

텍스트 데이터 분석

Team_Project

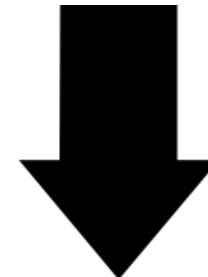
20202640 박원우

20202666 최민서

관심 주제

"국내 여행지 데이터를 수집해 실시간으로 많이 가는 장소들을 국내 여행을 다니는 사람들에게 추천할 수 있지 않을까?"

데이터소스



네이버 blog 및 cafe
내용 크롤링

MAIN TOPIC: 여행

01

경주여행

02

가평여행

03

제주여행

04

여수여행

05

춘천여행

06

부산여행

07

광주여행

08

강릉여행

텍스트 데이터 수집 blog

```
driver = webdriver.Chrome()
# query를 리스트 형식으로 받아 순차적으로 추출
base_url_blog = 'https://openapi.naver.com/v1/search/blog.json' # 네이버 블로그 검색 API의 기본 URL
# 여행지 키워드를 리스트로 저장
queries = ['부산여행', '여수여행', '제주여행', '강릉여행', '경주여행', '가평여행', '광주여행', '춘천여행']
n_display = 100 # 한 번의 요청에 가져올 결과 개수
sort = 'sim' # 결과 정렬 방식 ('sim'은 유사도 기준 정렬)

all_results = [] # 크롤링 결과를 저장할 리스트
i = 0
# 각 여행지 키워드(query)마다 최대 1000개의 블로그 글을 가져옴
for query in queries:
    encQuery = urllib.parse.quote(query) # 키워드를 URL 인코딩
    # 블로그 크롤링 시작 인덱스 (1부터 시작)
    start = 1
    while start <= 1000:
        # API 요청 URL 구성
        url = f'{base_url_blog}?query={encQuery}&display={n_display}&start={start}&sort={sort}'
        my_request = urllib.request.Request(url)
        my_request.add_header("X-Naver-Client-Id", client_id)
        my_request.add_header("X-Naver-Client-Secret", client_secret)

        # API 요청 보내기
        response = urllib.request.urlopen(my_request)
        rescode = response.getcode() # 응답 코드 확인
        if rescode == 200:
            response_body = response.read() # 응답 본문 읽기
        else:
            print("Error Code:" + rescode) # 오류 코드 출력 후 다음 루프 진행
            continue

        search_results = response_body.decode('utf-8') # 응답 본문을 JSON 형식으로 디코딩
        search_results = eval(search_results) # 문자열을 Python 객체로 변환

        # 각 블로그 글(item)에 대해 처리
        for item in search_results['items']:
            # 링크, 게시 날짜, 제목, 본문 내용 추출
            link = html.unescape(item['link']).replace('\n', '')
            postdate = item.get('postdate', 'Unknown') # 'postdate' 필드 사용, 없으면 'Unknown'
            title = remove_tag(item['title']) # HTML 태그 제거
            content = remove_tag(item['description']) # HTML 태그 제거
            # 결과 리스트에 추가
            all_results.append({
                'source': 'blog',
                'query': query,
                'url': link,
                'postdate': postdate,
                'title': title,
                'content': content
            })

        start += 100 # 다음 100개의 결과를 가져오기 위해 시작 인덱스 증가
        time.sleep(1) # API 호출 간격을 1초 대기하여 서버 과부하 방지
    
```

✓ 여행지 키워드를 리스트로 저장

여행지 키워드 8가지를 리스트로 저장해 각 키워드에 대해 블로그 글을 가져올 수 있게함

✓ 반복문을 통해 블로그 글 크롤링

네이버 api로 블로그 글을 크롤링, 이때 반복문을 사용하여 각 키워드 당 1000개의 데이터를 크롤링할 수 있도록 함

✓ 각 블로그 글에 대해 필요한 부분 추출

링크, 본문, 기사제목, 게시 날짜 등 필요한 부분을 간단한 전처리와 함께 추출

cafe

```
base_url_cafe = 'https://openapi.naver.com/v1/search/cafearticle.json'
# Cafe crawling
start = 1
while start <= 1000:
    url = f'{base_url_cafe}?query={encQuery}&display={n_display}&start={start}&sort={sort}'
    my_request = urllib.request.Request(url)
    my_request.add_header("X-Naver-Client-Id",client_id)
    my_request.add_header("X-Naver-Client-Secret",client_secret)
    response = urllib.request.urlopen(my_request)
    rescode = response.getcode()
    if rescode == 200:
        response_body = response.read()
    else:
        print("Error Code:" + rescode)
        continue

    search_results = response_body.decode('utf-8')
    search_results = eval(search_results)

    for item in search_results['items']:
        link = html.unescape(item['link']).replace('\\\\','')
        postdate = item.get('cafewrite', 'Unknown') # 'cafewrite' 필드 사용
        title = remove_tag(item['title']) # HTML 태그 제거
        content = remove_tag(item['description']) # HTML 태그 제거
        all_results.append({
            'source': 'cafe',
            'query': query,
            'url': link,
            'postdate': postdate,
            'title': title,
            'content': content
        })
    start += 100
    time.sleep(1) # 1초 대기
driver.quit()

df_all_results = pd.DataFrame(all_results)

# 중복제거
df_travel = df_all_results.drop_duplicates(subset = ['content'],keep = 'first', inplace= False).reset_index(drop = True)
idx_lst = []
for idx,i in enumerate(df_travel['content']):
    if type(i) == list:
        idx_lst.append(idx)
df_travel.drop(index = idx_lst, inplace= True)
df_travel.dropna(inplace=True)
```



