영화 스포일러 분류 프로그램

광운대학교 2019.06.16

팀 원 : 2017204002 김영서
2017204016 이지환
2017204068 김동영

I. 주제 설명 및 선택 배경

실시한 프로젝트의 주제는 영화 줄거리 리뷰 분석을 통한 영화 스포 문장 분류이다. 훈련 데이터로 네이버 블로그 영화 줄거리 리뷰 글과 나무위키 줄거리를 사용하여 각 영화의 스포 내용을 담고 있는 문장을 분석하고, 훈련 데이터에서 얻은 기준을 바탕으로 테스트 데이터인 네이버 영화 사이트의 평점 리뷰 데이터를 Text Classification을 실시한다. 스팸메일 분류기와 같이 최종적으로 스포 리뷰 분류기를 만들어 내는 것이 목적이다.

최근 개봉한 '어벤져스: 엔드게임'의 스포 내용이 인터넷을 떠돌아 많은 네티즌들이 피해를 입었고, 오프라인 상에서도 주변인에게 주요 내용을 듣게 되어 피해를 입는 문제가 발생했다. 팀원들도 영화를 관람하기 전에 결말을 알게 되어 크게 피해를 봤다. 이러한 문제를 온라인상에서라도 방지하기 위해 스포 내용을 포함하고 있는 문장을 필터링하는 프로그램을 고안하게 됐다. 또한, 영화가 매우 친근한 주제이므로 줄거리를 포함하고 있는 데이터를 쉽게 얻을 수 있다고 생각했다.

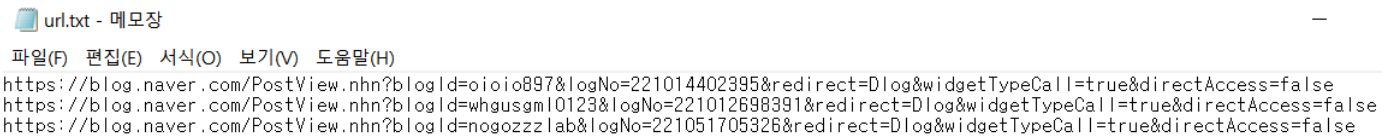
II. 원본 데이터 수집 방법 및 데이터 구성 요소 설명

원본 데이터에는 2가지 종류가 있다. 영화 줄거리, 리뷰 글과 네이버영화 평점 리뷰이다. 영화 줄거리, 리뷰 글은 각 영화의 나무위키 줄거리와 네이버 블로그 리뷰에서 가져와 한 영화당 21개의 다큐먼트를 가져왔다. 네이버영화 평점 리뷰는 각 영화에 해당하는 네이버 영화 사이트에서 호감순으로 정렬하여 약 1000개의 사용자 평점 리뷰 데이터를 가져왔다.

이렇게 2가지 종류의 데이터를 가져온 이유는 영화 줄거리, 리뷰 데이터는 직접 분석을 하여 스포에 해당하는 훈련 데이터를 추출하기 위한 선행 데이터인 셈이고, 네이버영화 평점 데이터는 앞서 추출한 훈련 데이터와 비교하여 그 리뷰가 스포인지 아닌지를 분류하기 위한 테스트 데이터이다.

가) 영화 줄거리, 리뷰 데이터

1. 각 영화의 줄거리와 리뷰가 써진 블로그 글의 URL을 복사해서 하나의 txt파일에 저장한다.



이런 식으로 한 영화당 20개의 URL, 총 100개의 URL이 하나의 txt파일에 저장되어야 한다.

2. 다음 코드는 블로그 URL을 불러와 그 글의 제목과 본문을 크롤링하는 함수이다.

```
from bs4 import BeautifulSoup
import requests
import urllib.request as req
import pandas as pd

def get_text(final_url):
    try :
    ##제목과 본문부분 추출
    res = req.urlopen(final_url)
    soup = BeautifulSoup(res, 'html.parser')
    temp = soup.select("#se_textarea")
    ##title 추출
    list=[]
    title = soup.head.find("meta", {"property":"og:title"}).get('content')
    list.append(title)
    ##본문 추출
    temp = soup.findAll("div", {"id":"postViewArea"})
    if temp == []:
            temp=soup.findAll("div", {"class":"se-module se-module-text"})
    if temp == []:
            temp = soup.findAll("p", {"class":"se_textarea"})

    text_str = ""
    for a in temp:
            text_str += a.get_text()
    list.append(text_str)
```

```
        list.append(final_url)
        return list
    except:
        print("크롤링 실패")
```

크롤링에 사용된 라이브러리는 requests와 beautifulsoup이다. requests 모듈을 이용하여 함수의 인자로 받는 URL 주소의 HTML 소스를 얻고, beautifulsoup를 사용하여 HTML 소스코드를 파싱한다. 우리가 필요한 블로그 글의 제목과 본문 글만을 가져오기 위해서는 그 텍스트가 속한 태그를 식별할 클래스명을 알아야한다. 직접 블로그 글의 html 소스코드를 살펴본 결과, 블로그 title은 meta태그의 "property", "og,title"이라는 속성이 있는 클래스의 "content" 속성에 있다는 것을 알아냈다. beautifulsoup를 이용하여 조건에 맡는 태그에 접근해 블로그 제목을 얻을 수 있었다. 이와 마찬가지로 블로그 본문 글의 태그와 클래스명을 찾았다. 하지만 네이버 블로그 게시글 에디터의 버전과 최신 글이냐 아니냐에 따라 각각 본문 글에 접근할 수 있는 태그들이 달랐다. 총 3가지 경우의 수가 있다는 걸 발견하고 3가지 경우 모두 소스코드에 포함시켰다. 이렇게 블로그 title과 블로그 본문 글을 끌어와 리스트에 저장시켜 결과 값으로 반환 시키는 함수이다.

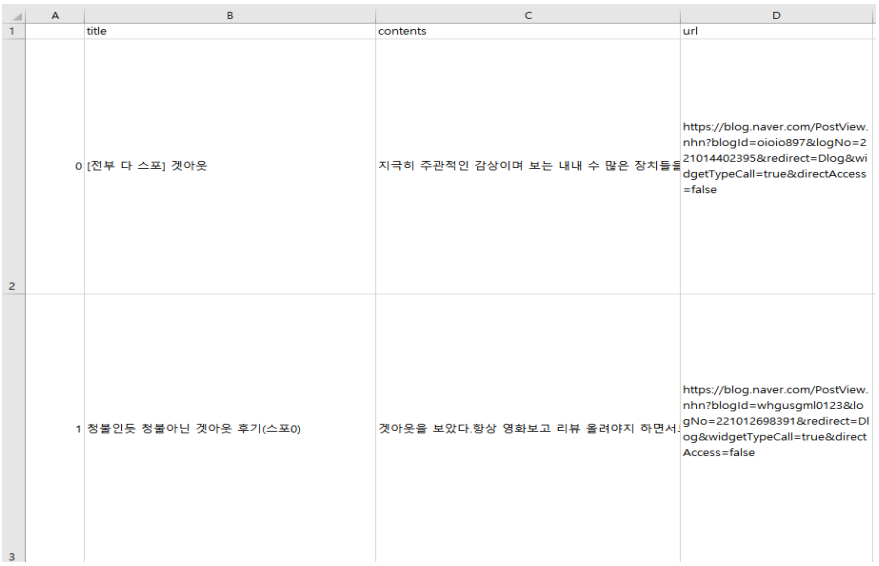
3. 다음 코드는 각 영화마다 리뷰 글을 크롤링하여 csv파일로 저장하는 코드이다.

```
f = open("url.txt", 'r')
lines = f.readlines()
results = []
movie_title={1:"겟아웃",2:"부산행",3:"서치",4:"해피데스데이",5:"곡성"}

for i, URL in enumerate(lines):
    i += 1
    if i%20 == 0:
            results.append(get_text(URL))
            data = pd.DataFrame(results)
            data = data.applymap(lambda x:
x.replace('₩xa0','').replace('₩xa9','').replace('₩u200b','').replace('₩ufeff','').replace('₩u119e',''))
            data.columns = ['title', 'contents', 'URL']
```

```
            title = "%s_네이버블로그리뷰.csv" %(movie_title[i/20])
            data.to_csv(title, encoding = 'cp949')
            print("%s 파일생성 완료" %(movie_title[i/20]))
            results = []
    else:
            results.append(get_text(URL))

    f.close()
```

