**문서 유사도 계산**

1. 유클리드 거리 활용 유사도 계산 실습

- TDM 만들어서 저장 해놓은 전파누리 뉴스 TDM호출

**from** **sklearn.externals** **import** joblib

**import** **pandas**

**def** get\_nouns(): #저장 할 때 'vectorizer': cv 저장 시 cv가 함수 포함, 없으면 에러

**pass**

**with** open('news.pkl', 'rb') **as** f:

data = joblib.load(f)

locals().update(data) #update는 사전 형태의 기존 디렉토리 데이터 한꺼번에 반영

- 원본 뉴스 호출

article= pd.read\_csv('한국방송통신전파진흥원\_전파누리\_최신뉴스\_20210129.csv',

encoding='cp949',

engine='python' )

- 유클리드 거리를 이용 **0번기사와 1번기사 거리**

**from** **sklearn.metrics.pairwise** **import** euclidean\_distances

euclidean\_distances(tdm[0], tdm[1])

- **0번 기사와 가장 가까운 거리**

**import** **numpy** **as** **np**

dist = euclidean\_distances(tdm[0], tdm[1:])

np.argmin(dist) + 1

- 내용 비교

article.loc[0, '내용']

2. 코사인 유사도 실습

**from** **sklearn.metrics.pairwise** **import** cosine\_similarity

이번에는 0번 기사와 1번 기사의 코사인 유사도를 구해본다.

cosine\_similarity(tdm[0], tdm[1])

array([[0.048795]])

0번 기사와 유사도 가장 높은 문서를 찾는다.

sim = cosine\_similarity(tdm[0], tdm[1:])

np.argmax(sim) + 1

1394

article.loc[1394, '본문']

**문서 클러스트링**

1. 스펙트럴 클러스트링(디폴트 유클리드 거리 활용)

- 저장 해 놓은 아마존 리뷰 TDM 호출 및 원본 호출

**import** **pandas** **as** **pd**

**from** **sklearn.externals** **import** joblib

**with** open('amazon.pkl', 'rb') **as** f:

data = joblib.load(f)

locals().update(data)

df = pd.read\_csv('amazon\_cells\_labelled.txt', sep="**\t**", header=None)

- 스펙트럴 클러스트링

**from** **sklearn.cluster** **import** SpectralClustering

문서들을 유클리드 거리를 기준으로 4개의 클러스터(n\_clusters)로 묶는다.

cl = SpectralClustering(n\_clusters=4, random\_state=1234)

labels = cl.fit\_predict(tdm[:100])

- 클러스트별 많이 나온 단어 확인->소스 복사

**from** **operator** **import** itemgetter

words = vectorizer.get\_feature\_names()

**def** top10(labels):

freq\_words = []

**for** i **in** range(4):

count = tdm[labels == i, :].sum(axis=0)

ws = [w **for** w, n **in** sorted(zip(words, count.flat), key=itemgetter(1), reverse=True)[:10]]

freq\_words.append(ws)

**return** pd.DataFrame(freq\_words)

top10(labels)

2. 스펙트럴 클러스트링(코사인 유사도)

**from** **sklearn.metrics.pairwise** **import** cosine\_similarity

cl\_cos = SpectralClustering(n\_clusters=4, affinity=cosine\_similarity, random\_state=1234)

labels\_cos = cl\_cos.fit\_predict(tdm[:100])

top10(labels\_cos)

3. KMeans 클러스트링

**from** **sklearn.cluster** **import** KMeans

km = KMeans(n\_clusters=4, random\_state=1234)

labels\_km = km.fit\_predict(tdm)

top10(labels\_km)

**주제 분석**

1. LSA 예제 분석

- 데이터 준비-tfidf로 vectorizer된 tdm

**import** **pandas** **as** **pd**

**from** **sklearn.externals** **import** joblib

**with** open('amazon.pkl', 'rb') **as** f:

data = joblib.lboad(f)

locals().update(data)

tdm.shape

(1000, 1000)

- LSA 적용

특이값 분해로 차원을 축소한다.