카페 글도 블로그 와 동일하게 진행

for문에 함께 포함시켜 base url주소만 cafe주소로 설정해
준뒤 동일하게 api크롤링 진행



결과 저장 및 중복 제거

크롤링 결과를 데이터프레임 형태로 저장한 후
drop_duplicate 사용해 중복 제거 결과적으로 총 15000개
가량의 데이터 추출

텍스트 데이터 전처리

```
import os
import konlpy

# konlpy 모듈 경로 확인
konlpy_path = os.path.dirname(konlpy.__file__)
print(f"konlpy 경로: {konlpy_path}")

# 작업 디렉토리를 konlpy java 디렉토리로 변경
java_dir = os.path.join(konlpy_path, 'java')
os.chdir(java_dir)
print(f"Current working directory: {os.getcwd()}")

# data 파일 경로 설정
file_path = os.path.join(java_dir, 'org/openkoreantext/processor/util/noun/names.txt')

# 기존 데이터 읽기
if os.path.exists(file_path):
    with open(file_path, 'r') as f:
        data = f.read()
else:
    data = ""
    print(f"{file_path} does not exist. A new file will be created.")

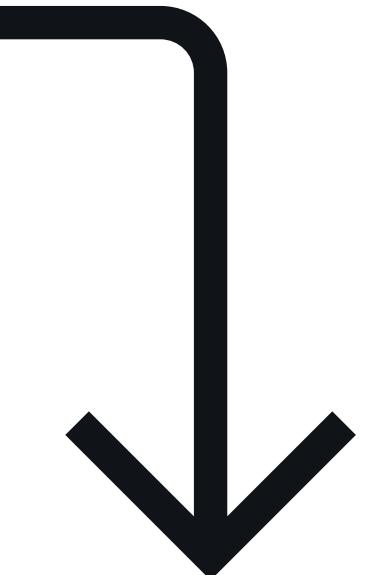
# 새 단어 추가
new_words = [
    '양떼목장', '수박', '서울근교', '풀빌라', '아난티', '아침고요수목원', '오색별빛정원', '카라반캠핑카이베이호텔', '아르데',
    '챔피언스필드', '김대중컨벤션센터', '궁전제과', '남양쟁', '국립문화전당', '민주화운동',
    '광안리', '요트투어', '해변열차', '조개구이', '부산타워', '밀락더마켓',
    '이순신광장', '루지데마파크', '풀빌라', '라마다호텔', '해상케이블카', '생선구이', '갈치조림', '라테라스', '거북선축제',
    '신화월드', '제주공항', '에이코스', '동화마을', '신라호텔', '함덕해수욕장',
    '스카이워크', '제이드가든', '풀물시장', '애견동반', '잭슨나인스호텔', '오월학교', '해피초원목장', '동궁과월지'
]

data += '\n' + '\n'.join(new_words) + '\n'

# 사전 저장
with open(file_path, 'w') as f:
    f.write(data)

# 저장된 데이터 출력
print(data)
```

단어 토큰화 진행시 복합 명사가 분리되는 현상으로
분석에 차질이 생기는 문제를 해결할 수 있는
방법



✓ konlpy에 새 단
어들 추가

분리된 단어들에 대해 임의로 단어들의 리스트를 만들어
konlpy에 추가해주는 작업을 추가로 실시함 → 워드클라우
드 및 lda모델 분석의 수월함

텍스트 데이터 전처리

```
# KoNLPy의 형태소 분석기 객체 생성
okt = Okt()

# 이모티콘 및 반복 문자열 제거를 위한 정규식 패턴
emoji_pattern = re.compile([
    u"\U0001F600-\U0001F64F", # emoticons
    u"\U0001F300-\U0001F5FF", # symbols & pictographs
    u"\U0001F680-\U0001F6FF", # transport & map symbols
    u"\U0001F1E0-\U0001F1FF", # flags (iOS)
    "+", flags=re.UNICODE)
repeat_pattern = re.compile(r'(.+)\1{1,}', re.IGNORECASE)

# 불용어 파일 읽기
with open('/Users/park/Desktop/text_data/project/stopwords.txt', 'r', encoding='utf-8') as f:
    stopwords = f.readlines()
stopwords = [x.strip() for x in stopwords]

def preprocess(text):
    # 이모티콘 제거
    text = emoji_pattern.sub('', text)

    # 반복되는 문자열 제거
    text = repeat_pattern.sub(r'\1', text)

    # 특수문자 제거
    text = re.sub(r'^\w\s', '', text)

    # 형태소 분석 및 품사 태깅
    pos_tags = okt.pos(text, norm=True, stem=True)

    # 명사, 동사, 형용사만 추출하여 불용어 제거
    tokens = [word[0] for word in pos_tags if word[1] in ['Noun'] and word[0] not in stopwords]

    return tokens

# 'content' 열이 존재하는 데이터프레임에 대해 전처리 함수를 적용하여 새로운 열에 저장
df_travel['tokens'] = df_travel['content'].apply(preprocess)

# 결과를 CSV 파일로 저장
df_travel.to_csv('text_data_project_tokens_2.csv', index=False, encoding='utf-8')
```

✓ 이모티콘 및 반복 문자열 제거

이모티콘, 반복문자열, 특수문자에 대한 제거 함수를 호출하고 형태소 분석 및 품사 태깅

✓ 불용어 제거 및 저장

명사,동사,형용사만 추출해 불용어 제거 후 데이터프레임에 tokens라는 새로운 열 저장

✓ csv 파일 저장

작업이 완료된 결과물을 csv파일에 저장해 공유 편리하게함

WORDCLOUD

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# 폰트 설정
plt.rcParams['font.family'] = 'AppleGothic'
plt.rcParams['axes.unicode_minus'] = False

# 쿼리별로 텍스트를 모아서 하나의 텍스트로 만들어 워드클라우드 생성
queries = df_travel['query'].unique()

for query in queries:
    # 해당 쿼리에 해당하는 행만 추출하여 텍스트 합치기
    tokens = df_travel[df_travel['query'] == query]['tokens']
    text = ' '.join(word for sublist in tokens for word in sublist if len(word) > 1)

    # 쿼리명에서 '여행' 등의 공통 단어 제거
    for word in query.split('여행'):
        text = text.replace(word, '')

    # '여행' 단어 제거
    text = text.replace('여행', '')

# 워드클라우드 생성
wordcloud = WordCloud(font_path='/System/Library/Fonts/Supplemental/AppleGothic.ttf', background_color='white', width=1000, height=600).generate(text)