앞서 저장한 100개의 URL모음 텍스트 파일을 불러오고 각 URL을 크롤링 함수의 인자로 넘겨주어 반환한 결과값(제목, 본문 내용)을 모두 모아 csv 파일로 만들게 된다. 이때 각 영화마다 딕셔너리로 번호를 부여하여 csv 파일이 따로따로 저장되게 하였다. 한 영화당 20개의 URL이 있는데 전체 URL은 100개이므로, 20개의 URL을 크롤링하면 다시 초기화되어 다음 영화로 넘어가는 식으로 반복문을 작성했다. 리스트를 사용하여 데이터들을 처리하였으므로 리스트를 csv 파일로 저장하기 위해 pandas 라이브러리를 사용했다.

4. 최종적으로 만들어진 "영화제목_네이버블로그리뷰.csv"파일은 다음과 같다.

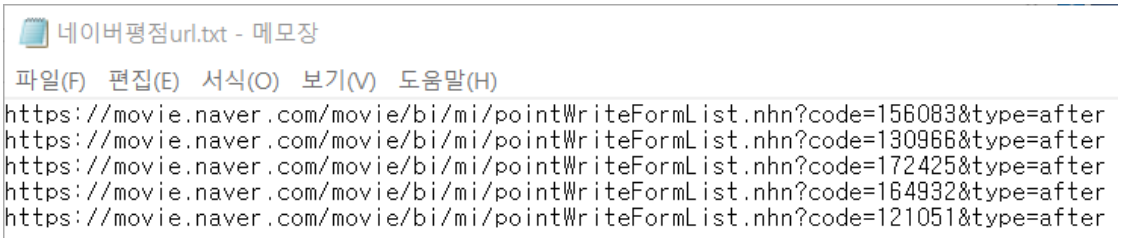


	A	B	C	D
1		title	contents	url
2	0	[전부 다 스포] 겟아웃	지극히 주관적인 감상이며 보는 내내 수 많은 장치들을	https://blog.naver.com/PostView.nhn?blogId=oioio897&logNo=221014402395&redirect=Dlog&widgetTypeCall=true&directAccess=false
3	1	정불인듯 정불아닌 겟아웃 후기(스포0)	겟아웃을 보았다.항상 영화보고 리뷰 올려야지 하면서	https://blog.naver.com/PostView.nhn?blogId=whgusgml0123&logNo=221012698391&redirect=Dlog&widgetTypeCall=true&directAccess=false

title 열은 블로그 글 제목, contents 열은 블로그 본문 내용, URL 열은 크롤링한 블로그 원본 주소를 의미한다. 총 20행이지만 21행 열에 나무위키 줄거리도 수작업(복사, 붙여넣기)으로 추가하여 총 21행의 데이터를 원본 데이터를 수집했다.

나) 네이버영화 평점 데이터

1. 각 영화의 네이버영화 평점 사이트의 URL을 복사해서 하나의 txt파일에 저장한다.



2. 다음 코드는 평점 사이트 URL을 불러와 평점 리뷰를 남긴 네티즌들의 닉네임, 날짜, 리뷰내용을 크롤링하는 함수이다.

```
import requests
from bs4 import BeautifulSoup
from datetime import datetime
import pandas as pd

def get_data(url,resp,html):
    resp = requests.get(URL)
    html = BeautifulSoup(resp.content, 'html.parser')
    score_result = html.find('div', {'class': 'score_result'})
    lis = score_result.findAll('li')
    results=[]
    for li in lis:
        nickname = li.findAll('a')[0].find('span').getText()
        created_at = datetime.strptime(li.find('dt').findAll('em')[-1].getText(), "%Y.%m.%d %H:%M")
        review_text = li.find('p').getText()
        review=[nickname,review_text,created_at]
        results.append(review)
    return results
```

전반적으로 앞서 설명했던 블로그 줄거리, 리뷰 크롤링과 크게 다르지 않다. 사용한 라이브러리, 원리가 모두 같고 단지 크롤링해서 가져와야 할 요소의 태그, 클래스명이 다를 뿐이다. 닉네임, 날짜, 평점 리뷰 내용을 가져오기 위해 html 코드를 보고 각각의 태그와 클래스명을 찾아냈다. 그리고

beautifulsoup를 이용하여 html을 파싱하여 각 태그에 접근해 데이터를 뽑아와 클래스에 저장하여 반환하도록 함수를 구성했다.

3. 다음 코드는 각 영화마다 네이버 평점 리뷰를 크롤링하여 csv파일로 저장하는 코드이다.

```
f = open("네이버평점url.txt", 'r')
final_url = f.readlines()
movie_title={1:"겟아웃",2:"부산행",3:"서치",4:"해피데스데이",5:"곡성"}

for i,url in enumerate(final_url):
    resp = requests.get(final_url[i])
    html = BeautifulSoup(resp.content, 'html.parser')
    print("₩n"+movie_title[i+1])
    final_result=[]
    for j in range(1,101):
        url = final_url[i] + '&page=' + str(j)
        print('url: "' + url + '" is parsing....')
        final_result.extend(get_data(url,resp,html))
    data = pd.DataFrame(final_result)
    data.columns = ['nickname', 'contents', 'datetime']
    title = "%s_네이버평점.csv" %(movie_title[i+1])
    data.to_csv(title, encoding = 'cp949')
    print("%s 파일 생성 완료" %(movie_title[i+1]))

    f.close()
```

우선 1번에서 저장한 네이버영화 평점 사이트 url을 모아둔 텍스트 파일을 불러온다. 그리고 각 영화마다 index를 붙인 딕셔너리를 생성한다. url에 하나 하나씩 접근해 각 영화 당 평점 리뷰를 크롤링 할 수 있도록 반복문을 썼다. 영화 평점 사이트 리뷰는 한 페이지에 약 10개의 데이터가 있으므로 100페이지에서 약 1000개의 데이터를 얻고 싶다면 url을 조작하는 과정을 거쳐야한다. 원본 url의 맨 끝에 "&page=(페이지 숫자)" 를 넣으면 되는데, 예를 들어 30페이지에서 데이터를 끌어오기 위해서

"https://movie.naver.com/movie/bi/mi/pointWriteFormList.nhn?code=156083&type=after&page=30" 와 같은 형식으로 변화하면 된다. 이런 방식으로 총 100페이지에 모두 접근하기 위해 반복문을 이용하여 get_data 함수를 실행시켰다. 100페이지의 모든 데이터가 크롤링한 결과를 하나의 리스트에 저장하였고 csv 파일로 저장하기 위해 pandas 라이브러리를 사용했다.

