Text Analytics Project

AI빅데이터융합경영학과
20195262 장예진

INDEX

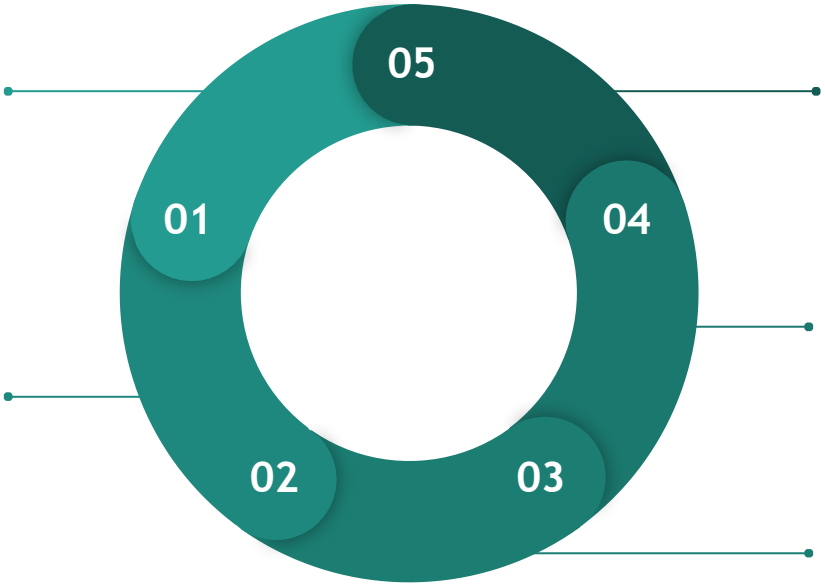
INDEX -details

Data Crawling

Choosing topics and extracting comments from select videos from the API

Data Preprocessing

Cleaning up the comments and performing natural language processing to reduce noise and redundancy in the data



Data Visualizations

Plotting graphs, creating word clouds, and building a dashboard to showcase results

Making Predictions

Using the models to make predictions on the classification of the comments based on fitted models

Data Modeling

Transforming the textual into numeric format and fitting machine learning algorithms on labeled

Introduction - Background of Idea(1)

Analysis of YouTube comments Korean movie ‘parasite’ review

최근 영상 매체와 쌍방향 소통 미디어의 발전으로 대중의 반응을 쉽게 확인할 수 있다. 여기서 ‘유튜브(Youtube)’는 다양한 컨텐츠 전달하는 세계 최대 규모의 영상 매체로 시청자(대중)의 반응이나 감정이 즉각적이고 적나라하게 드러나는 특징이 있다. 영화 리뷰 영상을 미리 보거나 관람 후에 감상평을 유튜브 댓글을 통해 공유하며 피드백을 남기는 새로운 트렌드가 만들어졌다.

칸 영화제에서 황금 종려상을 수상한 봉준호 감독의 <기생충(parasite)>의 해외 대중 반응을 유튜브 영화리뷰 영상 댓글을 통해 확인해보자.

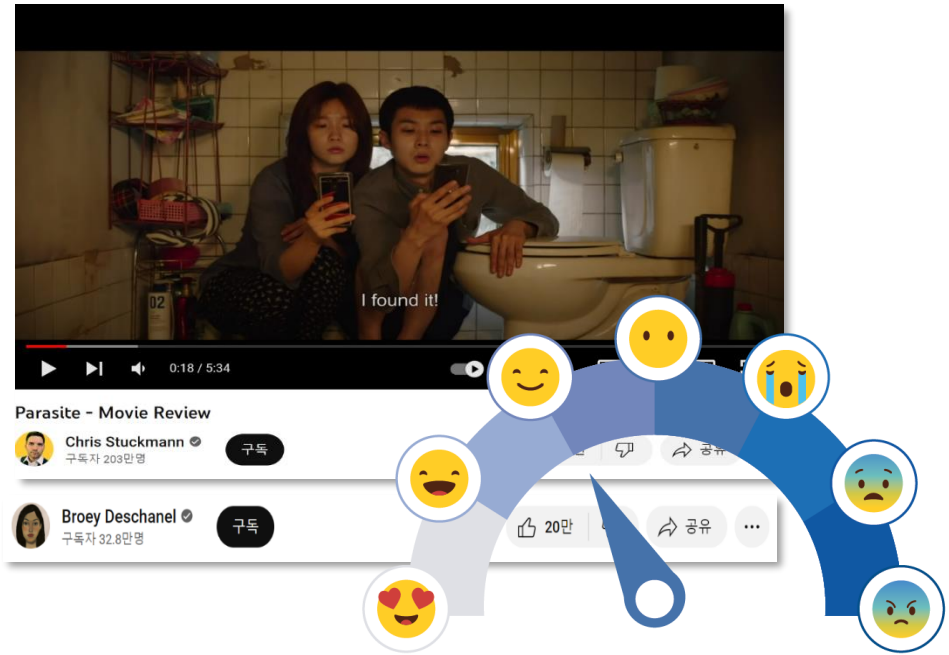
(*기생충은 해외에서 한국영화를 주목하게 만든 영화로 국내 반응은 객관성이 떨어질 것이라 판단하여 국내 댓글 데이터를 제외하였다.)

Introduction - Background of Idea(1)

Analysis of YouTube comments Korean movie ‘parasite’ review

본 프로젝트에서는 영화 ‘기생충(parasite)’ 리뷰를 다룬 유튜브 영상에 대한 해외 대중들의 관심과 특성을 알아보고자 한다.

‘parasite review’를 topic 으로 지정하여 그 중 top2의 영상 댓글 데이터를 수집하였다.



- Sentiment Analysis(감성분석) 진행
- Positive, Negative, and Neutral 3가지로 구분
- Modeling (SVM,LR,NB,MLP)
- Topic Modeling (LSA, LDA)

- > 분석을 통한 유의미한 인사이트 얻기

Data Crawling(1)

- Youtube API를 활용한 영상 댓글 데이터 수집

1. Google API console에 접속하여 Youtube Data API(V3)의 API key를 발급

2. API를 사용하기 위해 Google API Client 라이브러리 다운로드

3. 발급받은 API key를 입력하고 추출하고자 하는 동영상의 id(watch?v=뒤의 문자)를 입력

4. 각종 변수 설정

- - comments: 댓글들을 저장할 리스트형 변수
- - api_obj : import 한 build 함수로 생성할 Google API객체
- - response: 입력한 id의 동영상 관련 정보가 전달되는 변수
- - textDisplay: 댓글의 내용
- - authorDisplayName: 댓글 작성자
- - publishedAt: 댓글 작성 시간
- - likeCount: 좋아요 수

```python
import pandas as pd

# 비디오 댓글 데이터를 저장할 리스트
all_comments = []

# 각 비디오의 정보
video_info = [
    {'video_id': 'zgAK-4kPTb8', 'api_key': 'AIzaSyCe5QZ3LZsk9H-Xr-BIZPPspEGk_mLPGZQ'},
    {'video_id': 'l706YqckEyE', 'api_key': 'AIzaSyCe5QZ3LZsk9H-Xr-BIZPPspEGk_mLPGZQ'}
]

# YouTube API를 사용하여 댓글 데이터 추출
for info in video_info:
    video_id = info['video_id']
    api_key = info['api_key']

    comments = []
    api_obj = build('youtube', 'v3', developerKey=api_key)
    # 댓글 스레드 리스트 가져오기
    response = api_obj.commentThreads().list(part='snippet,replies', videoId=video_id, maxResults=100).execute()

    while response:
        for item in response['items']:
            # 상위 댓글 정보 가져오기
            comment = item['snippet']['topLevelComment']['snippet']
            comments.append([comment['textDisplay'], comment['authorDisplayName'], comment['publishedAt'], comment['likeCount']])

            if item['snippet']['totalReplyCount'] > 0:
                for reply_item in item['replies']['comments']:
                    # 대댓글 정보 가져오기
                    reply = reply_item['snippet']
                    comments.append([reply['textDisplay'], reply['authorDisplayName'], reply['publishedAt'], reply['likeCount']])

        # 다음 페이지 토큰이 있는 경우 다음 페이지로 이동
        if 'nextPageToken' in response:
            response = api_obj.commentThreads().list(part='snippet,replies', videoId=video_id, pageToken=response['nextPageToken'], ma
        else:
            break

    # 해당 비디오의 댓글 데이터를 전체 댓글 리스트에 추가
    all_comments.extend(comments)
```

참고문헌: https://velog.io/@ally0526/1.-Youtube-API%EB%A1%9C-%EB%8C%93%EA%B8%80-%EA%B0%80%EC%A0%B8%EC%98%A4%EA%B8%B0

Data Crawling(2)

- Youtube API를 활용한 영상 댓글 데이터 수집

6. commentThreads().list 메서드를 사용하여 video 댓글의 threads(스레드)를 가져옴.

