A Simple Document Classifier
using cosine similarity

박단우
Korea University
Dept. of Computer Science and Engineering

1. 코드 설명

```
training_files = ['category/entertainment_20.txt', 'category/golf_20.txt', 'category/politics_20.txt']

snowball_stemmer = SnowballStemmer("english")

# stopwords
stop = set(stopwords.words('english'))
stop = stop | set(['"', '--', "''", '``', ',', '.'])

word_set = set([]) # word global set
```

트레이닝 경로 지정, 스테머 객체 생성, 불용어 집합 구축
Vectorization 에 사용할 전체 단어 집합을 위한 word_set 변수 선언.

Crawling functions

```
def crawler(url):
    htmlData = urllib.request.urlopen(url)
    bs = bs4.BeautifulSoup(htmlData, 'lxml')
    text = ''# parsed_text
    # parse header
    header = bs.find('h1', 'pg-headline')
    if header is not None:
        text += header.getText()
    # parse image caption
    caption = bs.find('div', 'js-media__caption media__caption el__storyelement__title')
    if caption is not None:
        text += caption.getText()
    # parse image title
    imgTitle = bs.find('div', 'media__caption el__gallery_image-title')
    if imgTitle is not None:
        text += imgTitle.getText()
    # parse paragraphs
    first = bs.find('cite', class_="el-editorial-source")
    if first is not None:
        first.decompose()

    text += bs.find('p', 'zn-body__paragraph').getText()
    bodies = bs.findAll('div', 'zn-body__paragraph')
    for b in bodies:
        text += b.getText()
    return text.lower()
```

URL 을 인자로 주면 제목, 이미지 캡션 및 타이틀, 첫번째 문단을 포함한 모든 문단을 하나의 문자열로 만든 뒤 소문자로 바꾸어 반환하는 함수.

Preprocessing Functions

```
def preprocessing(text):
    tokens = nltk.word_tokenize(text)
    tagged = nltk.pos_tag(tokens)
    tokens = [i for (i,j) in tagged if i not in stop and j in ['NN','NNP','NNPS','NNS']]
    stemmed_tokens = []
    for token in tokens:
        stemmed_tokens.append(snowball_stemmer.stem(token))
    return stemmed_tokens
```

인자로 받은 문자열을 토큰화, 불용어 제거, 스테밍 하여 토큰 리스트로 반환하는 함수. 이 때, nltk 의 part of speech tagging 라이브러리를 사용하여 명사들만 뽑는다. 이는 우리가 명사들만 들어도 대충 주제가 무엇인지 현실에서 파악할 수 있기 때문이다.

```
def preprocessing_with_file(fname):
    with open(os.path.join(os.getcwd(), fname)) as f:
        return preprocessing(f.read())
```

파일 이름을 받아서 그 내용을 읽은 뒤 전처리하여 반환하는 함수.
반환값은 토큰 리스트가 된다.

```
def collect_words_from_url_file(wset, fname):
    with open(os.path.join(os.getcwd(), fname)) as f:
        urls = f.readlines()
        for u in urls:
            wset = wset | set(preprocessing(crawler(u.strip())))
    return wset
```

url 경로가 담긴 파일을 받아서 그 내용을 읽어서 각각의 url을 크롤링한 뒤 전처리하여 생성된 단어를 모두 모아서 반환하는 함수.

```
def construct_word_list(wlist, fname):
    with open(os.path.join(os.getcwd(), fname)) as f:
        urls = f.readlines()
        for u in urls:
            wlist = wlist + preprocessing(crawler(u.strip()))
    return wlist
```

url이 쓰여진 파일 이름을 받아서 모든 url에 대해 크롤링한 뒤 중복을 제거하지 않고 전처리만 하여 반환하는 함수

Vectorization

```
# construct global word set for 60 articles
for fname in training_files:
    word_set = word_set | collect_words_from_url_file(word_set, fname)
```