4. 최종적으로 만들어진 "영화제목_네이버평점.csv"파일은 다음과 같다.

	nickname	contents	datetime
0	냥냥이(slam****	크리스가 2층으로 올라갈때 떠들썩하던	2017-05-18 14:32
1	hhl4****	마지막에 경찰차가 오자 자연스럽게 두	2017-05-19 15:49
2	9759(yona****)	와, 초반에 여친이 흑인남친 면허증 안보	2017-05-17 2:05
3	ㅇㅇ(pon8****)	경찰차가 왔을때 모두 다 긴장했을듯 ㅋ	2017-05-17 1:31

nickname 열은 리뷰를 쓴 사람의 닉네임, contents 열은 평점 리뷰 내용, datetime 열은 리뷰를 쓴 날짜이다. 영화당 한 페이지에 존재하는 리뷰 수가 달라 행의 수는 모두 다르지만 약 1000행의 데이터를 수집했다.

III. 데이터 전처리 과정 및 방법

가) 데이터 전처리 및 토큰화

데이터를 분석하기에 앞서 분석에 용이할 수 있도록 전처리 및 토큰화를 진행하였다. 우선 매우 긴 블로그 줄거리, 리뷰 데이터를 형태소 분석하여 필요 없는 형태소는 모두 걸러내는 과정을 진행하였다. 전처리 과정에서 핵심점으로 쓰인 모듈은 konlpy 라이브러리에 존재하는 kkma (꼬꼬마 형태소 분석기)이다. kkma 이외의 mecab, twitter, komoran 등 다양한 종류의 형태소 분석기가 있었지만 분석성능, 실행환경, 속도 등을 고려했을 때 kkma가 프로젝트에 제일 적합하다는 결론을 내렸다. kkma를 이용하여 전처리 및 토큰화 한 과정은 다음과 같다.

1. 우선 불용어 리스트를 불러온다. 불용어 리스트는 아직 우리가 어떤 단어가 불용어인지에 대한 감이 잡히지 않아 https://bab2min.tistory.com/544 이 게시글 작성자가 분류 해 놓은 100개의 불용어

를 사용하였다.

2. 전처리를 진행할 원본 csv파일을 불러온다.

3. 불러온 본문에서 보통 블로그 글에서 제일 많이 쓰이는 'ㅋ', 'ㅠ', 'ㅜ'를 모두 제거하였다.

4. 본문에서 추출할 형태소 태그들을 리스트에 저장한다. 예를 들어 우리가 필요하다고 생각한 형태소 태그들은 'N'으로 시작하는 모든 명사 태그와, 'V'로 시작하는 모든 용언 태그 등이 있다. 이렇게 태그들을 지정하면 조사, 어미, 각종 특수문자, 기타 기호들과 같은 필요 없는 형태소들은 추출되지 않는다. 태그 정보들은 http://kkma.snu.ac.kr/documents/?doc=postag 에서 확인하였다.

5. 원본 데이터를 전체 형태소 분석한 것에서 추출할 태그에 해당하는 단어만을 리스트에 저장한다.

6. 5번에서 나온 리스트에서 불용어에 해당하는 단어들을 모두 지워준다.

7. 3번~6번까지가 전처리 및 토큰화 함수에 해당하는 과정이다. 반복문을 이용하여 원본 데이터의 모든 행에 접근하여 함수를 실행하면 필요 없는 형태소들이 제거된 단어들이 모두 하나의 리스트에 저장된다. 깔끔하게 전처리, 토큰화 된 이 데이터들은 앞으로 tf를 진행할 시 사용된다.

```
import csv
from konlpy.tag import Kkma
kkma = Kkma()

# 불용어 리스트 불러오기, stop_words 리스트에 저장
f = open("불용어 리스트.txt",'r', encoding='utf8')
stop_words = f.read()
stop_words = stop_words.split('₩n')

# 원본 csv파일 불러오기
blog = open('겟아웃_네이버블로그리뷰.csv', 'r')
rblog = csv.reader(blog)
next(rblog) # header 넘어가기

# 전처리 및 토큰화 함수
def preprocessing(line):
    text = line[2]
```

```
        text = text.replace('ㅋ','').replace('ㅠ','').replace('ㅜ','') # 'ㅋ', 'ㅠ', 'ㅜ' 제거
        text_pos = kkma.pos(text)
        text_morphs = kkma.morphs(text)
        words = []
        tags=['NNG','NNP','NNB','NNM','NP','VV','VA','VXV','VXA','VCP','VCN','MDT','MDN','MAG','MAC','XR','UN']
    # 추출할 품사 tag에 해당하는 단어 리스트 만들기
        for word, tag in text_pos:
                if tag in tags:
                        words.append(word)
    # 불용어 지우기
        result = []
        for w in words:
                if w not in stop_words:
                        result.append(w)
        return result

    # csv파일의 각 corpus에 접근하여 전처리
    result = []
    for line in rblog:
        result.append(preprocessing(line)) # result 리스트에 결과 저장
```

나) Term Frequency

Term frequency는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값을 나낸다. 값을 구하기 위한 방법으로 불린 빈도, 로그 스케일 빈도, 증가 빈도 이렇게 3가지가 존재하는데 그 중에서 로그 스케일 빈도를 이용하여 계산했다. 그 이유는 로그를 취함으로써 빈도의 차이를 줄어들기 때문이다. 로그 스케일 빈도를 구하는 공식은 [공식1] 같다. 이 공식에서 f_i는 term i의 빈도를 의미하며 f_i가 1인 경우 로그 값이 0이 되므로 1을 더했다.

원래 문서 별로 빈도를 구하는 방법이 일반화된 방법이지만 이번 프로젝트에서는 블로그 리뷰 별로 term frequency를 구한 것이 아니라 크게 영화 별로 전처리에 의해 선정한 형태소에 대한 빈도수를 구했다.

$$tf_i = \begin{cases} 1+\log f_i & \text{if } f_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

[공식 1]

구해진 tf 중에서 영화 '겟아웃'은 3.58, '곡성'은 3.81, '부산행'은 3.58, '서치'는 3.58, '**해피데스데이**'는 3.32 보다 높은 빈도수를 가지는 term을 이용하여 앞으로의 프로젝트를 진행하고자 한다.

```
movie_title={1:"겟아웃",2:"부산행",3:"서치",4:"해피데스데이",5:"곡성"}
for i in range(len(movie_title)):
    rtitle = "%s_최종본.csv" %(movie_title[i+1])
    wtitle = "%s_tf.csv" %(movie_title[i+1])
    blog = open(rtitle, 'r')
    rblog = csv.reader(blog)
    next(rblog) # header 넘어가기

    # csv파일의 각 corpus에 접근하여 전처리
    total=[]
    for line in rblog:
        total.extend(preprocessing(line[2])) # 영화 리뷰에 포함되어 있는 전체 단어

    # tf
    tf=[]
    countWord = Counter(total)
    for key in countWord:
        tf.append([key, round(1+math.log(countWord[key],2),2)])

    # tf csv파일 작성
```

```
tfcsv = open(wtitle, 'w', newline='')
wr = csv.writer(tfcsv)
wr.writerow(["word","tf"])
for i in range(len(tf)):
    wr.writerow(tf[i])
```

최종적으로 "영화제목_tf"는 [tf]와 같이 저장된다.

