과제 및 실습 1

(3) 실습문제 1

[1.4 실습 및 과제]

해당 TEXT 파일을 다른 실제 예제 문장 10개를 가져와서 **tf** 벡터와 **tfidf** 벡터로 바꾼 후 코사인 유사도를 비교하여라

```
TEXT = ['example1',
        'example2'
]

## Your code here
```

- **제출물:**
- 1) 작성 코드 스크린샷
- 2) 작성 코드에 대한 설명
- 3) 아웃풋 결과물 스크린샷

과제 및 실습 1

(3) 실습문제 1

```
TEXT = ['I don't know how william is gonna feel about this guy being in her apartment',
        'She has such an awesome sense of humor and personality!! Love her and Caleb together!! I'm not sure William would be happy about this tho…',
        'Is it me or was the energy/chemistry between these two insanely good!?, made the interview even more interesting to watch, love it!!! Awesome apart
        'The fact that she looked like that in middle school is probably why she has a great personality',
        'I love her personality she is so witty and genuine. So glad you did a longer video with her',
        'She's so well spoken, no doubt she makes millions',
        'This is one of the coolest girl you've ever featured. Her home is dope. She's got class and great taste. Best part of all relatable and unpretentio
        'scientist, model, entrepreneur with a dr strange window and 3 aussies. she''s living my dream life',
        'After a terrible 2022, shell-dazed monetary supporters have incidents to recuperate and a ton to consider, as an extension report and a heap of var
        'Her energy is great! Depending on what her subjects are, I would totally listen to a podcast with her! She''s just a boss! The guests usually annoy
]
```

```
#1 tf벡터 변환 후 코사인 유사도 계산
tf_vectorizer = CountVectorizer(min_df=0.2,max_df=0.9,ngram_range=(1,1))
tf_features = tf_vectorizer.fit_transform(TEXT)

feature_names = tf_vectorizer.get_feature_names_out()
#print(len(feature_names))

features = np.array(tf_features.todense())
#print(features.shape)

tf_cos_sims = []
for i in range(10):
    for j in range(i):
        sim = cos_sim(features[i],features[j])
        tf_cos_sims.append((f'문서{i}와 문서{j}',sim))

tf_cos_sims.sort(key=lambda x:x[1], reverse=True)

for comp in tf_cos_sims:
    print(comp)
```

1. 유튜브 댓글에서 발췌한 문장 10개를 TEXT에 저장해 두기
2. CounterVectorizer의 파라미터 min_df와 max_df를 조절하기 위해 feature_name의 length(고려할 단어들의 개수)를 확인하면서 진행
3. Tf_features.todense()를 통해 벡터 계산
4. Tf_cos_sims에는 10개의 문장들에서 모든 쌍 비교를 한 결과(두 문장의 인덱스 번호, 두 문장 tf벡터의 cosine유사도 값)를 저장해 두고 마지막에 유사도가 높은 순서대로 그 값과 해당 문서 쌍을 출력.

output

```
('문서6와 문서1', 0.5000000000000001)
('문서5와 문서3', 0.47809144373375745)
('문서8와 문서2', 0.47410465593076045)
('문서6와 문서4', 0.47140452079103173)
('문서9와 문서4', 0.4691574316284183)
('문서9와 문서2', 0.4626519455729923)
('문서5와 문서4', 0.4472135954999579)
('문서9와 문서6', 0.4423258684646915)
('문서8와 문서6', 0.4297722474802028)
('문서4와 문서1', 0.41247895569215276)
('문서6와 문서3', 0.37796447300922725)
('문서8와 문서1', 0.37605071654517747)
('문서9와 문서3', 0.33436692754521175)
('문서1와 문서0', 0.3333333333333333)
('문서6와 문서0', 0.3333333333333333)
('문서9와 문서8', 0.33267391956523024)
('문서9와 문서1', 0.2948839123097943)
('문서9와 문서0', 0.29488391230979427)
('문서4와 문서3', 0.2672612419124244)
('문서4와 문서0', 0.2651650429449553)
('문서6와 문서2', 0.2614881801842454)
('문서3와 문서1', 0.25197631533948484)
('문서7와 문서4', 0.25)
('문서8와 문서3', 0.24365796154926905)
('문서7와 문서1', 0.23570226039551587)
('문서7와 문서6', 0.23570226039551587)
('문서8와 문서7', 0.2279211529192759)
('문서3와 문서2', 0.22237479499833038)
('문서5와 문서1', 0.21081851067789195)
('문서6와 문서5', 0.21081851067789195)
('문서9와 문서7', 0.20851441405707477)
('문서2와 문서0', 0.19611613513818402)
('문서3와 문서0', 0.1889822365046136)
('문서9와 문서5', 0.18650096164806276)
('문서4와 문서2', 0.1386750490563073)
('문서2와 문서1', 0.1307440900921227)
('문서8와 문서4', 0.11396057645963795)
('문서8와 문서0', 0.04029114820126901)
('문서5와 문서0', 0.0)
('문서5와 문서2', 0.0)
('문서7와 문서0', 0.0)
('문서7와 문서2', 0.0)
('문서7와 문서3', 0.0)
('문서7와 문서5', 0.0)
('문서8와 문서5', 0.0)
```

