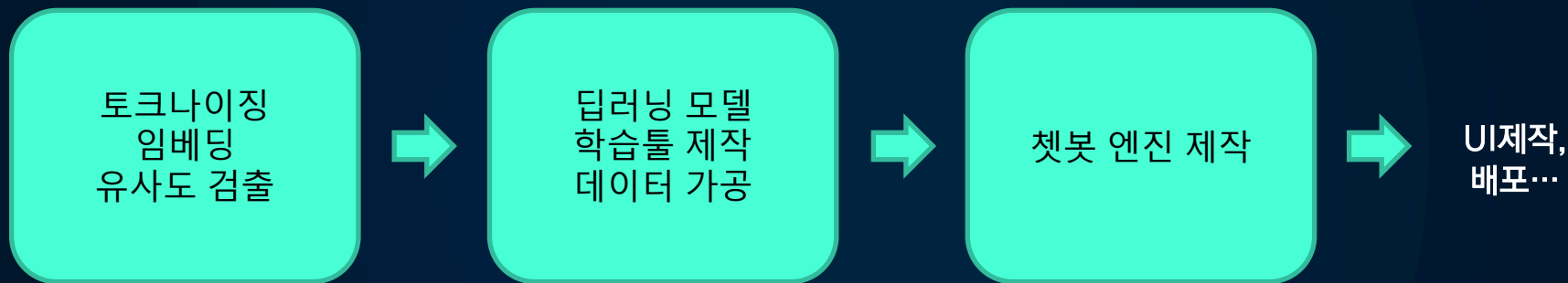




챗봇 프로젝트 07



자연어 처리 & 챗봇 엔진 구현



한글 자연어처리

KoNLPy 파이썬파일

```
[('아버지', 'Noun'), ('가', 'Josa'), ('방', 'Noun'), ('에', 'Josa'), ('들어가신다',  
'Verb'), ('.', 'Punctuation')]
```

한국어 정보를 처리를 위한 파이썬 라이브러리 패키지

자연어처리(NLP) 과정에서 문장을 명사, 조사, 동사 등 형태소 단위로 분리하는 역할
(토큰나이징, Tokenizing)

Hannanum, Kkma, Komoran, Mecab, Okt 5가지 클래스로 제공된다.

개발환경 세팅

가상환경 추가 설치 (chatbot2)

anaconda prompt 실행 후

```
conda create -n chatbot2 python=3.8 // python3.8 버전의 가상환경 생성
```

```
conda activate chatbot2 // chatbot2 가상환경 활성화
```

```
pip install konlpy
```

