

중간 보고서

데이터 마이닝과 분석을 통한 웹소셜 종합 인포 어플

Vol.6



제출일	2020. 04. 14	전공	컴퓨터공학과
과목	졸업작품 프로젝트	학번	2015722084
			2015722083
			20172020672
담당교수	이기훈	이름	한승주
			김성종
			조예슬

목 차

I 개요

1. 배경 및 필요성

2. 목적

ㄱ. 정보의 신뢰성

ㄴ. 접근의 용이성

3. 설계 내용

ㄱ. Flow Chart

ㄴ. 개념 설계

II 과제 수행

1. 수행 일정

2. 웹크롤링

3. DB 구축

4. 단어 사용 빈도 추출

5. UI 제작

III 과제 평가

1. 개선방안

2. 기대효과

ㄱ. 기업적 측면

ㄴ. 사용자 측면

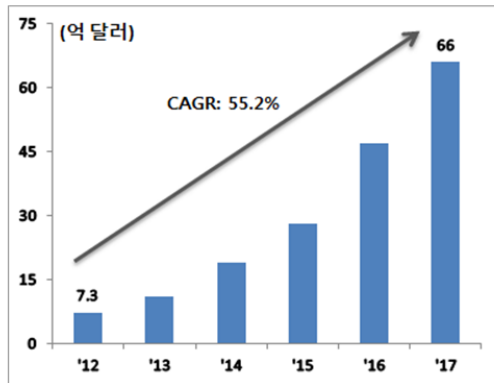
< 참고문헌 >

I. 개요

1. 배경 및 필요성

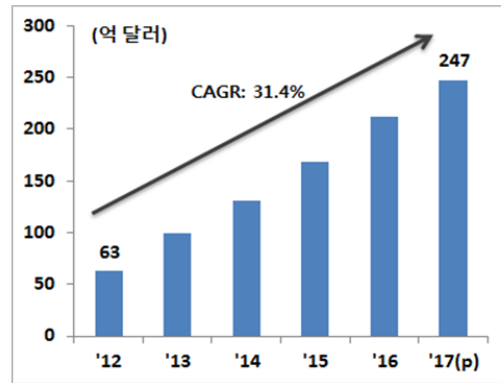
- 현대에 이르며 미디어 시장에 많은 변화가 이뤄졌다. 미디어의 다양화뿐만 아니라 미디어 플랫폼에도 변화가 이루어지며 현대인들이 더 쉽고 빠르게 그리고 편리하게 미디어에 접근할 수 있도록 환경이 만들어졌다. 음악에 있어 CD 앨범과 같은 아날로그식의 접근 방법이 음원 다운로드와 같은 디지털화를 넘어 스트리밍 플랫폼으로 진화하였고 영화 TV와 같은 영상 매체들은 TV, 영화관과 같은 제한적 접근이 “넷플릭스”와 같은 플랫폼이 생성되며 통합 스트리밍 플랫폼의 패러다임이 열리게 되었다고 할 수 있다.

< 전 세계 음악 스트리밍시장 규모 >



자료 : 국제음반산업협회(IFPI).

< 전 세계 OTT 서비스시장 규모 >



자료 : PwC(2017), ITU(2017), 정보통신진흥원(2018) 재인용.

그림 1-1 (출처 : 현대경제연구원)¹

- 출판업계 또한 마찬가지다. 책, 신문과 같은 인쇄물은 어느새 디지털화되어 신문은 웹으로 책은 이북과 오디오북 등 다양한 형태로 변화되어 출판업 시장이 많은 다양화를 이루어내고 있다.
- 그중 우리가 오늘 주목할 것은 웹소설시장이다. 웹소설의 전신은 과거 인터넷 소설과 그 전 PC 통신에서 퍼지던 소설부터 시작되었다고 할 수 있다. 스마트폰의 보급이 활성화되고 인터넷에 대한 접근이 쉬워지며 모바일 환경에서 쉽게 볼 수 있는 웹소설이라는 새로운 시장이 설립되었다. 기존의 인터넷 소설을 연재하던 사이트부터 카카오, 네이버 등 덩치가 큰 기업들이 뛰어들며 시장은 더욱 커지고 있다.

¹ [1] 콘텐츠 스트리밍 산업의 성장동력화가 시급하다. (현대경제연구원, 2019.02)

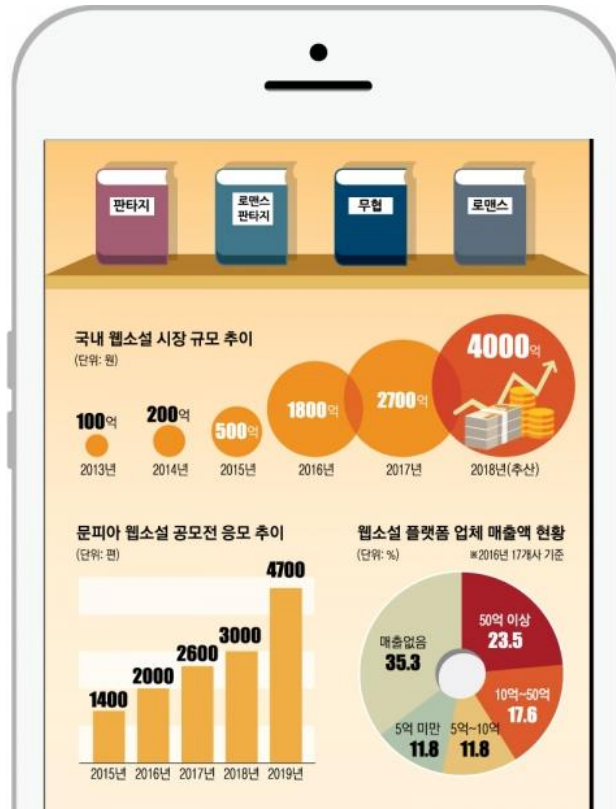


그림 1-2 (출처 : 서울신문, 2019.05)²

- 웹소설의 시장이 커질 수 있게 된 것에 원 소스 멀티 유즈, 즉 OSMU가 큰 역할을 한 것처럼 보인다. 2018년 한국 콘텐츠 진흥원의 조사에 따르면 국내 웹소설 시장은 2013년 100억 원 규모에서 2018년 4000억 원까지 40배 성장했다. 이러한 배경에는 웹소설을 바탕으로 제작된 드라마의 성공이 있다. 또한, 주변국인 일본은 남녀노소 상관없이 서브컬처 문화를 수용하고 이를 즐기고 있어 이러한 콘텐츠 시장이 이미 발달한 상태이고, 중국도 연간 2조 원의 시장 규모를 가지고 있을 만큼 웹툰 및 웹소설 시장이 큰 편이다. 최근에는 중국, 일본 외에도 세계 각국의 작품들을 수입 및 수출하고 있고 새로이 제작되는 웹툰이나 드라마, 영화 심지어는 게임까지도 웹소설을 기반으로 한 것이 굉장히 많다.
- 이렇게 커진 웹소설 시장을 방증하듯 독점 연재도 하며 인기도 많은 플랫폼이 약 6곳, 그 외에도 여러 작품의 연재 서비스를 제공하는 곳까지 합하면 그 수는 10곳을 훌쩍 넘는다. 이렇게 커진 플랫폼들은 저마다 신인 작가를 육성하기 위한 공모전을 열기도 하고 독점 연재작 계약도 진행하며 독자 유입을 위한 노력을 하고 있다.

² [2] '하루 5분' SNS 하트 쓰윽~ 4000억 시장 펼친 웹소설 (서울신문, 2019.05)

2. 목적

ㄱ. 너무 많아진 작품과 떨어지는 작품의 질

- 독자들은 문제점에 부딪히기 시작한다. 너무 많아진 작가, 작품, 플랫폼에 독자는 작품 선택에 대해 고려해야 할 사항이 많아졌다. 또한 많아진 웹소설의 인기와 트렌드에 따른 양산화에 따라 수적으로 선택지는 많아졌지만 작품의 질이 받쳐주지 않아 읽고 실망을 남겨지는 경우도 더러 있다. 그렇기 때문에 소비자는 이를 소비하는 방식과 작품에 대한 평가 방식도 다양해지고 있다. 플랫폼 내부의 리뷰와 별점을 여전히 이용하기도 하는가 하면 댓글 서비스와 별점을 이용해 불만을 터트리기도 하고 작품에 대한 얘기가 플랫폼 뿐 아닌 기타 소셜미디어와 커뮤니티, 블로그와 같은 곳에 퍼지기도 한다. 이렇게 퍼지는 리뷰에 따라 그 영향은 곧바로 작품에 적용이 되었다. 예를 들어 웹툰 [외모지상주의]는 초반 9점대 이상의 좋은 평으로 인기를 얻은 작품이다. 시간이 지날수록 줄거리에 지루함을 느낀 독자는 곧바로 댓글과 별점에 이를 표하며 현재는 7점대까지 평가가 떨어지며 작품에 타격을 주고 있다.





















	92화 [반익편] 체육대회 [02]	★★★★★ 9.93	2016.08.18		282화 호스텔 [마무리]	★★★★★ 8.55	2020.04.09
	91화 [반익편] 체육대회 [01]	★★★★★ 9.88	2016.08.11		281화 호스텔 [12]	★★★★★ 7.06	2020.04.02
	90화 불법 도도 [마무리]	★★★★★ 9.89	2016.08.04		280화 호스텔 [11]	★★★★★ 7.28	2020.03.26
	89화 불법 도도 [07]	★★★★★ 9.87	2016.07.28		279화 호스텔 [10]	★★★★★ 6.15	2020.03.19
	88화 불법 도도 [06]	★★★★★ 9.87	2016.07.21		278화 호스텔 [09]	★★★★★ 8.27	2020.03.12
	87화 불법 도도 [05]	★★★★★ 9.89	2016.07.14		277화 호스텔 [08]	★★★★★ 8.96	2020.03.05
	86화 불법 도도 [04]	★★★★★ 9.89	2016.07.07		276화 호스텔 [07]	★★★★★ 8.77	2020.02.27
	85화 불법 도도 [03]	★★★★★ 9.88	2016.06.30		275화 호스텔 [06]	★★★★★ 9.53	2020.02.20
	84화 불법 도도 [02]	★★★★★ 9.85	2016.06.23		274화 호스텔 [05]	★★★★★ 8.45	2020.02.13
	83화 불법 도도 [01]	★★★★★ 9.82	2016.06.16		273화 호스텔 [04]	★★★★★ 5.63	2020.02.06

그림 2-1 (출처 : 네이버 만화 외모지상주의)

ㄴ. 리뷰 플랫폼과 사용자의 다양화

- 리뷰를 남기는 방식도 각 플랫폼의 특성마다 다른 특성을 보이기 때문에 특정 작품에 대한 리뷰를 단 한 곳만 보고 작품에 대한 평가를 내리는 것은 불가능해졌다. 이런 이유 때문에 리뷰를 통합적으로 볼 수 있는 플랫폼에 대한 연구를 시작했다.
- 미디어를 접하는 플랫폼뿐만 아니라 소셜미디어나 커뮤니티에도 리뷰를 많이 남기는데 이를 이용하는 나이대나 성별 등이 각 소셜미디어나 커뮤니티에 따라 다르다 보니 리뷰를 남기는 방식에도 차이가 있고 이를 한 번에 확인할 수 있는 곳에 대한 필요성을 생각

하게 되었다.

(*실제 국내 페이스북 유저 비율은 남성이 여성 대비 14% 많고, 인스타그램은 여성이 남성 대비 4% 많은 비율을 가지고 있다. 또한 페이스북은 연령대가 고른 반면, 인스타그램은 20~30대 비율이 상대적으로 높다.)

- 우리는 각 플랫폼과 소셜미디어, 커뮤니티에 나와있는 리뷰를 모아 이를 분석하여 작품에 대한 트렌드를 분석해보고 사용자들에게 작품 판별에 대한 지표를 제공하고 더 나아가 현재 많은 미디어의 소스가 되고 있는 웹소설의 트렌드를 파악하여 앞으로 엔터테인먼트 시장의 트렌드를 먼저 파악해보려 한다.