7. 한번에 최대 100 개의 댓글 스레드를 가져오며 각 댓글의 댓글 내용 (textDisplay), 작성자 (displayName),
작성일(publishedAt),좋아요 수 (likeCount) 등의 정보가 포함.

8. While문을 통해 가져온 댓글 데이터를 리스트에 저장한 후, 답글이 있는 경우 답글 데이터로 함께 저장한다고 선언

9. NextPageToken 을 통해 페이지를 넘겨가며 모든 댓글의 스래드 수집

10. all_comments리스트에 모든 댓글 데이터과 답글의 전체 정보를 save

```python
import pandas as pd

# 비디오 댓글 데이터를 저장할 리스트
all_comments = []

# 각 비디오의 정보
video_info = [
    {'video_id': 'zgAK-4kPTb8', 'api_key': 'AIzaSyCe5QZ3LZsk9H-Xr-BIZPPspEGk_mLPGZQ'},
    {'video_id': 'I706YqckEyE', 'api_key': 'AIzaSyCe5QZ3LZsk9H-Xr-BIZPPspEGk_mLPGZQ'}
]

# YouTube API를 사용하여 댓글 데이터 추출
for info in video_info:
    video_id = info['video_id']
    api_key = info['api_key']

    comments = []
    api_obj = build('youtube', 'v3', developerKey=api_key)
    # 댓글 스레드 리스트 가져오기
    response = api_obj.commentThreads().list(part='snippet,replies', videoId=video_id, maxResults=100).execute()

    while response:
        for item in response['items']:
            # 상위 댓글 정보 가져오기
            comment = item['snippet']['topLevelComment']['snippet']
            comments.append([comment['textDisplay'], comment['authorDisplayName'], comment['publishedAt'], comment['likeCount']])

            if item['snippet']['totalReplyCount'] > 0:
                for reply_item in item['replies']['comments']:
                    # 대댓글 정보 가져오기
                    reply = reply_item['snippet']
                    comments.append([reply['textDisplay'], reply['authorDisplayName'], reply['publishedAt'], reply['likeCount']])

        # 다음 페이지 토큰이 있는 경우 다음 페이지로 이동
        if 'nextPageToken' in response:
            response = api_obj.commentThreads().list(part='snippet,replies', videoId=video_id, pageToken=response['nextPageToken'], ma
        else:
            break

    # 해당 비디오의 댓글 데이터를 전체 댓글 리스트에 추가
    all_comments.extend(comments)
```

참고문헌: https://velog.io/@ally0526/1.-Youtube-API%EB%A1%9C-%EB%8C%93%EA%B8%80-%EA%B0%80%EC%A0%B8%EC%98%A4%EA%B8%B0

Preprocessing(1)

1. raw data에서 comment만 추출한 comment_df를 csv로 저장

2. preprcoess_english 함수로 전처리 과정 정의

- 1) 소문자 변환 (대소문자를 구분하지 않기 위함)

- 2) 특수문자 , 숫자 , 이모티콘 제거 : 정규표현식 활용하여 텍스트에서 의미를 가지지 않거나 처리하기 어려운 부분인 특수문자, 숫자, 이모지를 제거

- 3) Tokenization(토큰화): treebankwordTokenizer 를 사용하여 문장을 단어로 토큰화 진행 (문장을 단어 단위로 분리하는 과정)

```
df = pd.read_csv('raw_comments_eng.csv')
# Select only the 'comment' column
comment_df = df[['comment']]

# # Save the DataFrame to a CSV file (comment만 있는 파일)
# comment_df.to_csv('comments_english.csv', index=False)
```

```
def preprocess_english(text):
    tokenizer = TreebankWordTokenizer() # TreebankWordTokenizer 객체 생성
    stop_words = set(stopwords.words('english')) # 영어 불용어(stop words) 집합 생성

    # Define the emoji pattern
    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)

    preprocessed_comments = []
    for comment in text:
        # Convert to lowercase
        comment = comment.lower()

        # Remove special characters, digits, and emojis
        comment = re.sub(emoji_pattern, '', comment)
        comment = re.sub(r'[^A-Za-z]', ' ', comment)

        # Tokenize the comment
        word_tokens = tokenizer.tokenize(comment)

        # Remove stopwords and short words
        filtered_words = [word for word in word_tokens if word not in stop_words and len(word) > 2]

        # Join the filtered words back into a comment
        preprocessed_comment = ' '.join(filtered_words)

        preprocessed_comments.append(preprocessed_comment)

    return preprocessed_comments
```

Preprocessing(2)

- 4) stopword removal(불용어 제거) : NLTK에 stopwords.words('english') 를 사용하여 영어의 불용어를 가져와 제거. 토큰화 된 단어 들 중에서 불용어에 해당하는 단어를 제거 진행

- 5) 단어길이 필터링 : 토큰화 된 단어의 길이가 2 보다 작은 경우 제거를 진행. 유튜브 댓글의 경우 oh, hh 등의 큰 의미를 가지지 않는 짧은 단어들을 제거하기 위한 필터링 과정

- 6) 전 처리된 텍스트 반환 : 전 처리된 단어들을 공백을 이용하여 다시 문장으로 연결하여 저장

```python
def preprocess_english(text):
    tokenizer = TreebankWordTokenizer() # TreebankWordTokenizer 객체 생성
    stop_words = set(stopwords.words('english')) # 영어 불용어(stop words) 집합 생성

    # Define the emoji pattern
    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)

    preprocessed_comments = []
    for comment in text:
        # Convert to lowercase
        comment = comment.lower()

        # Remove special characters, digits, and emojis
        comment = re.sub(emoji_pattern, '', comment)
        comment = re.sub(r'[^A-Za-z]', ' ', comment)

        # Tokenize the comment
        word_tokens = tokenizer.tokenize(comment)

        # Remove stopwords and short words
        filtered_words = [word for word in word_tokens if word not in stop_words and len(word) > 2]

        # Join the filtered words back into a comment
        preprocessed_comment = ' '.join(filtered_words)

        preprocessed_comments.append(preprocessed_comment)

    return preprocessed_comments
```

Preprocessing(3)

3. comment_df에 전처리된 preprocessed_comment column을 추가

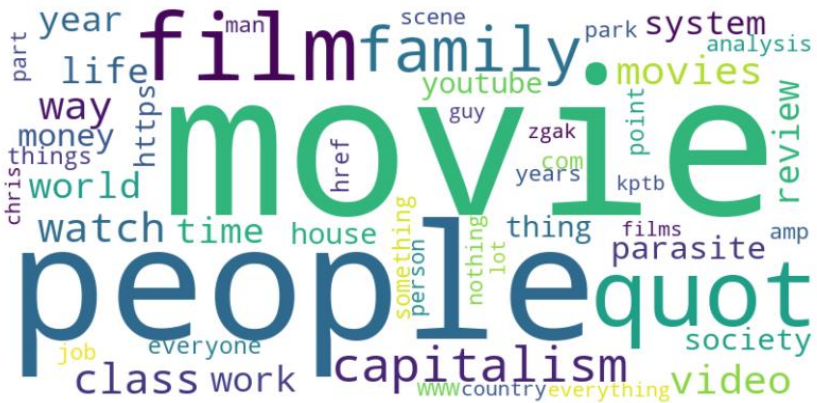
4. Pos-tag: 토큰화의 단어 추출을 통해 텍스트를 깔끔하게 cleaning한 이후, 이제 cleaning 된 text 에 품사를 태깅함으로서 단어의 역할과 의미를 파악 가능. 품사 태그에서 NN 이 가장 유 의미 할 것이라고 판단

