stemmedTokens_fromTxt(path):
    with open(os.path.join(os.getcwd(), path)) as f:
        runningText = f.read()
```

Vectorization

1. 주제별 vector의 생성

globalList의 생성이 끝나면 이와 같은 크기의 벡터(golf_Vec,ent_Vec,pol_Vec)를 주제별로 3개 생성합니다.그 뒤, 카테고리별로 문서(category폴더에 있는 문서들)를 순회하며 문서에 등장하는 단어들을 토큰화한 리스트를 얻어옵니다. 얻어온 토큰들을 바탕으로, 만약 golf카테고리의 문서들을 순회하였다면 golf_vec에서 단어가 있는 위치에 +1을 하여 가중치를 증가시킵니다

2. 테스트데이터(document폴더내의 문서)의 vector의 생성

```
golf_Vec = [0 for i in range(0, len(globalSet))]
ent_Vec = [0 for i in range(0, len(globalSet))]
pol_Vec = [0 for i in range(0, len(globalSet))]

## golf 주제의 topicVector생성 - count scheme
golfTxt=['category/golf_20.txt']
for fname in golfTxt:
    with open(os.path.join(os.getcwd(), fname)) as f:
        urls = f.readlines()
        for u in urls:
            text = get_stemmedTokens_fromUrl(u.strip())
            for w in text:
                if w in globalList:
                    golf_Vec[globalList.index(w)] += 1
```

위에서 주제별 벡터가 완성되었으면 이번에는 테스트하고자하는 데이터의 벡터를 생성합니다. 전체 단어사전(globalSet)과 같은 크기의 벡터를 선언하고, 테스트데이터의 토큰을 얻어옵니다. 토큰에 있는 단어는 그에 해당하는 자리의 테스트데이터_벡터에서 가중치가+1씩 증가합니다.

```
testText = get_stemmedTokens_fromTxt('document/golf.txt')
testText_Vec = [0 for i in range(0, len(globalSet))]

for w in testText:
    if w in globalList:
        testText_Vec[globalList.index(w)] += 1
```

Classification

위에서 주제별로 벡터와 테스트하고자 하는 데이터의 벡터가 완성되었다. 주제별 벡터와 테스트하고자 하는 데이터의 벡터를 내적시켜 cosine_similarity를 구하고, 그중 가장 높은 수치를 갖게하는 주제의 벡터가 그 데이터의 주제를 나타내는 것으로한다.

```
def cosineSimilarity(v1, v2):
        multi = (v1.dot(v2)).sum()
        x = math.sqrt((v1*v1).sum())
        y = math.sqrt((v2*v2).sum())

        result = multi/(x*y)
        return result
```

```
similarity_with_golf = cosineSimilarity(np.array(golf_Vec), np.array(testText_Vec))
print('The similarity with golf is',similarity_with_golf)

similarity_with_ent = cosineSimilarity(np.array(ent_Vec), np.array(testText_Vec))
print('The similarity with entertainment is',similarity_with_ent)

similarity_with_pol = cosineSimilarity(np.array(pol_Vec), np.array(testText_Vec))
print('The similarity with politics is',similarity_with_pol,'₩n')

if similarity_with_golf> similarity_with_ent  and similarity_with_golf>similarity_with_pol:
    print("This text's topic is ★golf★")
if similarity_with_ent> similarity_with_golf  and similarity_with_ent>similarity_with_pol:
    print("This text's topic is ★entertainment★")
if similarity_with_pol> similarity_with_golf  and similarity_with_pol>similarity_with_ent:
    print("This text's topic is ★politics★")
```

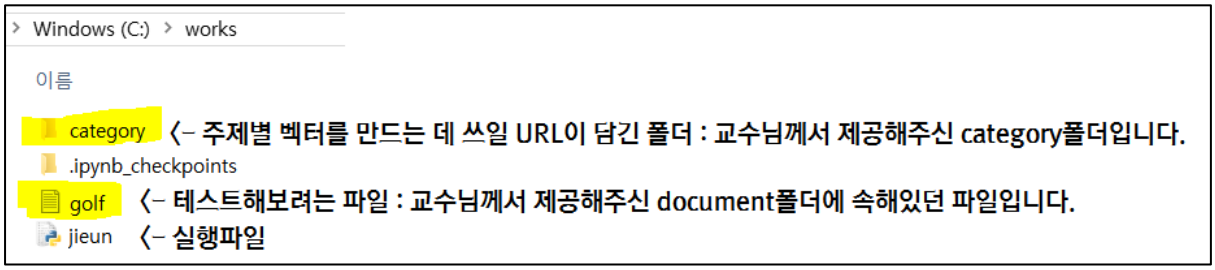
Visualizaiton

단어의 id(단어사전이 globalsSet에서의 단어위치를 나타냄)와 해당 단어의 빈도수를 묶는다. 빈도수를 기준으로 정렬하고, 상위랭크의 단어id를 globalList에서 찾아 해당 상위랭크의 단어(string)을 출력한다.