3. 설계내용

ㄱ. Flow Chart

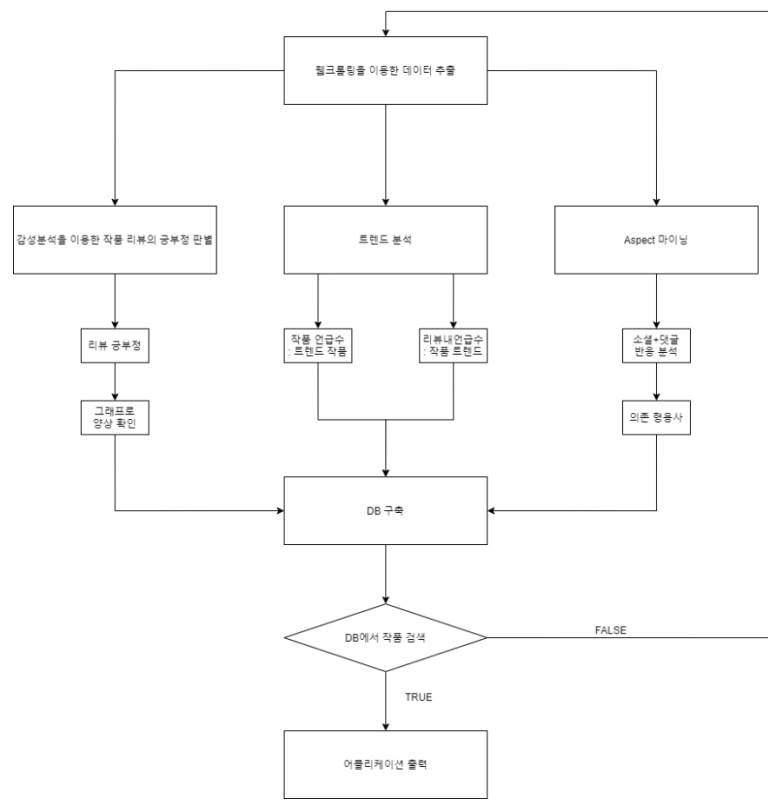


그림 3-1

L. 개념 설계

1. 플랫폼의 웹사이트 코드 분석 및 크롤링(Beautiful soup)

- 플랫폼 선정

플랫폼	추출 카테고리1	추출 카테고리2	비고
조아라	프리미어	완결작품	자유로운 연재 가능 정식 연재작 선정기준 필요
문피아	유료웹소설		자유로운 연재 가능 정식 연재작 선정기준 필요
카카오페이지			리뷰 크롤링 불가
리디북스			
네이버 시리즈			화수별 리뷰가 전체 리뷰에 포함
네이버 웹소설			네이버 단독 연재작

표 3-1

- 소셜 미디어 및 커뮤니티 선정 : 작품 애기가 많은 커뮤니티 중심 선정

플랫폼	비고
네이버 블로그	리뷰 중심
티스토리	리뷰 중심
트위터	독백형 추천작으로 언급이 多
인스타그램	해시태그 이용한 검색만 가능 리뷰 중심
디씨인사이드	독백, 리뷰, 추천 多
인스티즈	추천, 독백, 리뷰 多

표 3-2

- DB 를 구축하기 위해 해당 웹사이트 접속 (*리뷰 사이트, 블로그, 트위터, 인스타그램)
- 작품 기본 정보와 리뷰가 담긴 웹사이트 코드 분석
- BS4를 이용해 페이지 데이터 호출
- 작품의 기본 정보 tag 를 찾아 추출

- 각 사이트와 페이지별로 링크를 재귀적으로 검색하여 데이터를 추출

2. Kkma, Hannanum 을 이용한 KoNLP(키워드 생성)

- Kkma 나 Hannanum 모듈을 이용하여, 해당 모듈에 맞추어 입력된 문자열에서 키워드로 표현할 품사 추출

- 가장 빈도수가 높은 단어(키워드로 설정할 단어)를 DB 에 저장

- 오피니언 마이닝(감성 분석)을 이용하여 리뷰의 긍부정을 판단하고 전체적인 긍부정에 대한 평가를 진행한다. 긍부정 평가에 대한 점수를 기간에 대한 그래프로 표현하여 시간이 지남에 따라 작품의 평가 변화 양상을 확인한다.

- 이 과정에서 마이닝의 정확도를 높이는 작업이 필요하다. 이 부분은 현재 나와있는 다양한 모델 기반 접근법을 이용한 감성사전을 이용하여 데이터의 정확도를 끌어올릴 계획이다. 각 감성 사전마다 어떤 단어나 조사를 제거하고 비속어, 신조어, 오타의 처리를 어떻게 하나에 따라 정확도가 달라진다. 현 소셜미디어의 특성상 표준어를 사용하지 않고 발음을 있는 그대로 적어 두거나 비속어, 신조어, 약어 혹은 "이 장면 정말 사이다였다."와 같이 원래 가지고 있는 역할과 다르게 빗대어 많이 사용되고 있는 언어들이 있으므로 이 부분에서 더 효과적인 판별법과 사전이 어떤 것인지 정확도를 높이는 데 중점을 두고자 한다. 표준어만을 이용한 감성 분석 API 및 툴들은 많이 존재하지만 비표준어는 경우의 수가 굉장히 다양하기 때문에 이를 처리하는 API 나 사전 구축에 대한 조사는 더 필요하다.

- 참고 분석 예정 : word2vec, sentiwordnet, www.openhanquel.com, KTS

- 음소 단위 분할 조합을 통해 뜻을 파악하는 Trigram-Signature 를 사용하면 오타가 있는 "조ㅎ아", "실ㅎ어"와 같은 문맥을 파악하는 데 유용할 것이라고 생각한다. 이를 이용한 API 가 있는지 혹은 이와 관련된 사전을 제작하여 사용하는지 더 조사할 계획이다.

- word2vec 을 이용하여 Aspect 마이닝 분석을 한다. 이때 주인공, 작품, 분위기, 스토리 등 작품의 다양한 요소에 대해 의존적인 연결 형용사나 동사를 뽑으면서 작품 내 트렌드를 파악한다.

- matlab 을 이용한 데이터 분석을 통해 소셜미디어와 커뮤니티에서 작품의 언급수를 파악해보고 작품의 화제성을 판단한다.

- 위와 같은 방법으로 작품 내 명사들의 언급수를 통해 어떤 키워드가 화제가 되고 있는지 분석해본다.

3. 크롤링된 정보를 이용한 DB 구축(MySql or sqlite)

- MySQL 서버에 접속하여 데이터베이스 생성
- cursor 를 추출하여 execute 메서드로 SQL 을 실행, 테이블 생성
- Execute 메서드에 데이터를 계속 확장

4. Android UI 제작

- 자신이 보고 싶은 작품의 리뷰를 보기 위한 제품의 검색창을 사용자가 보기 편하도록 UI 로 구현.
- 검색 결과로 작품의 기본 정보와 함께 리뷰를 보여준다. 리뷰의 경우 긍정 평가에 대한 그래프를 제공하여 시간이 지남에 따라 작품 평가 양상을 확인 가능하도록 하며 또한 작품 내 트렌드는 어떤 것인지 보여준다. 메인화면에는 현재 많이 화제 되고 있는 작품 Top 10을 선정하여 보여준다.

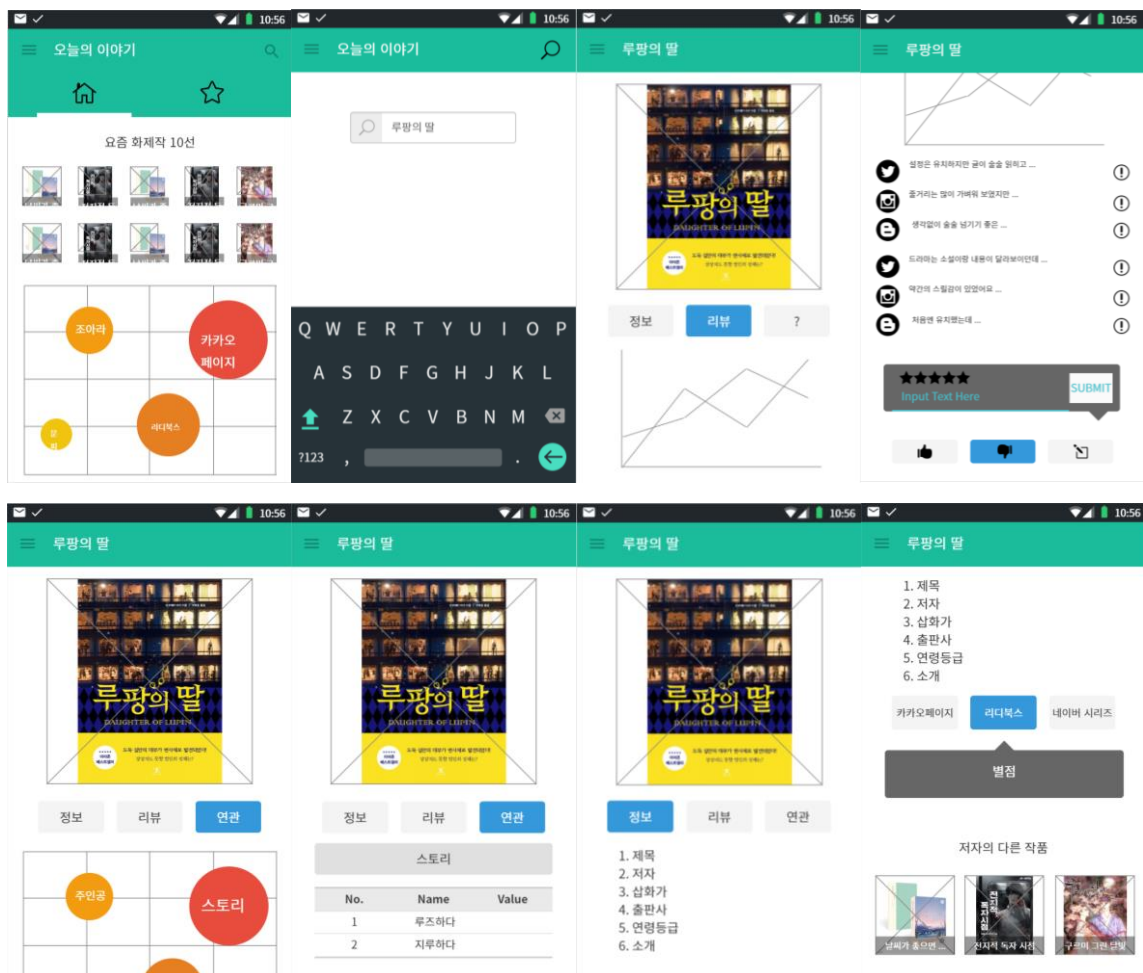


그림 3-2 예시 결과

II. 과제 수행

1. 수행 일정

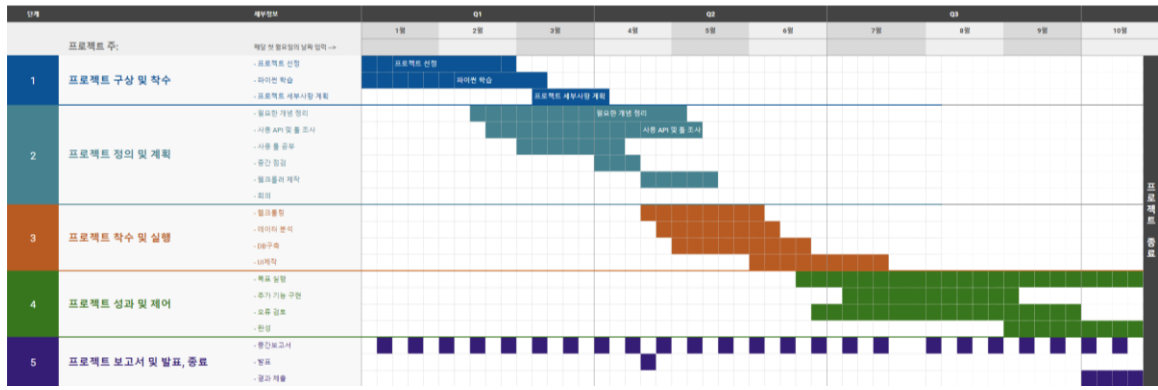


그림 1-3

2. 웹 크롤링

ㄱ. 트위터

- 트위터의 특징은 불특정 다수가 독백을 하는 경향이 있다.
- 불특정 다수의 독백을 통해 작품의 언급수는 자연스레 현재 작품의 화제성을 대변해줄 수 있다. 이를 통해 트렌드 작품을 파악하기 좋을 것이다.
- Aspect 마이닝에서도 작품에 관해 연관 지어 나오는 것들을 파악하기 좋은 소셜미디어라고 생각한다.
- 트위터 크롤링을 위한 API 는 여러 가지가 존재한다. 트위터의 정식 API 는 키 신청을 하면 이용이 가능하지만 최근 7일간의 트윗만 확인이 가능하다. 이외에도 현재 많이 사용되고 있는 사설 API GetOldTweets3³, twitterscraper⁴ 등 여러 가지가 있는데 이를 이용하여 이전 정보를

³ <https://github.com/Jefferson-Henrique/GetOldTweets-python>

⁴ <https://github.com/taspinar/twitterscraper>

가져오기 위해서는 사실 API 를 이후 계속되는 최신 정보를 가져오기 위해서는 Tweepy 를 사용할 계획이다.