5. pos-tagging된 NN에서 빈도 top50개의 단어를 WordCloud를 통한 Visualization

```
# pos-tag: 기본 전처리 후 품사 태깅
```

```
preprocessed_comments = comment_df['preprocessed_comment']
```

```
# POS-Tagging 수행
tokenizer = TreebankWordTokenizer()
tagged_comments = []
for comment in preprocessed_comments:
    tokens = tokenizer.tokenize(comment)
    tagged_comment = pos_tag(tokens)
    tagged_comments.append(tagged_comment)
tagged_comments_df = pd.DataFrame({'tagged_comment': tagged_comments})

# Initialize a counter to count noun occurrences
noun_counter = Counter()

# Iterate over the tagged comments
for tagged_comment in tagged_comments:
    # Extract the nouns from the tagged comment
    nouns = [word for word, pos in tagged_comment if pos.startswith('NN')]
    # Update the noun counter
    noun_counter.update(nouns)
```



Sentiment Analysis - Sentiment Labeling

1. 댓글의 긍,부정을 구분하는 Labeling을 진행

2. nltk의 패키지 SentimentIntensityAnalyzer 를 사용하여 VADER(Valence Aware Dictionary and sEntiment Reasoner) 감성분석기 구현을 진행.

3. 코드에서는 위의 SentimentIntensityAnalyzer 클래스의 인스턴스를 생성한 이후 이 인스턴스를 활용하여 VADER 감성분석 수행

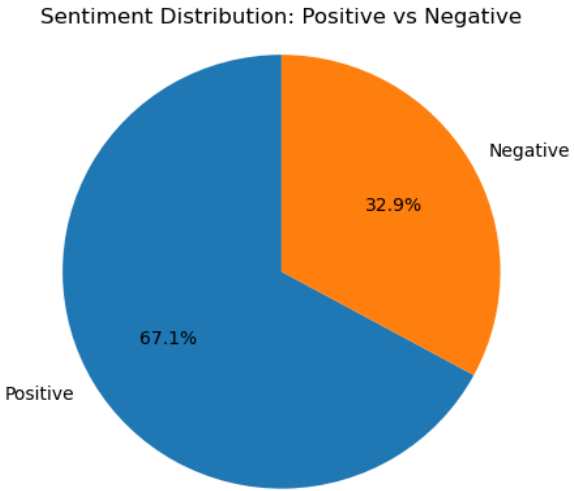
4. get_sentiment_label 함수는 텍스트를 입력으로 받아 해당 텍스트의 감성을 분류해주는 함수임. 함수 내에서 입력된 텍스트가 문자열인지 확인한 후에 sia.pol arity_scores() 메소드를 사용하여 해당 텍스트의 감성 점수를 계산

5. VADER 감성분석기는 주어진 텍스트의 compound 점수를 기준으로 긍정 부정 , 중립으로 분류함. Compound 점수가 0.05 이상인 경우 ‘positive’로 분류하고 , 0.05 이하인 경우 에는 ‘negative’ ,그 외는 ‘neutral’로 분류.

```python
# Create an instance of the VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Function to classify sentiment using VADER
def get_sentiment_label(text):
    if isinstance(text, str):
        sentiment_scores = sia.polarity_scores(text)

        if sentiment_scores['compound'] >= 0.05:
            return 'positive'
        elif sentiment_scores['compound'] <= -0.05:
            return 'negative'
        else:
            return 'neutral'
    else:
        return 'neutral'
```

```python
# Apply the sentiment label to the DataFrame
comment_df['sentiment_label'] = comment_df['preprocessed_comment'].apply(get_sentiment_label)
```

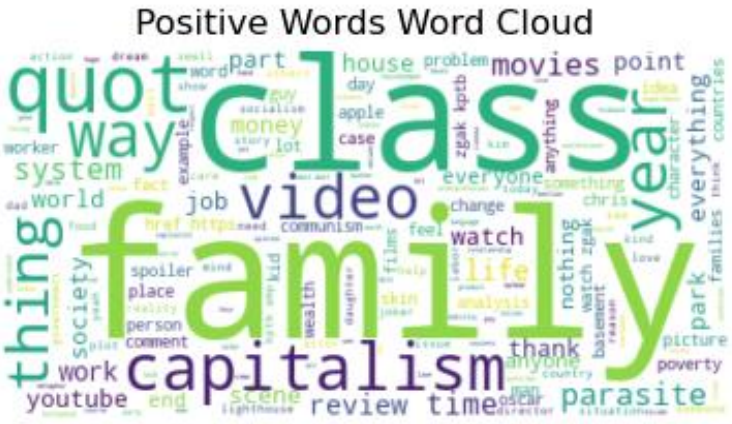
Sentiment Analysis - Sentiment Labeling

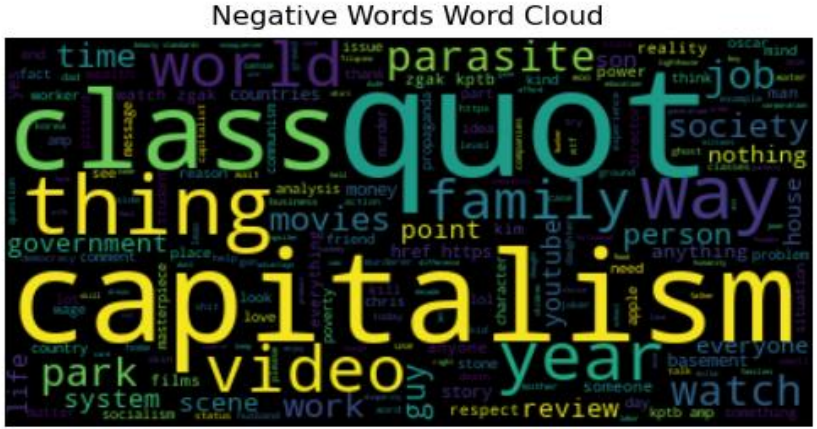


감성 분석 결과
positive(긍정)의 비율이
과반수 이상인 것으로 확인됨.

그렇다면 각 각의 키워드
단어는 무엇일까?

Sentiment Analysis - Sentiment Labeling





Pos-tagging을 활용한 긍정,부정의 키워드를 wordcloud를 통해 살펴보자. 압도적으로 높은 긍정의 비율에 비해서 그다지 다르지 않은 keyword를 확인할 수 있다.

왜 일까?
영화의 내용 자체가 사회 비판적인 부정적 내용 이기 때문이다!

