



네이버 영화 장르 분석

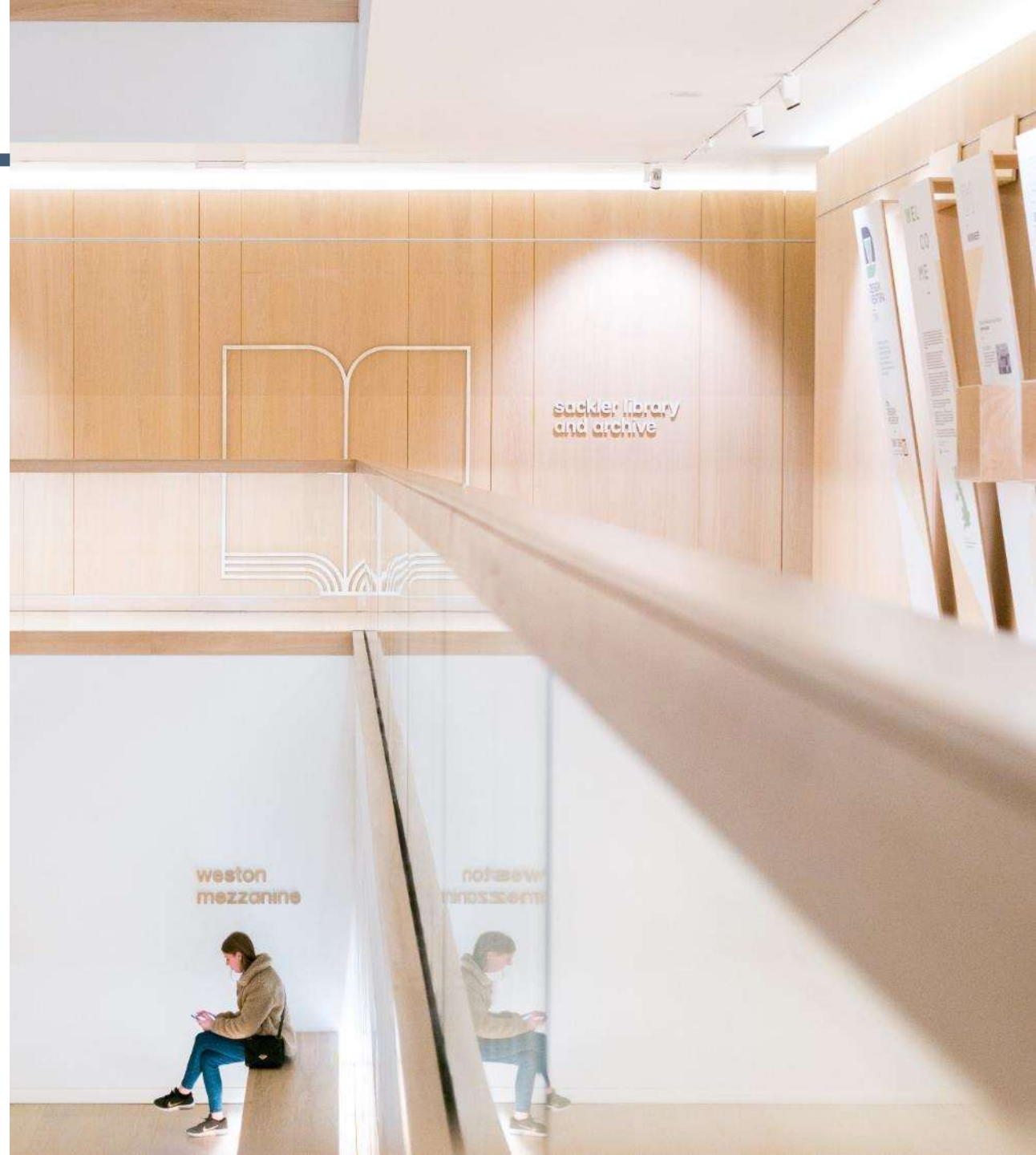
목차 a table of contents

1 웹 크롤링 내용

2 전처리 작업

3 나이트베이지 결과

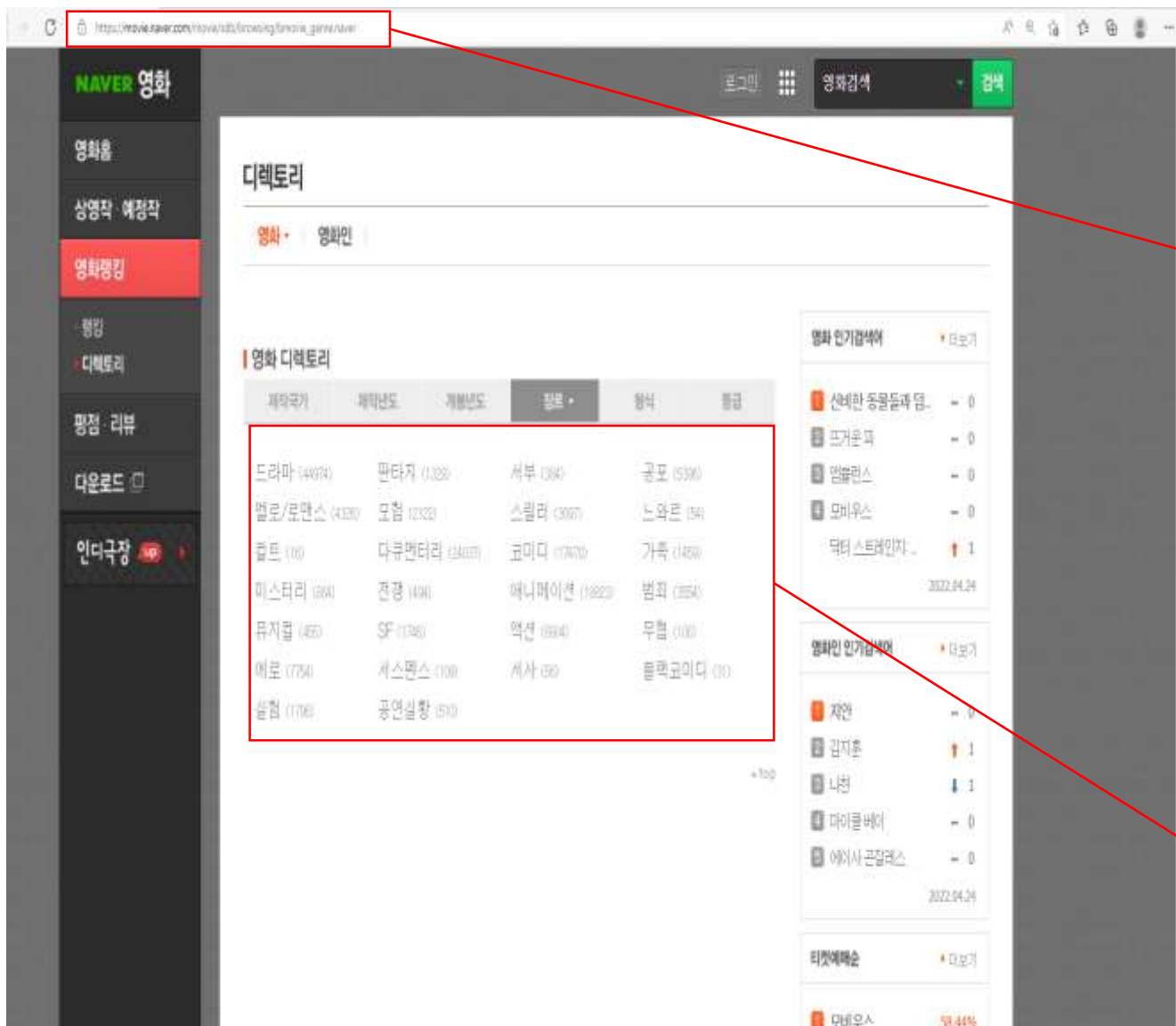
-형태소분석
-n-gram, bigram



PART 1

웹 크롤링





selenium을 활용하여 웹크롤링 진행

```
from selenium import webdriver
```

```
url = 'https://movie.naver.com/movie/sdb/browsing/bmovie_genre.naver'
driver = webdriver.Chrome('c:/data/chromedriver.exe')
driver.get(url)
driver.implicitly_wait(2)
```

```
html = driver.page_source
soup = bs(html, 'html.parser')
```

장르 : 판타지/공포/멜로/스릴/다큐멘터리/가족 6개 장르 웹크롤링

크롤링 코드

```
for i in range(1,4,2):
    driver.find_element(By.CSS_SELECTOR,'table.directory_item_other > tbody > tr:nth-of-type(2) > td:nth-of-type('+str(i)+') > a').click() # 장르 선택

for u in range(2,7): # 페이지 이동 for문
    for j in range(1,21): # 자료 수집 for문
        try:
            driver.find_element(By.CSS_SELECTOR,'ul.directory_list > li:nth-of-type('+str(j)+') > a').click() # 제목클릭
            html = driver.page_source
            soup = bs(html,'html.parser')
            time.sleep(2)
            name = soup.select_one('h3.h_movie > a').text # 제목 가져오기
            genre = soup.select_one('dl.info_spec > dd > p > span > a:nth-of-type(1)').text # 장르 가져오기
            text = soup.select_one('p.con_tx').text # 줄거리 가져오기
            movie_1 = movie_1.append({'name': name,'genre':genre,'text':text},ignore_index = True) # DataFrame 저장
            time.sleep(2)
            driver.back() # 뒤로가기
            time.sleep(2)

        except:
            failed_url.append(i) # 오류 페이지 저장
            driver.back() # 뒤로가기

    if u == 2: #u값이 2면 밑에 코드 진행
        driver.find_element(By.CSS_SELECTOR,'div.pagenavigation > table > tbody > tr > td:nth-of-type('+str(u)+') > a').click()
    else: # u값이 2가 아니면 밑에 코드 진행
        driver.find_element(By.CSS_SELECTOR,'div.pagenavigation > table > tbody > tr > td:nth-of-type('+str(u+1)+') > a').click()

driver.find_element(By.CSS_SELECTOR,'div.tab_type_6 > ul > li:nth-of-type(4)').click() # 장르 페이지로 가기
time.sleep(2)
```