- 연습해본 API 는 getoldtweet3이며, 해당 API 는 기간을 설정하여 트윗을 수집할 수 있다. 기간은 2019년 1년 동안 '로지텍'이라는 검색어가 들어간 트윗을 수집해보았다.

```
In [6]: try:
import GetOldTweets3 as got
except:
!pip install GetOldTweets3
import GetOldTweets3 as got

In [7]: import datetime

days_range = []

start = datetime.datetime.strptime("2019-01-01", "%Y-%m-%d")
end = datetime.datetime.strptime("2019-12-31", "%Y-%m-%d")
date_generated = [start + datetime.timedelta(days=x) for x in range(0, (end-start).days)]

for date in date_generated:
    days_range.append(date.strftime("%Y-%m-%d"))

print("=== 설정된 트윗 수집 기간은 {} 에서 {} 까지 입니다 ===".format(days_range[0], days_range[-1]))
print("=== 총 {}일 간의 데이터 수집 ===".format(len(days_range)))

=== 설정된 트윗 수집 기간은 2019-01-01 에서 2019-12-30 까지 입니다 ===
=== 총 364일 간의 데이터 수집 ===

In [8]: import time

# 수집 기간 맞추기
start_date = days_range[0]
end_date = (datetime.datetime.strptime(days_range[-1], "%Y-%m-%d")
            + datetime.timedelta(days=1)).strftime("%Y-%m-%d") # setUntil() 끝을 포함하지 않으므로, day + 1

# 트윗 수집 기준 정의
tweetCriteria = got.manager.TweetCriteria().setQuerySearch('로지텍')\
    .setSince(start_date)\
    .setUntil(end_date)\
    .setMaxTweets(-1)

# 수집 with GetOldTweets3
print("Collecting data start.. from {} to {}".format(days_range[0], days_range[-1]))
start_time = time.time()

tweet = got.manager.TweetManager.getTweets(tweetCriteria)

print("Collecting data end.. {0:0.2f} Minutes".format((time.time() - start_time)/60))
print("=== Total num of tweets is {} ===".format(len(tweet)))

Collecting data start.. from 2019-01-01 to 2019-12-30
Collecting data end.. 15.82 Minutes
=== Total num of tweets is 7521 ===
```

그림 2-2

가져온 트윗에서 사용할 정보는 유저, 트윗 개시 날짜, 트윗 링크, 내용이며 해당 내용을 csv 파일로 저장한다.

```
In [10]: from random import uniform
from tqdm import tqdm_notebook

# initialize
tweet_list = []

for index in tqdm_notebook(tweet):

    # 메타데이터 목록
    username = index.username
    link = index.permalink
    content = index.text
    tweet_date = index.date.strftime("%Y-%m-%d")
    tweet_time = index.date.strftime("%H:%M:%S")

    # 결과 합치기
    info_list = [tweet_date, tweet_time, username, content, link]
    tweet_list.append(info_list)

    # 휴식
    time.sleep(uniform(1,2))

HBox(children=(IntProgress(value=0, max=7521), HTML(value='')))
```

```
In [12]: # 파일 저장하기

import pandas as pd

twitter_df = pd.DataFrame(tweet_list,
                           columns = ["date", "time", "user_name", "text", "link"])

# csv 파일 만들기
twitter_df.to_csv("sample_twitter_data_{}_to_{}.csv".format(days_range[0], days_range[-1]), index=False)
print("=== {} tweets are successfully saved ===".format(len(tweet_list)))

=== 7521 tweets are successfully saved ===
```

그림 2-3

In [13]: # 파일 확인하기

```
df_tweet = pd.read_csv('sample_twitter_data_{}_to_{}.csv'.format(days_range[0], days_range[-1]))
df_tweet.head(10) # 위에서 10개만 출력
```

Out[13]:

	date	time	user_name	text	link
0	2019-12-30	18:37:18	s_NEGEV_s	와 근데 로지텍 마우스 G903이 HERO 센서 달고 오니까 배터리가 거의 달지...	https://twitter.com/s_NEGEV_s/status/121171791...
1	2019-12-30	16:29:14	mookibini	로지텍 빠순이라 키보드도 마우스도 헤드셋도 로지텍으로 하고싶었지만 만만...	https://twitter.com/mookibini/status/1211685685...
2	2019-12-30	16:09:43	KeepGoingMasiro	지금 게임용으로 쓰고 있는 키보드.. 사실 엄청 오래 되기는 했는데.. 아직도 잘...	https://twitter.com/KeepGoingMasiro/status/1211...
3	2019-12-30	14:03:32	RyuZU_Seod	음 그리고 키보드는 COX CK450 마우스는 로지텍 G102 정도면.. 80 살팍...	https://twitter.com/RyuZU_Seod/status/12116490...
4	2019-12-30	11:34:39	nabislife	마우스 로지텍 mx 버티컬 쓰고있음	https://twitter.com/nabislife/status/121161155...
5	2019-12-30	10:37:24	lulul_jd	안녕하세요.. 멍청하게 자기 아이디도 모르는 놈.. 팔로 해주셔서 감사해요...	https://twitter.com/lulul_jd/status/1211597147...
6	2019-12-30	09:18:35	sinrisoung	킹직히 유로트릭 최고세팅은 이거아님? 트리플모니터 하는 비용보다 vr저렴하...	https://twitter.com/sinrisoung/status/12115773...
7	2019-12-30	09:12:54	wannabecoolman	로지텍 블루키보드 고장났어.. 미쳤나... 내손에 남아나는게 없다는 뜻이니.....	https://twitter.com/wannabecoolman/status/1211...
8	2019-12-30	09:01:08	kkibaek	로지텍 키즈투고, 이거 사지 마세요 '블루투스 키보드'의 본질을 놓친 키보드	https://twitter.com/kkibaek/status/12115729177...
9	2019-12-30	09:00:22	binu_4_lviz	로지텍 keys to go 키보드 타자 느낌이 좋아서 계속 치고 있음 ㅋㅋㅋㅋ	https://twitter.com/binu_4_lviz/status/1211572...

그림 2-4

그림 2-3을 통해 저장한 파일을 확인하면 그림 2-4와 같은 결과를 얻을 수 있다.

ㄴ. 네이버 블로그 / 티스토리

- 네이버 블로그와 티스토리는 기타 소셜미디어 (트위터, 인스타그램)과 다르게 세세한 리뷰 중심의 글이 돋보인다. 상대적으로 더 정확한 연관어를 찾을 수 있을 것 같다고 예상되기에 Aspect 마이닝에 쓰일 계획이다.

- 1차시: 실제로 많은 사용자들이 많이 보는 리뷰는 네이버일 것이다. 따라서 네이버 블로그의 검색어에 따른 결과 크롤링을 해보면 제목과 블로그 링크를 가져올 수 있다. 무한정으로 많은 양의 검색 결과를 가져올 수 없는데 이는 좋은 검색 결과를 위해 네이버가 1000건의 검색 결과만을 보여주고 있기 때문이다. 하지만 이것만으로도 충분히 키워드를 추출한다던지, 제품 한 개를 분석함에 있어서 부족함이 보이지는 않는다.

```
In [30]: import requests
import pandas as pd
from bs4 import BeautifulSoup
from collections import OrderedDict
from itertools import count

def mycrawler(input_search):
    url='https://search.naver.com/search.naver'
    post_dict=OrderedDict()

    for page in count(1):
        params={
            'query': input_search,
            'where': 'post',
            'start': (page-1)*10+1,
            'date_from': 20191231,
            'date_to': 20200220,
        }
        print(params)
        response = requests.get(url, params=params)
        html=response.text

        soup=BeautifulSoup(html, 'html.parser')

        title_list=soup.select('.sh_blog_title')

        for tag in title_list:
            if tag['href'] in post_dict:
                return post_dict

            print(tag.text, tag['href'])
            post_dict[tag['href']]=tag.text

        return post_dict

In [31]: result=mycrawler('로지텍')
print(len(result))
```




그림 2-5

그림 2-5는 글에서 글의 제목과 링크 주소를 가져온 스크래핑의 결과인데, 해당 링크를 타서 본문을 가져올 수 있지만, 해당 기능은 다음인 인스타그램에서 구현해보고자 한다.

㉔. 인스타그램

- 인스타그램은 트위터와 같은 소셜미디어보다는 작품에 대한 얘기를 많이 하지만 네이버나 티스토리보다는 단순한 리뷰를 중심으로 많이 얘기한다. 이를 토대로 트렌드 지표를 알아보기 위한 언급수에 분석 결과를 쓸 예정이다.

- Vol1. 네이버 블로그가 긴 리뷰, 트위터가 짧은 리뷰라고 하면 인스타그램은 해쉬태그라는 자신만의 키워드를 표현하는 기능이 있지만, 실제로 분석함에 있어서는 해쉬태그를 따로 볼 것이 아니라 전체적인 리뷰로 보고 트위터와 유사한 짧은 리뷰라 판단하여 전체적인 분석을 하고자 한다.

```
def InstagramUrlFromKeyword(browser, keyword, numofpage):
    keyword_url_encode = quote(keyword) # 한글인식

    url = 'https://www.instagram.com/explore/tags/' + keyword_url_encode + '/?hl=ko'

    browser.get(url)

    arr_href = []

    body = browser.find_element_by_tag_name('body')

    for i in range(numofpage):
        body.send_keys(Keys.PAGE_DOWN)

        time.sleep(1)

    time.sleep(3)

    post = browser.find_elements_by_class_name('v1Nh3')

    for j in post:
        href_str = j.find_element_by_css_selector('a').get_attribute('href')

        arr_href.append(href_str) # append 추가시키는거

    return arr_href
```

그림 2-6

인스타그램에서 어떤 키워드로 검색어를 하면 해당 키워드인 한글이나 영어는 컴퓨터가 해석하기 난해하므로, 인코딩을 통해 해당 검색어에 맞추어 주소변환이 가능하다.

```
▼<div class="v1Nh3 kIKUG _bz0w">
  ▼<a href="/p/B1Fh3MvhY1J/"> == $0
    ▼<div class="eLAPa"> https://www.instagram.com/p/B1Fh3MvhY1J/
```

그림 2-7

여러 글의 본문을 찾기 위해서 해당 함수를 사용하게 되는데, 인스타그램 웹의 소스코드에는 v1Nh3 class 밑에 href 를 통해 모든 글의 주소를 가져올 수 있어서 해당 글의 ref 를 가져오기 위한 정의이다.

```
def IdHashTagFromInstagram(browser, url):
    browser.get(url)

    insta_id = ""

    hash_data = ""

    wait = WebDriverWait(browser, 20)

    wait.until(EC.presence_of_element_located((By.CLASS_NAME, "e1e1d"))) # e1e1d는 아이디가 적혀있는 소스코드

    id_href = browser.find_elements_by_class_name('e1e1d')

    insta_id = id_href[0].find_element_by_css_selector('a').text # id_href[0] 첫번째 있는 a 찾기

    wait.until(EC.presence_of_element_located((By.CLASS_NAME, "C4VMK")) # 댓글들

    href = browser.find_elements_by_class_name('C4VMK')

    total_hash_text = []
```