(*people,movie,film등의 무의미하고 반복된 단어들을 제외)

Modeling - setting

1. 위의 Sentiment Labeling을 통해 추가한 sentiment_label column을 활용하여 모델링 진행

2. Train_test_split 모듈을 사용하여 데이터셋을 train, test데이터로 분할

3. 전 처리된 댓글을 포함하는 특성행렬 X와 감성레이블을 포함하는 대상 벡터 y를 생성

4. TfidfVectorizer를 사용하여 TF-IDF 벡터화를 초기화 해준 이후 , 벡터화 객체를 사용하여 학습데이터와 테스트 데이터의 전 처리된 댓글을 TF IDF 벡터로 변환해줌.

```
# 학습 데이터와 레이블 생성
X = comment_df['preprocessed_comment']
y = comment_df['sentiment_label']

# 데이터를 학습 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#결측치 제거
comment_df.dropna(subset=['preprocessed_comment'], inplace=True)
```

```
C:₩Users₩shj06₩AppData₩Local₩Temp₩ipykernel_19172₩3194258765.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/in
-copy
  comment_df.dropna(subset=['preprocessed_comment'], inplace=True)
```

4-2) tf-idf vectorization

```
# TF-IDF 벡터화 객체 생성
vectorizer = TfidfVectorizer()

# TF-IDF 벡터화
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
```

각 분류기 모델을 텍스트 데이터인 'preprocessed_comment'를 TF-IDF 벡터화 하여 입력으로 사용하고, 이를 기반으로 감성 레이블을 예측한다. 예측 결과를 출력하고, 정확도(Accuracy)를 계산하여 모델의 성능을 평가한다. 이를 통해 다양한 분류기 모델을 비교하고, 가장 성능이 우수한 모델을 선택할 수 있다

Modeling - SVM(Support Vector Machine)

1. ‘SVC’를 사용하여 SVM분류기모델을 생성
2. X_train_Vectorized와 y_train을 사용하여 모델을 학습
3. 학습된 SVM분류기를 사용하여 테스트 데이터의 감성 레이블을 예측해줌
4. 댓글과 예측된 감성 레이블을 출력하여 확인
5. accuracy_score모듈을 사용하여 SVM분류기 모델의 정확도를 계산

```
# SVM 분류기 모델 생성
svm_classifier = SVC()

# SVM 모델 학습
svm_classifier.fit(X_train_vectorized, y_train)

# SVM 모델 예측
y_pred_svm = svm_classifier.predict(X_test_vectorized)

# 성능 평가
accuracy_svm = accuracy_score(y_test, y_pred_svm)
```

```
# svm결과 출력
for comment, label in zip(X_test, y_pred_svm):
    print(f'댓글: {comment}')
    print(f'감성 레이블: {label}')
    print('----------------------')
```

```
댓글: comments amazing guys really know valur realness movie amazing
감성 레이블: positive
----------------------
댓글: well everytime remember ending really terrifies especially promise kiwoo buy house fuckin sleep knowing never buy house take years save money buy house father hids
감성 레이블: positive
----------------------
댓글: watched film days ago forgot intense flood scene particularly interaction old housewife husband kicked stairs seriously concussed sadness anger thing existence loved dying front nothing could shattering seriously felt bad throughout whole film
감성 레이블: negative
----------------------
댓글: dissection movie literally gave chills true
감성 레이블: positive
----------------------
```

Modeling - Logistic Regression

1. 로지스틱 회귀분류기를 사용하여 LR분류기 모델을 생성

2. X_train_Vectorized와 y_train을 사용하여 모델을 학습

3. 학습된 LR분류기 모델을 사용하여 테스트 데이터의 감성 레이블을 예측해줌

4.댓글과 예측된 감성 레이블을 출력하여 확인

5.accuracy_score모듈을 사용하여 LR분류기 모델의 정확도를 계산

```
# LR 분류기 모델 생성
lr_classifier = LogisticRegression()

# LR 모델 학습
lr_classifier.fit(X_train_vectorized, y_train)

# LR 모델 예측
y_pred_lr = lr_classifier.predict(X_test_vectorized)

# 성능 평가
accuracy_lr = accuracy_score(y_test, y_pred_lr)
```

```
# LR 모델 결과 출력
print("LR 모델 결과:")
print("--------------------------------")
for comment, label in zip(X_test, y_pred_lr):
    print(f'댓글: {comment}')
    print(f'감성 레이블: {label}')
    print('----------------------')
```

```
LR 모델 결과:
--------------------------------
댓글: comments amazing guys really know valur realness movie amazing
감성 레이블: positive
----------------------
댓글: well everytime remember ending really terrifies especially promise kiwoo buy house fuckin sleep knowing never buy house take years save money buy house father hids
감성 레이블: positive
----------------------
댓글: watched film days ago forgot intense flood scene particularly interaction old housewife husband kicked stairs seriously concussed sadness anger thing existence loved dying front nothing could shattering seriously felt bad throughout whole film
감성 레이블: negative
----------------------
댓글: dissection movie literally gave chills true
감성 레이블: positive
```

Modeling - Naive Bayes

1. MultinomialNB() 를 사용하여 Naive Bayes 분류기 모델을 생성
2. X_train_Vectorized와 y_train을 사용하여 모델을 학습
3. 학습된 Naive Bayes 분류기 모델을 사용하여 테스트 데이터의 감성 레이블을 예측해줌
4. 댓글과 예측된 감성 레이블을 출력하여 확인
5. accuracy_score모듈을 사용하여 Naive Bayes 분류기 모델의 정확도를 계산

```
# Naive Bayes 분류기 모델 생성
nb_classifier = MultinomialNB()

# Naive Bayes 모델 학습
nb_classifier.fit(X_train_vectorized, y_train)

# Naive Bayes 모델 예측
y_pred_nb = nb_classifier.predict(X_test_vectorized)

# 성능 평가
accuracy_nb = accuracy_score(y_test, y_pred_nb)
```

```
# Naive Bayes 모델 결과 출력
print("Naive Bayes 모델 결과:")
print("---------------------------------")
for comment, label in zip(X_test, y_pred_nb):
    print(f'댓글: {comment}')
    print(f'감성 레이블: {label}')
    print('----------------------')
```

```
Naive Bayes 모델 결과:
---------------------------------
댓글: comments amazing guys really know valur realness movie amazing
감성 레이블: positive
----------------------
댓글: well everytime remember ending really terrifies especially promise kiwoo buy house fuckin sleep knowing never buy house take years save money buy house father hids
감성 레이블: positive
----------------------
댓글: watched film days ago forgot intense flood scene particularly interaction old housewife husband kicked stairs seriously concussed sadness anger thing existence loved dying front nothing could shattering seriously felt bad throughout whole film
감성 레이블: positive
----------------------
댓글: dissection movie literally gave chills true
감성 레이블: positive
----------------------
댓글: love everyone comment section trying sound deep awe inspired everything movie simple lol beauty good storytelling simple people think artist way thinking writing really write know know naturally underlying tones blossom reflect reality honestly people overthink especially new social media era everyone looking fresh new take observed
감성 레이블: positive
----------------------
댓글: laughed like times yes great movie understand people call quot funny quot thought disturbing depressing almost start
감성 레이블: positive
```

Modeling - MLP(Mulitple-Layer Perceptron)

1. MLPClassifier() 를 사용하여 신경망 모델을 생성
2. hidden_layer_sizes=(128, 64)로 설정하여 2 개의 은닉층을 가지는 신경망을 구성
3. activation='relu' 로 설정하여 은닉층의 활성화 함수로 ReLU 를 사용.
4. X_train_vectorized 와 y_train 을 사용하여 모델을 학습
5. X_test_vectorized 를 입력으로 사용하여 감성 레이블을 예측해줌
6. accuracy_score모듈을 사용하여 모델의 정확도를 계산

```
# 신경망 모델 생성
nn_classifier = MLPClassifier(hidden_layer_sizes=(128, 64), activation='relu')

# 모델 학습
nn_classifier.fit(X_train_vectorized.toarray(), y_train)

# 예측
y_pred_nn = nn_classifier.predict(X_test_vectorized.toarray())

# 성능 평가
accuracy_nn = accuracy_score(y_test, y_pred_nn)
```

```
# MLP 모델 결과 출력
print("MLP 모델 결과:")
print("--------------------------------")
for comment, label in zip(X_test, y_pred_nn):
    print(f'댓글: {comment}')
    print(f'감성 레이블: {label}')
    print('----------------------')
```

```
MLP 모델 결과:
--------------------------------
댓글: comments amazing guys really know valur realness movie amazing
감성 레이블: positive
----------------------
댓글: well everytime remember ending really terrifies especially promise kiwoo buy house fuckin sleep knowing never buy house take years save money buy house father hids
감성 레이블: negative
----------------------
댓글: watched film days ago forgot intense flood scene particularly interaction old housewife husband kicked stairs seriously concussed sadness anger thing existence loved dying front nothing could shattering seriously felt bad throughout whole film
감성 레이블: negative
----------------------
댓글: dissection movie literally gave chills true
감성 레이블: positive
```