JDK 설치

C:\Program Files\Java 폴더가 없다면 JAVA 설치

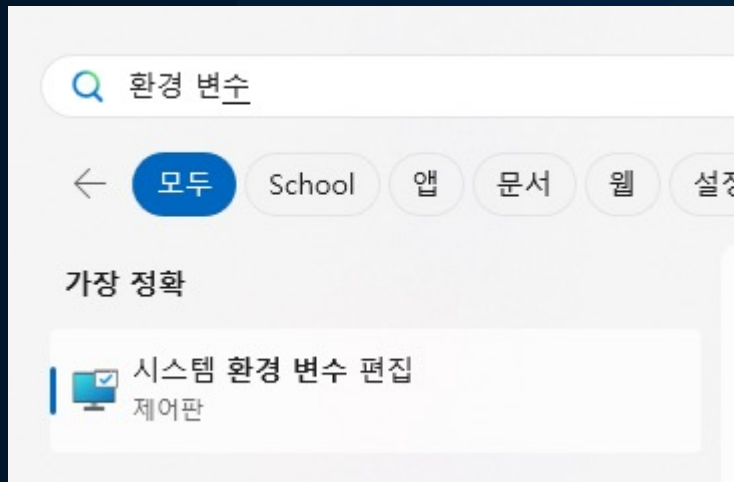
JAVA 설치 링크

https://www.java.com/ko/download/ie_manual.jsp?locale=ko

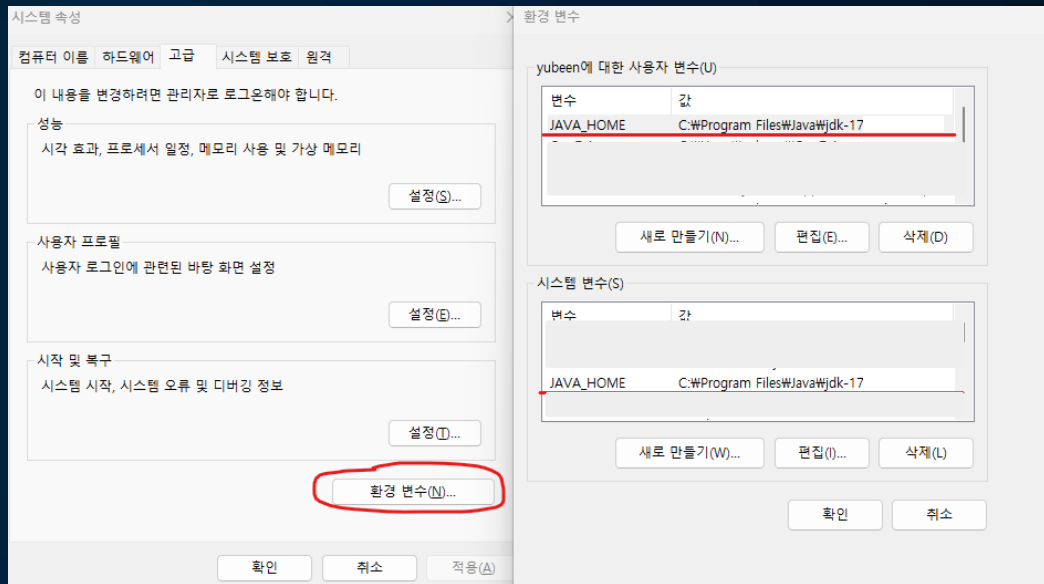
JDK 설치 (17.0.7)

<https://www.oracle.com/kr/java/technologies/downloads/#java17>

환경변수 설정



환경변수 시스템 변수 새로만들기
변수 이름 : JAVA_HOME
값 : C:\Program Files\Java\jdk-17



시스템 변수 > Path > 새로만들기
%JAVA_HOME%\bin 추가

JPYPE1 설치

설치 링크

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

JPyPe1-1.1.2-cp38-cp38-win_amd64.whl - 다운로드

Anaconda prompt, chatbot2 가상환경 활성화 상태에서

`pip install JPyPe1-1.1.2-cp38-cp38-win_amd64.whl` 입력

(`pip install {파일 이름}`)

```
(chatbot) C:\Users\yubeen\python-chatbot>pip install JPyPe1-1.1.2-cp38-cp38-win_amd64.whl
Processing c:\users\yubeen\python-chatbot\jpype1-1.1.2-cp38-cp38-win_amd64.whl
Installing collected packages: JPyPe1
```


설치 확인

Anaconda Prompt 실행 > chatbot2 활성화 *conda activate chatbot2*

python 입력 // 파이썬 실행 *python*

파이썬 실행 후 다음 코드 입력.

```
>>> from konlpy.tag import Kkkma  
>>> kkma = Kkma()  
>>> print(kkma.nouns('안녕하세요 좋은 아침입니다'))  
['안녕', '아침']
```

위와 같이 안녕, 아침 출력 시 정상 설치

실행 오류 시,

1.환경 변수 설정 확인.

2.프로그램 추가/제거 > python 삭제

3.Java 삭제 후 재설치
프로그램 추가/제거 > Java 삭제

4.Jpype 삭제 후 재설치
`pip uninstall Jpype1`
`pip install Jpype1-1.1.2-cp38-cp38-win_amd64.whl`

5.Java\jdk-17\bin\server 경로 안에 jvm.dll 파일이 있는 확인

그 외 기타 등등 오류...