문서 벡터의 column을 정하기 위해 일단 트레이닝 셋에 있는 모든 문서들(60개)에 대해서 등장하는 모든 단어들의 셋을 구성한다.

```
# create vector index
word_set = list(word_set)

# define topic vectors
ent_vec = [0 for i in range(0, len(word_set))]
golf_vec = [0 for i in range(0, len(word_set))]
pol_vec = [0 for i in range(0, len(word_set))]
```

각각의 Topic에 해당하는 벡터를 만든다. 이 때 벡터의 열 길이는 위에서 생성한 모든 단어들의 셋의 크기가 될 것이고 초기값은 단어가 한번도 등장하지 않았으므로 0으로 해준다.

```
# count word frequency
def counting(tokens, vec, global_list):
    for w in tokens:
        if w in global_list:
            vec[global_list.index(w)] += 1
    return vec
```

인자로 토큰 리스트와 업데이트할 Topic 벡터와 비교할 모든 단어들이 담긴 리스트를 받은 뒤 만약 token에 있는 단어가 전역 리스트에도 있다면 해당 단어의 위치에 해당하는 값을 Topic 벡터에서 증가시켜 준다. tokens는 중복이 제거되지 않은 리스트이다. 마지막엔 업데이트가 완료된 벡터를 반환한다.

```
ent_vec = counting(construct_word_list([], training_files[0]), ent_vec, word_set)
golf_vec = counting(construct_word_list([], training_files[1]), golf_vec, word_set)
pol_vec = counting(construct_word_list([], training_files[2]), pol_vec, word_set)
```

각각의 topic vector에 대해 위의 함수를 호출하여 값을 업데이트 해 준다. 결과는 각각의 주제를 나타내는 벡터가 된다. (단어의 빈도수가 반영된)

Classification

```
import numpy as np
import math

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
```

코사인 유사도를 구하는 함수 정의. 두 벡터를 받아서 코사인 유사도를 반환한다.

```
def classify(doc):
    sim_with_ent = cosineSimilarity(np.array(ent_vec), np.array(doc))
    sim_with_golf = cosineSimilarity(np.array(golf_vec), np.array(doc))
    sim_with_pol = cosineSimilarity(np.array(pol_vec), np.array(doc))

    print('sim_with_ent: '), print(sim_with_ent)
    print('sim_with_golf: '), print(sim_with_golf)
    print('sim_with_politics: '), print(sim_with_pol)

    if sim_with_ent > sim_with_golf and sim_with_ent > sim_with_pol:
        return 'entertainment'
    elif sim_with_golf > sim_with_ent and sim_with_golf > sim_with_pol:
        return 'golf'
    elif sim_with_pol > sim_with_ent and sim_with_pol > sim_with_ent:
        return 'politics'
```

문서 벡터를 받아서 각각의 토픽 벡터와의 코사인 유사도를 구한 뒤 비교하여 분류 결과를 문자열로 반환한다. 각 토픽 벡터와의 유사도도 출력해 준다.

```
# test
ent_test_vec = [0 for i in range(0, len(word_set))]
golf_test_vec = [0 for i in range(0, len(word_set))]
pol_test_vec = [0 for i in range(0, len(word_set))]

ent_test_vec = counting(preprocessing_with_file(test_files[0]), ent_test_vec, word_set)
golf_test_vec = counting(preprocessing_with_file(test_files[1]), golf_test_vec, word_set)
pol_test_vec = counting(preprocessing_with_file(test_files[2]), pol_test_vec, word_set)

print('entertainment is classified as : ')
print (classify(ent_test_vec))

print('golf is classified as : ')
print (classify(golf_test_vec))

print('politics is classified as : ')
print (classify(pol_test_vec))
```

```
entertainment is classified as :
entertainment
golf is classified as :
golf
politics is classified as :
politics
```

각각의 테스트 문서에 대해서 문서벡터를 구성하고 분류기에 넣어서 결과값을 확인한다.