그림 2-8

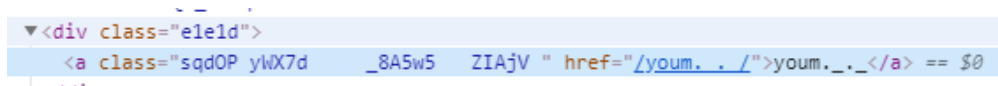


그림 2-9

아이디를 찾는 부분으로, e1e1d 클래스 영역에 href 로 text 만 뽑아내면 해당 부분이 인스타그램 id 이므로 이를 추출한다.

```

for i in range(0, len(href)): # 댓글 가져와서 하나씩 끝까지 보는 거 len 몇개 개수

    hash_text = href[i].find_element_by_css_selector('span').text

    total_hash_text.append(hash_text)

    image_src = ''

try:

    image_temp = browser.find_element_by_class_name('KL48h').find_element_by_css_selector('img') # 이미지 찾기

    image_list = image_temp.get_attribute('srcset') # srcset이란 속성을 가지고 있는 애를 가져와라

    temp = image_list.split(',') # ,로 구분해서 temp로 가져와라

    for i in temp:

        if '1080w' in i: # 사진의 많은 url중에서 1080w있는 문자열 찾기

            image_src = i.split(' ')[0] # url 1080w이 있는 링크에서 1080w를 떼고 공백 앞의 정보를 가져오기

except:

    image_src = '' # 동영상이면(이미지가 아니면) 빈칸으로 뒤라

pass

return insta_id, image_src, total_hash_text

```

그림 2-10

그림 2-10의 코드는 for 문을 이용하여 참조할 reference 가 있는 동안 해당 ref 의 본문을 축적하는 부분과 본문에 덧붙인 이미지의 src 를 찾는 부분으로 구성이 되어있다.

```

▼<span class title="수정됨"> == $0
    "저한테 로지텍 핑크 k380이 있지만"
    <br>
    "늘 화이트색상의 키보드를 가지고 싶다는 생각이 마음 저 구석에 있는데, 드디어 저도 화이트 키보드와 마우스를 하나 더 가지게 되었어요!"
    <br>
    <br>
    "제 책상과도 아주 찰떡인 k580과 m350"
    <br>

```

그림 2-11

본문의 부분으로 댓글 또한 위와 같이 span 의 영역에 글이 작성되어 있다.

```

▼<div class="KL48h" style="padding-bottom: 100%;">
    
    </div>

```


그림 2-12

Img_src 또한 KL4Bh 의 영역에 img 를 find 하여 찾을 수 있다.

```
browser = webdriver.Chrome('C:\chromedriver.exe')

keyword = input("검색어를 입력하세요 : ")

num_of_pages = 2

arr = InstagramUrlFromKeyword(browser, keyword, num_of_pages)

insta_df = pd.DataFrame(columns=['Insta ID', 'Image Src', 'Content'])

for url in arr:
```

그림 2-13

```
try:

    insta_id, image_src, hash_data = IdHashTagFromInstagram(browser, url)

    char = re.compile('[^0-9a-zㄱ-ㅣ-가-힣!#?]*') # 재정비
    """
    정규식을 두 번 이상 사용한다면, 모듈의 match, search 함수는 효율적이지 않다.
    매번 match 혹은 search를 수행할 때마다, 정규식을 분석해서 처리하기 때문이다.
    효과적인 처리 방법은 정규식을 내부 표현식으로 일단 변환하고, 그것을 계속 활용하는 것이다.
    compile 함수가 정규식을 내부 표현식으로 변환하여 정규식 객체를 리턴한다.
    """
    hash_data_str = ""
```

그림 2-14

```
for data in hash_data:
    hash_data_str = hash_data_str + data

hash_data_str = char.sub("", hash_data_str) # ""를 hash_data_str으로 바꿔주기

dic_insta = {"Insta ID": insta_id, "Image Src": image_src, "Content": hash_data_str}

temp_df = pd.DataFrame(dic_insta, index=[0]) # index=0은 dic을 temp로 바꾸는데 예러가 나지 않도록 하는 것

insta_df = insta_df.append(temp_df, ignore_index=True)

except:

    print(sys.exc_info()[0])

    pass

nsta_df.to_csv('insta_temp.csv', mode='w', encoding='euc-kr')
```

그림 2-15

[illegible]

그림 2-16과 같은 결과를 얻을 수 있다. 여기서, 활용한 것은 저번 2주 차 동안의 목표였던 링크를 타서 본문을 가져오는 것과, 검색어를 입력하여 해당 검색어에 맞는 리뷰를 가져오는 것을 사용해보았으며, 추가적으로 csv 파일로 저장하는 것까지 구현해보았다.

2. 디씨인사이드 / 인스티즈

- ### 3. DB 구축

18

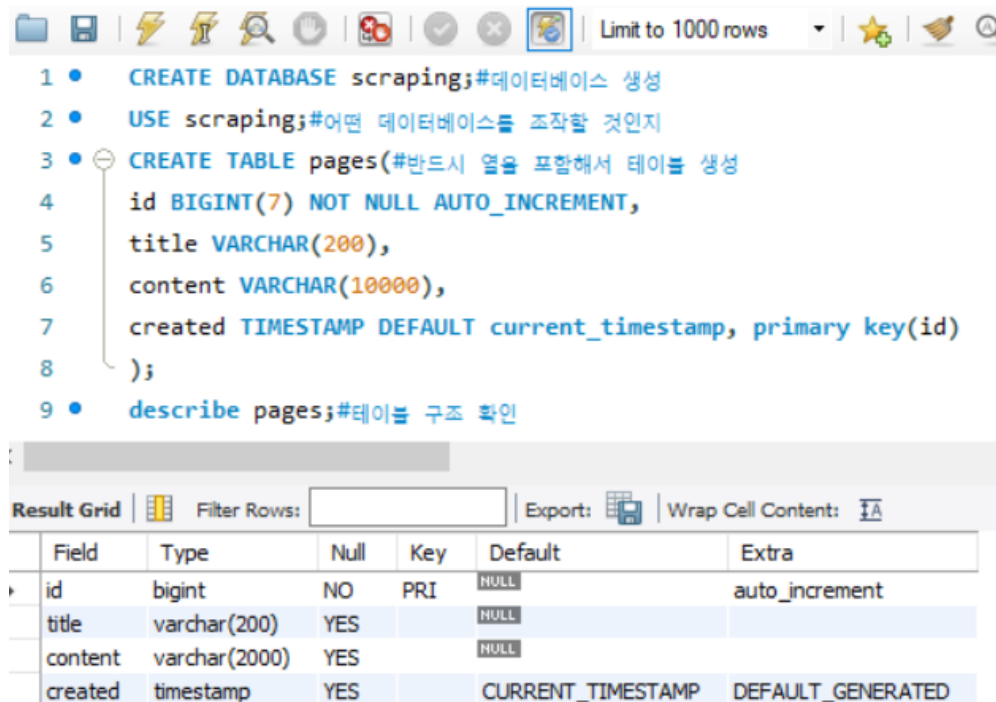


그림 3-3

여러 개의 테이블이 아닌 한 개의 테이블을 이용하여, 제목과 내용을 저장해 보기 위한 테이블을 구성하였습니다.

```

22 #####문자셋을 utf8mb4에서 unicode_ci로 바꿔주는 코드#####
23 • alter database scraping character set=utf8mb4 collate=utf8mb4_unicode_ci;
24 • use scraping;
25
26 • alter table pages convert to character set utf8mb4 collate utf8mb4_unicode_ci;
27 • alter table pages change title title varchar(200) character set utf8mb4 collate utf8mb4_uni
28 • alter table pages change content content varchar(200) character set utf8mb4 collate utf8mb4
29 #####
30 • alter table pages modify column content varchar(2000);

```

그림 3-4

이후, 데이터를 저장하는 과정에서 하나의 열에 내용을 포함시키지 못하는 경우가 많아 alter 기능을 이용하여 데이터 열의 정보를 수정하였습니다.

```

In [1]: pip install PyMySQL

Requirement already satisfied: PyMySQL in c:\users\#naruk\#anaconda3\#lib\site-packages (0.9.3)
Note: you may need to restart the kernel to use updated packages.

In [3]: import pymysql
conn=pymysql.connect(host='127.0.0.1',user='root',passwd='3721',db='mysql')

In [6]: cur=conn.cursor()
cur.execute("USE scraping")
cur.execute("SELECT * from pages WHERE id=2")

Out[6]: 1

In [7]: print(cur.fetchone()) #마지막에 실행한 쿼리 결과 출력
(2, 'A new title', 'Some new content', datetime.datetime(2020, 4, 2, 7, 28, 17))

In [8]: cur.close()
conn.close()

```

그림 3-5

pyMySQL 을 이용하여 cur 과 conn 이라는 변수를 활용하여 execute 문을 통한 MySQL 명령어를 실행해 보았습니다.

ㄷ. 크롤링된 결과를 DML 명령어를 사용하여 저장

```

insert into pages(title,content)#id는 자동증가, timestamp는 현재시간 자동 저장
values(
    "Test page title",
    "This is some test page content. It can be up to 10,000 characters long."
);
select * from pages where id=2;#id가 2인 행이없으므로 none return
select * from pages where title like "%test%";
select id,title from pages where content like "%page content%";
#delete를 실행하기전에는 select를 먼저 실행하는 것이 좋다.;
select * from pages where id=1;
delete from pages where id=1;
update pages set title="A new title", content="Some new content" where id=2;

```

그림 3-6

크롤링한 결과를 DB 에 저장해 보기 앞서서 DML 명령어를 기본적으로 활용해보았습니다.

```

In [5]: from urllib.request import urlopen
        from bs4 import BeautifulSoup
        import datetime
        import random
        import pymysql
        import re

        conn = pymysql.connect(host='127.0.0.1', user='root', passwd='3721', db='mysql', charset='utf8')
        cur = conn.cursor()
        cur.execute('USE scraping')

        random.seed(datetime.datetime.now())

        def store(title, content):
            cur.execute('INSERT INTO pages (title, content) VALUES ("%s", "%s")', (title, content))
            cur.connection.commit()

        def getLinks(articleUrl):
            html = urlopen('http://en.wikipedia.org'+articleUrl)
            bs = BeautifulSoup(html, 'html.parser')
            title = bs.find('h1').get_text()
            content = bs.find('div', {'id': 'mw-content-text'}).find('p').get_text()
            store(title, content)
            return bs.find('div', {'id': 'bodyContent'}).findAll('a', href=re.compile('^(/wiki/)((?!:).)*$'))

        links = getLinks('/wiki/Kevin_Bacon')
        try:
            while len(links) > 0:
                newArticle = links[random.randint(0, len(links)-1)].attrs['href']
                print(newArticle)
                links = getLinks(newArticle)
        finally:
            cur.close()
            conn.close()

/wiki/List_of_gothic_festivals
/wiki/Moldova
/wiki/Taralia_District
/wiki/Romani_people
/wiki/Turkey
/wiki/Karachays
/wiki/Soyot
/wiki/Chechens
/wiki/Berbers_in_Belgium
/wiki/Aghul_people
/wiki/Archil_people
/wiki/Rutul_people
/wiki/Poles_in_Azerbaijan
/wiki/Azerbaijan
/wiki/United_Nations_Development_Program
/wiki/UNICEF
/wiki/Chapter_XIV_of_the_United_Nations_Charter

```

그림 3-7

그림 3-5와 동일하게, conn 과 cur 변수를 활용하여 pyMySQL 의 데이터에 접근합니다. 기본적으로 위키백과의 데이터를 하나 참조하되, 데이터 내부의 href 를 모두 찾아서 ref 의 제목과 내용을 모두 가져와서 DB 에 저장하는 코드를 작성해 보았습니다. 이는, 추후에 네이버 블로그나 티스토리 등에서 작품을 검색하고, 그 내용을 수집하는 과정과 매우 유사할 것이라 판단합니다. 특히, 네이버 블로그 위키백과와 동일하게 request 를 활용합니다.

이때, DML 명령어를 사용하기 위해서 execute 문을 사용하여 insert 문을 실행하고 있음을 확인할 수 있습니다.