과제 및 실습 1

(3) 실습문제 1

```
tfidf_vectorizer = TfidfVectorizer(min_df=0.2,max_df=0.9,ngram_range=(1,1))
tfidf_features = tfidf_vectorizer.fit_transform(TEXT)

tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
#print(len(tfidf_feature_names))

tfidf_features = np.array(tfidf_features.todense())
#print(tfidf_features.shape)

tfidf_cos_sims = []
for i in range(10):
    for j in range(i):
        sim = cos_sim(tfidf_features[i],tfidf_features[j])
        tfidf_cos_sims.append((f'문서{i}와 문서{j}',sim))

tfidf_cos_sims.sort(key=lambda x:x[1], reverse=True)

for comp in tfidf_cos_sims:
    print(comp)
```

Tf-idf 벡터를 만들기 위해 `TfidfVectorizer` 함수를 사용. 파라미터는 tf에서 썼던 것과 동일하게 진행했고 마찬가지로 45가지 모든 쌍대 비교 결과를 출력

```
('문서5와 문서4', 0.511349813707771)
('문서9와 문서2', 0.4318546075697167)
('문서6와 문서1', 0.42837309217756303)
('문서8와 문서2', 0.4137002880169129)
('문서8와 문서6', 0.39039790155928145)
('문서6와 문서4', 0.3766928159728779)
('문서8와 문서1', 0.3611394067147043)
('문서9와 문서6', 0.36019141059339094)
('문서9와 문서4', 0.3508719105761806)
('문서1와 문서0', 0.3412223000147285)
('문서5와 문서3', 0.31292686561857347)
('문서4와 문서1', 0.3091686460203514)
('문서9와 문서8', 0.3012921575451338)
('문서6와 문서3', 0.2776228092354855)
('문서9와 문서1', 0.26042070109625287)
('문서9와 문서7', 0.25971429715483174)
('문서8와 문서3', 0.251376270599634)
('문서9와 문서3', 0.24634535062627344)
('문서6와 문서0', 0.2452365188826978)
('문서3와 문서1', 0.21910668594579089)
('문서7와 문서4', 0.20461194714851894)
('문서9와 문서0', 0.19163075570987165)
('문서8와 문서7', 0.18255782766330683)
('문서4와 문서3', 0.18255374120775644)
('문서6와 문서2', 0.17399187088351856)
('문서2와 문서0', 0.17252804484597636)
('문서3와 문서0', 0.17139329676450907)
('문서7와 문서6', 0.1668611778668887)
('문서4와 문서0', 0.15694328100797852)
('문서6와 문서5', 0.15268678375616201)
('문서2와 문서1', 0.15192256513206176)
('문서7와 문서1', 0.1463617419364331)
('문서3와 문서2', 0.13820390077905745)
('문서5와 문서1', 0.13392871803320705)
('문서9와 문서5', 0.131372379775806)
('문서4와 문서2', 0.10905269980242338)
('문서8와 문서4', 0.09695828978553762)
('문서8와 문서0', 0.05181663940118292)
('문서5와 문서0', 0.0)
('문서5와 문서2', 0.0)
('문서7와 문서0', 0.0)
('문서7와 문서2', 0.0)
('문서7와 문서3', 0.0)
('문서7와 문서5', 0.0)
('문서8와 문서5', 0.0)
```