Modeling - Accuracy scores

- Training and testing on YouTube comment data

	SVM	LR	NB	MLP
Accuracy	0.8024	0.8106	0.5749	0.8010

Modeling - visualization

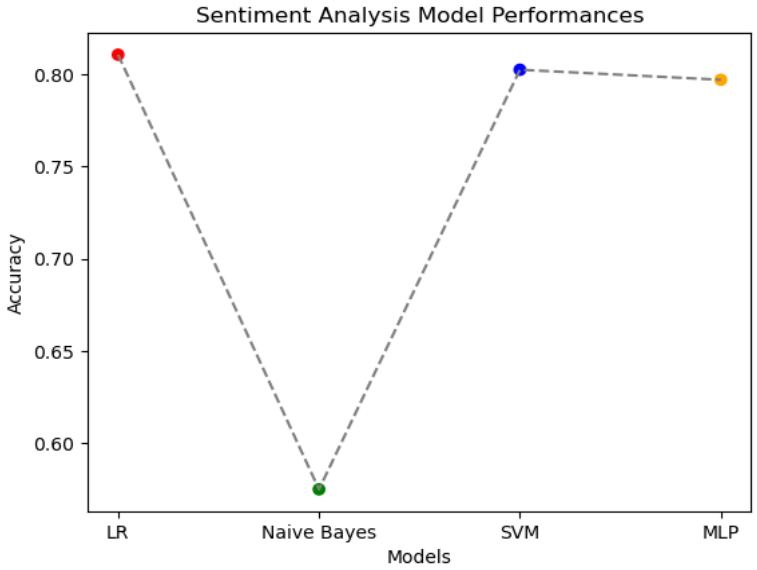
우수 모델: Logistic Regression

Why?

LR은 모델 파라미터 의 개수가 적어 비교적 간단한 모델 조직을 가지기 대문에 과적합(overfitting)의 가능성이 적다. 학습과 예측 속도가 빨라서 대용량 데이터도 빠른 학습이 가능하다. 또한, 희소(sparse)한 데이터에 대해 잘 작동하는 경향이 있다.

내가 분석하는 텍스트 데이터는 일반적으로 희소행렬(sparse matrix)의 형태이기 때문에 실제로 사용되는 단어의 비율이 상대적으로 적다.

그렇기에 Logistic Regression이 성능이 잘 나온 것이 아닐까?



Topic Modeling -LSA(Latent Semantic Analysis)

LSA란?

텍스트 문서의 잠재의미를 추출. 행렬분해기법을 사용하여 문서의 저차원의 의미 공간으로 표현하는 기법

1. analyze_topic_association 이라는 함수를 정의하여 대상 단어와 주제 간의 연관성을 분석할 예정 (해당 함수는 두 매개변수 사용, data 는 텍스트데이터, target_word를 연관성을 분석하려는 대상 단어)

2. TfidfVectorizer 를 사용하여 텍스트데이터를 TF-IDF 로 변환(여기서 최대 특성 수는 2000)

3. TfidfVectorizer객체를 fitting하고 변환된 결과를 x변수에 저장해줌

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
import numpy as np

def get_keyword_by_topic(components, feature_names, n=5):
    for idx, topic in enumerate(components):
        sorted_keyword_idx = np.argsort(topic)[-1:-n-1:-1]
        sorted_keyword_result = [(feature_names[x], topic[x].round(4)) for x in sorted_keyword_idx]
        print(f'Topic {idx}: {sorted_keyword_result}')
```

```python
def analyze_topic_modeling(data, n_topics=20, n_top_keywords=5):
    # TF-IDF 벡터라이저 생성
    vectorizer = TfidfVectorizer(stop_words='english', max_features=2000, max_df=0.5, smooth_idf=True)
    # 벡터라이저를 사용하여 데이터를 fit_transform합니다.
    X = vectorizer.fit_transform(data)

    # 토픽 모델링을 위한 TruncatedSVD 모델 생성
    model = TruncatedSVD(n_components=n_topics, n_iter=100, random_state=42)
    # TF-IDF로 변환된 데이터에 모델을 fit합니다.
    model.fit(X)

    # 데이터를 토픽 공간으로 변환합니다.
    result = model.transform(X)
    print(result.shape)

    # 벡터라이저에서 feature_names를 가져옵니다.
    feature_names = vectorizer.get_feature_names_out()
    # 각 토픽의 상위 키워드를 출력합니다.
    get_keyword_by_topic(model.components_, feature_names, n_top_keywords)

# 사용 예시
analyze_topic_modeling(comment_df['preprocessed_comment'], n_topics=20, n_top_keywords=5)
```

Topic Modeling -LSA(Latent Semantic Analysis)

4. TfidfVectorizer객체의 vocab_ 속성을 사용하여 단어와 해당하는 인덱스를 가져와 줌 . 단어는 인덱스 기준으로 정렬됨

5. TruncatedSVD 를 사용하여 TF IDF 벡터의 차원을 축소하고 여기서 27 개의 주성분을 사용하며 random_seed 는 42 로 고정

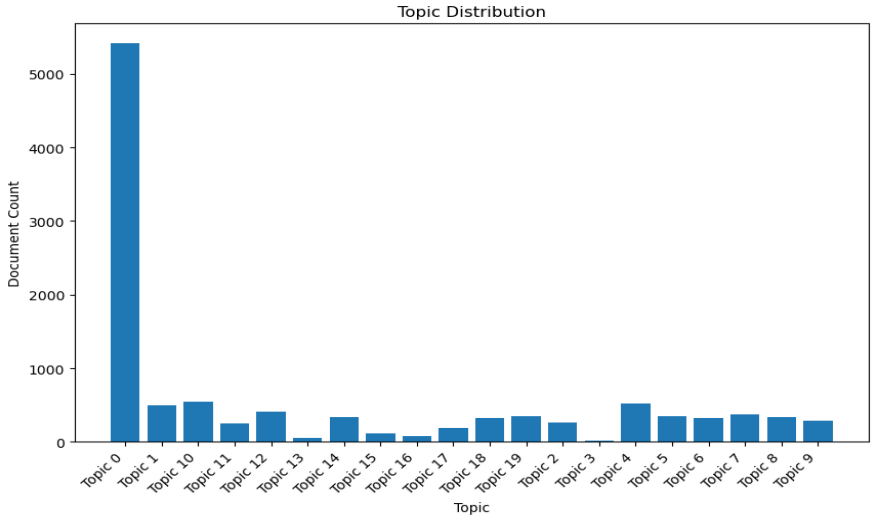
6. svd 객체에 fitting 하고 TF-IDF벡터에 적용

```python
def analyze_topic_modeling(data, n_topics=20, n_top_keywords=5):
    # TF-IDF 벡터라이저 생성
    vectorizer = TfidfVectorizer(stop_words='english', max_features=2000, max_df=0.5, smooth_idf=True)
    # 벡터라이저를 사용하여 데이터를 fit_transform합니다.
    X = vectorizer.fit_transform(data)

    # 토픽 모델링을 위한 TruncatedSVD 모델 생성
    model = TruncatedSVD(n_components=n_topics, n_iter=100, random_state=42)
    # TF-IDF로 변환된 데이터에 모델을 fit합니다.
    model.fit(X)