**from** **sklearn.decomposition** **import** TruncatedSVD

svd = TruncatedSVD(n\_components=30)

svd.fit\_transform()에 tdm을 넘겨줘서 차원을 축소한다.

pos = svd.fit\_transform(tdm)

pos.shape

- 시각화

**import** **matplotlib.pyplot** **as** **plt**

% matplotlib inline

dim1 = 0

dim2 = 1

plt.plot(pos[sentiment == 1, dim1], pos[sentiment == 1, dim2], 'ro')

*# 긍정적인 코멘트 (r 빨간색, o 원형)*

plt.plot(pos[sentiment == 0, dim1], pos[sentiment == 0, dim2], 'bo') *# 부정적인 코멘트 (파란색)*

- 데이터 정규화

LSA로 만들어진 좌표는 문서의 길이에 영향을 받는다. 이 영향을 제거하기 위해 문서의 원점에서 거리를 1로 변환을 해줄 수 있다.

**from** **sklearn.preprocessing** **import** Normalizer

norm = Normalizer(copy=False)

# 입력이 이미 numpy배열이거나 CSR매트릭스인 경우 False로 설정

pos2 = norm.fit\_transform(pos)

dim1 = 0

dim2 = 1

plt.plot(pos2[sentiment == 1, dim1], pos2[sentiment == 1, dim2], 'ro') *# 긍정적인 코멘트 (빨간색)*

plt.plot(pos2[sentiment == 0, dim1], pos2[sentiment == 0, dim2], 'bo') *# 부정적인 코멘트 (파란색)*

- 파이프라인 만들어 스텝 줄이기

LSA와 노멀라이징처럼 2단계의 변환이 필요한 경우 파이프라인으로 만들어두면 한 번에 처리를 할 수 있다.

사이킷런의 Pipeline 클래스는 연속된 변환을 순차적으로 처리할 수 있는 기능을 제공

**from** **sklearn.pipeline** **import** make\_pipeline

lsa = make\_pipeline(svd, norm)

lsa.transform(tdm)

2. LDA 분석

- genism 설치 필요 ‘!conda install gensim’

- 데이터 로드(뉴스 데이터)

**import** **pandas** **as** **pd**

**from** **sklearn.externals** **import** joblib

**def** get\_nouns():

**pass**

**with** open('news.pkl', 'rb') **as** f:

data = joblib.load(f)

locals().update(data)

- 단어 가져 오기

.get\_feature\_names() 을 통해 각 차원에 해당하는 단어 목록을 가져오자.

words = vectorizer.get\_feature\_names()

단어 목록을 {단어 번호: 단어} 형태의 딕셔너리로 변환한다.

word\_dict = dict(enumerate(words))

word\_dict[0]

'4차 산업'

word\_dict[100]

'광주'

- genism 형태로 변환

tdm을 gensim 형식으로 변환한다.

**from** **gensim.matutils** **import** Sparse2Corpus

Sparse2Corpus()에 tdm의 전치행렬을 넘겨준다.

corpus = Sparse2Corpus(tdm.T)

corpus

- 분석

이제 LDA 분석을 위해 gensim.models에서 LdaModel을 불러오자.

**from** **gensim.models.ldamodel** **import** LdaModel

LdaModel() 함수에 몇 가지 옵션을 넘겨준다.

corpus 옵션에는 생성해놓은 코퍼스를 넘겨준다.

num\_topics 옵션에는 찾는 주제의 수를 넘겨주면 된다. 예제에서는 100가지 주제를 찾도록 지정하자.

기존에 생성한 인덱스와 단어로 이루어진 단어 사전인 word\_dict를 id2word 옵션에 넘겨준다.

passes와 iterations는 계산 횟수를 정해준다. 기본값은 각각 1과 50이나 증가시켜주면 결과가 더 안정적으로 나온다.

LdaModel()은 계산에 랜덤한 부분이 있기 때문에 random\_state를 고정시켜 계산 결과를 일정하게 만들어준다.

lda = LdaModel(corpus=corpus,

num\_topics=100,

passes=3,

iterations=100,

id2word=word\_dict,

random\_state=123)

- 토픽이 포함한 단어 확인

lda.show\_topic(0)

lda.show\_topic(2)

- 문서에 어떤 비율로 주제가 있는지 확인

- 문서의 인덱스와 빈도 생성

row = tdm[0]

doc = list(zip(row.indices, row.data))

doc

- 인덱스를 단어로 변환

doc\_words = [(words[i], n) **for** i, n **in** doc]

doc\_words

- 문서의 주제어 토픽 생성

lda.get\_document\_topics(doc)

- 문서의 토픽 확인

lda.show\_topic(28)

- 모형 저장

lda.save('lda\_test.lda')

- 모형 호출

lda2 = LdaModel.load('lda\_test.lda')