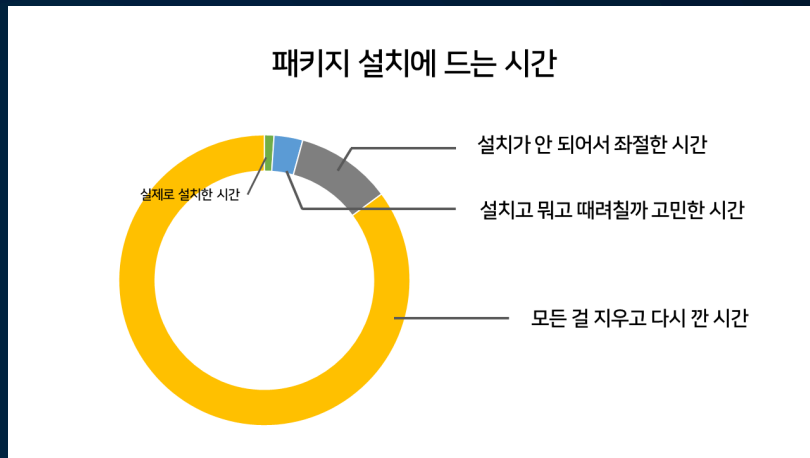
확인해도 오류가 발생할 경우...

konlpy 공식 설치 가이드 참고
<https://konlpy.org/ko/latest/install/>

차선책으로 구글 Colab 환경에서 간단하게 설치 및 실습가능

(세팅 방법 안내 블로그)

<https://couplewith.tistory.com/entry/Python-KoNLPy-%ED%98%95%ED%83%9C%EC%86%8C-%EB%B6%84%EC%84%9D%EA%B8%B0-%EB%B9%84%EA%B5%90-Komoran-Okt-Kkma>



kkma.py

```
from konlpy.tag import Kkma

kkma = Kkma()
text = "아버지가 방에 들어갑니다."

morphs = kkma.morphs(text)
print(morphs) 형태소 추출

pos = kkma.pos(text)
print(pos) 형태소 + 태그 (ex) 명사, 동사

nouns = kkma.nouns(text)
print(nouns) 명사

sentences = "오늘 날씨는 어때요? 내일은 덥다던데."
s = kkma.sentences(sentences)
print(s)
```

anaconda prompt창에서 py파일이 있는 경로로 이동 후,
python kkma.py 입력

품사	설명
NNG	일반 명사
JKS	주격 조사
JKM	부사격 조사
VV	동사
EFN	평서형 종결 어미
SF	마침표, 물음표, 느낌표

형태소 분석

(chatbot2) C:\Users\yubeen\python-chatbot>python Kkma.py

['아버지', '가', '방', '에', '들어가', '버니다', '.']

[('아버지', 'NNG'), ('가', 'JKS'), ('방', 'NNG'), ('에', 'JKM'), ('들어가', 'VV'), ('버니다', 'EFN'), ('.', 'SF')]

['아버지', '방']

['오늘 날씨는 어 때요?', '내일은 덥다 던데.']

1문자씩

Komoran.py

```
from konlpy.tag import Komoran

komoran = Komoran()
text = "아버지가 방에 들어갑니다."

morphs = komoran.morphs(text)
print(morphs)

pos = komoran.pos(text)
print(pos)

nouns = komoran.nouns(text)
print(nouns)
```

anaconda prompt창에서 py파일이 있는 경로로 이동 후,
python komoran.py 입력

komoran과 kkma 클래스는 함수와 품사 태그 등의
인터페이스가 동일하다

품사	설명
NNG	일반 명사
JKS	주격 조사
JKM	부사격 조사
VV	동사
EFN	평서형 종결 어미
SF	마침표, 물음표, 느낌표

```
(chatbot2) C:\Users\yubeen\python-chatbot>python komoran.py
['아버지', '가', '방', '에', '들어가', '됩니다', '.']
[('아버지', 'NNG'), ('가', 'JKS'), ('방', 'NNG'), ('에', 'JKB'), ('들어가', 'VV'), ('됩니다', 'EF'), ('.', 'SF')]
['아버지', '방']
```