또한, 기준에 따라 상위tf를 구하여 생성된 파일 "영화제목_상위tf"는 [상위tf]와 같다.

	A	B
1	word	tf
2	부산행	8
3	관람	4.46
4	일자	1
5	극장	3
6	전주	1
7	고사	1

[tf]

	A	B
1	word	tf
2	영화	9.4
3	좀비	8.81
4	석우	8.06
5	열차	8.06
6	부산행	8
7	공유	7.85
8	수안	7.78

[상위 tf]

IV. 알고리즘 설명 및 선택 배경

가) Apriori 알고리즘 (연관 규칙 분석 알고리즘)

첫번째로 사용된 알고리즘은 연관 규칙 분석 알고리즘(Apriori algorithm) 다른 이름으로 장바구니 분석으로 알려진 알고리즘이다. 이 알고리즘은 어떤 두 아이템 집합이 자주 발생하는가를 알려주는 일련의 규칙들을 생성하는 알고리즘이다. 예를 들어, 소비자들의 구매이력 데이터를 토대로 "X 아이템을 구매하는 고객들은 Y 아이템 역시 구매할 가능성이 높다"는 식의 결론을 내는 알고리즘이다. 이 알고리즘을 이용하여 쇼핑을 할 때 어떤 상품을 고르면 그 상품을 구매한 사람들이 선택한 다른 상품을 제안해주는 **컨텐츠 기반 추천**(contents-based recommendation)의 기본이 되는 방법론이다. 또한, 기저귀를 사는 사람들이 맥주를 같이 구매하는 경우가 크다는 것을 알아낼 때에도 이 알고리즘이 사용되었다.

연관 규칙 분석을 하기 위해서는 우선 아이템 집합을 만들어야 한다. 아이템 집합에는 조건절과 결과

절을 구성하는 각각의 아이템이 포함되어야 한다. 조건절(Antecedent)은 '만일 ~라면'에 해당하는 부분이고, 결과절(Consequent)은 조건에 따른 결과를 나타낸다. 예를 들어, "기저귀를 사면 맥주도 같이 산다."인 경우에 아이템 집합은 기저귀와 맥주이다. 이와 같이 여러 개의 아이템 집합을 [아이템 집합]과 같이 만들어 볼 수 있다. 그 다음에 만들어진 집합의 모든 아이템을 [조합]과 같이 서로 짝을 지어서 짝이 지어진 아이템끼리의 관계가 있는가에 대한 확률을 구한다. 이 과정 중에 Minimum support threshold를 이용하여 자주 반복되는 패턴을 구하게 된다.

ID	Item
A	'M', 'O', 'N', 'K', 'E', 'Y'
B	'D', 'O', 'N', 'K', 'E', 'Y'
C	'T', 'U', 'K', 'E', 'Y'
D	'L', 'I', 'K', 'E', 'Y'
E	'M', 'O', 'N', 'E', 'Y'
F	'T', 'U', 'N', 'E'

[아이템 집합]

pattern	support
'M'	
'O'	
'N'	
⋮	
'M','O'	
⋮	
'M','T','U'	
⋮	
'T','K','E','Y'	
⋮	
'M','O','N','K','E','Y','T' • • •	

[조합]

이때, 아이템끼리에 관계가 있는지 알려주는데 사용되는 3 가지의 확률이 사용되는데 지지도(support)와 신뢰도(confidence), 향상도(lift)가 해당된다. 자주 반복되는 패턴을 구하기 위해서는 '지지도(support)'를 사용한다. '지지도(support)'는 특정 패턴이 발생하는 확률로 정의된다.

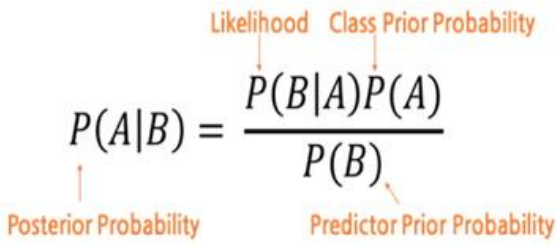
Apriori 알고리즘은 이 알고리즘이 나오기 이전(1994년 이전)의 알고리즘보다 패턴을 찾는데 걸리는 시간을 효율적으로 줄일 수 있다. 하지만, DB를 읽는 횟수가 많다는 것이 단점이다. 이 단점을 보안하기 위해 등장한 알고리즘으로 FD-growth알고리즘이 있다.

이 알고리즘을 선택한 이유는 전처리 후의 형태소 간의 연관성을 이용하여 일정확률보다 큰 경우의 패턴들을 모아서 다음 알고리즘인 나이브 베이즈 정리에 spo인지 nonspo인지를 정하는 기준으로 사용하고자 Apriori 알고리즘을 이용하였다. Apriori 알고리즘은 아이템 간의 인과관계를 구하는 것까지 실행 가능하지만 이번 프로젝트에서는 빈번히 등장하는 패턴을 구하기 위해 사용했다.

나) Naïve Bayes 알고리즘

두번째로는 네이버 review를 spo와 nonspo로 분류하기 위해 나이브 베이즈를 사용하였다. 나이브 베이즈는 분류문제에 베이즈 이론을 적용한 단순한 방법이고 가장 일반적인 방법이다. 실제로 나이브 베이즈는 스팸 이메일 필터링과, 컴퓨터 네트워크에서 침입행위 탐지, 일련의 관찰된 증상에 대한 의학적 질병 등에 자주 사용된다. 우리가 이번에 프로젝트 주제로 삼은 스포 분류 역시 스팸 분류와 매우 비슷하기 때문에, 스포 분류가 이 알고리즘에 유용하게 적용될 수 있을 것 같은 생각이 들었다. 베이즈안은 '사건에 대한 우도(likelihood)는 복수시행에서 즉시 이용할 수 있는 증거를 기반으로 해서 추정해야 한다'는 아이디어에 기반하였다.

베이즈 정리를 통한 조건부 확률 계산법



나이브 베이즈 알고리즘은 기존 데이터들을 가지고 예측변수에 대한 교차표를 작성하여 사전 확률을 구한 뒤, 새로운 데이터가 주어졌을 때 각 특징 값들에 대한 우도(likelihood)와 사전 확률(prior probability)을 가지고 사후확률(posterior probability)을 분류한다. 가장 중요한 점은 데이터 셋의 모든 특징은 동등하게 중요하고 독립적이라고 여긴다는 점이다. 나이브 베이즈에는 라플라스 추정량이 등장하는데, 라플라스 추정량이란, 각클래스에 대한 발생 확률이 0이 되지 않게 보장하기위해 더하는 수로, 보통 추정량을 1로 설정하여 데이터의 각 클래스 특징 조합이 최소 한번은 나타나게 보장하는 것이다.

나이브 베이즈의 장점은 간단하고 빠르고 매우 효율적이며, 잡음과 누락 데이터를 잘 처리하며, 훈련에는 상대적으로 적은 예시가 필요하지만. 대용량의 예시에도 매우 잘 작동된다는 장점이 있다. 그러나 단점은 모든 특징이 동등하게 중요하고 독립적이라는 가정이 잘못된 경우가 자주 있고, 수치 특징이 많은 데이터셋에는 이상적이지 않으며, 추정된 확률이 예측된 클래스보다 덜 신뢰할 만하다는 단점이 있다.