Visualization

```
import matplotlib.pyplot as plt

x = [i for i in range(0, len(word_set))]
freq_list = [(x,y) for x,y in zip(x, ent_vec)]
freq_list.sort(key=lambda x:x[1])
freq_list = freq_list[-5:]

WordOrder = [5,4,3,2,1]
WordFrequency = [x[1] for x in freq_list]

LABELS = [word_set[x[0]] for x in freq_list]

plt.bar(WordOrder, WordFrequency, align='center')
plt.xticks(WordOrder, LABELS)
plt.show()
```

단어 id, 빈도수 로 구성된 투플을 만들고 (zip 사용), 이것을 두번째 원소로 정렬하여 (key = lamda x : x[1]) 시각화 한다. 이 때 order는 [5,4,3,2,1]로 하여 빈도수 내림차순으로 시각화 한다.

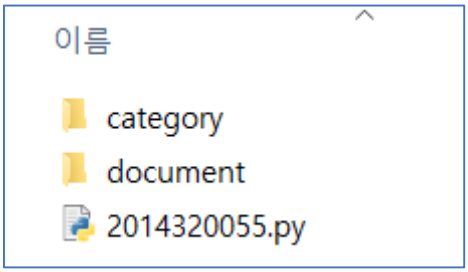
```
x = [i for i in range(0, len(word_set))]
freq_list = [(x,y) for x,y in zip(x, golf_vec)]
freq_list.sort(key=lambda x:x[1])
freq_list = freq_list[-5:]

WordOrder = [5,4,3,2,1]
WordFrequency = [x[1] for x in freq_list]

LABELS = [word_set[x[0]] for x in freq_list]

plt.bar(WordOrder, WordFrequency, align='center')
plt.xticks(WordOrder, LABELS)
plt.show()
```

위의 과정을 golf 토픽 벡터에 대해서 동일하게 수행한 것이다.

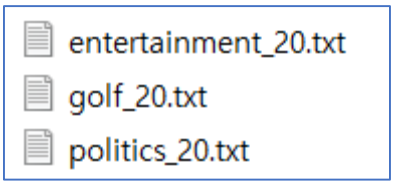
```
x = [i for i in range(0, len(word_set))]
freq_list = [(x,y) for x,y in zip(x, pol_vec)]
freq_list.sort(key=lambda x:x[1])
freq_list = freq_list[-5:]

WordOrder = [5,4,3,2,1]
WordFrequency = [x[1] for x in freq_list]

LABELS = [word_set[x[0]] for x in freq_list]

plt.bar(WordOrder, WordFrequency, align='center')
plt.xticks(WordOrder, LABELS)
plt.show()
```

위의 과정을 정치 토픽 벡터에 대해서 동일하게 수행한 것이다.

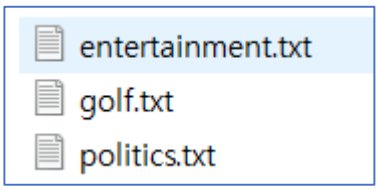
3.실행 방법



위의 그림과 같이 스크립트 파일을 category와 document 디렉토리와 함께 위치시킨다.



(category 디렉토리의 내용)



(document 디렉토리의 내용)

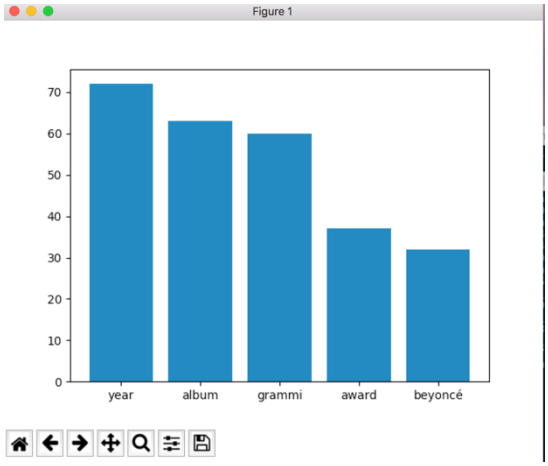
python 2014320055.py document/테스트파일명.txt
로 스크립트를 실행시키면 된다.
ex) 명령어: python 2014320055.py document/golf.txt