Okt.py

```
from konlpy.tag import Okt

okt = Okt()
text = "아버지가 방에 들어갑니다."

morphs = okt.morphs(text)
print(morphs)

pos = okt.pos(text)
print(pos)

nouns = okt.nouns(text)
print(nouns)

text = "오늘 날씨가 좋아요 ㅎㅎ~"
print(okt.normalize(text))
print(okt.phrases(text))
```


anaconda prompt창에서 py파일이 있는 경로로 이동 후,
python okt.py 입력

phrases로 어구를 추출하면 띄어쓰기를 포함한
여러 개의 단어를 추출하기도 한다.

표 3-7 Okt 품사 태그 표

품사	설명
Noun	명사
Verb	동사
Adjective	형용사
Josa	조사
Punctuation	구두점

```
(chatbot2) C:\Users\yubeen\python-chatbot>python okt.py
['아버지', '가', '방', '에', '들어갑니다', '.']
[('아버지', 'Noun'), ('가', 'Josa'), ('방', 'Noun'), ('에', 'Josa'), ('들어갑니다', 'Verb'), ('.', 'Punctuation')]
['아버지', '방']
오늘 날씨가 좋아요 ㅎㅎ~
['오늘', '오늘 날씨', '날씨']
```

쭝도 - komoran < Okt < Kkma

custom_word.py 새로운단어

```
from konlpy.tag import Komoran

komoran = Komoran()
text = "자연어처리는 엔엘피라고 하지."
pos = komoran.pos(text)
print(pos)
```

실행

```
(chatbot2) C:\Users\yubeen\python-chatbot>python customword.py
[('자연어', 'NNP'), ('처리', 'NNP'), ('는', 'JX'), ('엔', 'NNB'), ('엘', 'NNP'), ('피', 'NNG'), ('이', 'VCP'), ('라', 'EC'), ('고', 'EC'), ('하', 'VX'), ('지', 'EF'), ('.', 'SF')]
```

custom_word.py 와 같은 경로에 user_dic.tsv 파일 생성

생성 후, 추가 하고 싶은 고유명사 등을 형식에 맞춰 작성
([단어] Tab [품사])

user_dic.tsv 파일 작성 후,
custom_word.py 코드도 아래와 같이 수정.

콤마
↓
CSV

```
#user_dic.tsv
엔엘피 NNG
스즈메의 문단속 NNG
```

```
from konlpy.tag import Komoran

komoran = Komoran(userdic='./user_dic.tsv')
text = "자연어처리는 엔엘피라고 한다."
text2 = "나 저번주에 스즈메의 문단속을 봤어."
pos = komoran.pos(text)
pos2 = komoran.pos(text2)
print(pos)
print(pos2)
```

단어의 경우 추가가 되었지만,

공백이 포함 된 여러 개의 단어는 지정이 되지 않음

```
(chatbot2) C:\Users\yubeen\python-chatbot>python customword.py
[('자연어', 'NNP'), ('처리', 'NNP'), ('는', 'JX'), ('엔엘피', 'NNG'), ('이', 'VCP'), ('라고', 'EC'), ('하', 'VX'), ('니다', 'EF'), ('.', 'SF')]
[('나', 'NP'), ('저', 'XPN'), ('번주', 'NNG'), ('에', 'JKB'), ('스즈메의', 'NA'), ('문단속', 'NNG'), ('을', 'JKO'), ('보', 'W'), ('았', 'EP'), ('어', 'EF'), ('.', 'SF')]
```

단어 임베딩

말뭉치에서 각각의 단어를 벡터로 변환하는 기법.

