Week15_예습과제_우정연

8.6 토픽 모델링(Topic Modeling) - 20 뉴스그룹

- 토픽 모델링(Topic Modeling)
  - 문서 집합에 숨어 있는 주제를 찾아내는 것
  - 많은 양의 문서에 머신러닝 기반의 토픽 모델링을 적용해 숨어 있는 중요 주제를 효과적으로 찾아낼 수 있음
  - 사람이 수행하는 토픽 모델링: 더 함축적인 의미로 문장을 요약
  - 머신러닝 기반의 토픽 모델링: 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
- LSA(Latent Semantic Analysis)
- LDA(Latent Dirichlet Allocation)
  - 차원 축소의 LDA(Linear Discriminant Analysis)와 서로 다른 알고리즘임을 유의!
- 20 뉴스 그룹 데이터 세트를 이용한 토픽 모델링 적용
  - 사이킷런은 LDA 기반의 토픽 모델링을 LatentDirichletAllocation 클래스로 제공
  - LDA 토픽 모델링을 위해 fetch_20newsgroups() API는 categories 파라미터를 통해 필요한 주제만 필터링해 추출하고 추출된 텍스트를 Count 기반으로 벡터화 변환
  - LDA는 Count 기반의 벡터화만 사용

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# 모토사이클, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 전자공학, 의학 8개 주제를 추출
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x',
        'talk.politics.mideast', 'soc.religion.christian', 'sci.electronics', 'sci.med']

# 위에서 cats 변수로 기재된 카테고리만 추출. featch_20newsgroups()의 categories에 cats 입력
news_df = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'),
                             categories = cats, random_state = 0)

# LDA는 Count 기반의 벡터화만 적용
count_vect = CountVectorizer(max_df = 0.95, max_features = 1000, min_df = 2, stop_words = 'english',
                             ngram_range = (1,2))
feat_vect = count_vect.fit_transform(news_df.data)
print('CountVectorizer Shape: ', feat_vect.shape)
```

```
CountVectorizer Shape:  (7862, 1000)
```

    - CountVectorizer 객체 변수인 feat_vect 모두 7862개의 문서가 1000개의 피처로 구성된 행렬 데이터임
- 피처 벡터화된 데이터 세트 기반으로 LDA 토픽 모델링 수행
- 토픽 개수는 8개
- LatentDirichletAllocation 클래스의 n_components 파라미터를 이용해 토픽 개수를 조정

```
lda = LatentDirichletAllocation(n_components = 8, random_state = 0)
lda.fit(feat_vect)
print(lda.components_.shape)
lda.components_
```

```
(8, 1000)
array([[3.60992018e+01, 1.35626798e+02, 2.15751867e+01, ...,
        3.02911688e+01, 8.66830093e+01, 6.79285199e+01],
       [1.25199920e-01, 1.44401815e+01, 1.25045596e-01, ...,
        1.81506995e+02, 1.25097844e-01, 9.39593286e+01],
       [3.34762663e+02, 1.25176265e-01, 1.46743299e+02, ...,
        1.25105772e-01, 3.63689741e+01, 1.25025218e-01],
       ...,
       [3.60204965e+01, 2.08640688e+01, 4.29606813e+00, ...,
        1.45056650e+01, 8.33854413e+00, 1.55690009e+01],
       [1.25128711e-01, 1.25247756e-01, 1.25005143e-01, ...,
        9.17278769e+01, 1.25177668e-01, 3.74575887e+01],
       [5.49258690e+01, 4.47009532e+00, 9.88524814e+00, ...,
        4.87048440e+01, 1.25034678e-01, 1.25074632e-01]])
```

    - components_의 형태와 속성값
        - array[8,4000]으로 구성
        - 8개의 토픽별로 1000개의 word 피처가 해당 토픽 별로 연관도 값을 가지고 있음
- lda_model.components_ 값만으로는 각 토픽 별 word 연관도를 보기가 어려움
- display_topics() 함수 만들어 각 토픽 별로 연관도가 높은 순으로 Word를 나열

```python
def display_topics(model, feature_names, no_top_words):
  for topic_index, topic in enumerate(model.components_):
    print('Topic #', topic_index)

    # components_array에서 가장 값이 큰 순으로 정렬했을 때, 그 값의 array 인덱스를 반환
    topic_word_indexes = topic.argsort()[::-1]
    top_indexes = topic_word_indexes[:no_top_words]

    # top_indexes 대상인 인덱스 별로 feature_names에 해당하는 word feature 추출 후 join으로 concat
    feature_concat = ' '.join([feature_names[i] for i in top_indexes])
    print(feature_concat)