V. 분석 프로그램 설명 및 캡처 화면

가) 전처리 실시

나) tf를 이용하여 연관성 분석에 사용할 itemset 생성

다) 줄거리 리뷰 속 문장마다 포함되어 있는 영화별 상위tf를 가지는 단어의 분포만을 이용하여 연관성 분석

1. Kkma 분석기의 오류로 인해 품사가 잘못 분류된 경우가 존재하기 때문에 2차 전처리를 실시

코드는 [2차 전처리]에서 확인 할 수 있다.

2. 형태소 분석과 전처리를 통해 생성된 result 속 원소들을 2차 전처리의 결과물인 newTF와 비교하여 newTF속 형태소와 같은 형태소가 존재하는 문장만을 선별

코드는 [비교]에서 확인할 수 있다.

3. 연관성 분석 실시

선별된 문장 속 형태소를 apriori 알고리즘에서 item, 선별된 문장을 itemset으로 생각한다. 이때, mlxtend 라이브러리 속 apriori함수를 이용한다. 단, 분석을 실시하여 얻은 결과 중 item set들의 type는 frozenset이다. 다양한 item들간의 관계 중에 이번 프로젝트에서는 1개의 패턴을 이루는 item의 개수가 3개인 패턴을 사용한다.

Apriori함수에는 1개의 패턴이 item을 최대한 몇 개까지 가지고 있을 수 있는지에 대한 매개변수를 작성할 수 있으나 특정한 숫자에 대한 경우는 따로 식을 사용하여 구해야 한다.

4. 최종적으로 "영화 제목_연관성.csv"와 같은 파일이 다음과 같다

	support	itemsets
2536	0.02843	frozenset({'흑인', '남자', '친구'})
2974	0.02843	frozenset({'로즈', '크리스', '흑인'})
3214	0.024722	frozenset({'백인', '흑인', '친구'})
2960	0.023486	frozenset({'로즈', '크리스', '친구'})
3487	0.023486	frozenset({'크리스', '흑인', '친구'})

5. 추후 진행을 위한 데이터 처리

연관성 분석의 결과를 이용하여 Naïve Bayesian algorithm을 사용하므로 데이터의 접근과 계산이 편리하게 만들기 위해 apriori함수로 인해 나오는 결과 값의 type을 list로 바꿔준다.

6. 3개의 item으로 이루어진 패턴을 모두 사용하는 것이 아니라 support값이 일정한 수 이상인 경우에 스포를 나타내는 패턴으로 채택한다.

겟아웃과 서치의 최소 지지도는 0.007, 해피데스데이의 최소 지지도는 0.011, 부산행의 최소 지지도는 0.004, 곡성의 최소 지지도는 0.005으로 정한다.

영화	코드
겟아웃	frequentThree=frequentThree[frequentThree["support"]>=0.007]
부산행	frequentThree=frequentThree[frequentThree["support"]>=0.004]
서치	frequentThree=frequentThree[frequentThree["support"]>=0.007]
해피데스데이	frequentThree=frequentThree[frequentThree["support"]>=0.011]
곡성	frequentThree=frequentThree[frequentThree["support"]>=0.005]

```
import csv
from konlpy.tag import Kkma
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
kkma = Kkma()

movie_title={1:"겟아웃",2:"부산행",3:"서치",4:"해피데스데이",5:"곡성"}
for i in range(len(movie_title)):
    rtitle = "%s_최종본.csv" %(movie_title[i+1])
    uptitle= "%s_상위tf.csv" %(movie_title[i+1])
   # 겟아웃, 부산행, 서치 3.58이상, 곡성 3.81이상, 해피데스데이 3.32이상
```

```
    wtitle = "%s_연관성.csv" %(movie_title[i+1])

    csv_file = open(rtitle, 'r')
    doc = csv.reader(csv_file)
    next(doc) # header 넘어가기

# 문장별로 분리
def split_sentence(doc):
        text = doc[2]
        sentence=kkma.sentences(text)
        return sentence
# 문장별로 저장
result = []
for line in doc:
        result.extend(split_sentence(line))

# 상위tf의 2차 전처리
f=open(uptitle,'r')
rf=csv.reader(f)
next(rf)
TF=[]
for row in rf:
        TF.append(row[0])

    stop=['이', '을', '스', '여','노', '은', '아', '터', '드', '마', '자', '고', '레','거','개방','관람','씬','내용','결국','그런데', '거','늘', '의', '라', '누', '어', '건','진','자', '마','로','스','까','은','시','드','곽','왜','그리하','약간','갑자기', '마','저','거','라','까','분','늘','자','의','어','스','여','을','몇','몇몇','걸','시','드','단','서','인','블록버스터', '늘','메','파','자','김','티','리','바','스','건','대','새','어','보','시','디','르','을','고','예고편','스포일러','장르',  '어쩌','듯하','
```

경','어느','결말','이때','재미있','그리하','이번','조금', '스','자','누','터','라','뜨','다','막','분','임','위','하다','리','사','유','면','감독','스포','줄거리','영화']

```
newTF=[]
for word in TF:
    if word not in stop:
        newTF.append(word)


# 문장별로 나눈 result를 형태소 별로 분석
def split_morphs(doc):
    doc_morphs = kkma.morphs(doc)
    return doc_morphs


# 각 문장별로 어떤 상위 tf를 가지는 단어를 가지고 있는지 확인
dataset2=[]
for i in range(len(result)):
    a=split_morphs(result[i])
    dataset= []
    for w in a:
        if w in newTF:
            dataset.append(w)
    dataset2.append(dataset)

# newTF속 단어와 일치하는 단어가 존재하지 않은 문장 제거
dataset3=[]
for k in range(len(dataset2)):
```

```
            if dataset2[k] != []:
                dataset3.append(dataset2[k])


    # 연관성 분석 실시
    te = TransactionEncoder()
    te_ary = te.fit(dataset3).transform(dataset3)
    df = pd.DataFrame(te_ary, columns=te.columns_)  # 데이터프레임으로 변경

    frequent_itemsets3 = apriori(df, min_support=0.003, max_len=3, use_colnames=True)
    frequent_itemsets2 = apriori(df, min_support=0.003, max_len=2, use_colnames=True)
    frequentThree=frequent_itemsets3[len(frequent_itemsets2):len(frequent_itemsets3)]

    # 모든 결과 값을 확인하기 위해 저장(필수 아님)
    frequentThree.to_csv(wtitle, encoding = 'cp949')
    print(wtitle,"저장 완료")

frequentThree=frequentThree[frequentThree["support"]>=0.007] # 겟아웃, 서치
frequentThree=frequentThree[frequentThree["support"]>=0.004] # 부산행
frequentThree=frequentThree[frequentThree["support"]>=0.011] # 해피데스데이
frequentThree=frequentThree[frequentThree["support"]>=0.005] # 곡성

# frozenset를 list로 변화
sets=[]
for line in frequentThree['itemsets']:
    sets.append(line)
association=[list(x) for x in sets]
```

다) Multinomial Naive Bayes 에 적용

연관성 분석과정에서 한 영화의 블로그 리뷰를 문장당 토큰화 및 형태소 분석을 진행하여 document 당 word들을 생성을 하였고, 연관성 있는 단어가 함께 있는 문장을 spo, 그렇지않은 문장은 모두 nonspo로 분류 하였다. 즉 연관성 분석을 통해 training dataset을 얻은 것이다. 이 경우에는 사전확률을 쉽게 구할 수도 있고 데이터도 크기 때문에, 'MAP'을 사용하였다. 네이버 영화평에서 영화당 1000개의 영화평을 가져와 영화평도 토큰화 및 형태소 분석을 통해 주요 단어를 분리하였다.