    # 데이터를 토픽 공간으로 변환합니다.
    result = model.transform(X)
    print(result.shape)

    # 벡터라이저에서 feature_names를 가져옵니다.
    feature_names = vectorizer.get_feature_names_out()
    # 각 토픽의 상위 키워드를 출력합니다.
    get_keyword_by_topic(model.components_, feature_names, n_top_keywords)

# 사용 예시
analyze_topic_modeling(comment_df['preprocessed_comment'], n_topics=20, n_top_keywords=5)
```

```
(11033, 20)
Topic 0: [('movie', 0.6008), ('quot', 0.3383), ('good', 0.1694), ('watch', 0.1684), ('like', 0.1572)]
Topic 1: [('quot', 0.6791), ('youtube', 0.1358), ('https', 0.1356), ('com', 0.1355), ('www', 0.1345)]
Topic 2: [('watch', 0.4383), ('https', 0.2872), ('youtube', 0.2854), ('com', 0.2851), ('www', 0.2835)]
Topic 3: [('quot', 0.5017), ('movie', 0.3031), ('review', 0.1209), ('year', 0.064), ('chris', 0.0521)]
Topic 4: [('best', 0.3768), ('parasite', 0.3454), ('review', 0.3389), ('film', 0.3287), ('year', 0.2777)]
Topic 5: [('review', 0.6037), ('video', 0.3273), ('great', 0.2525), ('chris', 0.1036), ('thank', 0.0958)]
Topic 6: [('video', 0.5788), ('good', 0.2814), ('great', 0.2668), ('really', 0.12), ('analysis', 0.1023)]
Topic 7: [('parasite', 0.8339), ('thought', 0.1734), ('anime', 0.1307), ('joker', 0.0703), ('lighthouse', 0.0643)]
Topic 8: [('good', 0.7859), ('movies', 0.1768), ('year', 0.1661), ('really', 0.1222), ('chris', 0.0599)]
Topic 9: [('film', 0.6408), ('watch', 0.413), ('great', 0.1527), ('family', 0.0718), ('analysis', 0.0677)]
Topic 10: [('like', 0.5576), ('capitalism', 0.2621), ('thought', 0.1661), ('movies', 0.1642), ('film', 0.1606)]
Topic 11: [('watch', 0.5897), ('movies', 0.3512), ('like', 0.1493), ('capitalism', 0.1337), ('chris', 0.1118)]
Topic 12: [('capitalism', 0.513), ('good', 0.2222), ('watch', 0.2186), ('film', 0.1982), ('best', 0.1791)]
Topic 13: [('great', 0.4389), ('capitalism', 0.4132), ('year', 0.1817), ('parasite', 0.1553), ('communism', 0.1023)]
Topic 14: [('thought', 0.4671), ('year', 0.3315), ('lol', 0.2635), ('capitalism', 0.227), ('anime', 0.2095)]
Topic 15: [('great', 0.5787), ('thought', 0.2757), ('best', 0.1791), ('watch', 0.1732), ('good', 0.1434)]
Topic 16: [('respect', 0.8727), ('love', 0.1829), ('movies', 0.1681), ('best', 0.0797), ('thought', 0.0767)]
Topic 17: [('love', 0.5237), ('yes', 0.2999), ('movies', 0.2268), ('lol', 0.1421), ('chris', 0.1231)]
Topic 18: [('lol', 0.5958), ('anne', 0.3307), ('curtis', 0.3124), ('yes', 0.2403), ('really', 0.1185)]
Topic 19: [('yes', 0.5965), ('thank', 0.4367), ('analysis', 0.2451), ('really', 0.1864), ('amazing', 0.1839)]
```

Topic Modeling -LSA(Latent Semantic Analysis)

7. words 에서 target_word 의 인덱스를 찾아서 word_idx 에 저장. plotting 을 통해서 targetword와 다른 주제 간의 연관성을 그래프로 시각화




```
Topic 0: [('movie', 0.6008), ('quot', 0.3383), ('good', 0.1694), ('watch', 0.1684), ('like', 0.1572)]
Topic 1: [('quot', 0.6791), ('youtube', 0.1358), ('https', 0.1356), ('com', 0.1355), ('www', 0.1345)]
Topic 2: [('watch', 0.4383), ('https', 0.2872), ('youtube', 0.2854), ('com', 0.2851), ('www', 0.2835)]
Topic 3: [('quot', 0.5017), ('movie', 0.3031), ('review', 0.1209), ('year', 0.064), ('chris', 0.0521)]
Topic 4: [('best', 0.3768), ('parasite', 0.3454), ('review', 0.3389), ('film', 0.3287), ('year', 0.2777)]
Topic 5: [('review', 0.6037), ('video', 0.3273), ('great', 0.2525), ('chris', 0.1036), ('thank', 0.0958)]
Topic 6: [('video', 0.5788), ('good', 0.2814), ('great', 0.2668), ('really', 0.12), ('analysis', 0.1023)]
Topic 7: [('parasite', 0.8339), ('thought', 0.1734), ('anime', 0.1307), ('joker', 0.0703), ('lighthouse', 0.0643)]
Topic 8: [('good', 0.7859), ('movies', 0.1768), ('year', 0.1661), ('really', 0.1222), ('chris', 0.0599)]
Topic 9: [('film', 0.6408), ('watch', 0.413), ('great', 0.1527), ('family', 0.0718), ('analysis', 0.0677)]
Topic 10: [('like', 0.5576), ('capitalism', 0.2621), ('thought', 0.1661), ('movies', 0.1642), ('film', 0.1606)]
Topic 11: [('watch', 0.5897), ('movies', 0.3512), ('like', 0.1493), ('capitalism', 0.1337), ('chris', 0.1118)]
Topic 12: [('capitalism', 0.513), ('good', 0.2222), ('watch', 0.2186), ('film', 0.1982), ('best', 0.1791)]
Topic 13: [('great', 0.4389), ('capitalism', 0.4132), ('year', 0.1817), ('parasite', 0.1553), ('communism', 0.1023)]
Topic 14: [('thought', 0.4671), ('year', 0.3315), ('lol', 0.2635), ('capitalism', 0.227), ('anime', 0.2095)]
Topic 15: [('great', 0.5787), ('thought', 0.2757), ('best', 0.1791), ('watch', 0.1732), ('good', 0.1434)]
Topic 16: [('respect', 0.8727), ('love', 0.1829), ('movies', 0.1681), ('best', 0.0797), ('thought', 0.0767)]
Topic 17: [('love', 0.5237), ('yes', 0.2999), ('movies', 0.2268), ('lol', 0.1421), ('chris', 0.1231)]
Topic 18: [('lol', 0.5958), ('anne', 0.3307), ('curtis', 0.3124), ('yes', 0.2403), ('really', 0.1185)]
Topic 19: [('yes', 0.5965), ('thank', 0.4367), ('analysis', 0.2451), ('really', 0.1864), ('amazing', 0.1829)]
```

Topic Modeling -LDA(Latent Dirichlet Allocation)

LDA란?
토픽모델링을 위한 확률적 모델로서 문서의 토픽 구조를 추론하는 기법

```
from gensim import corpora

# 개별 텍스트를 토큰화하여 배열로 저장
tokenized_comments = [comment.split() for comment in comment_df['preprocessed_comment']]

# 전체 텍스트 배열을 Dictionary의 입력으로 사용
word_dict = corpora.Dictionary(tokenized_comments)

# 각 텍스트를 Bag-of-Words 형식으로 변환
corpus = [word_dict.doc2bow(text) for text in tokenized_comments]
```

1. tokenized_comments 리스트에는 각 텍스트를 토큰화하여 배열로 저장

2. word_dict는 모든 텍스트를 담고 있는 Dictionary 객체로서 토큰화된 단어들을 이용하여 Dictionary를 생성

3. Bag-of-Words 변환: corpus 리스트에는 각 텍스트를 Bag-of-Words 형식으로 변환한 결과가 저장한 후, doc2bow() 함수를 사용하여 각 텍스트를 단어의 빈도 수로 표현된 벡터로 변환