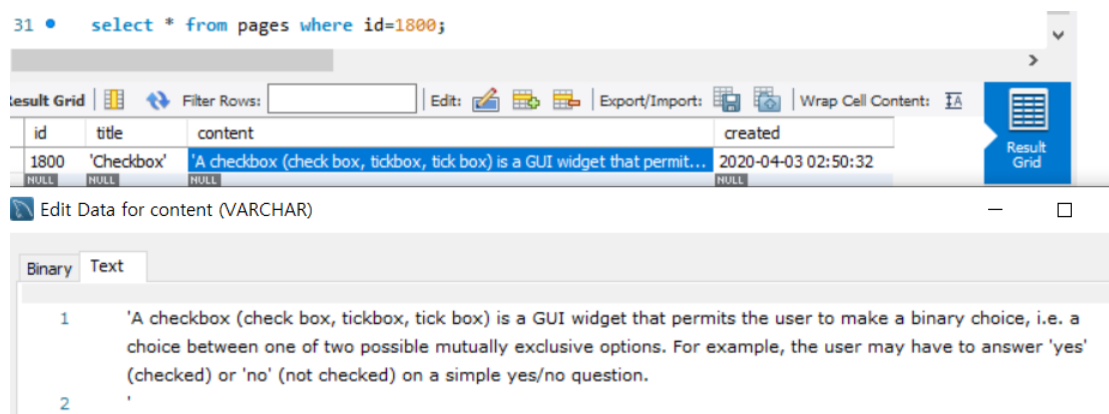


그림 3-8

컴퓨터의 속도에 비해 데이터의 길이가 크다고는 생각을 안 했는데, 생각보다 긴 시간이 소요되는 것을 확인하였습니다. 웹페이지당 1초 정도 소요되긴 하지만 추후에는 더 많은 데이터를 저장할 것이라 판단하므로 빠른 작업이 필요할 것 같습니다.

ㄹ. DB Logical-ERD 구상

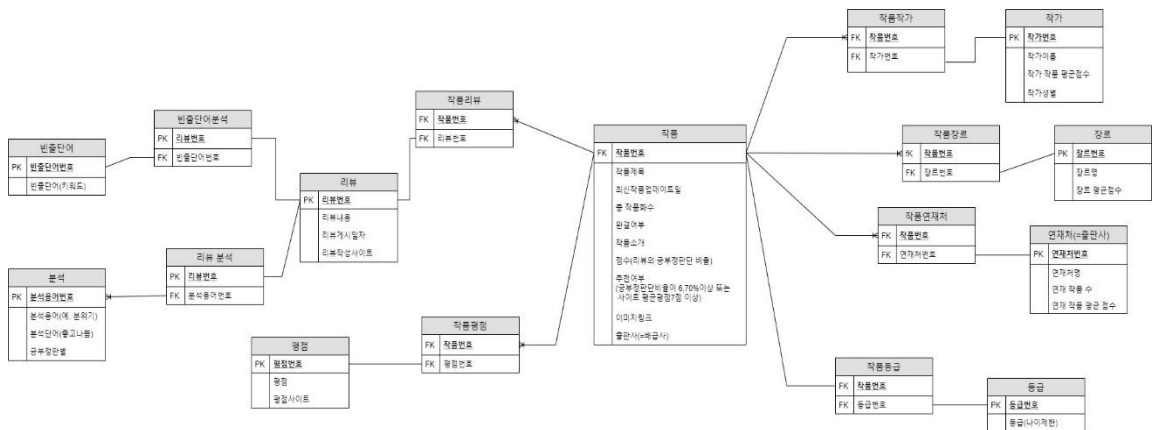


그림 3-9

기본적인 작품 정보로 작품 이미지, 제목, 작가, 출판사, 연재사, 완결 여부, 현재 총 작품 화수, 최신화 업데이트일, 작품 소개, 장르가 들어가게 됩니다. 이 중 작가, 장르와 연재처는 단일 테이블로 만들어 간단한 작가 정보와 연재처 정보를 가지게 됩니다. 이 테이블에는 장르, 작가와 연재처의 다른 작품을 포함한 통합적 평균 점수가 개재되게 됩니다. 가장 중요한 리뷰의 경우 해당 작품에서 화제가 되며 많은 언급이 일어나고 있는 키워드와 리뷰의 긍정적 판단을 위해 두 가지 테이블을 만들었습니다. 또한 각 리뷰 사이트마다 가지고 있는 평점에 대한 평균을 측정하여 종합적인 리뷰 점수를 판단하기 위해 평점에 관련된 테이블도 만들었습니다.

*이는 추후 제작과정에서 변경될 수 있으며 계속 진행되는 회의를 통해 상세 내역을 수정해 나가고 있습니다.

4. 단어 사용 빈도 추출

ㄱ. 전처리 과정(불용어 제거, 어근 동일화, N-gram)

```

In [3]: import nltk
        nltk.download('stopwords')

[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\Naruk\AppData\Roaming\nltk_data...
[nltk_data] Unzipping corpora\stopwords.zip.

Out[3]: True

In [5]: words_Korean=["추석","연휴","민족","대이동","시작","눌어","교통량","교통사고","특히","자동차"]
        stopwords=["가다","눌어","나타","것","특히"]
        [i for i in words_Korean if i not in stopwords]

Out[5]: ['추석', '연휴', '민족', '대이동', '시작', '교통량', '교통사고', '자동차']

In [7]: from nltk.corpus import stopwords
        words_English=["apple","banana","chief","roberts",',','president','you','.']
        print([w for w in words_English if not w in stopwords.words('english')])

['apple', 'banana', 'chief', 'roberts', ',', 'president', '.']

```

그림 4-1

Nltk api 의 stopwords 를 이용하여 불용어를 제거한 결과입니다. 이 외에도 re.compile 과 정규식의 조합을 이용하여 특수문자나 특정 조합(메일)등을 지우는 처리를 해보았습니다.

```

In [1]: ### ### ### ### ### ### ### ### ### ### N-gram ### ### ### ### ### ### ### ### ### ###
        from nltk.stem import PorterStemmer
        from nltk.tokenize import word_tokenize

In [4]: import nltk

In [7]: nltk.download('punkt')

[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\Naruk\AppData\Roaming\nltk_data...
[nltk_data] Unzipping tokenizers\punkt.zip.

Out[7]: True

In [9]: ps_stemmer=PorterStemmer()
        new_text="It is important to be immersed while you are pythoning with python. All pythoners have pythoned poorly at least once."
        words=word_tokenize(new_text)
        for w in words:
            print(ps_stemmer.stem(w),end=" ")

It is import to be immers while you are python with python . all python have python poorli at least onc .

In [10]: from nltk.stem.lancaster import LancasterStemmer
         LS_stemmer=LancasterStemmer()

In [11]: for w in words:
         print(LS_stemmer.stem(w),end=" ")

it is impoort to be immers whil you ar python with python . al python hav python poor at least ont .

In [15]: from nltk.stem.regexp import RegexpStemmer
         RS_stemmer=RegexpStemmer("python")#python이라는 글자제거
         for w in words:
             print(RS_stemmer.stem(w),end=" ")

It is important to be immersed while you are ing with . All ers have ed poorly at least once .

```

그림 4-2

해당 실행문은 영어의 경우 시제에 따라 같은 동사임에도 형태가 달라지는 경우가 있는데, 이때 달라지게 하는 원인을 제거하는 것을 실행해 보았습니다. 한국어에도 이러한 형태가 있는데 형태소 분석 뒤에 가능할 것이라고 판단합니다.

```

In [2]: """ ### ### ### ### ### ### ### ### ### ### N-gram ### ### ### ### ### ### ### ### ### ###
        n-gram이란 n번 연이어 등장하는 단어들의 연쇄를 의미한다.
        ex) Republic of korea 트라이그램, 보통 바이(2)그램까지만 사용한다.
        """

        from nltk import ngrams
        sentence="Chief Justice Roberts, President Clinton, President Obama, President Carter, the citizen of Americans and people of the world"
        grams=ngrams(sentence.split(),2)
        for gram in grams:
            print(gram,end=" ")
        #Chief, President와 같이 두개의 단어가 연이어 생성되는 단어를 바이그램을 통해 추출할 수 있다.

('Chief', 'Justice') ('Justice', 'Roberts,') ('Roberts,', 'President') ('President', 'Clinton,') ('Clinton,', 'President') ('President', 'Obama,') ('Obama,', 'President') ('President', 'Carter,') ('Carter,', 'the') ('the', 'citizen') ('citizen', 'of') ('of', 'Americans') ('Americans', 'and') ('and', 'people') ('people', 'of') ('of', 'the') ('the', 'world,') ('world,', 'thank') ('thank', 'you,')

```

그림 4-3

n-gram 을 이용하여 '대통령 트럼프'와 같이 한 단어로 취급해야 할 필요가 있는 단어를 표기해주는 전처리 과정을 실행해 보았습니다.

ㄴ. KoNLPy 한국어 분석 中 Hannanum

```
In [24]: import pandas as pd
import nltk
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
```

그림 4-4

```
In [30]: from konlpy.tag import Hannanum
hannanum=Hannanum()

In [31]: temp = []
for i in range(len(lines)):
    temp.append(hannanum.nouns(lines[i]))#명사만추출

In [32]: # 중첩 리스트(개념을 알 것) 하나의 리스트로 변환하는 함수
def flatten(l):
    flatList = []
    for elem in l:
        if type(elem) == list:
            for e in elem:
                flatList.append(e)
        else:
            flatList.append(elem)
    return flatList

word_list=flatten(temp)
# 두글자 이상의 단어만 추출
word_list=pd.Series([x for x in word_list if len(x)>1])
word_list.value_counts().head(10)

Out [32]: 대통령    29
국민    19
대한민국    9
우리    8
여러분    7
역사    6
국민들    6
나라    6
대통령의    5
세상    5
dtype: int64
```

그림 4-5

대통령 답화문 중, 명사만을 추출하여 빈도를 추출하고자 한 실행문입니다. 답화문의 경우 문단별 구성에 따라서 리스트가 중첩이 되므로 이를 하나로 만들어줄 필요가 있습니다. 따라서 def 문을 통해 함수를 정의하여 실행하였습니다.

```
In [37]: from wordcloud import WordCloud
from collections import Counter

In [38]: font_path = 'C:\Users\#naruk\Desktop#잡아라! 텍스트마이닝#NanumBarunGothic.ttf'

In [39]: wordcloud = WordCloud(
    font_path = font_path,
    width = 800,
    height = 800,
    background_color="white"
)

In [40]: count = Counter(stopped_tokens2)
wordcloud = wordcloud.generate_from_frequencies(count)

In [41]: def __array__(self):
    """Convert to numpy array.
    Returns
    -----
    image : nd-array size (width, height, 3)
           Word cloud image as numpy matrix.
    """
    return self.to_array()

def to_array(self):
    """Convert to numpy array.
    Returns
    -----
    image : nd-array size (width, height, 3)
           Word cloud image as numpy matrix.
    """
    return np.array(self.to_image())
array = wordcloud.to_array()
```

그림 4-6

```
In [43]: count = Counter(word_list)
wordcloud = wordcloud.generate_from_frequencies(count)
array = wordcloud.to_array()
```

```
In [44]: get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10, 10))
plt.imshow(array, interpolation='bilinear')
plt.show()
fig.savefig('wordcloud.png')
```

그림 4-7

자연어 처리를 한 결과만을 이용하여 UI 를 구성하고자 했는데, 시각적인 처리가 있다면 UI 가 좀 더 다양화할 수 있지 않을까 하여 수행해보았습니다. 이 외에도 matplotlib 를 이용하여 그래프 등의 시각적인 자료를 제작해보았는데, 이러한 시각적인 자료가 트렌드를 파악할 때 사용될 수 있으리라 판단합니다.

ㄴ. 감성분석(감성사전이용)

```
In [1]: #사건기반감성분석
import pandas as pd
import glob
from afinn import Affin
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
import numpy as np
import matplotlib.pyplot as plt

In [2]: pos_review=(glob.glob("C:\Users\naruk\Desktop\잡아라!엑스트마이닝\데이터\imdb\train\pos\*.txt"))[20]
#IMDB 리뷰의 긍정(pos)데이터셋중 20번째 데이터의 경로를 받아옴
f = open(pos_review, 'r')
lines1 = f.readlines()[0]
#해당 문자열을 받아옴
f.close()

In [3]: affinn = Affin()
affinn.score(lines1)#감성점수산출(긍정이나까 하이스코어)

Out[3]: 8.0

In [4]: #부정
neg_review=(glob.glob("C:\Users\naruk\Desktop\잡아라!엑스트마이닝\데이터\imdb\train\neg\*.txt"))[20]
f = open(neg_review, 'r')
lines2 = f.readlines()[0]
f.close()
affinn.score(lines2)

Out[4]: -4.0
```

그림 4-8

영화사이트 IDMB 의 리뷰 자료를 이용하였고, 2500개의 data set 을 가지고 있는 Affin 을 이용하여 불러온 리뷰의 긍부정 척도를 계산하는 것을 실행해 보았습니다.