과제 및 실습 1

(3) 실습문제 2

[실습 및 과제 2]

위 documents_filtered에 들어 있는 문서 4개를 1.3과 같이 tf-idf vectorizer를 통해서 벡터화하고 해당 결과를 1.3의 데이터프레임처럼 만들어라.

+ Code + Markdown

```
tfidf_vectorizer = TfidfVectorizer(min_df=2, max_df=0.75, ngram_range=(1,1))
tfidf_features = tfidf_vectorizer.fit_transform(documents_filtered)

tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
#print(len(tfidf_feature_names))

tfidf_features = np.array(tfidf_features.todense())
print(tfidf_features.shape)

df = pd.DataFrame(data=tfidf_features, columns=tfidf_feature_names)
df
```

1. TfidfVectorizer 함수를 사용
2. Documents_filtered 에 들어 있는 문서 4가지에 대한 tf-idf 벡터의 형태가 4x27 임을 확인.(고려된 word의 종류가 27가지)
3. Tfidf_features.todense()의 결과를 np.array를 이용해 넘파이 배열 형태로 변환 후에 이를 pandas dataframe으로 만들어 주었다. 컬럼은 tfidf_feature_names 즉 워드들

[111...

	apple	back	battery	bottom	case	cases	center	courts	day	degrees	...	max	part	phone	silicone	thursday	today	way	wednesday	xs	years
0	0.169187	0.042297	0.338374	0.084594	0.676748	0.296077	0.042297	0.000000	0.042297	0.000000	...	0.126890	0.000000	0.422968	0.042297	0.042297	0.000000	0.034243	0.000000	0.169187	0.034243
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.817071	0.204268	0.000000	0.000000	...	0.000000	0.204268	0.000000	0.000000	0.204268	0.204268	0.165372	0.204268	0.000000	0.165372
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.227761	0.000000	0.455521	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.368784	0.227761	0.000000	0.184392
3	0.056196	0.168588	0.505765	0.112392	0.280981	0.056196	0.000000	0.000000	0.112392	0.056196	...	0.112392	0.056196	0.168588	0.056196	0.000000	0.056196	0.000000	0.000000	0.280981	0.000000

4 rows × 27 columns

과제 및 실습 1

(3) 실습문제 3

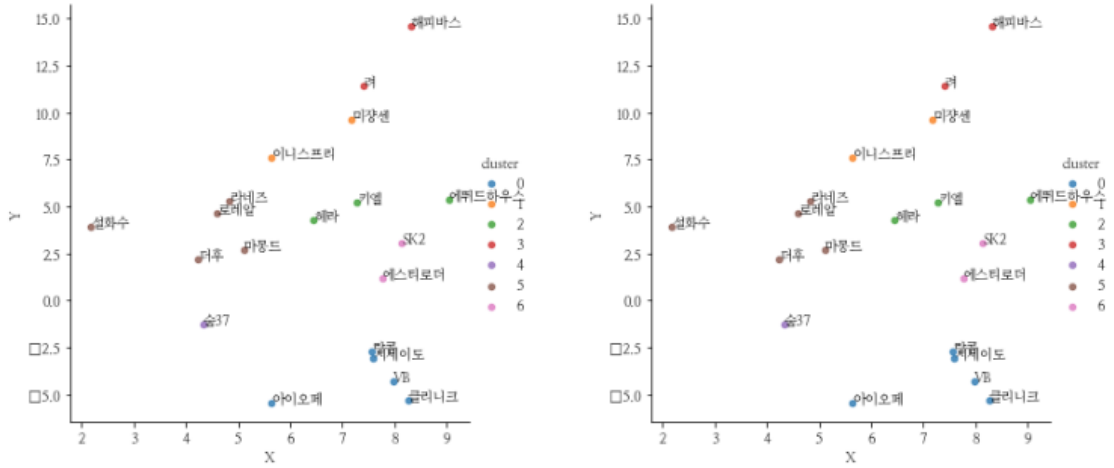
[실습 및 과제 3]

실습에서 학습한 word2vec을 모델을 기반으로 본인이 원하는 단어 5개를 선정하고, 이에 대해 most_similar를 통해 가장 가까운 단어 10개를 포함하여 시각화한 결과물을 보여라.