# 워드클라우드 시각화
plt.figure(figsize=(12, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title(f'Word Cloud for Query: {query}')
plt.axis('off')
plt.show()
```

8개 키워드에 대한 워드클라우드 생성

반복문을 사용해 각 키워드에 관한 워드클라우드 결과물을 한번에 확인할 수 있도록 코드 생성

여행이 들어간 단어 제거

여행이라는 단어가 들어갈 시 쿼리명이 워드클라우드에 너무 많이 노출→여행이 포함된 단어를 제거하고 쿼리명을 제목으로 설정해 쿼리별 워드클라우드를 알아볼 수 있게 함

한글자 단어 제거

한글자단어들이 분석에 크게 의미있지 않다고 판단해 한글자 이하의 단어들을 제거하고 워드클라우드 생성

WORDCLOUD



해운대, 광안리, 해동용궁
사 등 부산 유명 관광지에
대한 단어가 크게 노출

▶ 여행지 추천이라는 목표달성, 높은 성능



오동도, 향일암, 이순신광장, 바다 등의 여수 관광지 단어 크게 노출

WORDCLOUD



황리단길, 첨성대, 불국사 등 경주의 유명 유적지와 거리에 대한 단어들이 매우 크게 노출되는 것을 확인



주문진, 중앙시장, 경포
대, 강문해변 등 강릉
의 주요 관광지인 바
다에 관련된 단어들과
지역단어들이 눈에 띠
게 노출되어 있음을 확
인

경주의 여행지 추천이라는 목표 달성, 높은 성능

강릉의 여행지 추천 목표달성, 높은 성능

WORDCLOUD



제주도의 여러 관광명
소들을 대표하지 못하
고 주로 크게 의미없
는 단어들이 노출되어
있음



제주의 여행지 추천이라는 목표달성을 어려움
→ LDA모델 분석으로 원인 파악 필요



아침고요수목원, 뽀띠
프랑스, 남이섬 등 특정
장소에 대한 단어들이
크게 노출되어 있음을
확인



가평의 여행지 추천 목표달성가능, 높은 성능

WORDCLOUD



펭귄마을, 무등산, 송정역시장 등의 단어가
큰 편의 단어에 속해 있는 것을 확인



광주의 여행지 추천이라는 목표달성을 가능,
준수한 성능



춘천을 대표하는 닭갈비에 대한 키워드가 많이 노출되어 있지만 여행지에 관한 키워드를 쉽게 파악하기는 어려움



춘천 여행지추천이라는 목표달성이려움,
낮은 성능 → LDA모델 분석으로 원인파악

워드클라우드 성능분석

전반적으로 높은 성능이 나왔다고 할 수 있다. 그 이유는 결과를 확인해보면서 추가적인 전처리를 수행한 것에 있다고 생각한다. 불용어들의 제거를 확실히 했고 토큰화하는 과정에서 분리된 명사들을 사전에 추가로 정의해주는 작업을 실시하면서 각 지역의 주요단어들이 워드클라우드에 잘 나타날 수 있게 작업을 진행한 것이 주요했다고 볼 수 있다.

반면 상대적으로 낮은 성능이 나온 결과물들이 있는데 특정 단어들의 빈도가 매우 높게 나온 것이 낮은 퍼포먼스의 이유라고 분석했다. 이에 대한 해결책으로는 너무 자주 나오는 단어들을 제거하거나 단어의 빈도수를 조절하여 성능을 올리는 방법을 사용하면 더 나은 결과물을 도출할 수 있을 것 같다.

LDA MODEL

```
import pandas as pd
import gensim
from gensim import corpora
import pyLDAvis
import pyLDAvis.gensim_models
from IPython.display import display