단어를 표현하고 변환하는 기법에는 다양한 종류가 있으며 이 방법에 따라 다양한 모델이 존재.

희소 표현, 분산 표현

원-핫 인코딩

Word2Vec

FastText

BERT(Bidirectional Encoder Representations from Transformers)

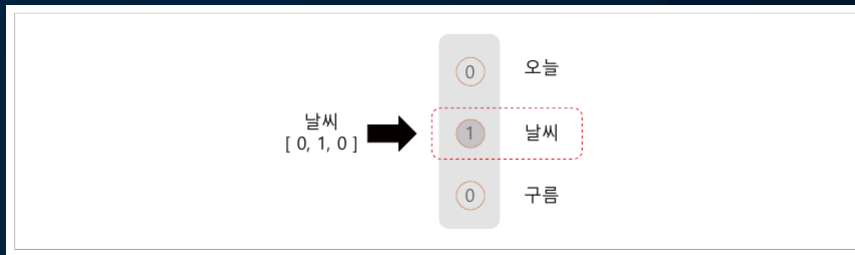
Transformers

...

임베딩 방식

희소표현과 분산표현

	원핫	임베딩
표현	희소	밀집
차원	고차원	저차원
학습방법	수동	훈련 데이터로 학습
값의 타입	1, 0	실수



희소표현



분산표현
(밀집표현)

원-핫 인코딩

단어를 숫자 벡터로 변환하는 가장 기본적인 방법.

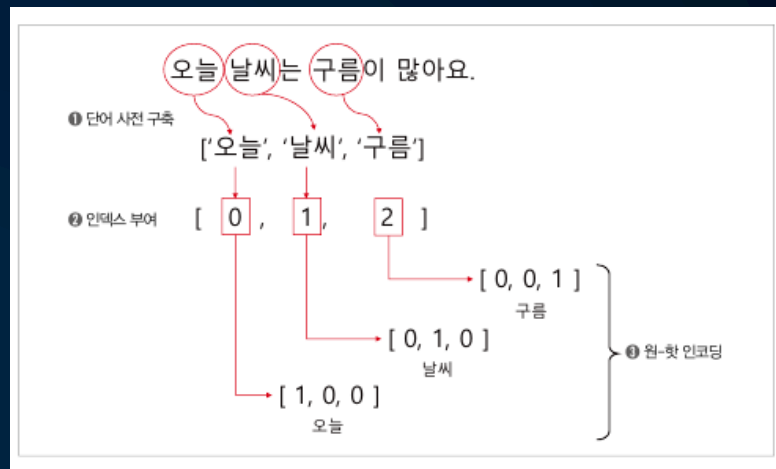
요소들 중 단 하나의 값만 1이고 나머지는 0인 인코딩 방식

말뭉치 전체에서 가져온 단어들의 집합이 필요

간단한 방식에 비해 성능이 빠르다.

단어의 수 만큼 벡터의 차원이 결정됨
(단어가 100개면 벡터의 차원도 100개)

차원이 많아지면 성능 저하.



단어	단어 인덱스	원-핫 벡터
you	0	[1, 0, 0, 0, 0, 0, 0]
say	1	[0, 1, 0, 0, 0, 0, 0]
goodbye	2	[0, 0, 1, 0, 0, 0, 0]
and	3	[0, 0, 0, 1, 0, 0, 0]
I	4	[0, 0, 0, 0, 1, 0, 0]
say	5	[0, 0, 0, 0, 0, 1, 0]
hello	6	[0, 0, 0, 0, 0, 0, 1]

oneandhot.py

```
from konlpy.tag import Komoran
import numpy as np

komoran = Komoran()
text = "오늘 날씨는 구름이 많네요"

nouns = komoran.nouns(text)
print(nouns)

dics = {}
for word in nouns:
    if word not in dics.keys():
        dics[word] = len(dics)
print(dics)

nb_classes = len(dics)
targets = list(dics.values())
one_hot_targets = np.eye(nb_classes)[targets]
print(one_hot_targets)
```