```
# Dictionary에 있는 단어와 인덱스 출력
word_dict_items = [(idx, word) for word, idx in word_dict.items()]
print(word_dict_items)
```

```
[('broeydeschanel', 0), ('com', 1), ('consider', 2), ('content', 3), ('guys', 4), ('hey', 5), ('href', 6), ('https', 7), ('like', 8), ('means', 9), ('patreon', 10
('supporting', 12), ('thanks', 13), ('watching', 14), ('www', 15), ('great', 16), ('movie', 17), ('terrifying', 18), ('beautiful', 19), ('instance', 20), ('loved'
e', 22), ('ooh', 23), ('park', 24), ('resolved', 25), ('sex', 26), ('worker', 27), ('anyone', 28), ('bong', 29), ('case', 30), ('checking', 31), ('coming', 32), (
('filmmaker', 34), ('first', 35), ('hyomin', 36), ('interested', 37), ('interview', 38), ('joon', 39), ('korea', 40), ('meet', 41), ('parasite', 42), ('scnmfbassf
44), ('son', 45), ('video', 46), ('watch', 47), ('youtube', 48), ('add', 49), ('open', 50), ('option', 51), ('portuguese', 52), ('put', 53), ('subs', 54), ('subti
d', 56), ('lecture', 57), ('socialist', 58), ('vile', 59), ('anne', 60), ('commercial', 61), ('curtis', 62), ('hahahaha', 63), ('tho', 64), ('american', 65), ('de
ough', 67), ('family', 68), ('may', 69), ('never', 70), ('parasitic', 71), ('poor', 72), ('thought', 73), ('affected', 74), ('backed', 75), ('cant', 76), ('corner
8), ('get', 79), ('idea', 80), ('intentionally', 81), ('ironic', 82), ('kept', 83), ('part', 84), ('people', 85), ('see', 86), ('stupid', 87), ('truly', 88), ('un
specially', 90), ('landlords', 91), ('parasites', 92), ('rich', 93), ('society', 94), ('alright', 95), ('cleanliness', 96), ('constructs', 97), ('garbage', 98), (
utting', 100), ('said', 101), ('social', 102), ('care', 103), ('politician', 104), ('battle', 105), ('becomes', 106), ('dinner', 107), ('every', 108), ('potential
l', 110), ('agenda', 111), ('appreciate', 112), ('capitalism', 113), ('characters', 114), ('due', 115), ('explaining', 116), ('film', 117), ('liberal', 118), ('li
g', 120), ('medical', 121), ('plot', 122), ('read', 123), ('reason', 124), ('simultaneously', 125), ('sure', 126), ('unaware', 127), ('unknown', 128), ('absolute'
0), ('among', 131), ('asian', 132), ('back', 133), ('bad', 134), ('bait', 135), ('believe', 136), ('belong', 137), ('bottom', 138), ('cataclysmic', 139), ('change
141), ('communistic', 142), ('concept', 143), ('confuse', 144), ('day', 145), ('decisions', 146), ('designed', 147), ('disappear', 148), ('dos', 149), ('economic'
151), ('either', 152), ('els', 153), ('event', 154), ('false', 155), ('friendly', 156), ('gamble', 157), ('gatekeepers', 158), ('hard', 159), ('harder', 160), ('h
d', 162), ('hours', 163), ('ideas', 164), ('integrity', 165), ('keep', 166), ('know', 167), ('long', 168), ('loose', 169), ('make', 170), ('many', 171), ('member'
g', 173), ('much', 174), ('need', 175), ('norm', 176), ('nothing', 177), ('obscurity', 178), ('observe', 179), ('one', 180), ('plus', 181), ('pore', 182), ('premi
e', 184), ('puts', 185), ('right', 186), ('seem', 187), ('seen', 188), ('smarter', 189), ('sorts', 190), ('starting', 191), ('succeed', 192), ('successful', 193),
hings', 195), ('top', 196), ('transitory', 197), ('ultra', 198), ('want', 199), ('war', 200), ('wealth', 201), ('word', 202), ('work', 203), ('love', 204), ('vid'
06), ('effects', 207), ('following', 208), ('ignore', 209), ('last', 210), ('late', 211), ('lockdowns', 212), ('pretty', 213), ('saw', 214), ('stage', 215), ('the
r', 217), ('better', 218), ('away', 219), ('buy', 220), ('echo', 221), ('hear', 222), ('meters', 223), ('mic', 224), ('real', 225), ('room', 226), ('always', 227)
ies', 229), ('host', 230), ('kills', 231), ('came', 232), ('defeating', 233), ('price', 234), ('say', 235), ('soviets', 236), ('child', 237), ('culture', 238), ('
a', 240), ('interpreting', 241), ('intricacies', 242), ('korean', 243), ('summer', 244), ('sweet', 245), ('talks', 246), ('thats', 247), ('westerner', 248), ('bru
ed', 250), ('glasses', 251), ('life', 252), ('look', 253), ('merciless', 254), ('rose', 255), ('analysis', 256), ('dimensional', 257), ('heavily', 258), ('ideolog
gent', 260), ('nuanced', 261), ('projects', 262), ('really', 263), ('sounds', 264), ('figuring', 265), ('lot', 266), ('waking', 267), ('actively', 268), ('america
e', 270), ('backs', 271), ('bail', 272), ('burned', 273), ('chemicals', 274), ('class', 275), ('contrast', 276), ('control', 277), ('east', 278), ('events', 279),
```

Topic Modeling -LDA(Latent Dirichlet Allocation)

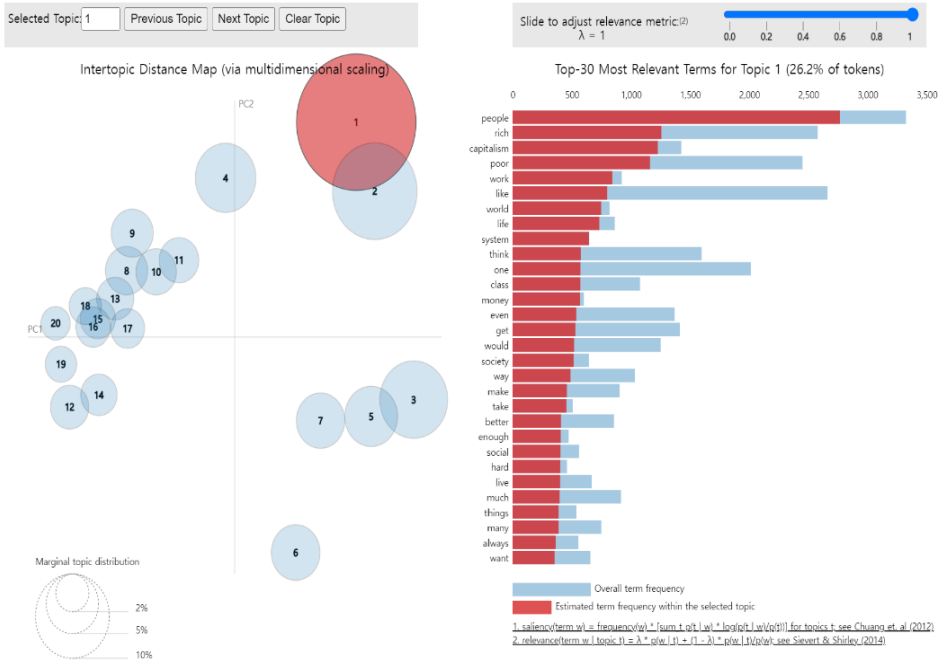
4.LDA model train: ldamodel은 LDA 모델 객체임. ldaModel 클래스를 사용하여 토픽의 개수를 지정하고, 훈련 데이터인 corpus와 Dictionary 객체인 word_dict를 입력으로 사용하여 모델을 훈련

5.토픽 추출: topics는 훈련된 LDA 모델에서 추출된 토픽들임. print_topics() 함수를 사용하여 각 토픽의 단어와 가중치를 출력