# CountVectorizer 객체 내의 전체 word의 명칭을 get_feature_names()를 통해 추출
feature_names = count_vect.get_feature_names_out()

# 토픽 별 가장 연관도가 높은 word를 15개만 추출
display_topics(lda, feature_names, 15)
```

```
Topic # 0
year 10 game medical health team 12 20 disease cancer 1993 games years patients good
Topic # 1
don just like know people said think time ve didn right going say ll way
Topic # 2
image file jpeg program gif images output format files color entry 00 use bit 03
Topic # 3
like know don think use does just good time book read information people used post
Topic # 4
armenian israel armenians jews turkish people israeli jewish government war dos dos turkey arab armenia 000
Topic # 5
edu com available graphics ftp data pub motif mail widget software mit information version sun
Topic # 6
god people jesus church believe christ does christian say think christians bible faith sin life
Topic # 7
use dos thanks windows using window does display help like problem server need know run
```

8.7 문서 군집화 소개와 실습(Opinion Review 데이터 세트)

[문서 군집화 개념]

- 문서 군집화(Opinion Clustering)
  - 비슷한 텍스트 구성의 문서를 군집화(Clustering)하는 것
  - 동일한 군집에 속하는 문서를 같은 카테고리 소속으로 분류할 수 있으므로 텍스트 분류 기반의 문서 분류와 유사
  - 텍스트 분류 기반의 문서 분류는 사전에 결정 카테고리 값을 가진 학습 데이터 세트가 필요한 데 반해, 문서 군집화는 학습 데이터 세트가 필요없는 비지도 학습 기반으로 동작함

[Opinion Review 데이터 세트를 이용한 문서 군집화 수행하기]

- UCI 머신러닝 리포지터리의 Opinion Review 데이터 세트 사용
- 여러 개의 파일을 한 개의 Dataframe으로 로딩해 데이터 처리
  - 해당 디렉터리 내 파일을 하나씩 읽어서 파일명과 파일 리뷰를 하나의 Dataframe으로 로드하여 파일 명 별로 어떤 리뷰를 담고 있는지 살펴봄

```
import pandas as pd
import glob, os

from google.colab import drive
drive.mount('/content/drive')

# 각자 디렉터리 설정
path = 'drive/My Drive/Colab Notebooks/data/OpinosisDataset1.0/topics'
# path로 지정한 디렉터리 밑에 있는 모든 .data 파일의 파일명을 리스트로 취합
all_files = glob.glob(os.path.join(path, "*.data"))
filename_list = []
opinion_text = []

# 개별 파일의 파일명은 filename_list 로 취합
# 개별 파일의 파일 내용은 Dataframe 로딩 후 다시 string으로 변환해 opinion_text list로 취합
for file_ in all_files:
  # 개별 파일을 읽어서 DataFrame으로 생성
  df = pd.read_table(file_, index_col = None, header = 0, encoding = 'latin1')

  # 절대 경로로 주어진 파일명을 가공. 리눅스에서 수행할 때는 다음 \\를 /로 변경
  # 맨 마지막 .data 확장자도 제거
  filename_ = file_.split('/')[-1]
  filename = filename_.split('.')[0]