- 공식 대입 설명

(1) Prior probabilities:

p(spo)=p(spo 블로그 문장 개수)/p(전체 블로그 문장 개수)

p(nonspo)=p(nonspo 블로그 문장 개수)/p(전체 블로그 문장 개수)

(2) Likelihoods:

$$P(k_i|C_k) = \frac{\sum tf(k_i, d \in C_k) + \alpha}{\sum N_{d \in C_k} + \alpha V}$$

<multinomial naive bayes의 likelihood 공식>

where:

ki : 네이버 영화 평에 있는 하나의 단어

∑tf(ki, d∈Ck): 각 spo class ,nonspo class각각에 있는 모든 document에서의 word ki 의 TF의 총합

∑N(d∈Ck): spo class 및 nonspo class 각각의 TF의 합

α: an additive smoothing parameter(여기서도 laplace smoothing α=1 적용)

V: spo와 nonspo class의 전체 단어 수 (단, 중복된 단어 없음)

*likelihoods 적용 예시 (영화 겟아웃 중 中)

review: '로드같은 친구 한명 있으면 인생 성공한거다.'

P(로드|spo),p(친구|spo), p(인생|spo), p(성공|spo), p(거|spo)

P(로드|nonspo),p(친구|nonspo), p(인생|nonspo), p(성공|nonspo), p(거|nonspo)계산

(3) posterior probabilities:

P(spo|review)= P(로드|spo)*p(친구|spo)*p(인생|spo)*p(성공|spo)*p(거|spo)* P(spo)

P(nonspo|review)= P(로드|nonspo)*p(친구|nonspo)*p(인생|nonspo)
p(성공|nonspo) p(거|nonspo)* p(nonspo)

-> 결론: P(spo|review) 과 P(nonspo|review)중 큰 값을 review에 해당 클래스로 분류

```
#블로그 리뷰를 문장당 spo, nonspo 리스트에 추가(이중리스트)
spo = []
nonspo = []
for row in sentence_morphs:
    s1 = set(row)
    for dataset in association:
        s2 = set(dataset)
        isSpo = False
        if s2.issubset(s1) == True:
            spo.append(row)
            isSpo = True
            break
    if not isSpo:
        nonspo.append(row)
# 영화 리뷰 불러오고 리스트에 저장
```

```
csv_file = open(ctitle, 'r')
doc = csv.reader(csv_file)
next(doc) # header 넘어가기

review_token = []
totalreview = []
for line in doc:
    text = line[2]
    totalreview.append(text) # totalreview 리스트에 결과 저장
index = 0
for comment in totalreview[0:1000]:
    review=preprocessing(comment) # 형태소 분석
    index+=1

    #_________prior probabilities
    prior_spo=len(spo)/(len(spo)+len(nonspo))
    prior_nonspo=len(nonspo)/(len(spo)+len(nonspo))

    #________training data
    #spo의 중복단어 리스트 (the sum of term frequency for class SPO)
    spo_overlap=[]
    for i in range(len(spo)):
        spo_overlap.extend(spo[i])

    #nonspo의 중복단어 리스트 (the sum of term frequency for class NONSPO)
    nonspo_overlap=[]
```

```
for i in range(len(nonspo)):
    nonspo_overlap.extend(nonspo[i])

#중복 없는 전체단어 갯수: Vocabulary
all_spo=[]
for i in range(len(spo)):
    for j in range(len(spo[i])):
        if(spo[i][j] not in all_spo):
            all_spo.append(spo[i][j])

all_nonspo=[]
for i in range(len(nonspo)):
    for j in range(len(nonspo[i])):
        if(nonspo[i][j] not in all_nonspo):
            all_nonspo.append(nonspo[i][j])

overall=[]
overall.extend(all_spo)
overall.extend(all_nonspo)

nonover=[]
for i in range(len(overall)):
    if(overall[i] not in nonover):
        nonover.append(overall[i])
all_count=len(nonover)
```

```
#_____likelihood

#spo의  likelihood

spo_like_result=1

for i in range(len(review)):

    count=1# Laplace smoothing

    for j in range(len(spo_overlap)):

        if (review[i]==spo_overlap[j]):

            count=count+1

    spo_like_result=spo_like_result*(count/(len(spo_overlap)+all_count))

    #단어별 spo likelihoods


#nonspo의  likelihood

nonspo_like_result=1

for i in range(len(review)):

    ncount=1# Laplace smoothing

    for j in range(len(nonspo_overlap)):

        if (review[i]==nonspo_overlap[j]):

            ncount=ncount+1

    nonspo_like_result=nonspo_like_result*(ncount/(len(nonspo_overlap)+all_count))

                                    #단어별 nonspo likelihoods


#_____posterior probabilities

#P(SPO | REVIEW)=p(REVIEW|SPO)P(SPO)

post_spo=spo_like_result*prior_spo


#P(NONSPO | REVIEW)=P(REVIEW|NONSPO)P(NONSPO)
```

```
    post_nonspo=nonspo_like_result*prior_nonspo
    if(post_spo>post_nonspo):
        print(index,"번째 평점리뷰:",comment)
        print("분류결과: 스포 O")
        print("다음 review는 spo:{} >nonspo:{}이므로 스포이다.".format(post_spo,post_nonspo))
        print("==========================================================
==================================")
    elif(post_spo<post_nonspo):
        print(index,"번째 평점리뷰:",comment)
        print("분류결과: 스포 X")
        print("다음 review는 spo:{} <nonspo:{}이므로 스포가 아니다.".format(post_spo,post_nonspo))
        print("==========================================================
==================================")
```

연관성 분석을 통해 나누어진 class를 spo와 nonspo list에 이중 리스트로 저장한다. 네이버 영화평도 이중리스트로 저장한다. 그 뒤 3개의 vocabulary 리스트(∑(d∈spo), ∑(d∈nonspo),V)를 생성하고, likelihood 식을 계산하는 코드를 작성하고, 최종적으로 post probabilities를 계산하는 순차적 단계의 코드를 작성

결과)

```
1 번째 평점리뷰: 관람객관객에게 미끼를 던진 영화
분류결과: 스포 X
다음 review는 spo:5.782005490905095e-17 <nonspo:1.646324406603334e-140이므로 스포가
아니다.
==========================================================================
```

⋮

```
1000 번째 평점리뷰: 1개도 아까운 쓰레기 영화
분류결과: 스포 X
다음 review는 spo:1.1431519800749018e-11 <nonspo:2.4306875186948343e-100이므로 스포
가 아니다.
==========================================================================
```

이러한 형태로 1000개의 영화평에 대한 class 분류 결과가 print된다.