```
#LDA 모델 훈련
import gensim
N_TOPICS = 20
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = N_TOPICS, id2word=word_dict, passes = 15)

topics = ldamodel.print_topics(num_words=4)
for topic in topics:
    print(topic)
```

```
(0, '0.220*"quot" + 0.022*"korea" + 0.014*"korean" + 0.012*"south"')
(1, '0.025*"parasites" + 0.019*"maybe" + 0.015*"put" + 0.012*"host"')
(2, '0.061*"movie" + 0.045*"year" + 0.039*"movies" + 0.022*"love"')
(3, '0.071*"best" + 0.066*"film" + 0.062*"parasite" + 0.035*"movie"')
(4, '0.062*"guy" + 0.046*"thought" + 0.038*"scene" + 0.023*"basement"')
(5, '0.152*"movie" + 0.045*"good" + 0.036*"really" + 0.034*"amazing"')
(6, '0.037*"people" + 0.017*"rich" + 0.016*"capitalism" + 0.015*"poor"')
(7, '0.087*"movie" + 0.059*"seen" + 0.026*"see" + 0.024*"ever"')
(8, '0.028*"opinion" + 0.022*"mean" + 0.018*"like" + 0.015*"videos"')
(9, '0.040*"god" + 0.035*"plot" + 0.024*"totally" + 0.022*"john"')
(10, '0.050*"joker" + 0.020*"pretty" + 0.018*"still" + 0.017*"better"')
(11, '0.168*"great" + 0.088*"video" + 0.032*"wow" + 0.028*"watching"')
(12, '0.032*"skin" + 0.026*"white" + 0.022*"class" + 0.019*"anne"')
(13, '0.067*"waiting" + 0.043*"capitalism" + 0.022*"late" + 0.017*"reviewing"')
(14, '0.141*"watch" + 0.082*"amp" + 0.074*"youtube" + 0.065*"https"')
(15, '0.044*"family" + 0.035*"rich" + 0.034*"poor" + 0.016*"like"')
(16, '0.032*"respect" + 0.027*"shirt" + 0.025*"looking" + 0.016*"want"')
(17, '0.057*"two" + 0.017*"funny" + 0.015*"left" + 0.012*"theater"')
(18, '0.028*"second" + 0.027*"bad" + 0.026*"ending" + 0.025*"favorite"')
(19, '0.185*"review" + 0.140*"chris" + 0.051*"please" + 0.032*"thanks"')
```

Topic Modeling -LDA(Latent Dirichlet Allocation)




```
#LDA 모델 시각화
import pyLDAvis.gensim_models

pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim_models.prepare(ldamodel, corpus, word_dict)
pyLDAvis.display(vis)
```

6. pyLDAvis 라이브러리를 사용하여 LDA 모델의 결과를 시각화해줌. prepare() 함수를 사용하여 시각화에 필요한 데이터를 생성하고, display() 함수를 사용하여 시각화 결과를 표시

Topic Modeling -LDA(Latent Dirichlet Allocation)

```
def get_topic_ratio_for_each_document(ldamodel, corpus):
    results = []
    for i, topic_list in enumerate(ldamodel[corpus]):
        sorted_topic_list = sorted(topic_list, key = lambda x: x[1], reverse=True)
        most_important_topic, most_important_ratio = sorted_topic_list[0]
        doc_result = [i, most_important_topic, most_important_ratio, sorted_topic_list]
        results.append(doc_result)

    results = pd.DataFrame(results, columns = ['INDEX', '가장 비중이 높은 토픽', '가장 높은 토픽의 비중', '각 토픽의 비중'])
    return results
```

```
df_results = get_topic_ratio_for_each_document(ldamodel, corpus)
df_results
```

	INDEX	가장 비중이 높은 토픽	가장 높은 토픽의 비중	각 토픽의 비중
0	0	19	0.393274	[(19, 0.39327437), (14, 0.34346095), (5, 0.085...
1	1	5	0.512473	[(5, 0.51247287), (11, 0.2625141), (0, 0.01250...
2	2	15	0.704276	[(15, 0.70427614), (5, 0.10525825), (4, 0.1052...
3	3	14	0.408356	[(14, 0.40835637), (15, 0.39811322), (5, 0.061...
4	4	17	0.459345	[(17, 0.4593452), (3, 0.18648009), (6, 0.10257...
...	...	...	...	...
11028	11028	1	0.508344	[(1, 0.5083445), (19, 0.17501038), (5, 0.17496...
11029	11029	7	0.445581	[(7, 0.4455807), (2, 0.32941276), (0, 0.012500...
11030	11030	5	0.524998	[(5, 0.52499753), (0, 0.02500013), (1, 0.02500...
11031	11031	2	0.524969	[(2, 0.52496874), (0, 0.025001645), (1, 0.0250...
11032	11032	0	0.050000	[(0, 0.05), (1, 0.05), (2, 0.05), (3, 0.05), (...

11033 rows × 4 columns

Type *Markdown* and LaTeX: α^2

각 문서의 토픽 비중 계산:

•get_topic_ratio_for_each_document() 함수는 각 문서에 대해 가장 비중이 높은 토픽과 해당 토픽의 비중, 그리고 모든 토픽의 비중을 계산한 후, 결과는 데이터프레임으로 반환하는 코드

Conclusions

- ✓ 댓글의 감성은 약 81%의 정확도로 분류할 수 있었다.
- ✓ Youtube댓글은 독특한 데이터와 의사소통의 형태로서 모델들은 긍정적인 댓글과 부정적인 댓글의 유의미한 차이점을 예측하는 데에 어려움을 겪었다.
- ✓ 긍정 -부정 단어 분류에서 그다지 구분되는 단어들이 나오지 않는 이유는 영화 자체의 주제가 긍정적인 분위기가 아닌 사회 비판적인 내용이다.
- ✓ 또한 영화에서는 코믹적, 감동적, 비극적 요소가 혼합된 장면이 자주 등장한다. 이러한 장면의 연속은 배경지식이 없는 해외 네티즌들로 하여금 복잡하며 긍정 부정 단어를 분리하기 어렵게 만들 수 있다.
- ✓ 전반적으로 국 내외에서 높은 평가와 호평을 받은 영화로서 긍정적인 반응의 비율이 높은 것을 확인하였다.
- ✓ 해당 영화 ‘기생충’이 사회비판적, 사회주의 적인 주제를 가진 만큼 ‘Capitalism’의 빈도가 높은 것은 이 영화에 대한 관심과 긍정의 비율이 높은 것으로 판단 할 수 있을 것이다.

Conclusions -보완점 & 성능에 관한 고찰

“보완점“

: 대용량 데이터에 더욱 적합한 토픽 모델링을 위해서 추가로 또 다른 parasite review영상 댓글 데이터를 수집할 수 있습니다.

: 해당 영화 리뷰 맞춤 감성 사전을 임의로 제작해서 더욱 유의미한 차이점을 예측 가능합니다.

: 모델 앙상블을 활용하여 성능을 향상 시킬 수 있습니다.

“성능에 관한 고찰“

위의 언급 한 것 과 같이
LR은 모델 파라미터 의 개수가 적어 비교적 간단한 모델 조직을 가지기 대문에 과적합(overfitting)의 가능성이 적습니다.
학습과 예측 속도가 빨라서 대용량 데이터도 빠른 학습이 가능합니다. 또한, 희소(sparse)한 데이터에 대해 잘 작동하는 경향이 있습니다.

우리가 분석하는 텍스트 데이터는 일반적으로 희소행렬(sparse matrix)의 형태이기 때문에 실제로 사용되는 단어의 비율이 상대적으로 적다. 그렇기의 LR모델의 성능이 높게 나온 것이라는 결론을 도출했습니다.

End of documents