# CSV 파일을 읽어옵니다.
df_travel_tokens = pd.read_csv('text_data_project_tokens_2.csv')

# 토큰이 담긴 컬럼을 리스트로 변환합니다.
df_travel_tokens['tokens'] = df_travel_tokens['tokens'].apply(eval)

# 한글자 이하 단어를 제거하는 함수
def remove_short_tokens(tokens):
    return [token for token in tokens if len(token) > 1]

# 토큰 리스트에서 한글자 이하 단어를 제거합니다.
df_travel_tokens['tokens'] = df_travel_tokens['tokens'].apply(remove_short_tokens)

# 쿼리별로 데이터를 그룹화합니다.
grouped = df_travel_tokens.groupby('query')

# 각 쿼리에 대해 LDA 모델을 훈련하고 시각화합니다.
for query, group in grouped:
    print(f"Processing query: {query}")

    # 토큰 리스트 생성
    tokens = group['tokens'].tolist()

    # 단어 사전을 생성합니다.
    dictionary = corpora.Dictionary(tokens)

    # 말뭉치를 생성합니다.
    corpus = [dictionary.doc2bow(text) for text in tokens]

    # LDA 모델을 훈련합니다.
    lda_model = gensim.models.LdaModel(corpus, num_topics=10, id2word=dictionary, passes=50)

    # 토픽 모델링 결과를 시각화합니다.
    lda_vis = pyLDAvis.gensim_models.prepare(lda_model, corpus, dictionary)

    # 시각화를 주피터 노트북에 표시합니다.
    display(pyLDAvis.display(lda_vis))
```



한글자단어제거

워드클라우드 생성 때와 같은 이유로 한글자 단어를 제거



여행이 들어간 단어 제거하지 않음

여행이 들어간 단어를 제거하면 한토픽의 비중이 매우 커지고 나머지 토픽이 매우 작은 원으로 생성되는 문제 발생 → 여행이 들어간 단어를 제거함으로써 특정 단어의 빈도가 커지면서 한 토픽이 지배적이 되었을 것이라고 추측



각 쿼리에 대해 lda모델 생성

반복문을 통해 워드클라우드 생성시와 마찬가지로 각 쿼리 당 lda모델이 생성될 수 있도록 함