VI. 결과 분석 및 작성한 알고리즘 또는 프로그램의 장단점 및 한계점 설명

가) 결과

1. 겟아웃

```
3 번째 평점리뷰: 와, 초반에 여친이 흑인남친 면허증 안보여 주려고 했던거 설계였네 경찰한테 꼬리 잡힐까봐,, 백인경찰이 편견으로 흑인 검문 때리는 줄 알았는데 지리네 설계
분류결과: 스포 0
이 review는 (spo:1.682202060949524e-61) > (nonspo: 1.3356409665272398e-63)이므로 스포이다.
==========================================================================================

------------------------------------------------------------------------------------------
242 번째 평점리뷰: 뇌를 지배당한 육체의 길잃은 눈빛 소름
분류결과: 스포 0
이 review는 (spo:6.62408156636864e-23) > (nonspo: 2.1744843258459737e-24)이므로 스포이다.
==========================================================================================


364 번째 평점리뷰: 최면이라는 설정만 기억하고 뇌수술하는 장면까지 보여줬는데도. 하녀가 차를 막고 하인이 쫓아오는데도 뇌수술한 할아버지. 할머니라는 건 완전 반전;; 최고의 심리스릴러 영화입니다.
분류결과: 스포 0
이 review는 (spo:4.1371292822908135e-67) > (nonspo: 1.993455271760045e-69)이므로 스포이다.
==========================================================================================


129 번째 평점리뷰: 곱씹을수록 더 소름끼치는 영화진짜 이런게 스릴러지..
분류결과: 스포 X
이 review는 (spo:6.953333555649048e-29) < (nonspo: 2.635713682202765e-26)이므로 스포가 아니다.
==========================================================================================
130 번째 평점리뷰: 저는 스포를 당하고 영화를 봤는데도 괜찮았어요. 스토리 자체가 참 기괴하고 공포스럽고 신선해요. 보고난 후 한장면씩 회상하며 풀이하는 재미가 있네요ㅋㅋ
분류결과: 스포 X
이 review는 (spo:4.9066770187010156e-57) < (nonspo: 1.4414703390045337e-52)이므로 스포가 아니다.
==========================================================================================
131 번째 평점리뷰: 방금 보고 왔는데 말이필요없음요~ 필관람 닥추!!!!
분류결과: 스포 X
이 review는 (spo:4.035579201656432e-24) < (nonspo: 2.2839851436832175e-22)이므로 스포가 아니다.
==========================================================================================
```

2. 부산행

```
123 번째 평점리뷰: 권혁수 닮은 할매가 이 영화 최고 악역이다. 감염자 무리들 뚫고 안전구역에 들어왔으면 격리조치 취하는건 당연한거지 지 언니 못들어왔다고 문 열어 재껴서 무고한 생존자들 죽인 살인자지 그게 ㅋㅋ 그걸 사이다니 뭐니 하고있으니 어휴
분류결과: 스포 0
이 review는 (spo:2.0852401289164298e-117) > (nonspo: 6.476220952099603e-120)이므로 스포이다.
==========================================================================================


127 번째 평점리뷰: ㅋ최악 흔한 여자아이 눈물로 감성팔이하고 부성애 모성애 연인들의 사랑 등등 아주 신파의 절정을 보여주네 공유뿐 아니라 배우들 연기는 극혐 몰입이 안됨.. 좀비들이  끌려가는씬에서는 고무보트에 타고 있는거 다보임  엉성한 cg....
분류결과: 스포 0
이 review는 (spo:3.096771927311355e-106) > (nonspo: 1.076397257151438e-108)이므로 스포이다.
==========================================================================================
```

151 번째 평점리뷰: 관람객마동석형님때문에 재미는있으나 애기가 답답하고 짜증남 평소에는 문도 잘 닫더니 문 안닫아서 물리고 물리고 떠 물리고 어쩔땐 깨지는 문 어쩔땐 안깨지는 문
분류결과: 스포 0
이 review는 (spo:1.414502998520607 2e-74) > (nonspo: 1.739718309875799e-76)이므로 스포이다.
==========

287 번째 평점리뷰: 스토리 너무 질질 끌고 할머니도 들어올수 있는데 그 열차칸까지 잘 오다가 문 앞에서 가만히 서있어서 죽는것도 어이없고조연 주연급들은 아름다운 좀비 변신으로 몇분 동안 버티고일반 좀비들은 바로바로 변하는 것도 어이없음
분류결과: 스포 0
이 review는 (spo:1.3909570574915567e-89) > (nonspo: 2.17258190979129 38e-91)이므로 스포이다.
==========

310 번째 평점리뷰: 난 제일 어이없던게.. 주연들이 하나 같이 바보같다라는거다.마동석 좀비될때도 머리앞에 손을 갖다주지않나..소희 좀비 될때도 좀비가 올걸 알면서 왜 뒷문은 안닫았나 싶고.공유 좀비될때도 왜 굳이 입에다가 손을 대주나 싶었다.
분류결과: 스포 0
이 review는 (spo:5.259894618705425e-79) > (nonspo: 5.1740642283248406e-80)이므로 스포이다.
==========

348 번째 평점리뷰: 역대 최악의 영화. 감동을 주려는건지 웃길려는건지 구분이 안갑니다.공유 죽을때 / 애기가 노래부르면서 터널 지나올땔 ㅋㅋㅋ역대최악의 영화. 여배우들/부산행에휴,
분류결과: 스포 0
이 review는 (spo:6.8393305494229 46e-74) > (nonspo: 6.77172220258481e-77)이므로 스포이다.
==========

3. 서치

185 번째 평점리뷰: 관람객아빠ㅠㅠㅠㅠ 아빠가 넘 잘 찾아
분류결과: 스포 0
다음 review는 spo:7.326745981304657e-16 >nonspo:7.216361383717455e-16이므로 스포이다.
==========

228 번째 평점리뷰: 방금 2천 얼마주고 티비로 다운받아봤는데 ㄹㅇ... 1만원도 아깝지 않을 작품이였음 나는 애가 없어졌는데 컴퓨터로만 어케찾나 했더닣ㅋㅋㅋ 인터넷 뉴스 생중계..ㄷㄷ 무엇보다 좋았던건 마지막 아버지가 타자칠때 보이는 (...) 말풍선과 대사가 좋았다.
분류결과: 스포 0
다음 review는 spo:5.311645220337958e-93 >nonspo:1.6690337355499677e-93이므로 스포이다.
==========

368 번째 평점리뷰: 내가 실종됐는데 아빠가 sns 찾는게 더 공포라는 댓글이 정확하다.
분류결과: 스포 0
다음 review는 spo:1.927701297288 4887e-29 >nonspo:1.4550966413783733e-29이므로 스포이다.
==========

9 번째 평점리뷰: 관람객아버님 정보검색대회 금메달감
분류결과: 스포 X
다음 review는 spo:9.898290935102103e-24 <nonspo:1.0733985500140807e-22이므로 스포가 아니다.
==========
10 번째 평점리뷰: 관람객입소문을 타야만 하는 영화. 끝까지 긴장을 놓치못하는 오래만에 재밌게 본.. 모두들 봐줬으면
분류결과: 스포 X
다음 review는 spo:3.22985642275649e-47 <nonspo:1.6960291133925172e-44이므로 스포가 아니다.
==========

11 번째 평점리뷰: XP로 시작해서 MAC으로 끝나는 영화
분류결과: 스포 X
다음 review는 spo:2.0004090674788853e-10 <nonspo:6.913816755043378e-09이므로 스포가 아니다.
==========

4. 해피데스데이

13 번째 평점리뷰: 리셋을 알리는 아침의 알람음이 가장 큰 소름일 듯.
분류결과: 스포 0
다음 review는 spo:2.2217883198586047e-27 >nonspo:2.090268698394561 4e-280이므로 스포이다.
==========