실행 결과

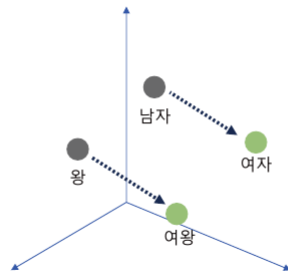
```
(chatbot2) C:\Users\yubeen\python-chatbot>python onehot.py  
['오늘', '날씨', '구름']  
{'오늘': 0, '날씨': 1, '구름': 2}  
[[1. 0. 0.]  
 [0. 1. 0.]  
 [0. 0. 1.]]
```

Word2Vec

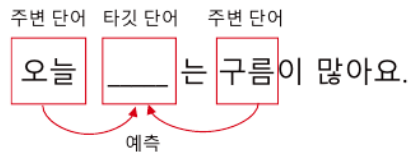
분산 표현 형태의 임베딩 모델 중 하나.

Word2Vec은 2013년 구글에서 발표한 모델이며
현재도 가장 많이 사용되고 있다.

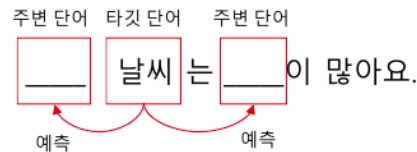
벡터 수치의 특성상 의미에 따라 방향성을 띄며,
특정 단어 간의 관계를 계산할 수 있고,
이로 인해 자연어의 의미를 파악할 수 있다.
(남자 - 여자 / 왕 - 여왕)



CBOW모델 & skip-gram모델



CBOW 모델



skip-gram 모델

라이브러리 설치
`pip install gensim`

id	document	label
8112052	어릴때보고 지금다시봐도 재밌어요ㅋㅋ	1
8132799	디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업이 부러웠는데	
4655635	폴리스스토리 시리즈는 1부터 뉴까지 버릴게 하나도 없음.. 최고.	1
9251303	와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이렇게 진짜 영화지	1
10067386	안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.	1
2190435	사람을 해본사람이라면 처음부터 끝까지 웃을수 있는영화	1
9279041	완전 감동입니다 다시봐도 감동	1
7865729	개들의 전쟁2 나오나요? 나오면 1빠로 보고 싶음	1
7477618	굿	1
9250537	바보가 아니라 병 현 인듯	1
9730759	내 나이와 같은 영화를 지금 본 나는 감동적이다..하지만 훗날 다시보면대사하나하나 그 감정을완벽하게	
640794	재밌다	1
9537008	고질라나무 귀엽다능ㅋㅋ	1
4911311	영화의 오페라화라고 해야할 작품. 극단적 평갈림은 어쩔 수 없는 듯.	1
6686673	3도 반전 좋았제 ^^	1
9034036	평점 왜 낮아? 긴장감 스릴감 진짜 최고인데 진짜 전장에서 느끼는 공포를 생생하게 전해준다.	1
979683	네고시메이터랑 소재만 같을 뿐.. 아무런 관련없음..	1
165498	단연 최고	1

단어 임베딩에 사용할 데이터 다운로드

(네이버 영화 리뷰 데이터)

