**KoNLPy 설치**

1. java 설치

- <https://java.com/ko/download/manual.jsp>

- Python이 64비트면 Java도 64비트여야 한다. 32비트인 경우도 같다

2. 환경변수 설정

- java 설치 후 환경 변수 설정 필요  
 - C:\Program Files\Java\jre1.8.0\_241 설치 경로 확인

- 제어판/시스템 및 보안/고급 시스템 설정/고급/환경 변수

- 새로만들기(N) 클릭

- 변수명:JAVA\_HOME, 값: C:\Program Files\Java\jre1.8.0\_241 입력

3. Jpype 설치

- 이제 JAVA와 Python을 연결해주는 역할을 하는 JPype를 설치해야 합니다.

- 설치 주소 : <https://www.lfd.uci.edu/~gohlke/pythonlibs/#jpype>

- JPype1‑1.3.0‑cp38‑cp38‑win\_amd64.whl Python3.8/window64 bit

- pip install JPype1‑1.3.0‑cp38‑cp38‑win\_amd64.죄

4. KoNLPy 설치

- pip install konlpy

**형태소 분석**

pos는 part of speech의 약자로 한국어로는 품사를 의미한다.

**from** **konlpy.tag** **import** Komoran

tagger = Komoran() *# 형태소 분석기*

tagger.pos('한국어로 텍스트를 분석해 봅시다.') *# 형태소 분석*

[('한국어', 'NNP'),

('로', 'JKB'),

('텍스트', 'NNG'),

('를', 'JKO'),

('분석', 'NNG'),

('하', 'XSV'),

('아', 'EC'),

('보', 'VX'),

('ㅂ시다', 'EF'),

('.', 'SF')]

많은 경우 명사만을 뽑아 텍스트 분석을 실시한다.

코모란의 경우 .nouns 메소드로 명사만을 추출할 수 있다.

tagger.nouns('한국어로 텍스트를 분석해 봅시다.') *# 명사 추출*

['한국어', '텍스트', '분석']

만약 **형태소 단위**로 문자열을 끊고 싶다면, .morphs()를 사용하면 된다.

tagger.morphs()에 이전에 사용했던 문장을 넘겨주자.

tagger.morphs(' 한국어로 텍스트를 분석해 봅시다.')

['아버지', '가방', '에', '들어가신', '다']

**아마존 제품 리뷰 TDM**

1. 데이터 다운로드

- Amazon 사용자들이 올린 제품 리뷰를 TDM으로 만들어보자. UCI 머신러닝 리포지토리에서에서 Sentiment Labelled Sentences 데이터를 다운받아 'amazon\_cells\_labelled.txt'를 사용한다.

- <https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences>

2. 데이터 불러 오기

- 아마존 제품 리뷰 자료는 한줄 댓글과 함께 해당 제품에 대한 긍/부정 점수가 1과 0으로 기록되어 있다. 따라서 긍/부정 감정사전을 만드는데 용이하다.

- 자료는 탭으로 구분이 이뤄져있으며, 열마다 제목이 없기 때문에 파일을 읽을 때 sep="\t" 옵션과 header=None 옵션을 넘겨줘야 한다

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

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

- 자료를 확인해보니, 댓글이 첫번째 열에 들어있으며 감정 분류가 두번째 열에 들어가있다.

content = df[0]

sentiment = df[1]

3. TDM 만들기

**from** **sklearn.feature\_extraction.text** **import** TfidfVectorizer

tfidf = TfidfVectorizer(stop\_words='english',

lowercase=True,

max\_features=1000)

tdm = tfidf.fit\_transform(content)

tdm

tdm의 정보 확인 가능

- tfidf 뒤에 .get\_feature\_names() 메써드를 부르면 TDM에 사용된 feature 단어 목록을 볼 수 있다.

- 마지막 10개의 단어만 살펴보자

tfidf.get\_feature\_names()[-10:]

4. 저장

향후에 감정분석에서 사용하기 위해 tfidf, tdm, 그리고 df 의 'sentiment' 열을 피클로 저장해두자. joblib은 pickle과 비슷하게 파이썬의 값들을 저장해주는 라이브러리다. pickle에 비해서 수치 데이터를 더 효율적으로 저장한다.