총 50개의 단어에 대한 벡터를 시각화하면 된다.

Hint1: t-nse를 이용한 시각화

Hint2: [예시]



- **제출물:**
- 1) 작성 코드 스크린샷
- 2) 작성 코드에 대한 설명
- 3) 아웃풋 결과물 스크린샷

과제 및 실습 1

(3) 실습문제 3

```
## your code here
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.manifold import TSNE

selected_words = ['좀비', '로맨스', '연출', '지루', '공포']

allwords_50 = []
word_vectors_list = []
associated_words = []

for word in selected_words:
    similar_words = model_sg_n10.wv.most_similar(word, topn=10)
    for sw, _ in similar_words:
        allwords_50.append(sw)
    similar_word_vectors = np.array([model_sg_n10.wv[word] for word, _ in similar_words])
    word_vectors_list.extend(similar_word_vectors)
    associated_words.extend([word]*10)

word_vectors_list = np.array(word_vectors_list)
print(word_vectors_list.shape)
#word_vectors_list = word_vectors_list.reshape(50,100)
#print(word_vectors_list.shape)
#지금 sim_words_list는 3차원 array(5,10,100) -> 50 x 100으로 flatten?
```

1. 차원축소를 위해 sklearn.manifold의 TSNE를 import

주요 저장정보

1) Selected_words에 선택한 단어 5개를 저장
2) Allwords_50은 단어 5개에서 각 단어마다 최고 유사 단어 10개씩 즉 총 50개의 단어를 저장
3) Word_vectors_list 워드 임베딩 벡터 결과를 저장
4) Associated_words 는 추후 그래프에서 hue값을 리스트로 넘겨주기 위해 각 기준 단어를 10번 반복하는 형태로 저장

2. wv.most_simila를 이용해서 해당 단어마다 가장 유사한 단어 10개의 목록(단어-유사도 쌍들)을 얻는다.

3. Wv 에서 10개 단어들의 임베딩 값을 얻고 이를 np.array로 변환 후 word_vector_list에 추가한다. 이때 넘파이 array를 3차원이 아닌 2차원으로 유지하기 위해 append()가 아닌 extend()를 사용