  # 파일명 list와 파일 내용 list에 파일명과 파일 내용을 추가
  filename_list.append(filename)
  opinion_text.append(df.to_string())

# 파일명 list와 파일 내용 list 객체를 DataFrame으로 생성
document_df = pd.DataFrame({'filename':filename_list, 'opinion_text':opinion_text})
document_df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
```

	filename	opinion_text
0	battery-life_ipod_nano_8gb	...
1	speed_windows7	...
2	voice_garmin_nuvi_255W_gps	...
3	accuracy_garmin_nuvi_255W_gps	...
4	display_garmin_nuvi_255W_gps	...

    - 각 파일 이름 자체만으로 의견(opinion)의 텍스트(text)가 어떤 제품/서비스에 대한 리뷰인지 잘 알 수 있음

- 문서를 TF-IDF 형태로 피처 벡터화
    - tokenizer: Lemmatization을 구현한 LemNormalize() 함수를 이용
    - ngram: (1, 2)
    - min_df와 max_df 범위를 설정해 피처의 개수를 제한
    - TfidfVectorizer의 fit_transform()의 인자로 document_df DataFrame의 opinion_text 칼럼을 입력하면 개별 문서 텍스트에 대해 TF-IDF 변환된 피처 벡터화된 행렬을 구할 수 있음
- 문서별 텍스트가 TF-IDF 변환된 피처 벡터화 행렬 데이터에 대해서 군집화를 수행해 어떤 문서끼리 군집되는지 확인

- 군집화 - Kmeans 적용
- 5개의 중심(Centroid) 기반, 최대 반복 횟수 max_iter = 10000

[군집 별 핵심 단어 추출하기]

- 각 군집에 속한 문서는 핵심 단어를 주축으로 군집화되어 있을 것
- KMeans 객체는 각 군집을 구성하는 단어 피처가 군집의 중심을 기준으로 얼마나 가깝게 위치해 있는지 cluster_centers_라는 속성으로 제공
  - cluster_centers_는 배열 값으로 제공
  - 행은 개별 군집을, 열은 개별 피처를 의미
  - 각 배열 내의 값은 개별 군집 내의 상대 위치를 숫자 값으로 표현한 일종의 좌표 값
  - cluster_centers_는 (3, 2409) 배열 → 군집이 3개 word피처가 2409개로 구성 의미
    - 1에 가까운 값을 가질수록 중심과 가까운 값을 의미
- 핵심 단어 피처의 이름 출력 위해
  - ndarray의 argsort( )[:,::-1]을 이용 → cluster_centers 배열 내 값이 큰 순으로 정렬된 위치 인덱스 값을 반환
- get_cluster_details() 함수 생성
  - cluster_centers_ 배열 내에서 가장 값이 큰 데이터의 위치 인덱스를 추출

    → 해당 인덱스를 이용해 핵심 단어 이름과 그때의 상대 위치 값을 추출해 cluster_details라는 Dict 객체 변수에 기록하고 반환
  - dictionary를 원소로 가지는 리스트인 cluster_details를 반환
    - cluster_details에는 개별 군집번호, 핵심 단어, 핵심단어 중심 위치 상댓값, 파일명 속성 값 정보가 있음

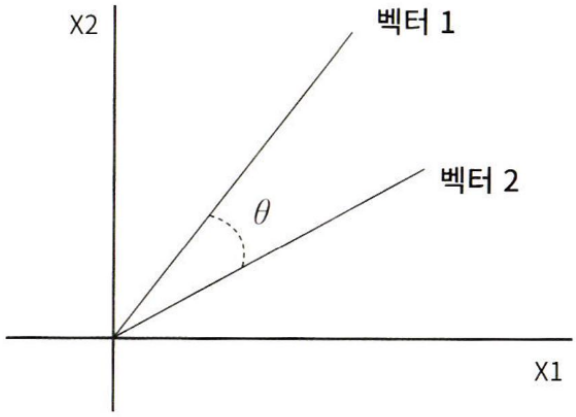
      → 더 보기 좋게 표현하기 위해 별도의 print_cluster_details() 함수 생성

8.8 문서 유사도

[문서 유사도 측정 방법 - 코사인 유사도]

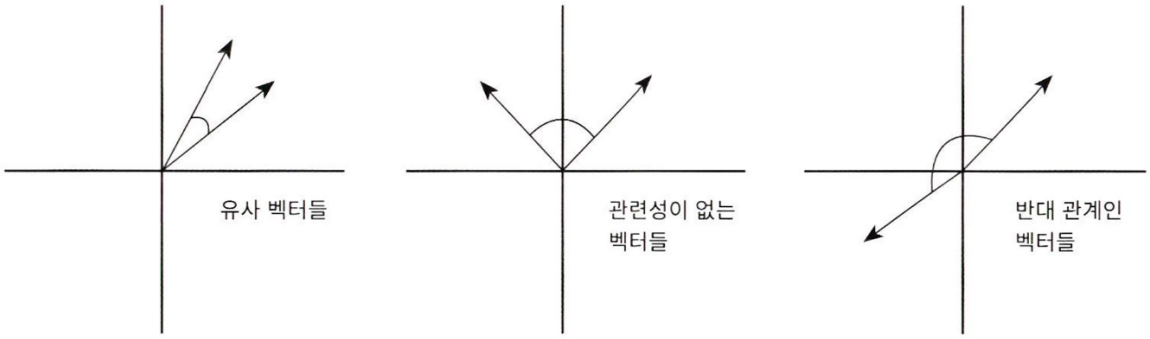
- 문서와 문서 간의 유사도 비교는 일반적으로 코사인 유사도(Cosine Similarity)를 사용

- 코사인 유사도
  - 벡터와 벡터 간의 유사도를 비교할 때 벡터의 크기보다는 벡터의 상호 방향성이 얼마나 유사한지에 기반
  - 두 벡터 사이의 사잇각을 구해서 얼마나 유사한지 수치로 적용한 것



[두 벡터 사잇각]

- 두 벡터의 사잇각에 따라 상호 관계는 유사하거나 관련이 없거나 아예 반대 관계가 될 수 있음