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

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

joblib.dump(

{'vectorizer': tfidf, 'tdm': tdm, 'sentiment': sentiment},

f

)

뉴스 TDM 생성

1. 데이터 다운로드

- 개방 공공 데이터 활용(전파누리 최신 뉴스)

- <https://www.data.go.kr/data/15076857/fileData.do>

2. 데이터 불러오기

import pandas as pd

df2 = pd.read\_csv('한국방송통신전파진흥원\_전파누리\_최신뉴스\_20210129.csv',

encoding='cp949',

engine='python' )

- 탐색 : df2.shape, df2.head(), df2[‘내용’]

3. 형태소 분석

**from** **konlpy.tag** **import** Komoran *# 형태소 분석기 불러오기*

tagger = Komoran() *# Komoran 형태소 분석기를 불러 tagger 에 저장한다.*

형태소 분석을 할 때 내장되어있는 명사 추출 함수를 사용할 수 있다.

tagger.nouns('아침에 해를 보며 버스를 탔다') *# 명사 추출 함수 확인*

['아침', '해', '버스']

혹은 직접 만든 함수를 사용할수도 있다. 예시로 단어의 길이가 2 이상인 단어만 추출하는 함수를 만들어보자.

**def** get\_nouns(text):

nouns = tagger.nouns(text)

nouns = [word **for** word **in** nouns **if** len(word) > 1]

**return** nouns

get\_nouns('아침에 해를 보며 버스를 탔다')

['아침', '버스']

4. countvectorizer

Scikit-learn에서 제공하는 TDM 만드는 모듈인 CountVectorizer를 불러온다. CountVectorizer는 토큰이 문서별로 몇 번 등장했는지 행렬로 정리해주는 모듈이다.

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer

cv = CountVectorizer()

CountVectorizer 안에 많은 옵션을 지정해 설정할 수 있다.

* max\_features=: TDM에 사용할 최대(max) 단어(feature) 수를 지정할 수 있다.
* stop\_words=: 영어의 경우 'a', 'the' 와 같은 불필요한 단어를 제거할 수 있다.
* tokenizer=: token(feature)를 만들 때 적용하고 싶은 함수를 지정할 수 있다. 한국어의 경우 형태소 분석 함수를 지정한다.

cv = CountVectorizer(max\_features=1000, tokenizer=get\_nouns)

5. TDM생성

TDM은 cv 뒤에 .fit\_transform() 메써드로 만들 수 있다. 형태소 분석을 같이 하기 때문에 시간이 오래 걸릴 수 있다.

* .fit(): 단어 문서 행렬의 형태를 정하는 메소드로, 각 열에 들어갈 단어를 결정한다. 이 정보는 cv의 내부에 저장된다.
* .transform(): .fit()으로 정한 단어에 데이터를 끼워 맞춘다.
* fit\_transform은 fit과 transform 두 개의 메소드를 합친 것이다.

tdm = cv.fit\_transform(df['본문'])

cv.get\_feature\_names()[:10] *# 첫 10 단어*

6. TF-IDF 변환

**from** **sklearn.feature\_extraction.text** **import** TfidfTransformer

tfidf = TfidfTransformer()

tdm2 = tfidf.fit\_transform(tdm)

- CountVectroizer와 TF-IDF 단어 비교 비교

words = cv.get\_feature\_names()

# 0번째 인덱스 슬라이스(문자열 가져오기 위해), data는 벡터 값

**for** i, n **in** sorted(zip(tdm[0].indices, tdm[0].data)):

**print**(words[i], n)

**for** i, n **in** sorted(zip(tdm2[0].indices, tdm2[0].data)):

**print**(words[i], n)

7. 저장

첫 10열, 10행을 보기 좋은 표 형태로 나타내보자.

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

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

joblib.dump({'vectorizer': cv, 'tdm': tdm}, f)