과제 및 실습 1

(3) 실습문제 3

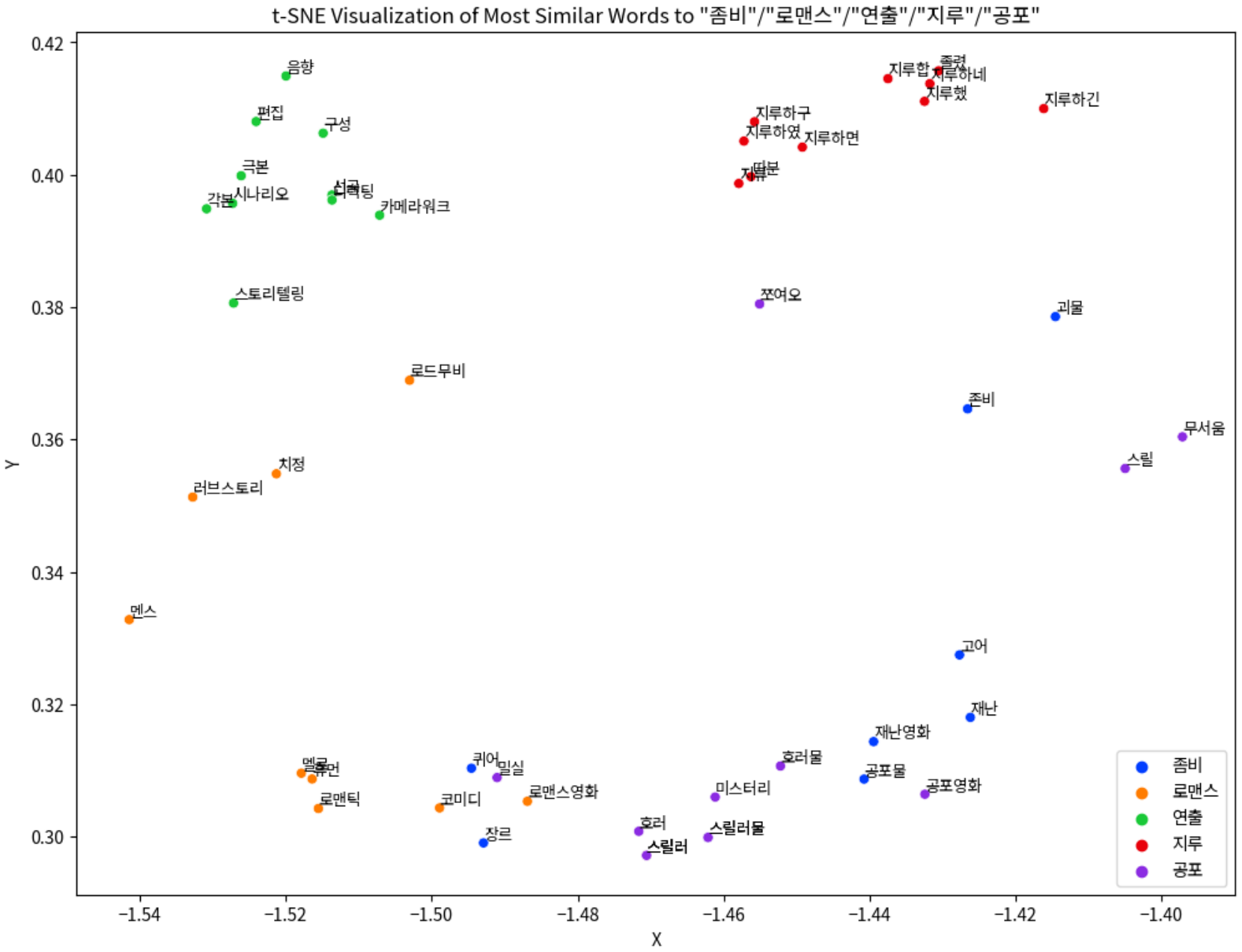
```
#에러 문제--'perplexity must be less than n_samples'
#TSNE의 파라미터 perplexity 조절 default=30.0인데

tsne = TSNE(n_components=2, random_state=42, perplexity = len(word_vectors_list)-1)
embeddings_2d = tsne.fit_transform(word_vectors_list)


plt.figure(figsize=(12,9))
sns.scatterplot(x=embeddings_2d[:,0], y=embeddings_2d[:,1], hue=associated_words, palette=sns.color_palette('bright'))
plt.title('t-SNE Visualization of Most Similar Words to "좀비"/"로맨스"/"연출"/"지루"/"공포"')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend(loc='lower right')

for i, (x,y) in enumerate(embeddings_2d):
    plt.text(x, y, allwords_50[i], fontsize=9, ha='left',va='bottom')

plt.show()
```

1. TSNE를 이용해 100차원->2차원 차원 축소, 이때 perplexity가 샘플 수보다 크면 안 되기 때문에 len(word_vectors_list)-1로 설정해줌
2. Matplotlib와 seaborn 라이브러리를 활용해 2차원 평면에 시각화
3. 다섯 개 단어들 각각의 유사단어 10개가 어느 정도로 클러스터링 되는지 확인할 수 있었다.

과제 및 실습 1

(3) 실습문제 3

output



과제 및 실습 1

(3) 실습문제 4

[실습 및 과제 4]

실습 4의 코드를 수정하여 조금 더 많은 양의 문서를 학습하여 더 정확한 비슷한 리뷰 랭킹 결과가 나오게 한다.

Hint1: 학습하는 리뷰의 수를 늘리기 (현재는 10,000개의 리뷰만 학습된 상태이고, 총 788,189개 리뷰가 존재한다.)