LDA MODEL - 가평여행

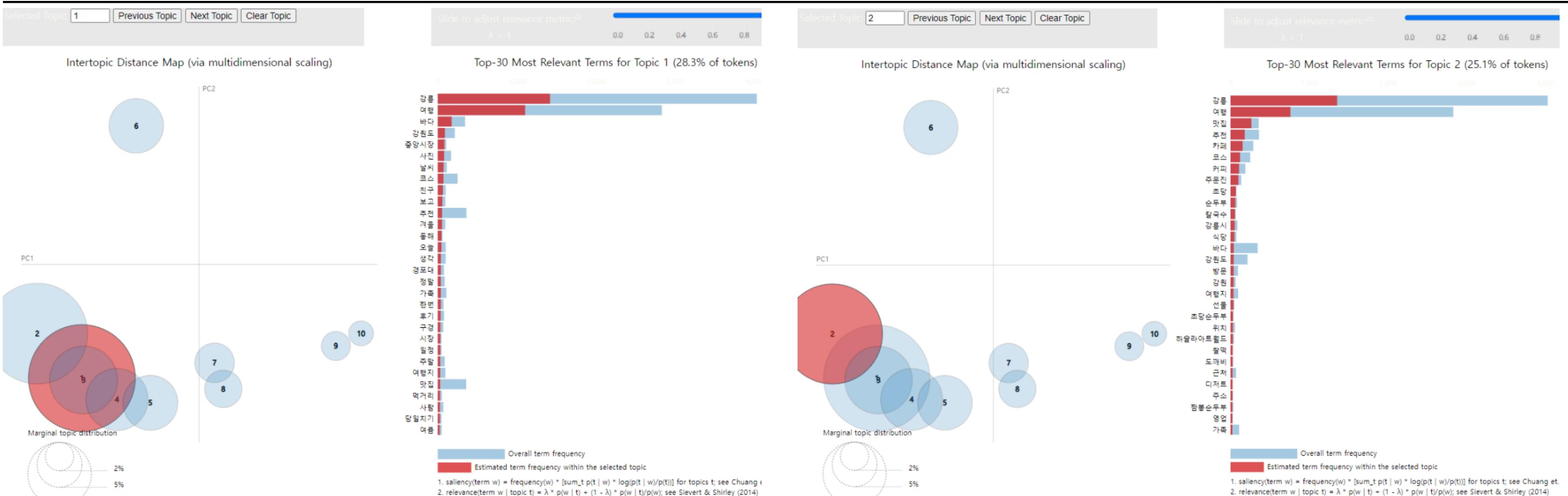
**토픽 1**

주로 음식에 관련된 내용이 많이 있는 것으로 보아 토픽 1은 음식일 것으로 추측

**토픽 2**

토픽 2는 가족, 친구, 반려동물과 함께 하기 좋은 활동에 관한 것으로 추측

LDA MODEL - 강릉여행



토픽 1

토픽 1은 강릉의 주요 관광지에 대한 토픽일 것이
라고 추측



토픽 2

음식에 관련된 단어들이 많이 있는 것으로 보아 토픽2는 음식에 관한 것으로 추측

LDA MODEL - 경주여행



토픽 1

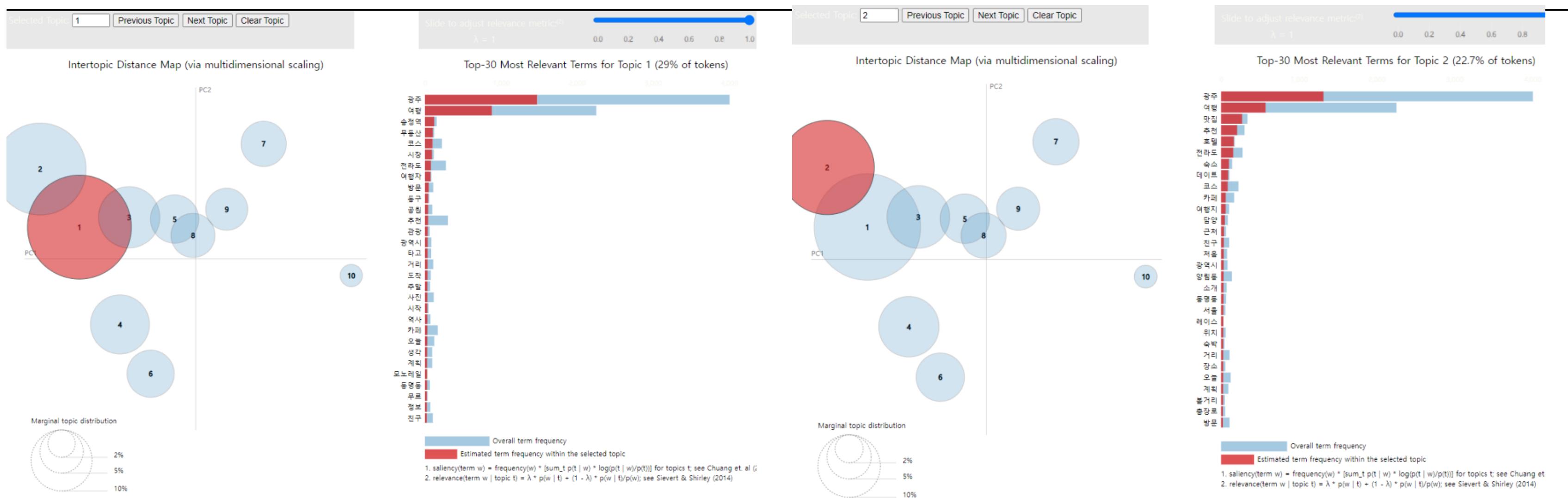
토픽 1은 기억, 구경, 수학여행 등의 전반적인 개인의 경험이나 추억에 기반한 단어들이 많이 나타남
→ 과거 학창시절 수학여행지로 많이 선정되었던 것이 영향을 미쳤을 것이라 추측



토픽 2

토픽 2는 실제 관광지명이 많이 등장하며 실제로 방문한 관광지에 대한 내용이 많이 나타나는 것으로 확인됨

LDA MODEL - 광주여행

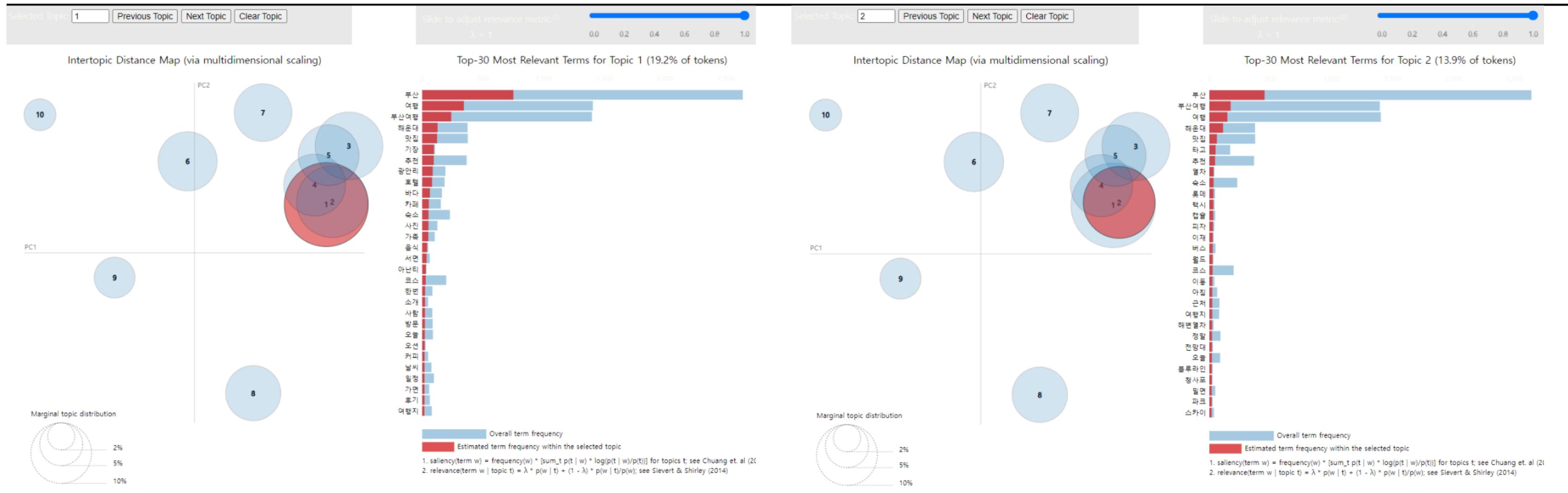
**토픽 1**

토픽 1은 추천 관광지에 대한 토픽일 것이라 추측