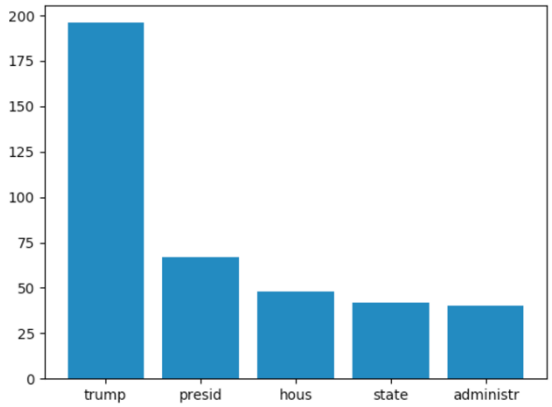
결과

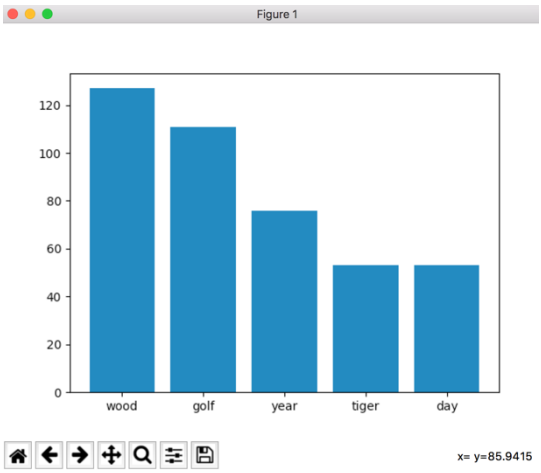
```
(venv-data-science)  veritas@Danwoos-MacBook-Pro  ~/python_과제  python 201432
0055.py document/golf.txt
sim_with_ent:
0.0797585037191
sim_with_golf:
0.707363176528
sim_with_politics:
0.0651662477195
document/golf.txt  is classified as :  golf
(venv-data-science)  veritas@Danwoos-MacBook-Pro  ~/python_과제  python 201432
0055.py document/entertainment.txt
sim_with_ent:
0.190377117244
sim_with_golf:
0.139895336425
sim_with_politics:
0.0766716847264
document/entertainment.txt  is classified as :  entertainment
(venv-data-science)  veritas@Danwoos-MacBook-Pro  ~/python_과제  python 201432
0055.py document/politics.txt
sim_with_ent:
0.0506207463553
sim_with_golf:
0.0565862179292
sim_with_politics:
0.209780047215
document/politics.txt  is classified as :  politics
```

위에서부터 테스트 문서와 토픽 벡터간의 코사인 유사도 값이 나오고
그 결과 어떤 토픽으로 분류되었는지 결과가 나온다.

4.Visualization







(왼쪽 상단) entertainment 토픽 벡터에 대해 상위 빈도수 단어 5개 시각화. Year 는 정치에서도 많이 나온 단어인데, 트레이닝 셋에 그래미 시상식 관련 기사가 많은지 grammi, award, year (this year's grammi award winner) 와 같은 단어들이 많이 나온다.

(왼쪽 하단)

politics 토픽 벡터 상위 빈도수 5개 단어 시각화. House는 white와 같이 갈 것 같은데 (white house) pos tagging 으로 명사만 추출해서 white가 날아 간 것 같다. ~행정부 할 때 의 표현 때문에 administer 도 상위권 임을 확인할 수 있다.

(오른쪽 하단)

golf 토픽 벡터 상위 5개 단어 시각화. Tiger와 wood 는 타이거 우즈 때문 인 것으로 생각된다. (woods 가 스테밍되어 wood로)