Hint2: doc2vec의 파라미터 변경

1. 개선된 모델에 대한 분석하는 리포트를 작성하라. (어떤 부분을 개선 했는지)

2. 개선의 결과 중 어떤 부분이 비슷한지 자연어처리 관점에서 설명하라.

- **제출물:**
- 1) 작성 코드 스크린샷
- 2) 작성 코드에 대한 설명
- 3) 아웃풋 결과물 스크린샷

(3) 실습문제 4

실험 1: 학습 문장수를 10000개에서 20000개로 늘려 보았다

```
reviews = [doc[1].strip() for doc in docs][:20000]


preprocessed_reviews = [preprocess(review) for review in reviews]

from gensim.models.doc2vec import TaggedDocument
tagged_docs = [TaggedDocument(doc, tags=[i]) for i, doc in enumerate(preprocessed_reviews) if doc != None]

from gensim.models.doc2vec import Doc2Vec
model_2 = Doc2Vec(tagged_docs, vector_size=100, min_count=3, epochs=100, dm=1, negative=5,
                  alpha=0.001)
```

```
results = model_2.dv.most_similar(100, topn=10)

print("\nOriginal Review:")
print(reviews[100])
for i, result in enumerate(results):
    print("\nSimilar Reviews", i+1)
    print(result)
    review_id = result[0]
    print(reviews[review_id])
```

과제 및 실습 1

(3) 실습문제 4

실험 1 결과: 학습 문장 수가 10000 개 였을 때보다 유의미한 개선이 보였지만(부정적 리뷰들로 추려짐), 10개 문장 중 5개 문장만 유사도 0.6 이상이 나왔다

Original Review:
중반 루즈 하게 늘어지 했 좀더 높은 평점 될 있었 조금 아쉬운 전개

Similar Reviews 1
(66, 0.6302624940872192)
원작 만화 먼저 봐 런가 내용 똑같 영화 재미 없네 좀 뭐랄 계속 진지하던 갑갑하기 하고 상황 설정 너무 말 안되 부산 행 만들었 싶은 생각 계속 했 좀비 매니아 로써 좀 실망 크

Similar Reviews 2
(1262, 0.6288757920265198)
일본 문화 정도 알 있는 사람 라면 볼 만 영화 한국 편 부산 행 기대하고 있으 아쉬움 없지 않은 부분 있 수 있다 일본 특유 지루하다 지루한 부분 있기 때문 와 닿는 줄거리 마음 드는 편 속한

Similar Reviews 3
(615, 0.6132833957672119)
영화 보면 나가고 싶 생각 해본 적 없는 온몸 베베 꼬면 볼 정도 나가고 싶었 진짜 미친 것 같다 만화책 원작 것 이제 알 평점 높은 사람 만화책 빠져 나오는 잔인 거 볼게 없다 내용 기 노잼

Similar Reviews 4
(634, 0.6046154499053955)
개인 기대 않고 봤 정말 재밌게 봤 깔끔한 스토리 선택 집중 잘해서 필요한 부분 잘 보여 재미 챙겼 다만 원작 본 사람 너무 잘랐 덧 히로 미가 왜 무것 하지 않냐 아직 원작 반도 따라가서 그런

Similar Reviews 5
(1200, 0.6007057428359985)
좀비 영화 좋아해서 봤 약간 지루한 있지 그 안 나름 볼거리 있다

Similar Reviews 6
(520, 0.5989741086959839)
하도 만들었 다해서 봤 좀비 물 안 본 일반인 설레 발 이었 지루하다 점 낮추기 위해 점준

Similar Reviews 7
(13871, 0.5977314710617065)
대박 비스티보이즈 저리 가라 임 뭐 있겠 설마 에이 거 아니 했 너무 함 뭘 말하고 싶은 지는 알 이건 심함

Similar Reviews 8
(15723, 0.5917060971260071)
전체 흐름 성공하게 된 에피소드 부분 좀 약해서 큰 감동 받 순 없었 중간 중간 감동 부분 있었