**토픽 2**

토픽 2는 주로 연인들의 관심사인 데이트코스, 호텔, 카페와 같은 단어들이 많이 나타나는 것으로 확인
→연인들의 데이트코스에 대한 토픽으로 추측

LDA MODEL - 부산여행

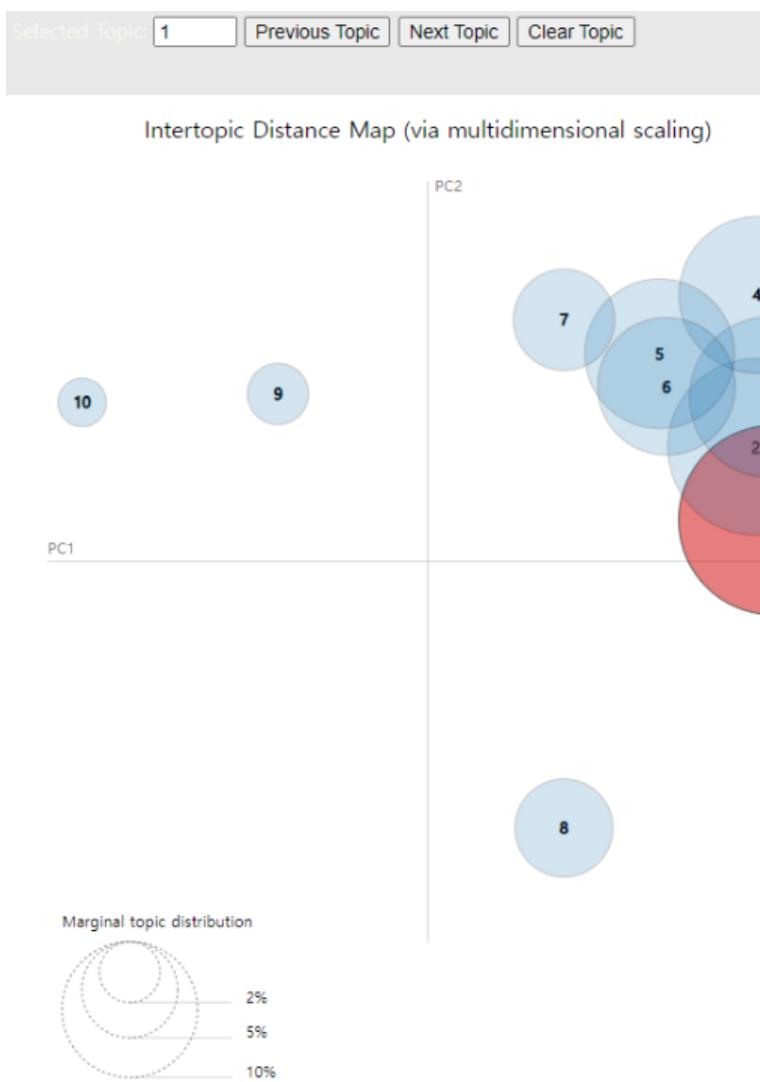
**토픽 1**

토픽 1은 여행의 전반적인 일정에 관한 내용이나 타난 것을 확인

**토픽 2**

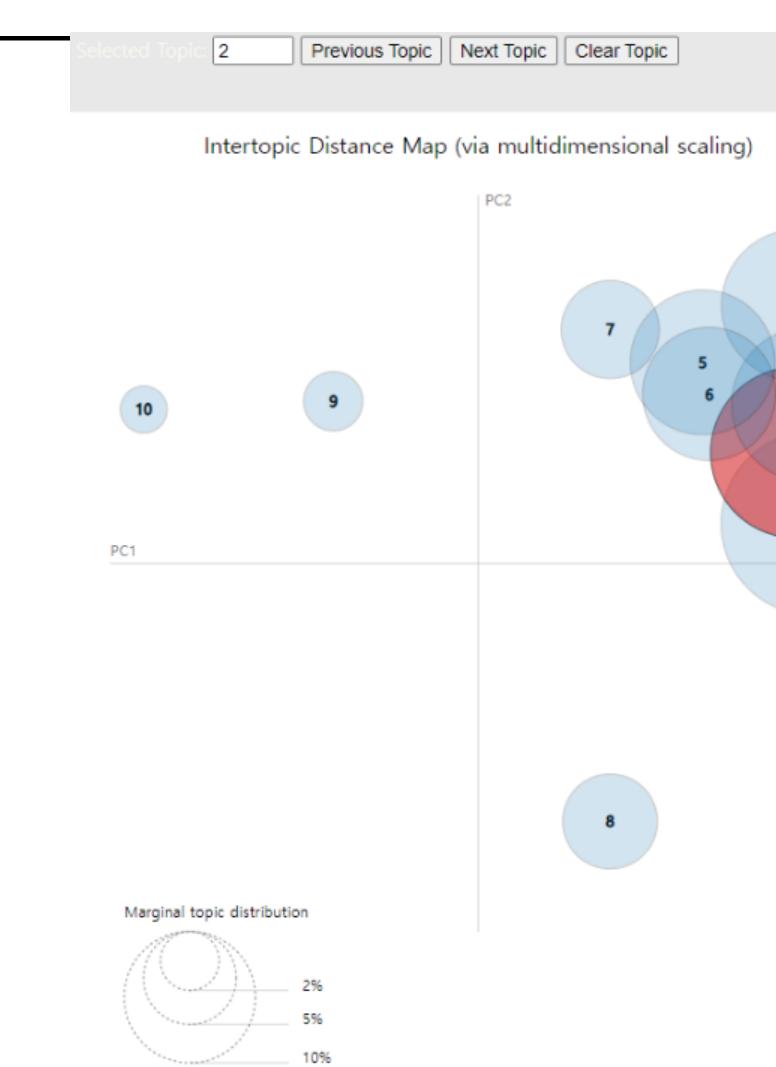
토픽 2는 구체적인 컨텐츠명과 지명들이 나타나 있는 것을 확인

LDA MODEL - 여수여행



토픽 1

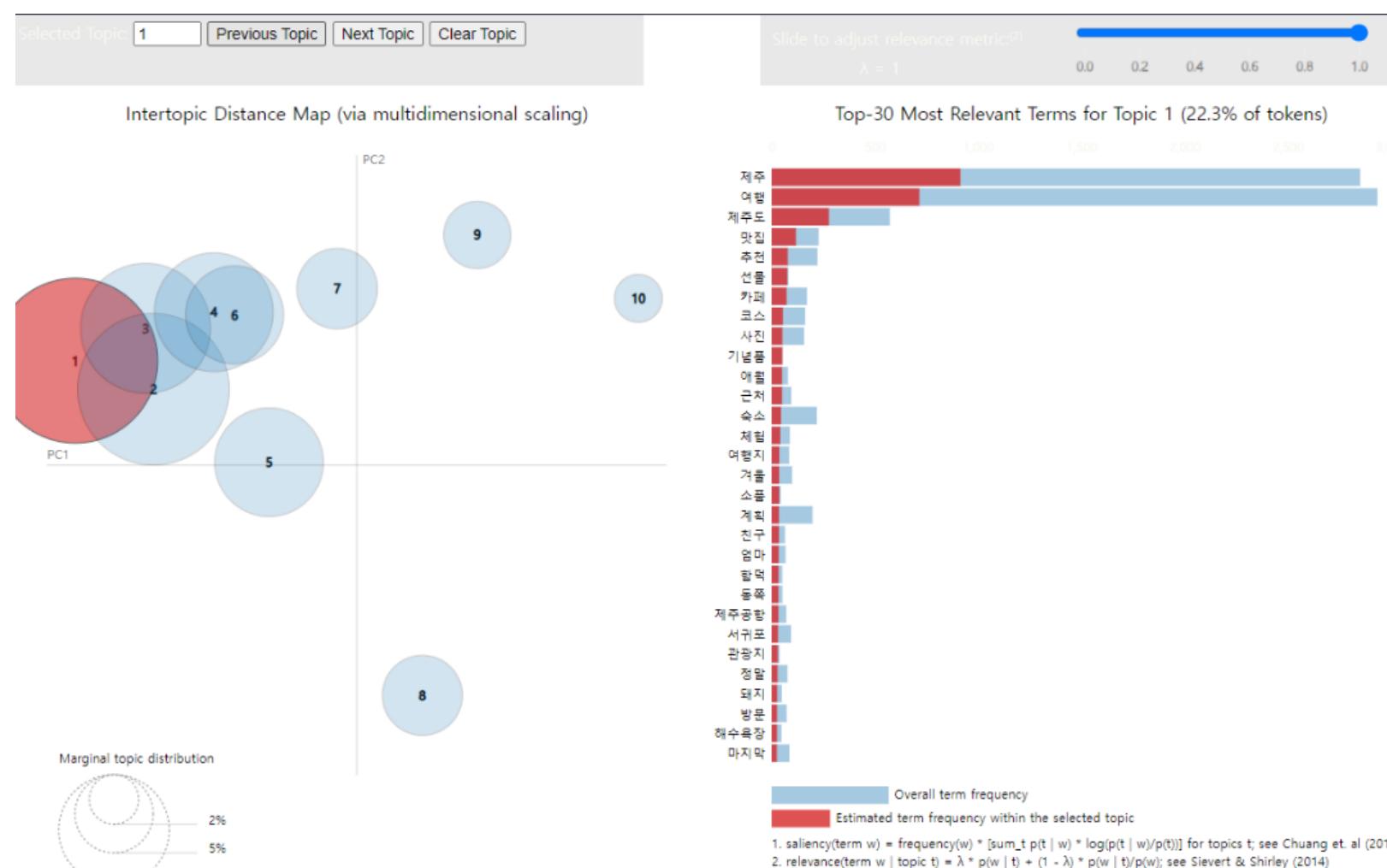
토픽 1은 주로 리조트, 호텔, 풀빌라같은 단어들이 많이 보이는 걸로 보아 숙소에 관한 토픽이라고 추



토픽 2

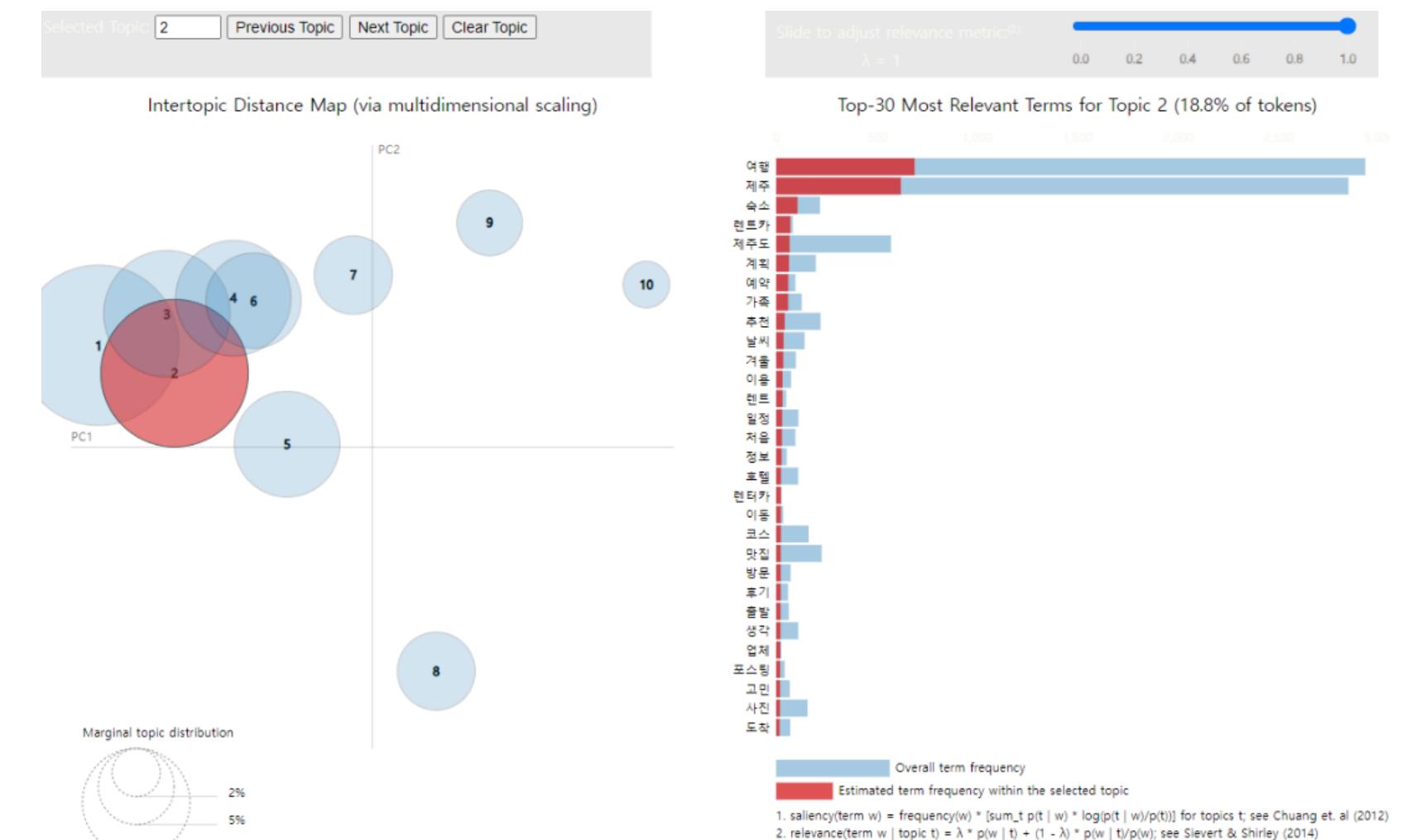
토픽2는 전반적인 여행의 계획에 관한 내용일 것이라고 추측

LDA MODEL - 제주여행



토픽 1

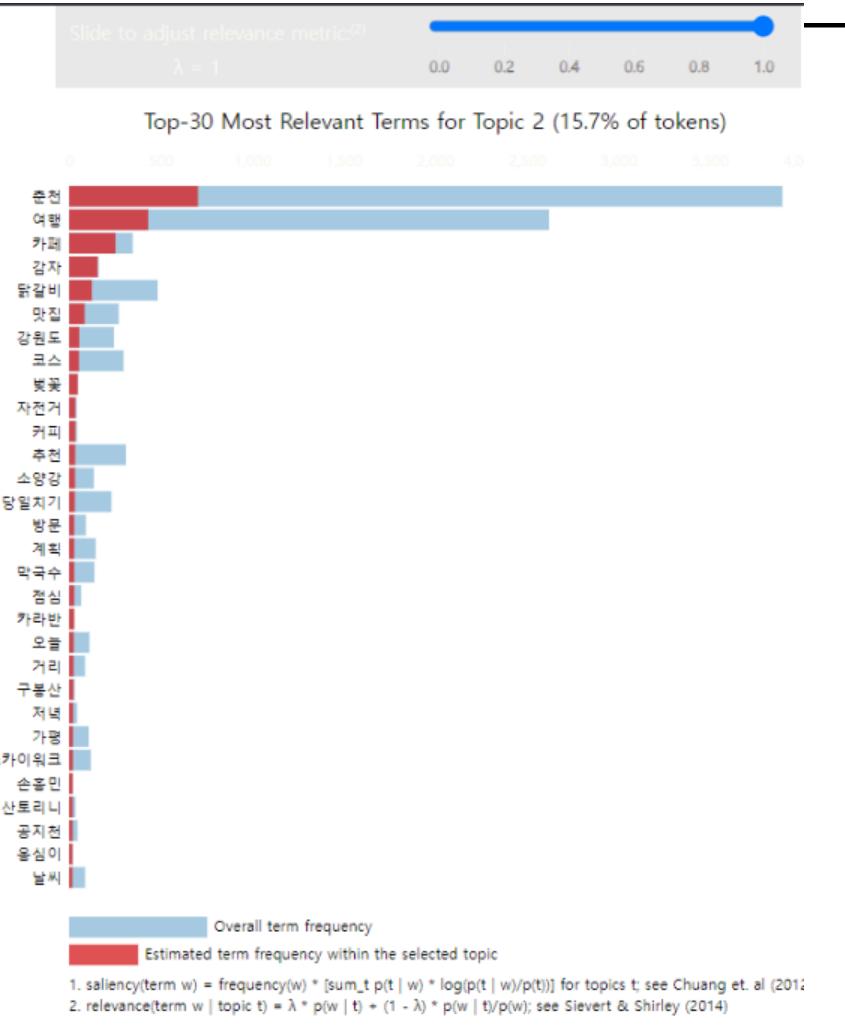
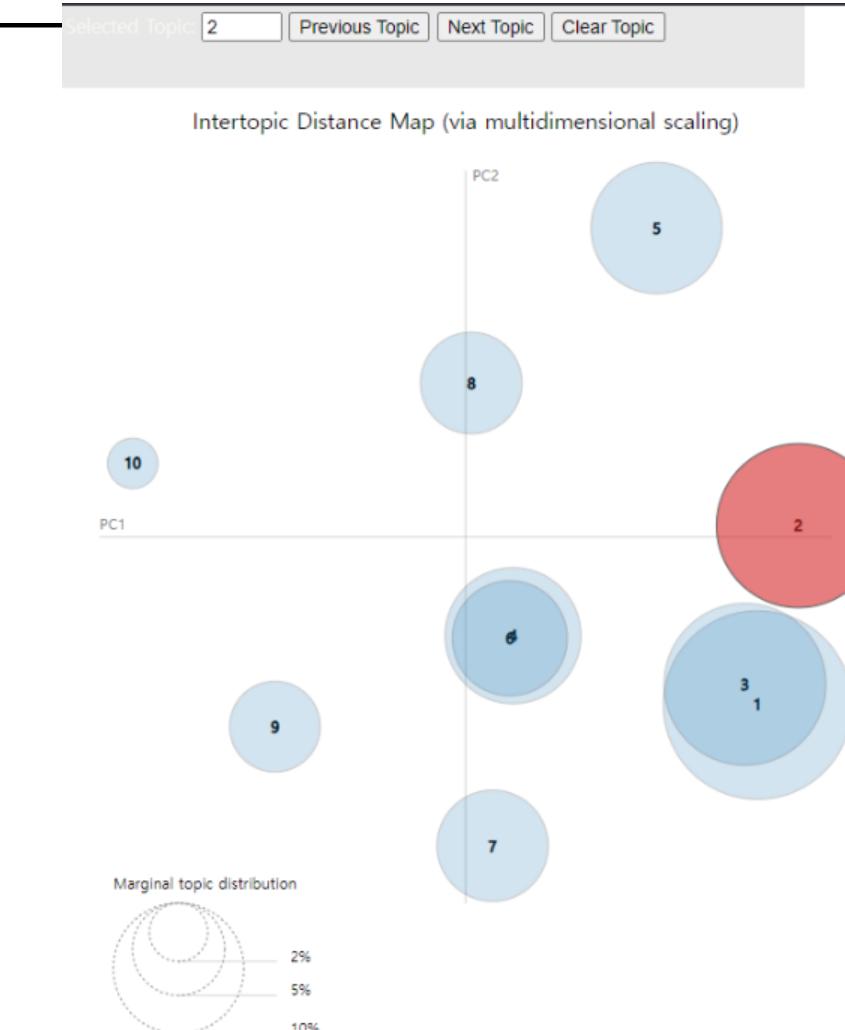
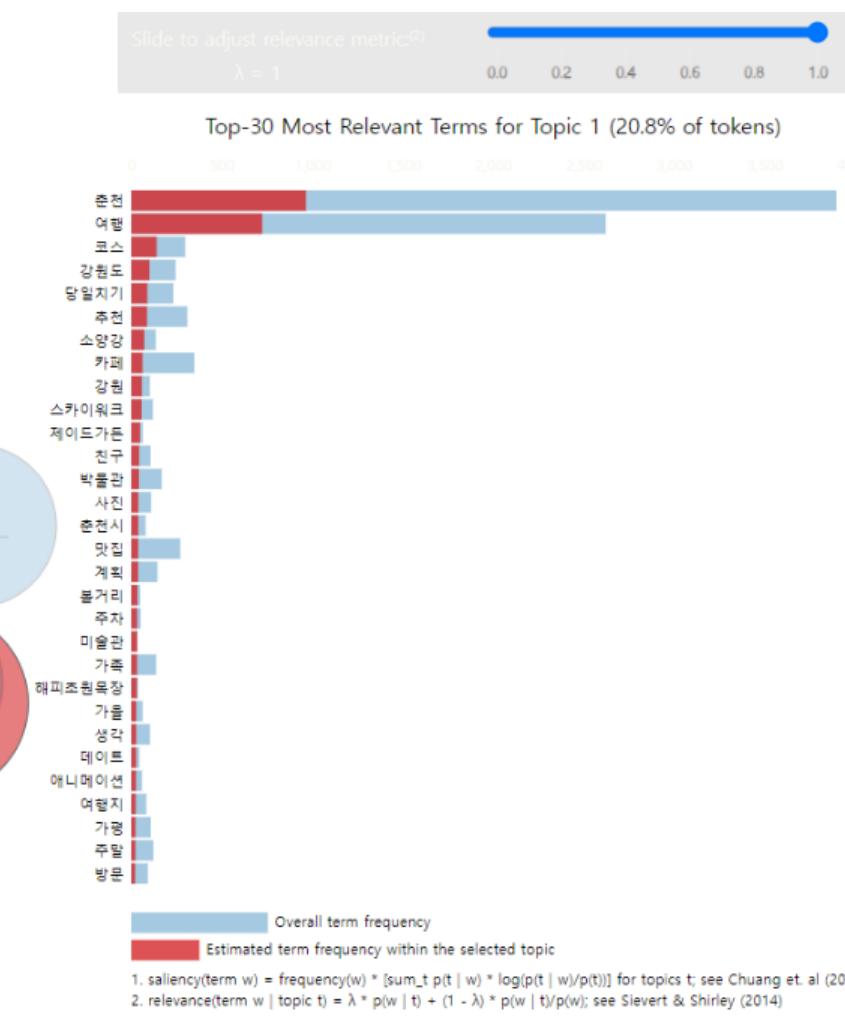