저희는 한국어 감성사전을 이용하여 이러한 척도를 계산하는 방법도 있지만 한국어를 영어로 번역하여 해당 사전을 활용하는 방안도 고려 중에 있습니다.

```

In [7]: #EmoLex
NRC=pd.read_csv('C:\Users\hmaruk\Desktop\잡아라! 텍스트 마이닝 데이터\nrc.txt', engine="python", header=None, sep="\t")
#감성사전 오픈

NRC=NRC[(NRC != 0).all(1)]
NRC=NRC.reset_index(drop=True)
#감성어와 감성표현이 유의미한 라벨만 추출

tokenizer = RegexpTokenizer('[\w]+')
stop_words = stopwords.words('english')
#불용어 처리

p_stemmer = PorterStemmer()

raw = lines1.lower()
tokens = tokenizer.tokenize(raw)
stopped_tokens = [i for i in tokens if not i in stop_words]

#금정 텍스트 전처리

match_words = [x for x in stopped_tokens if x in list(NRC[0])]
emotion=[]
for i in match_words:
    temp=list(NRC.iloc[np.where(NRC[0] == i)[0],1])
    for j in temp:
        emotion.append(j)
#감성사전과 텍스트의 감성어들의 매핑

sentiment_result1=pd.Series(emotion).value_counts()
#감성표현 출력 함수

sentiment_result1

```

```

Out[7]:
positive      8
trust         7
negative      5
joy           4
anticipation  4
sadness       3
fear          3
anger         2
surprise      2

```

그림 4-9

자연어 처리를 하기 전에 불용어 처리를 하여 단어별 감정을 매핑하여 감정이 몇 번 매핑되었나를 확인하는 실행문입니다.

㉔. 감성분석(지도 기계학습기반 감성 분석)

```

In [2]: import pandas as pd
import glob
from afinn import Afinn
from nltk.corpus import stopwords
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

```

```

In [3]: pos_review=(glob.glob("C:\Users\hmaruk\Desktop\잡아라! 텍스트 마이닝 데이터\aclImdb\train\pos\*.txt"))
# 긍정 리뷰 텍스트 읽어오기
lines_pos=[]
for i in pos_review:
    try:
        f = open(i, 'r')
        temp = f.readlines()[0]
        lines_pos.append(temp)
        f.close()
    except Exception as e:
        continue
len(lines_pos)

```

```

Out[3]: 12490

```

```

In [4]: neg_review=(glob.glob("C:\Users\hmaruk\Desktop\잡아라! 텍스트 마이닝 데이터\aclImdb\train\neg\*.txt"))
lines_neg=[]
for i in neg_review:
    try:
        f = open(i, 'r')
        temp = f.readlines()[0]
        lines_neg.append(temp)
        f.close()
    except Exception as e:
        continue
len(lines_neg)

```

```

Out[4]: 12489

```

```

In [5]: total_text=lines_pos+lines_neg
len(total_text)

```

```

Out[5]: 24979

```

그림 4-10

그림 4-10의 IDMB 데이터셋을 불러오는 과정입니다.

```
In [7]: x = np.array(["pos", "neg"])
class_index = np.repeat(x, [len(lines_pos), len(lines_neg)], axis=0)
#금. 부정 클래스 라벨링
stop_words = stopwords.words('english')

vect = TfidfVectorizer(stop_words=stop_words).fit(total_text)
X_train_vectorized = vect.transform(total_text)
#TF-IDF가중치를 준 후에 문서-단어 매트릭스로 바꾸어줌

In [8]: from sklearn.linear_model import LogisticRegression, SGDClassifier
model = LogisticRegression()
model.fit(X_train_vectorized, class_index)
#로지스틱 회귀모델을 세움

C:\Users\naruk\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)

Out[8]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=None, solver='warn', tol=0.0001, verbose=0,
warm_start=False)

In [17]: pos_review_test = (glob.glob("C:\Users\naruk\Desktop\잡아라! 텍스트 마이닝 데이터\ac\imdb\test\pos*.txt"))[10]

test = []
f = open(pos_review_test, 'r')
test.append(f.readlines()[0])
f.close()

predictions = model.predict(vect.transform(test))
predictions

Out[17]: array(['pos'], dtype='<U3')

In [18]: neg_review_test = (glob.glob("C:\Users\naruk\Desktop\잡아라! 텍스트 마이닝 데이터\ac\imdb\test\neg*.txt"))[20]

test2 = []
f = open(neg_review_test, 'r')
test2.append(f.readlines()[0])
f.close()

predictions = model.predict(vect.transform(test2))
predictions

Out[18]: array(['neg'], dtype='<U3')
```

그림 4-11

여러 모델 중 로지스틱 회귀분석 모델을 사용한 결과로 불러온 리뷰의 감정을 나타낸 결과입니다.

위와 동일하게 서포트벡터머신, 의사결정 나무 모형 등 지도 기계학습 기반의 여러 모델을 동일한 데이터에 대해 수행해보았습니다.

```

In [19]: #의사결정나무모형으로 위와 동일한 실험
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train_vectorized, class_index)
predictions = clf.predict(vect.transform(test))
predictions

Out[19]: array(['neg'], dtype='<U3')

In [20]: predictions = clf.predict(vect.transform(test2))
predictions

Out[20]: array(['neg'], dtype='<U3')

In [15]: #서포트벡터머신-리나오래결합
from sklearn.svm import SVC
clf = SVC(gamma='scale') #이런식으로 변경가능
clf.fit(X_train_vectorized, class_index)
predictions = clf.predict(vect.transform(test))
predictions

C:\Users\#naruk\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Out[15]: array(['pos'], dtype='<U3')

In [16]: predictions = clf.predict(vect.transform(test2))
predictions

Out[16]: array(['pos'], dtype='<U3')

```

그림 4-12

그 결과로 로지스틱 회귀분석의 경우 원하는 대로 긍정, 부정의 결과를 가져왔고 수행 속도도 해당 데이터에 한해서는 다른 두 모델보다 속도도 빨랐습니다. 하지만, 나머지 두 개의 모델은 수행 속도도 굉장히 느리고 원하는 결과를 도출해내지도 않았습니다. 이를 통해서 모델별로 적용해서 실제 데이터의 예측률을 파악하는 게 중요할 것이라고 생각하였으며, 저희가 수행하고자 할 프로젝트에서 사용할 수 있는 api 나 모델들이 여러 가지 있으므로 이를 적용하여 최선의 값을 도출해낼 필요가 있다고 판단하여 추후에 작업해보고자 합니다.

㉑. Word2vec을 이용한 단어 임베딩 中 단어 유사도 판단

추후에 Aspect Analysis 를 할 때 단어 유사도 판단의 과정이 포함될 것이라 생각하여 Word2vec 을 이용하여 단어 임베딩을 실시해보았습니다. 그중에서 이번에는 단어 유사도를 판단하여, 특정 단어와 유사한 단어가 무엇이 있는지 확인해보았으며, 좌표로서 나타낼 수 있음을 확인하였습니다. 이때, 네이버 영화 말뭉치 training 이 완료된 Set 를 이용하여 Word2vec 과정을 수행하였으며, Konlpy 를 이용하여 한글에 대한 분석을 실시해 보았습니다. 특히, konlpy 사용 중에 norm 이나 stem 을 이용하여 오타나 형태소의 원형을 이용하였습니다.

```

In [5]: import codecs
        #konlpy 0.5.0 버전 이후부터 이름이 Twitter에서 Okt로 바뀌었다.
        from konlpy.tag import Okt
        from gensim.models import word2vec
        from konlpy.utils import pprint

In [6]: def read_data(filename):
        with codecs.open(filename, encoding='utf-8', mode='r') as f:
            data = [line.split('ㄷ') for line in f.read().splitlines()]
            data = data[1:] # header 제외
        return data

In [7]: #파일 위치. 본인의 파일경로로 변경필요
        ratings_train = read_data('ratings_train.txt')
        #konlpy 중에서 트위터 형태소분석기 사용 (1)
        tw_tagger = Okt()

In [8]: # 토큰나이즈(의미단어검색) 함수. 트위터 형태소 분석기 사용 (2)
        # 형태소 / 품사 형태로 리스트화
        def tokens(doc):
            return ['/'.join(t) for t in tw_tagger.pos(doc, norm=True, stem=True)]
        #norm 기능을 이용해 오다를 정경(ex. 사중해를 사강해로), stem을 이용해 원형으로 반환(ex. 입니다를 이다로)

In [9]: # 파일중에서 영화 리뷰 데이터만 찾기
        docs = []
        for row in ratings_train:
            docs.append(row[1])

        data = [tokens(d) for d in docs]

In [11]: # [TRAIN] word2vec 으로 모델 생성 (3)
        w2v_model = word2vec.Word2Vec(data)

        # init_sims 명령어로 필요있는 메모리 반환
        w2v_model.init_sims(replace=True)

        # [TEST] 가장 유사한 단어 출력 (4)
        pprint(w2v_model.wv.most_similar(positive=tokens(u'남자 여배우'),
            negative=tokens(u'배우'), topn=1))

        [('여자/Noun', 0.8150866031646729)]

```

그림 4-13

```

In [14]: pprint(w2v_model.wv.most_similar(positive=tokens(u'주인공'),topn=10))

[('기선/Noun', 0.7544881105422974),
 ('얼굴/Noun', 0.6906263828277588),
 ('행동/Noun', 0.6644679307937622),
 ('남자/Noun', 0.6600486636161804),
 ('여자/Noun', 0.6532753705978394),
 ('악역/Noun', 0.6524346470832825),
 ('음장인물/Noun', 0.6311010122299194),
 ('성격/Noun', 0.630095899105072),
 ('캐릭/Noun', 0.6252544522285461),
 ('악당/Noun', 0.6231570639681897)]

In [16]: from sklearn.manifold import TSNE
        from matplotlib import font_manager as fm
        from matplotlib import rc
        movie_tsne = TSNE(n_components=2)
        movie_tsne

Out[16]: TSNE(angle=0.5, early_exaggeration=12.0, init='random', learning_rate=200.0,
          method='barnes_hut', metric='euclidean', min_grad_norm=1e-07,
          n_components=2, n_iter=1000, n_iter_without_progress=300, perplexity=30.0,
          random_state=None, verbose=0)

In [18]: movie_vocab = w2v_model.wv.vocab
        movie_similarity = w2v_model[movie_vocab]
        movie_similarity

C:\Users\Naruk\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning: Call to deprecated `__getitem__` (Method w
ill be removed in 4.0.0, use self.wv.__getitem__() instead).

Out[18]: array([[ 0.01583514, -0.01967263,  0.01102952, ..., -0.06493731,
                  -0.11728221, -0.10047553],
                 [ 0.02601962, -0.07917922, -0.02239081, ..., -0.11629011,
                  -0.03626932,  0.07144445],
                 [ 0.2009029 , -0.05290657,  0.02421443, ..., -0.1569652 ,
                  0.07382585, -0.17336367],
                 ...,
                 [ 0.10578212,  0.0687996 , -0.027095 , ...,  0.02845743,
                  -0.18845902, -0.03497209],
                 [ 0.00156862,  0.11541089, -0.0601042 , ...,  0.02754588,
                  -0.17417371,  0.06905237],
                 [ 0.00297765, -0.00140684,  0.05425932, ..., -0.0171057 ,
                  -0.16980827,  0.06580625]], dtype=float32)

```

그림 4-14

자료를 나타내 주는 코드를 수행하였습니다. 단어의 유사도 이므로 특정 단어와 유사한 단어를 보여주는 것인데, Aspect Analysis 의 경우 문법적인 연관성을 판단해야 하므로 임베딩에서 다른 과정을 추가적으로 요구함을 확인하였습니다.

```
In [20]: import pandas as pd
movie_transform_similarity = movie_tsne.fit_transform(movie_similarity)
movie_df = pd.DataFrame(movie_transform_similarity, index=movie_vocab, columns=['x', 'y'])
movie_df[0:10]
```

Out [20]:

	x	y
아/Exclamation	22.172894	-57.955566
덕빙/Noun	6.981616	45.611614
./Punctuation	18.876213	-60.003292
진짜/Noun	17.515112	-63.976002
짜증나다/Adjective	36.130859	53.686295
목소리/Noun	-56.858418	-29.822529
홀/Noun	23.910069	-57.069225
.../Punctuation	18.831528	-59.998028
포스터/Noun	46.868126	26.451733
보고/Noun	50.604397	-23.869593

그림 4-15

작업하고자 하는 타겟이 리뷰이다 보니, 'ㅋㅋㅋ'나 '꿀잼'과 같은 은어들에 대해서 처리하는 과정이 필요하리라 판단하였는데 Okt 의 경우 이러한 은어들도 한국어 조사 등으로 처리가 되고 있음을 따로 확인하였습니다. 그 외의 분석기에 대해서도 확인이 필요할 듯 싶습니다. 분석기 별로 수행 속도가 다름을 확인하였고, 분석을 하여 보여주는 결과의 종류도 각기 다르기 때문입니다.