Similar Reviews 9
(1226, 0.5909866094589233)
잔인 거 봐 거의 좀비 싸우는 장면 자막 나온 같지 재미 있어 웃긴 부분 있고 좀 병맛 같기 하구 시간 감

Similar Reviews 10
(478, 0.5886700749397278)
좀비 영화 드라마 광 기대하고 봤 영화 긴장감 전혀 없었 워킹데드 드라마 편 정도 본 느낌 랄 까 주인공 총 하나로 좀비 때 잡 수 있다 무협 영화 아니

(3) 실습문제 4

실험 2: 학습 문장수를 10000개에서 20000개로 늘림과 동시에 에폭 수를 100에서 150으로 늘리기

```
reviews = [doc[1].strip() for doc in docs][:20000]


preprocessed_reviews = [preprocess(review) for review in reviews]

from gensim.models.doc2vec import TaggedDocument
tagged_docs = [TaggedDocument(doc, tags=[i]) for i, doc in enumerate(preprocessed_reviews) if doc != None]

from gensim.models.doc2vec import Doc2Vec
model_2 = Doc2Vec(tagged_docs, vector_size=100, min_count=3, epochs=150, dm=1, negative=5,
                  alpha=0.001)
```

```
results = model_2.dv.most_similar(100, topn=10)

print("\nOriginal Review:")
print(reviews[100])
for i, result in enumerate(results):
    print("\nSimilar Reviews", i+1)
    print(result)
    review_id = result[0]
    print(reviews[review_id])
```

과제 및 실습 1

(3) 실습문제 4

실험 2 결과: 모든 문장의 유사도가 0.7 이상으로 나왔다. 본 문장처럼 '아쉽다', '지루하다'는 감정을 표현하는 유사한 문장들이 결과로 나온 것으로 보인다. 다만 가장 유사하다고 나온 1번문장은 본 문장보다 더 부정적인 느낌을 내포하는 것으로 보인다. 오히려 하위 순위의 리뷰들이 비슷한 감정을 표현하고 있다

Original Review:
중반 루즈 하게 늘어지 했 좀더 높은 평점 될 있었 조금 아쉬운 전개

Similar Reviews 1
(12733, 0.7491954565048218)
내용 때문 스트레스 받은 아니 영화 개연 성도 부족하고 어 설 보면 스트레스 받 평론가 평점 낮은 이유 있었 기대하고 봤 화남

Similar Reviews 2
(855, 0.7437376379966736)
원작 부족한 점 잘 살리 원작 잘됬 부분 죽 여러 모로 아쉬운 영화 잔인 건 덤

Similar Reviews 3
(520, 0.7373504042625427)
하도 만들었 다해서 봤 좀비 물 안 본 일반인 설레 발 이었 지루하다 점 낮추기 위해 점준

Similar Reviews 4
(15263, 0.7347068786621094)
몰입 감 좀 떨어지 나쁘 않은

Similar Reviews 5
(740, 0.733656644821167)
아쉬운 점 있다 원작 만화 다른 스 피디 전개 원작 느긋 심리 묘사 잔잔 시간 전개 영화 특수성 급 박하 다급하게 전개된 원작 모르 보는 관객 입장 인물 몰입 조금 버거

Similar Reviews 6
(990, 0.722673237323761)
돈 아깝 않고 나름 괜찮 영화 시간 때문 칼질 좀 게 아까웠

Similar Reviews 7
(3847, 0.7203417420387268)
너무 순정 스럽 하기 좀 그렇 너무 밋밋 감 평점 높은 이해 조금 안되 전문가 관객 평점 너무 차이 가나 점 드려 조금 평점 잘못된 같다 기분

Similar Reviews 8
(17897, 0.7161885499954224)
역사 사실 민족 시사 점등 영화 통해 얻고 했 것 너무 큰 욕심 일 음악 단편 상황 눈물 빼려 하고 깊이 전혀 없는 영화 였 차라리 부산 행 같은 영화 깊이 없어 아예 기대하지 않고 가서 보는 영화 괜찮 역