PART 2

전 처리 과정



전처리 작업

- 불필요한 단어 및 공백 제거 작업

```

movie['text'] = movie['text'].apply(lambda x : re.sub('\xa0', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('\n|\t+', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub("\{'I Think, Therefore I am \\\(나는 생각한다, 고로 존재한다\\)" - Descartes \\\(데카르트\\), 1596-1650\\}', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('\s{2,}', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('\(.+?\)', '', x))
movie['text'] = movie['text'].apply(lambda x : x.strip())
movie['text'] = movie['text'].apply(lambda x : re.sub('\(.+?\)', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('[, \"]', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('\d+\w', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('\[.+\]', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('\s{2,}', '', x))
movie['text'] = movie['text'].apply(lambda x : x.strip())
movie['text'] = movie['text'].apply(lambda x : re.sub('...', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub("[']-|?<|>|!'", '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('MEM:', '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub("[']'", '', x))
movie['text'] = movie['text'].apply(lambda x : re.sub('[\"']', '', x))

```


A dark, atmospheric photograph of an art gallery. The walls are lined with numerous framed black and white photographs. The lighting is low, creating a moody environment. A bright red horizontal bar is superimposed over the upper right portion of the image, containing the text 'PART 3' and '나이프베이스' in white.

PART 3

나이프베이스


```

okt = Okt()

def okt_pos(arg):
    token_cpr = []
    for i in okt.pos(arg):
        if i[1] in ['Noun','Adjective']:
            token_cpr.append(i[0])
    token_cpr = [j for j in token_cpr if len(j) >= 2]
    return token_cpr
tokenizer = okt_pos

x_train,x_test,y_train,y_test= train_test_split(movie['text'],movie['genre'],test_size = 0.2)

cv = CountVectorizer(tokenizer = okt_pos)
x_train = cv.fit_transform(x_train)
cv.get_feature_names()
x_train.toarray()

x_test = cv.transform(x_test)
x_test.toarray()

nb = MultinomialNB()
nb.fit(x_train,y_train)

y_predict = nb.predict(x_test)
accuracy_score(y_test,y_predict) # 1자료

print(classification_report(y_test,y_predict)) # 2번 자료

```



형태소 분석(품사태깅)을 활용

명사/형용사만 사용하여 나이브베이지를 확인한 결과 51% 확인

	precision	recall	f1-score	support
가족	0.67	0.87	0.62	7
공포	0.75	0.55	0.63	11
다큐멘터리	0.60	0.27	0.37	11
멜로/로맨스	0.50	1.00	0.67	11
스릴러	0.25	0.21	0.23	14
판타지	0.53	0.57	0.55	14
accuracy			0.51	68
macro avg	0.55	0.53	0.51	68
weighted avg	0.53	0.51	0.50	68

```
x_train,x_test,y_train,y_test= train_test_split(movie['text'],movie['genre'],test_size = 0.2)
```

```
cv = CountVectorizer(ngram_range=(2,2))
x_train = cv.fit_transform(x_train)
cv.get_feature_names()
x_train.toarray()
```

```
x_test = cv.transform(x_test)
x_test.toarray()
```


```
nb = MultinomialNB()
nb.fit(x_train,y_train)
```

```
y_predict = nb.predict(x_test)
accuracy_score(y_test,y_predict) # 1번 자료
```

```
print(classification_report(y_test,y_predict)) # 2번 자료
```

n-gram,bigram을 활용

명사/형용사만 사용하여 나이브베이지스를 확인한 결과 26% 확인



	precision	recall	f1-score	support
가족	0.00	0.00	0.00	12
공포	0.17	0.10	0.12	10
다큐멘터리	0.00	0.00	0.00	8
멜로/로맨스	0.29	0.92	0.44	12
스릴러	0.10	0.09	0.10	11
판타지	0.42	0.33	0.37	15
accuracy			0.26	68
macro avg	0.16	0.24	0.17	68
weighted avg	0.18	0.26	0.19	68



Inspiration