A. 트위터에서 마이닝한 트윗 정보로 유사도 판단하기.

```
In [1]: #https://jeongwookie.github.io/2019/06/10/190610-twitter-data-crawling/
# GetOldTweet3 사용 준비
try:
    import GetOldTweets3 as got
except:
    !pip install GetOldTweets3
    import GetOldTweets3 as got

In [2]: # datetime을 사용해 가져올 범위를 정의
# 예제 : 2020-01-01 ~ 2020-04-01

import datetime

days_range = []

start = datetime.datetime.strptime("2020-01-01", "%Y-%m-%d")
end = datetime.datetime.strptime("2020-04-01", "%Y-%m-%d")
date_generated = [start + datetime.timedelta(days=x) for x in range(0, (end-start).days)]

for date in date_generated:
    days_range.append(date.strftime("%Y-%m-%d"))

print("=== 설정된 트윗 수집 기간은 {} 에서 {} 까지 입니다 ===".format(days_range[0], days_range[-1]))
print("=== 총 {}일 간의 데이터 수집 ===".format(len(days_range)))

=== 설정된 트윗 수집 기간은 2020-01-01 에서 2020-03-31 까지 입니다 ===
=== 총 91일 간의 데이터 수집 ===
```

그림 4-16

```
In [4]: # 특정 검색어가 포함된 트윗 검색하기 (query search)
# 검색어 : 기묘한이야기

import time

# 수집 기간 맞추기
start_date = days_range[0]
end_date = (datetime.datetime.strptime(days_range[-1], "%Y-%m-%d")
            + datetime.timedelta(days=1)).strftime("%Y-%m-%d") # setUntil() 끝을 포함하지 않으므로, day + 1

# 트윗 수집 기준 정의
tweetCriteria = got.manager.TweetCriteria().setQuerySearch('기묘한이야기')#
            .setSince(start_date)#
            .setUntil(end_date)#
            .setMaxTweets(-1)

# 수집 with GetOldTweet3
print("Collecting data start.. from {} to {}".format(days_range[0], days_range[-1]))
start_time = time.time()

tweet = got.manager.TweetManager.getTweets(tweetCriteria)

print("Collecting data end.. [0:0.2f] Minutes".format((time.time() - start_time)/60))
print("=== Total num of tweets is {} ===".format(len(tweet)))

Collecting data start.. from 2020-01-01 to 2020-03-31
Collecting data end.. 21.67 Minutes
=== Total num of tweets is 7758 ===
```

그림 4-17

```
In [5]: # GetOldTweet3 에서 제공하는 기본 변수
# 유저 아이디, 트윗 링크, 트윗 내용, 날짜, 리트윗 수, 관심글 수
# 원하는 변수 골라서 저장하기

from random import uniform
from tqdm import tqdm_notebook

# initialize
tweet_list = []

for index in tqdm_notebook(tweet):

    # 메타데이터 목록
    username = index.username
    link = index.permalink
    content = index.text
    tweet_date = index.date.strftime("%Y-%m-%d")
    tweet_time = index.date.strftime("%H:%M:%S")

    info_list = [tweet_date, tweet_time, username, content, link]
    tweet_list.append(info_list)

# 휴식
time.sleep(uniform(1,2))

HBox(children=(IntProgress(value=0, max=7758), HTML(value='')))
```

그림 4-18

예시로 2020년 1월 1일부터 3월 31까지 “기묘한이야기”를 검색한 결과를 마이닝하였습니다.


```
In [6]: # csv 파일로 결과 저장 - pandas
# 파일 저장하기

import pandas as pd

#twitter_df = pd.DataFrame(tweet_list,
#                           # columns = ["date", "time", "user_name", "text", "link", "retweet_counts", "favorite_counts",
#                           # "user_created", "user_tweets", "user_followings", "user_followers"])

twitter_df = pd.DataFrame(tweet_list,
                          columns = ["date", "time", "user_name", "text", "link"])

# csv 파일 만들기
twitter_df.to_csv("sample_twitter_data_{}_to_{}.csv".format(days_range[0], days_range[-1]), index=False)
print("=== {} tweets are successfully saved ===".format(len(tweet_list)))

=== 7758 tweets are successfully saved ===

In [7]: #생성 파일 확인
# 파일 확인하기

df_tweet = pd.read_csv('sample_twitter_data_{}_to_{}.csv'.format(days_range[0], days_range[-1]))
df_tweet.head(10) # 위에서 10개만 출력

Out[7]:
```

	date	time	user_name	text	link
0	2020-03-31	23:52:03	H	요즘 청담사리지가 유행하던데 난 넷을 잘, 많이 안봐서 킹덤, 기묘한이야기, 내가	https://twitter.com/.../status/124...
1	2020-03-31	21:07:31		혁 기묘한 이야기 보러구 했는데!!! 도연 패럴리 재미있어! 애들이 너무 귀엽습미	https://twitter.com/BB.../2450954049...
2	2020-03-31	20:58:52		기묘한 이야기 재밌고 다 좋은데 시즌 3는 특히 엄청 그로테스크하고 고어하니 주	https://twitter.com/rz.../us/12450932...
3	2020-03-31	20:57:28		공 속 인물한테 꿈에서 꿈 좀 거 꿈꿨다고 말했는데 자기도 똑같은 꿈 꿔다고 하더	https://twitter.com/hy.../s/124509287...
4	2020-03-31	20:08:00		눈아 기묘한이야기 심령어를 보고자염	https://twitter.com/91.../2450804244...
5	2020-03-31	19:13:12		월모야 기묘한이야기 시즌3 촬영예정인줄알았는데 진작에 출시당테	https://twitter.com/M.../s/124506663...
6	2020-03-31	19:03:14		혈 넷을 보세요?!! 산타클라리타다이어트 / 엘리트들 / 나를 자버린 스파이 / ...	https://twitter.com/m.../1245064127...
7	2020-03-31	18:22:17		넷플릭스 추천해주세용 문서운거 빼고 ... 저 겁 완전 많아서 기묘한이야기도 못보	https://twitter.com/i.../2450538196...

그림 4-19

크롤링한 내용을 csv 파일로 저장하여 위와 같이 저장하고 저장한 파일에서 내용과 관련된 부분만 추출하여 konlpy 와 word2vec 을 이용한 분석을 진행해보았습니다.

```
In [24]: from konlpy.tag import Twitter
twitter = Twitter()

word_dic={}

for i in df["text"]:
    mallist=twitter.pos(i)
    for word in mallist:
        if word[1]=="Noun" or word[1]=="Verb" or word[1]=="Adjective":
            if not(word[0] in word_dic):
                word_dic[word[0]]=0
            word_dic[word[0]]+=1
print(word_dic)
```

```
{'같은데': 48, '있지': 9, '나': 519, '어제': 68, '학교': 28, '귀신': 30, '봤다': 71, '년': 9, '도': 119, '헛소리': 4, '아뎀다고': 1, '그래': 21, '일정': 2, '팔로워': 2, '도달': 2, '시': 24, '개장': 4, '확정': 3, '됩니다': 13, '너': 102, '그': 339, '소문': 7, '들은': 5, '작': 23, '있어': 36, '아가': 5, '발음': 4, '출귀': 2, '길': 15, '모어': 7, '결산': 26, '임': 118, '넷': 73, '오케이': 80, '브루클린': 68, '나인': 141, '글라': 10, '엘리스': 1, '릴뉴블': 1, '있는데': 63, '더스타': 21, '알마': 64, '둘': 51, '나와서': 20, '괴리관': 1, '들어요': 2, '본다': 41, '나탈': 1, '리아': 1, '커피': 7, '나오는': 85, '뒤': 33, '일제': 14, '사귀게': 1, '멤': 33, '취향': 76, '비슷하면': 1, '아메리칸': 14, '반달': 1, '리즈': 6, '봐줘': 21, '좋아하면': 31, '반': 16, '뵤': 222, '더': 207, '불리': 12, '티선': 11, '종이': 220, '집': 265, '오유': 80, '불': 81, '오줌': 3, '지리': 3, '재밌음': 51, '보고싶은데': 25, '진도': 6, '나감': 1, '글루': 15, '리스': 17, '뭐': 209, '레이스': 170, '내일': 33, '만나': 4, '보실수': 3, '보신': 28, '분': 100, '있나올': 1, '다음': 61, '불까': 78, '말까': 3, '고민': 33, '중': 297, '뵤더니': 12, '가발': 2, '바느질': 1, '꿀냈서': 1, '용': 6, '와': 93, '왜': 196, '레오': 3, '색': 5, '국내': 3, '팔지': 1, '알아서': 8, '이런': 66, '별짓': 1, '하게': 29, '만드': 1, '컷팅': 1, '해야하는데': 5, '자야': 5, '해서': 113, '일단': 81, '글러다니던': 1, '인형': 24, '머리': 175, '씩씩': 1, '품': 149, '꼭같다': 1, '게이': 10, '튼이다': 1, '보시나요': 10, '곰': 1, '이제': 156, '이집트': 1, '파라미드': 1, '세줄': 1, '있게': 6, '되있습니': 4, '비정환': 1, '표정': 11, '진상': 42, '열': 25, '동네': 4, '부': 81, '다봤다': 1, '왕': 16, '보고싶다': 32, '보려고': 31, '달': 49, '무로': 17, '하는건데': 2, '제': 176, '글래고': 8, '운다': 4, '이': 211, '복': 6, '불': 1, '아시': 10, '검': 131, '수박': 2, '겉': 1, '할기로': 1, '파는': 7, '만아': 16, '스킨스': 11, '같은것도': 4, '그냥': 83, '형': 2, '분위': 45, '출방': 2, '이려고': 5, '좋아함': 31, '성격': 9, '대사': 39, '관계도': 1, '이런건': 4, '알바': 3, '무지개': 1, '전구': 4, '배송': 5, '와라': 1, '느낌': 108, '내면': 2, '방': 32, '쳐': 14, '박해있을거라고': 1, '데이고': 8, '결제': 49, '헛다': 1, '봐라': 16, '안보': 28, '아남': 48, '기묘한이야기': 1, '존나': 111, '좋아한다고': 2, '오빠': 22, '나가고': 4, '나가자': 1, '배우': 58, '투하연': 8, '마지막': 7, '한국': 54, '요괴': 27, '대박': 10, '프로젝트': 9, '후원자': 8, '님': 33, '공유': 19, '님': 182, '결': 89, '쓰요': 1, '재밌음': 4, '바': 51, '쓸': 16, '다뵤습': 1, '게': 84, '외국': 1, '맞아는': 1, '아닌것': 3, '같': 33, '달았지만': 1, '머가': 10, '다
```

그림 4-20

그림 4-20과 같이 명사, 동사, 형용사 같은 리뷰와 작품과 연관 지어 중요한 품사의 단어들만 다시 추출하였고 그 단어가 마이닝한 결과 내에서 얼마만큼의 횟수만큼 사용되었는지 파악하였습니다.

```
In [26]: for word, count in keys[:20]:
        print("{} {}".format(word, count))
```

```
이야기 7966
기묘한 7753
시즌 942
전파 607
나 519
거 511
넷플릭스 465
보고 462
추천 459
저 418
넷플 369
것 352
그 339
드라마 338
내 313
중 297
때 265
집 265
안 259
곳 238
```