Similar Reviews 9
(16573, 0.7149726748466492)
이런 글 쓰는 너무 별로 씁 좋 배우 나오는 정말 별로 였 일단 흐름 개연 떨어지 쌩뚱 맞게 흘러갑 좀 지루하고 여 튼 보지 마세 시간 아까운

Similar Reviews 10
(880, 0.7146745324134827)
좀비 영화 좋아하는 잔인 괜찮 약간 긴장감 떨어지 주후 부산 행 더 긴장감 있고 좋 것 같아

(3) 실습문제 4

실험 3: 학습 문장수 20000, 에폭 수를 150, vector_size=160으로

```
reviews = [doc[1].strip() for doc in docs][:20000]


preprocessed_reviews = [preprocess(review) for review in reviews]

from gensim.models.doc2vec import TaggedDocument
tagged_docs = [TaggedDocument(doc, tags=[i]) for i, doc in enumerate(preprocessed_reviews) if doc != None]

from gensim.models.doc2vec import Doc2Vec
model_2 = Doc2Vec(tagged_docs, vector_size=160, min_count=3, epochs=150, dm=1, negative=5,
                  alpha=0.001, )
```

```
results = model_2.dv.most_similar(100, topn=10)

print("\nOriginal Review:")
print(reviews[100])
for i, result in enumerate(results):
    print("\nSimilar Reviews", i+1)
    print(result)
    review_id = result[0]
    print(reviews[review_id])
```

학습하는 문장 수를 늘렸기 때문에 모델이 더 많은 정보를 학습할 수 있음을 고려해 vector size를 더 크게 설정하였다.

과제 및 실습 1

(3) 실습문제 4

실험 3 결과: 전반적으로 유사도가 높아졌고, 가장 유사한 1번문장이 실험 2에서보다 덜 부정적인 느낌이다 나머지 문장들도 3번을 제외하면 '지루하다' 거나 '아쉬운' 감정을 잘 드러내고 있는 것으로 보인다.

```
Original Review:
중반 루즈 하게 늘어지 했 좀더 높은 평점 될 있었 조금 아쉬운 전개

Similar Reviews 1
(740, 0.8055727481842041)
아쉬운 점 있다 원작 만화 다른 스 피디 전개 원작 느긋 심리 묘사 잔잔 시간 전개 영화 특수성 급 박하

Similar Reviews 2
(14758, 0.8029325008392334)
좀 억지 부분 있었 나름 꽤 재밌게 봤 마무리 너무 뜸금 없고 떡밥 꽤 흘린 같은 회수 안되 끝나서 완성

Similar Reviews 3
(12733, 0.7940245866775513)
내용 때문 스트레스 받은 아니 영화 개연 성도 부족하고 어 설 보면 스트레스 받 평론가 평점 낮은 이유

Similar Reviews 4
(5898, 0.7876927852630615)
후반 전투씬 좀 짧은 감 느껴져 아쉬웠 그것 제외하면 전체 만족함

Similar Reviews 5
(15717, 0.7705382704734802)
좋 배우 아까운 영화 스토리 너무 허 접하고 지루하고 코미디 데 코믹 요소 하나 없 시간 돈 아까웠 어디

Similar Reviews 6
(855, 0.7691242694854736)
원작 부족한 점 잘 살리 원작 잘됐 부분 죽 여러 모로 아쉬운 영화 잔인 건 덤

Similar Reviews 7
(4178, 0.7654222249984741)
솔직히 점줄 영화 아니 평점 조절하려 점 드려 내용 전개 억지 부분 있었 장면 장면 끊어지는 느낌 후반

Similar Reviews 8
(880, 0.763227641582489)
좀비 영화 좋아하는 잔인 괜찮 약간 긴장감 떨어지 주후 부산 행 더 긴장감 있고 좋 것 같아

Similar Reviews 9
(3847, 0.7616686224937439)
너무 순정 스럽 하기 좀 그렇 너무 밋밋 감 평점 높은 이해 조금 안되 전문가 관객 평점 너무 차이 가나

Similar Reviews 10
(307, 0.7476649880409241)
원작 살렸 원작 뒤 더 내용 있지 스토리 자르는 부분 적당하게 잘 자른 같고 일본 원작 영화 하면 죽쑤는
```