- 내적 값

$$A * B = \| A \| \| B \| cos\theta$$

- 유사도 코사인

$$\text{similarity} = cos\theta = \frac{A \cdot B}{\| A \| \| B \|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- 코사인 유사도가 문서의 유사도 비교에 가장 많이 사용되는 이유
  - 문서를 피처 벡터화 변환하면 차원이 매우 많은 희소 행렬이 되기 쉬움

    → 이러한 희소 행렬 기반에서 문서와 문서 벡터 간의 크기에 기반한 유사도 지표는 정확도가 떨어지기 쉬움

    → 또한 문서가 매우 긴 경우 단어의 빈도수도 더 많을 것이므로 빈도수에만 기반해서는 공정한 비교를 할 수 없음
- 사이킷런의 sklearn.metrics.pairwise.cosine_similarity API
  - 코사인 유사도 측정을 위해 제공

[Opinion Review 데이터 세트를 이용한 문서 유사도 측정]

- 문서를 벡터화해 변환하면 문서 내 단어에 출현 빈도와 같은 값을 부여해 각 문서가 단어 피처의 값으로 벡터화됨
- 각 문서가 피처 벡터화된 데이터를 cosine_similarity( )를 이용해 상호 비교해 유사도 확인

8.9 한글 텍스트 처리 - 네이버 영화 평점 감성 분석

- 네이버 영화 평점 데이터를 기반으로 감성 분석 적용
- 한글 NLP 처리에서 주의할 점과 대표적인 파이썬 기반의 한글 형태소 패키지인 KoNLPy를 소개

[한글 NLP 처리의 어려움]

- 한글 언어 처리가 영어 등의 라틴어 처리보다 어려운 이유
  - '띄어쓰기'와 '다양한 조사'
  - 한글: 띄어쓰기를 잘못하면 의미가 왜곡되어 전달될 수 있음, 어려움

영어: 띄어쓰기를 잘못하면 의미왜곡이 아닌 잘못된 또는 없는 단어로 인식함 → 분석에서 제외 가능, 쉬움

- 조사는 주어나 목적어를 위해 추가되며, 경우의 수가 많아 어근 추출(Stemming/Lemmatization)등의 전처리 시 제거하기가 까다로움

[KoNLPy 소개]

- KoNLPy
  - 파이썬의 대표적인 한글 형태소 패키지
    - 형태소: 단어로서 의미를 가지는 최소 단위
    - 형태소 분석: 말뭉치를 형태소 어근 단위로 쪼개고 각 형태소에 품사 태깅을 부착하는 작업
  - KoNLPy 이전에는 파이썬 기반의 형태소 분석 프로그램이 거의 없었음
    - 대부분 C/ C++, Java 기반 패키지로 개발됨
    - KoNLPy는 기존의 C/C++, Java로 만들어진 한글 형태소 엔진을 파이썬 래퍼 기반으로 재작성한 패키지
    - 기존의 엔진을 그대로 유지한 채 파이썬 기반에서 인터페이스 제공 → 검증된 패키지의 안정성을 유지할 수 있음
    - 꼬꼬마(Kkma), 한나눔(Hannanum), Komoran, 은전한닢 프로젝트(Mecab), Twitter - 5개의 형태소 분석 모듈을 모두 사용 가능
      - Mecab의 경우 윈도우 환경에서 구동 X - 리눅스 환경에서만 가능

[데이터 로딩]

- 네이버 영화 평점 데이터: https://github.com/e9t/nsmc 에서 다운로드 가능
- 0/1 비율이 균등한 분포
- train_df, test_df 데이터 전처리
  - ‘document’ 칼럼에 Null이 일부 존재 → 공백으로 변환
  - 숫자의 경우 단어적인 의미로 부족하므로 파이썬의 정규 표현식 모듈인 re를 이용해 공백으로 변환
- TF-IDF 방식 벡터화

    - 각 문장을 한글 형태소 분석을 통해 형태소 단어로 토큰화
    - SNS 분석에 적합한 Twitter 클래스 이용
        - Twitter 객체의 morphs() 메서드를 이용하면 입력 인자로 들어온 문장을 형태소 단어 형태로 토큰화해 list 객체로 반환
- 로지스틱 회귀를 이용해 분류 기반의 감성 분석 수행
    - 로지스틱 회귀의 하이퍼 파라미터 C의 최적화를 위해 GridSearchCV를 이용
- test 세트를 이용해 최종 감성 분석 예측 수행
    - 학습할 때 적용한 TfidfVectorizer를 그대로 사용해야 함

      → 학습 시 설정된 TfidfVectorizer의 피처 개수와 테스트 데이터를 TfidfVectorizer로 변환할 피처 개수가 같아짐