```
#golf
print('Golf Text Top 5 words')
x = [i for i in range (0,len(globalSet))] #x에 단어의 id(위치)를 담는다

# zip을 사용해서 x와 ent_Vec을 묶음으로써 -> id와 단어빈도수를 묶는다
topList = [(x,y) for x,y in zip(x, golf_Vec)]
#sort를 사용해서 빈도수로 정렬(오름차순)한다.
topList.sort(key=lambda x:x[1])
#오름차순 정렬에서 뒤에서 5개(빈도수 최다 5개)를 뽑는다.
topList = topList[-5:]

labelOrder = [5,4,3,2,1]
#현재 topList는 (x,y)의 투플형태로 (id(위치),단어)의 데이터를 오름차순 정렬해서 가지고있다

#topList에서 id(위치)를 가져와서 globalList에 해당위치의 단어를 가져온다
topWordLabels = [globalList[x] for (x,y) in topList]

#topList에서 단어(y)를 가져와서 globalList에 해당위치의 단어를 가져온다
topWordCounting = [y for (x,y) in topList]


plt.bar(labelOrder, topWordCounting)
plt.xticks(labelOrder, topWordLabels)
plt.show()
```

실행방법

0. 실행파일이 있는 위치에는 category폴더와 테스트해보려는파일이 함께 위치해 있어야한다.



1. anaconda prompt 실행.

2.실행파일이 있는 위치로 이동.

3. python jieun.py(실행파일명) golf.txt(테스트파일) 입력.

```
(C:₩Users₩primo₩Anaconda3) C:₩works>python jieun.py golf.txt
```

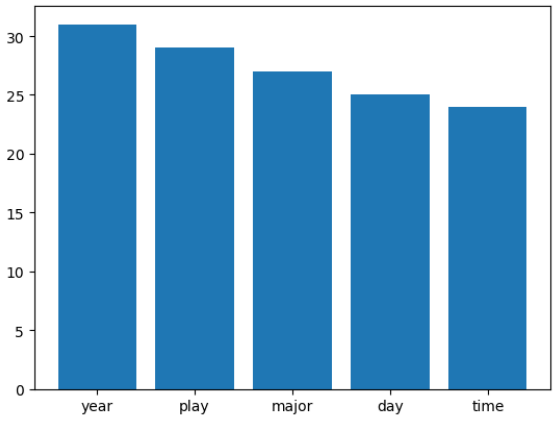
결과

```
(C:₩Users₩primo₩Anaconda3) C:₩works>python jieun.py golf.txt
The similarity with golf is 0.453105549295
The similarity with entertainment is 0.267290053018
The similarity with politics is 0.275733781912

This text's topic is ★golf★
```

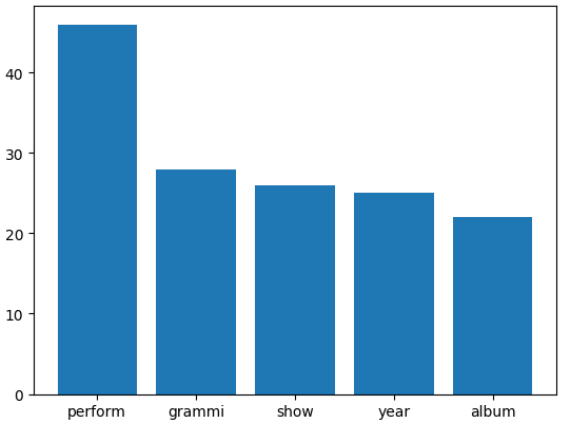
golf.txt데이터(교수님께서 제공해주신 데이터)를 테스트해봤다. golf주제벡터와의 similarity가 다른 주제에 비해 2배가량 높게나왔다. 따라서 golf.txt데이터의 주제를 나타내는 것은 golf이다.

Visualization

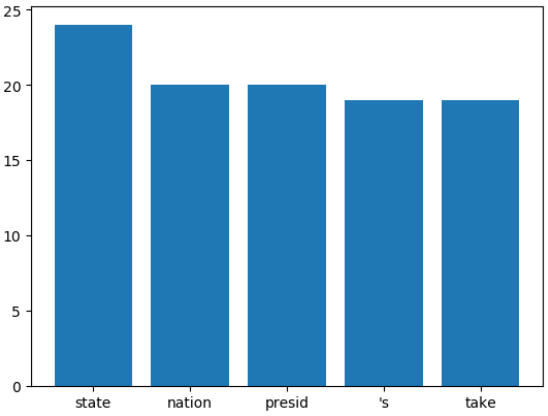
〈Golf〉



〈Entertainment〉



〈Politics〉



이번 과제를 통한 나의 생각

주제별로 보다 뚜렷하게 유사도가 나타난다면 더 좋겠다는 생각이 들었다. 자연어처리 시간에 part of speach tagging에 대하여 배운적이 있는데, 그것을 통해 토큰들을 품사로 태깅한 뒤, 그중에서 명사만을 뽑아낼 수 있다면 보다 주제별 유사도를 뚜렷하게 나타낼 수 있을 것으로 생각된다. 왜냐하면, 주제별로 자주 나타나는 고유명사가 존재할 것이고, 동사 같은 경우에는 주제를 불분하고 general한 것이 보다 많을 것으로 예상되기 때문이다.