135 번째 평점리뷰: 마지막에 카터 화요일아닌척 연기할때 소름이었다.
분류결과: 스포 0
다음 review는 spo:1.5030851403935123e-20 >nonspo:1.3165682812960 05e-200이므로 스포이다.
==========

559 번째 평점리뷰: 관람객꾸르잼 생일이다가오면 컵케잌은먹지말아야지
분류결과: 스포 0
다음 review는 spo:8.9615770165154 58e-24 >nonspo:1.62539293987161 13e-240이므로 스포이다.
==========

23 번째 평점리뷰: 관람객되게신선했고 꼭 우리한테. 하루하루가 의미있는하루보내라는신호인거같았습니다. 너무재밌었어요
분류결과: 스포 X
다음 review는 spo:8.377551197005689e-39 <nonspo:7.903343074516401e-380이므로 스포가 아니다.
==========
24 번째 평점리뷰: 관람객왜 공포영화 아니라고 했는지 보면 안다 ㅋㅋ 좀 무섭기는 했는데 재미가 더 큼
분류결과: 스포 X
다음 review는 spo:6.631257972324319e-20 <nonspo:7.14422612162464 4e-170이므로 스포가 아니다.
==========
25 번째 평점리뷰: 영화장난없네 이거 흥하겠다 잘만들었음
분류결과: 스포 X
다음 review는 spo:3.7699002926498266e-14 <nonspo:1.5225001200209812e-130이므로 스포가 아니다.
==========

5. 곡성

496 번째 평점리뷰: 아니 나홍진 멍청아. 이것에 대답이나하라고. 일광이 살을 날리때 누구한테 날린거야? 이게 효진이한테 날릴수도있는거라 본다고? 관객이? 어떤 근거로?? 일광이 일본 놈과 맞서 싸우다가 마지막에도 곽도원집까지 왔었는데 그게 곽도원을 죽이기 위했던거였나??
분류결과: 스포 0
다음 review는 spo:6.026604839415157e-112 >nonspo:2.106389122067747 8e-1120이므로 스포이다.
==========

574 번째 평점리뷰: 경찰 2명이 살인혐의 일본인의 집에 가서 마을사람들을 죽인 확실한 증거를 발견하고도 일본인을 붙잡아 오지도 않고, 며칠뒤 일본인이 증거물을 다 없앤 후에 찾아가서는 '너 이 마을 얼른 떠나라'? 그게 사람 여럿 죽인 살인용의자한테 경찰이 할 소리냐??
분류결과: 스포 0
다음 review는 spo:5.213350832492199e-86 >nonspo:1.900700892471435 2e-890이므로 스포이다.
==========

18 번째 평점리뷰: 이것저것 짬뽕시켜서 이리저리 꼬다가 에라모르겟다 돌디로 되라!
분류결과: 스포 X
다음 review는 spo:3.506447471302225e-20 <nonspo:5.359934517603540 5e-200이므로 스포가 아니다.
==========

20 번째 평점리뷰: 절때 보지미세요 후회합니다.내용을 미스터리로 최고조 까지 끌어올려놓고 아무것도 설명안해주고 반전만 선사하고 엔딩크레딧이 올라가네요;; 진심 자극적이게만 만들아놓고 명작소리 듣네
분류결과: 스포 X
다음 review는 spo:6.744760276140778e-80 <nonspo:7.793670938365402e-760이므로 스포가 아니다.
==========
21 번째 평점리뷰: 내가진짜 처음 평점쓰는데.진짜 좀비물이냐 귀신영화냐ㅎㅎㅎ 철학적으로 풀지마라.ㅋㅋㅋ 이딴 영화
분류결과: 스포 X
다음 review는 spo:3.198771578072421e-47 <nonspo:1.3380414269571508e-420이므로 스포가 아니다.
==========

495 번째 평점리뷰: 끝까지 "진실이 뭘까" 긴장하며 보게 하다가 끝에가서는 "이영화를 만든 진짜이유가 뭐지?"라는 의문을 들게 하네요. 그리고 지금까지 본 영화중에 가장 뒤끝이 안좋은... 찝찝한 영화였습니다. 추천은 하지 않습니다...배우들의 열연으로 6점.
분류결과: 스포 X
다음 review는 spo:1.910727537965259e-64 <nonspo:2.8335359445918614e-570이므로 스포가 아니다.
==========

나) 프로그램의 장단점

이 프로그램은 사람들이 스포성 댓글을 마음대로 선택해서 볼 수 있다는 장점이 있다. 대부분의 사람들은 스포성 댓글을 원하지 않기 때문에 원하지 않는 사람은 스포성 댓글을 따로 보지 않아도 되며, 또한 사람들이 결말을 모르는 경우 영화를 보러 갈 때 기대감을 높일 수 있게 된다. 요즘 영화에 대한 스포적 글에 예민한 사람들이 많기 때문에, 이 프로그램을 네이버 영화평 이외에도 네이버 기사나 사람들이 쉽게 노출될 만한 곳에 프로그램을 사용하면 사람들에게 유익한 도움이 될 수 있을 것이다. 또한 네이버 블로그 이외에도, 실시간 글들로 빠르게 데이터들을 수집하여 스포와 관련된 training data를 빠르게 수집할 수도 있다. 이러한 training data를 통해 다른 사용자가 리뷰를 입력하는 순간 빠른 속도로 분류하여 스포인 경우에는 숨김 기능이 실행되어 스포인 댓글과 스포가 아닌 댓글을 구별할 수 있게 된다.

단점은 이 프로그램이 구현되기 위해서는 어느 정도의 데이터가 필요하기 때문에 오래 걸린다는 단점이 있다. 물론 실시간 글이나 적은 블로그 평들로 training data를 만들 수는 있지만, 이 data는 정확성이 떨어지게 된다. 즉 정확성과 빠른 생성을 둘 다 잡기는 힘들다는 단점이 있다. 따라서 개봉한지 얼마 되지 않은 영화들은 정확하게 스포성 글들을 분류하기가 힘들다.

다) 한계점

1. kkma 형태소 분석기의 잘못된 분석

'을', '이', '은', '의'와 같이 조사로 분류되어야 하는 형태소들이 명사로 분류되는 경우가 존재한다. 사람의 이름을 성과 이름으로 따로 작성할 경우 따로따로 분류하여 형태소 분석을 실시한다.

2. 확률의 기준을 정하는데 확실하지 않은 기준

효율적인 결과를 얻기 위해 선택한 확률의 기준이 사람마다 다른 경우가 존재한다. 그로 인해 스포의 기준이 되는 기준이 달라진다. 또한 단어 선택의 과정에서 스포가 아닌 단어들이 선택되어 스포라고 분류가 되는 경우가 있다.

3. 같은 뜻의 단어들이 무수히 많이 존재

한국어에서는 같은 뜻의 다른 단어들이 많이 존재한다. 예를 들어, 동일하게 '죽다'의 의미를 나타내지만 '사망하다.', '세상을 떠나다.',' 목숨을 잃다.' 등이 있다. 이렇게 '죽다'라는 뜻은 같지만 사용되는 단어의 tf가 적어 중요하다고 채택되지 못할 수도 있다.

4. 나이브 베이즈를 통해 최종으로 스포인지 아닌지 여부가 100% 정확하지가 않음

근소한 소수점 차이로 spo와 nonspo가 분류되는 경우도 있다.

5. 띄어쓰기와 맞춤법이 맞지 않고 오타가 존재하는 원본 데이터가 많음