그림 4-21

위와 같이 리뷰에서 가장 많은 쓰인 단어를 추출할 수 있었고 이를 토대로 다음 과정에서는 유용한 단어에 대한 추출방법이 과제가 될 것 같습니다.

```
In [31]: results = []
        for i in dff["text"]:
            mallist = twitter_pos(i, norm=True, stem=True)
            # stem=True -> 어근으로 출력하라는 의미
            # ex) '그려요' -> '그을다'
            # print(mallist)
            r = []
            for word in mallist:
                if not word[0] in ["Josa", "Punctuation", "Foreign", "Suffix", "Eomi"]:
                    r.append(word[0])
            # 결과에서 제외 할 품사 입력하기
            print(r)
            rl = (" ".join(r)).strip() #공백제거
            results.append(rl)
        print(results)
```

```
['기묘하다', '이야기', '시즌', '1', '2', '3', '준서대로', '새비', '술어불다', '줄애', '시즌', '4', '불다', '기', '대점', '모기']
['하이큐', '회지', '하산', '기묘하다', '이야기', '하산', 'In', 'our', 'time', '하산', '한지봉', '세남자', '양도', '판매', '구함',
'니', '델', '연선', '부탁드리다']
['그냥', '보다', '거', '계속', '보구', '요즘', '기묘하다', '이야기', '불', '까말다', '생각', '중', '이다', '닷', '썩연님', '뽀',
'보다']
['기묘하다', '이야기']
['발레', '영상', '유튜브', '보다', '저', '찾다', '보다', '바', '젬다', '기묘하다', '이야기', '저', '줄다', '거의', '보다', '같다',
'에', 'ㅋㅋ', 'ㅠ', '새롭다', '나오다', 'ㅠ', '노래', '저', '방탄', '소년단', '원', '들다', 'ㅋㅋ', 'ㅇ', 'ㅋㅋ']
['버즈', '오브', '프레이', '할리퀸', '에브다', '그냥', '할리퀸', '다', '함', '클로켓', '한국판', '기묘하다', '이야기', '일드', '감
특취', '더', '이상', '노를', 'ㅠ']
['기묘하다', '이야기', '하나', '영기다', '있다', '영역', '오다']
['앗', 'ㅋㅋ', '저', '그때', '그때', '다르다', '요즘', '짧다', '발레', '영상', '물릴', '몇개', '주', '구', '장황', '둘리다', '보
구', '드라마', '기묘하다', '이야기', '시즌', '4', '나오다', '같다', '재향', '중이', 'at', 'It', '영화', '요즘', '재향하다', '없
다', 'ㅋㅋ', '뉴님', '요즘', '뽀', '들다', 'at', 'It']
['기묘하다', '이야기', '미치다', '마관', '하다', '되다', '더', '과', 'ㅡ', '실', 'ㅇㅇ', 'ㅇ', 'ㅇㅇ']
['기묘하다', '이야기', '한번', '보다', '보다']
['마블', '제시카', '존스', 'DC', '타이탄', '기묘하다', '이야기', '한니발']
['기묘하다', '이야기', '4', '언제', '나오다', 'ㅠ', '내', '기', '달리', '있다']
['하이큐', '회지', '양도', '발다', '레스큐일', '원더풀', '데이즈', '기묘하다', '이야기', '마법사', '기록', 'oh', 'kids', '디엠',
'줄다', 'ㅠ']
```

그림 4-22

다음은 형용사와 동사를 어근으로 바꿔주는 작업을 하였습니다.

```

In [33]: from gensim.models import word2vec
data = word2vec.LineSentence(data_file)
print(data)
model = word2vec.Word2Vec(data, size=100, window=10, hs=1, min_count=2, sg=1)
# CBOW, Skip-gram(0)
model.init_sims(replace=True) #필요없는 메모리는 unload
model.save("news.model")
print("ok")

<gensim.models.word2vec.LineSentence object at 0x00000176B8BC6488>
ok

In [36]: model = word2vec.Word2Vec.load("news.model")
print(model.similarity("기묘하다", "넷플릭스"))
print(model.similarity("기묘하다", "밀리"))
#두 단어의 유사도 -> 1에 근접할 수록 서로 상관관계가 있다.
print(model.most_similar("기묘하다"))

print(model.most_similar(positive=["기묘하다"]))
print(model.most_similar(positive=["기묘하다", "이 야기"], negative=["정 그럽다"], topn=5))

0.5386249
0.32553965
[('개굴', 0.6190359592437744), ('역시', 0.6100665926933289), ('월', 0.6065816283226013), ('기정', 0.6044149398803711), ('이따', 0.6040357947349548), ('알다', 0.6020412445068359), ('이러다가', 0.6007715463638306), ('거왕', 0.6000221967697144), ('도리', 0.5988008975982666), ('축', 0.5978043079376221)]
[('개굴', 0.6190359592437744), ('역시', 0.6100665926933289), ('월', 0.6065816283226013), ('기정', 0.6044149398803711), ('이따', 0.6040357947349548), ('알다', 0.6020412445068359), ('이러다가', 0.6007715463638306), ('거왕', 0.6000221967697144), ('도리', 0.5988008975982666), ('축', 0.5978043079376221)]
[('πππ', 0.4821982681751251), ('열', 0.46736279129981995), ('좋아하다', 0.45916640758514404), ('추광', 0.44668009877204895), ('헨즈', 0.4345983564853668)]

```

그림 4-23

이전의 작업을 토대로 단어 간의 유사성을 판단해보았습니다.

이번 과정에서는 추출한 데이터를 konlpy 및 word2vec 과 같은 분석에 유용한 툴과 연결시키는 작업을 해보았고 이 결과 유용한 데이터의 선별 및 축약어와 같은 비속어에 대한 판별 또한 다음 과제가 될 것으로 보입니다.

5. UI 제작

(진행도가 있을 때 추가 작성예정)

III. 과제 평가

1. 개선방안

- 이번 주차는 진행하면서, 이전과 같이 단순히 API를 사용해보는 것이 아니라, 주제를 확장하다 보니 개념적인 부분의 이해가 부족한 것 같아 개념적인 이해를 코딩하면서 이해하고 정리하였으며 실제로 어떤 식으로 적용할 수 있을까 생각을 많이 한 주차입니다.

<input type="checkbox"/> bs4, selenium, lxml 크롤링.ipynb	12시간 전
<input type="checkbox"/> practice.ipynb	Running 한 시간 전
<input type="checkbox"/> Twitter 품사분석(phrases 문장을 구단위로 쪼개주는 메서드가 있는게 특징).ipynb	Running 18시간 전
<input type="checkbox"/> 감성분석(감성사전이용).ipynb	Running 15시간 전
<input type="checkbox"/> 감성분석(지도 기계학습기반).ipynb	Running 14시간 전
<input type="checkbox"/> 군집분석.ipynb	Running 13시간 전
<input type="checkbox"/> 꼬꼬마 연습.ipynb	Running 18시간 전
<input type="checkbox"/> 불용어제거, n-gram(n개단어), 품사분석, 어근동일화(의미는 같은데 생김새 달라. s나 es같은게 붙어서).ipynb	Running 한 시간 전
<input type="checkbox"/> 빈출단어 추출.ipynb	Running 15시간 전
<input type="checkbox"/> 소셜커머스후기분석.ipynb	11시간 전
<input type="checkbox"/> 연관어분석.ipynb	Running 13시간 전
<input checked="" type="checkbox"/> 정규표현식 연습.ipynb	Running 18시간 전
<input type="checkbox"/> 트위터 크롤링.ipynb	12시간 전

그림 1-4

- 이번의 경우 konlpy보다는 nltk, 영화 리뷰(영어) data set과 api를 이용한 긍부정 점수 산출, 감성 사전 data set의 인덱스화 등 영어를 사용한 분석을 다뤄서 자연어 처리와 감성 분석의 개념을 이해하는데 시간을 활용했기에, 다음에는 konlpy와 한국어 감성사전을 이용해 보고자 합니다.
- DB구축에 있어서는 교수님께서 조언해주신 대로 기능을 확장시키고 있습니다. 또한, data별로 어떻게 활용할지를 생각하는 과정에 있어서 기초적인 ERD모델은 구축되었고 소소한 수정 이후에 확정되는 대로 데이터베이스를 구축하고자 합니다.
- 어떤 데이터를 추출할지(웹소셜 리뷰)를 정했기 때문에 사이트별 크롤러를 제작하고자 합니다.
- 이번 주차는 캡스톤 설계 전공과목 프로젝트 중간보고서 발표 준비와 함께 구축 시작 전 팀원과 회의를 통해 세부 사항 정리에 집중했습니다. 다음 주차부터 웹 크롤러 제작부터 구축에 들어갈 예정입니다.

2. 기대효과

ㄱ. 기업적 측면

- 즉각적인 피드백이 필요한 문화 산업에서 소셜미디어와 커뮤니티 같은 독자층의 실시간 반응이 보이는 곳의 리뷰를 통합적으로 확인 가능함으로써 앞으로의 홍보, 제작, 투자 방향 선택에 도움이 되는 지표가 될 것이다.

ㄴ. 사용자 측면

- 별점 테러와 같이 실제 작품에 대한 후기가 아닌 평가 반영으로 실제 작품의 후기를 원하는 사용자에게 더욱 사실적인 후기를 각기 다른 플랫폼에서 검색해 볼 필요 없이 한 곳에서 확인이 가능할 것이다.
- 리뷰에서 자주 언급된 단어를 통해 중요 키워드를 산출해내기 때문에 선호하는 양상의 작품을 기호에 맞춰 선택하기 쉽다.
- 현재 작품에 대한 주요 평가가 어떻게 되는지 시각적으로 확인 가능합니다

■ 참고문헌

- [3] 파이썬을 활용한 클로러 개발과 스크레이핑 입문 (카토 카츠야, 요코야마 유우키, 위키북스, 2019)
- [4] 파이썬 데이터 수집 자동화 한방에 끝내기 한입에 웹크롤링 (김경록, 서영덕, 비제이퍼블릭, 2018)
- [5] 파이썬을 이용한 웹크롤링과 스크레이핑 (카토 코타, 위키북스, 2018)
- [6] 파이썬을 이용한 머신러닝, 딥러닝 실전 개발 입문 (쿠지라 히코우즈쿠에, 위키북스, 2019)
- [7] Web Scraping with Python (라이언미첼, 한빛미디어, 2019)
- [8] 잡아라! 텍스트 마이닝 with 파이썬 (서대호, 비제이퍼블릭, 2019)
- [9] <https://www.crummy.com/software/BeautifulSoup/bs4/doc.ko/>
- [10] 오피니언 마이닝 기술을 이용한 효율적 상품평 검색 기법 (윤홍준, 김한준, 장재영, 2010)
- [11] 한글 텍스트의 오피니언 분류 자동화 기법 (김진옥, 이선숙, 용환승, 2011)
- [12] 상품평가 텍스트에 암시된 사용자 관점추출 (장경록, 이강욱, 맹성현, 2013)
- [13] 텍스트 마이닝을 이용한 2012년 한국대선 관련 트위치 분석 (배정환, 손지은, 송민, 2013)
- [14] 한글 감성어 사전 api구축 및 자연어 처리의 활용 (안정국, 김희웅, 2014)
- [15] 한글 음소단위 trigram-signature 기반의 오피니언 마이닝 (장두수, 김도연, 최용석, 2015)
- [16] 소셜네트워크서비스에 활용할 비표준어 한글처리 방법연구 (이종화, 레환수, 이현규, 2016)

- [17] 인공지능을 활용한 오피니언 마이닝 - 소셜 오피니언 마이닝은 무엇인가?⁵ (윤병운, 2017)
- [18] 한국어 비정형 데이터 처리를 위한 효율적인 오피니언 마이닝 기법 (남기훈, 2017)
- [19] A study on Sentiment Analysis with Multivariate ratings in Online Reviews (임소현, 2020)

⁵ https://www.samsungsds.com/global/ko/support/insights/1195888_2284.html