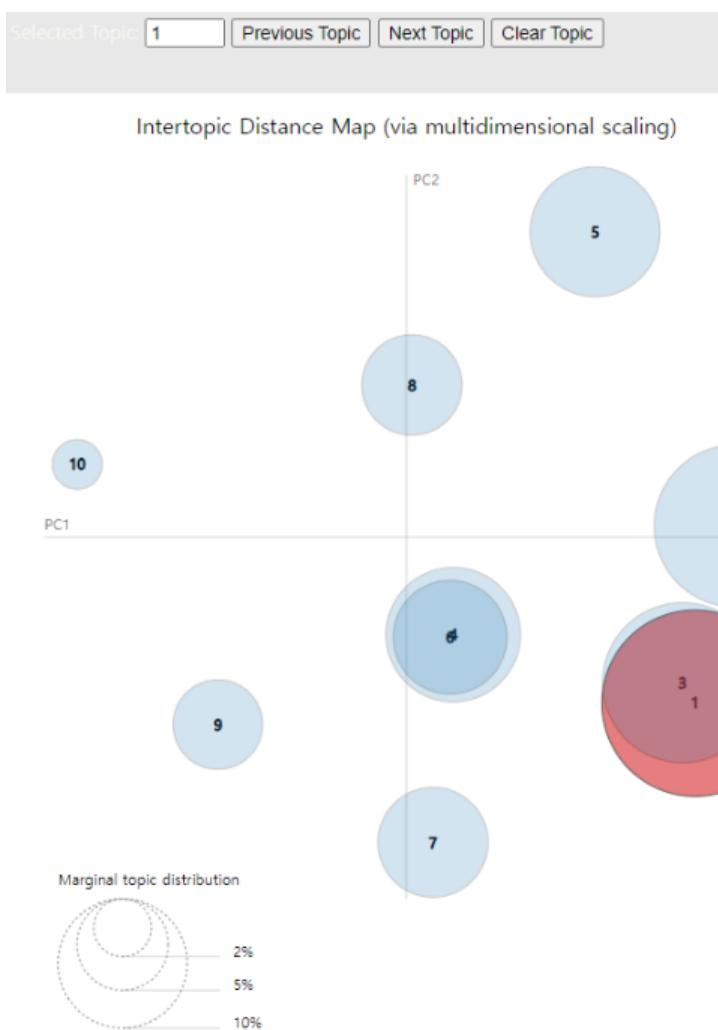
토픽 1은 주로 여행지, 기념품과 같은 내용으로 구성되어 있음



토픽 2

토픽 2는 렌트에 관한 단어들이 많이 보임→제주도에서 많이 하는 렌트에 대한 언급이 많은 것을 확인

LDA MODEL - 춘천여행



토픽 1

토픽 1은 여러 관광지명이 자세히 나와있는 것으로 보아 춘천의 주요관광지에 대한 토픽으로 추측



토픽 2

토픽2는 닭갈비, 막국수, 손흥민등의 단어가 나온 것으로 보아 춘천하면 떠오르는 것들에 대한 토픽이지 않을까 추측

LDA MODEL 성능분석

LDA모델에서도 워드클라우드와 동일하게 추가적인 전처리의 영향으로 전반적으로 높은 성능이 나왔던 것 같다. 상대적으로 분석에 큰 의미가 없는 단어들을 배제하고 여행지에 관련된 단어들을 추린 상태에서 모델링을 진행해 토픽을 추론하기 용이했다.

하지만 토픽을 추론하기 어려운 모델들도 물론 있었는데 이는 전처리를 지속적으로 진행했음에도 완벽하게 할 수는 없기 때문에 추가적인 전처리가 더 진행되었다면 더 높은 성능이 나왔을 것이라고 생각하고 토픽의 수를 조절하고 하이퍼파라미터를 조정하는 등 여러방면에서 모델 수정을 하면서 결과를 확인해봤다면 더 좋은 결과를 이끌어낼 수 있지 않았나라는 생각이 들었다.

Thank you!

감사합니다.