<https://github.com/e9t/nsmc>

ratings.txt 다운로드 -> 프로젝트 폴더로 이동

word2vec.py

```
from gensim.models import Word2Vec
from konlpy.tag import Komoran
import time

def read_review_data(filename):
    with open(filename, 'r', encoding='UTF-8') as f:
        data = [line.split('\t') for line in
f.read().splitlines()]
        data = data[1:]
    return data

start = time.time()

print('1) 말뭉치 데이터 읽기 시작')
review_data = read_review_data('./ratings.txt')
print(len(review_data))
print('1) 말뭉치 데이터 읽기 완료: ', time.time() - start)
```

```
print('2) 형태소에서 명사만 추출 시작')
komoran = Komoran()
docs = [komoran.nouns(sentence[1]) for sentence in review_data]
print('2) 형태소에서 명사만 추출 완료: ', time.time() - start)
```

```
print('3) word2vec 모델 학습 시작')
model = Word2Vec(sentences=docs, vector_size=200, window=4,
min_count=2, sg=1)
print('3) word2vec 모델 학습 완료: ', time.time() - start)
```

```
print('4) 학습된 모델 저장 시작')
model.save('nvmc.model')
print('4) 학습된 모델 저장 완료: ', time.time() - start)
```

```
print("corpus_count : ", model.corpus_count)
print("corpus_total_words : ", model.corpus_total_words)
```

실행 결과

```
(chatbot2) C:\Users\yubeen\python-chatbot>python word2vec.py
1) 말뭉치 데이터 읽기 시작
200000
1) 말뭉치 데이터 읽기 완료: 0.18602347373962402
2) 형태소에서 명사만 추출 시작
2) 형태소에서 명사만 추출 완료: 58.07240581512451
3) word2vec 모델 학습 시작
3) word2vec 모델 학습 완료: 63.51826500892639
4) 학습된 모델 저장 시작
4) 학습된 모델 저장 완료: 63.542256355285645
corpus_count : 200000
corpus_total_words : 1076896
```

(컴퓨터 사양에 따라 1~2분 정도 소요)
폴더에 nvmc.model 파일이 생성되었으면 성공

word2vec_model.py

```
from gensim.models import Word2Vec

model = Word2Vec.load('nvmc.model')
print("corpus_total_words : ", model.corpus_total_words)

print('사랑 : ', model.wv['사랑'])

print("일요일 = 월요일\t", model.wv.similarity(w1='일요일',
w2='월요일'))
print("안성기 = 배우\t", model.wv.similarity(w1='안성기',
w2='배우'))
print("대기업 = 삼성\t", model.wv.similarity(w1='대기업',
w2='삼성'))
print("일요일 != 삼성\t", model.wv.similarity(w1='일요일',
w2='삼성'))
print("히어로 != 삼성\t", model.wv.similarity(w1='히어로',
w2='삼성'))

print(model.wv.most_similar("안성기", topn=5))
print(model.wv.most_similar("시리즈", topn=5))
```

실행 결과

‘사랑’이라는 단어의 임베딩 벡터

두 단어의 유사도 비교
일요일 = 월요일
(단어 형태 자체가 비슷함)

대기업 = 삼성
(단어 형태는 다르지만
의미가 비슷함)

안성기, 시리즈와 비슷한 단어를
각각 5가지 찾아서 출력.

```
0.08543662 -0.10086995 -0.23271309 -0.37835297 0.19552492 -0.01878669
-0.01988132 -0.04438308 -0.3952234 0.14974214 -0.15447822 -0.06966704
-0.31716323 0.02849556 -0.27588794 0.762574 0.00937104 0.03902356
0.43278044 0.22078958 0.35730737 0.06051953 0.34498915 -0.36040634
-0.06767131 -0.16212747 0.07908221 -0.04671841 -0.05188153 0.10275475
-0.49082613 -0.10639554 -0.11896364 0.2954785 -0.02673793 0.12025648
0.5379815 0.34574434 0.23822188 0.14135623 0.389118 -0.00964424
0.1412058 0.05262238 -0.069635 -0.13130623 -0.43337086 0.52195394
0.10052752 0.2712004 -0.17768018 -0.4920221 0.1180975 0.34720775
-0.2845204 -0.32095894 0.03526663 -0.43811068 0.14176121 0.08034869
0.18980308 -0.42759204 -0.16700494 -0.3894627 0.3049087 -0.07269979
0.11370472 0.0223815 -0.30764586 -0.38868737 0.13398424 0.16194387
-0.03043736 -0.12015959 -0.38251522 0.01722985 -0.06052242 0.31545544
-0.04467312 0.40782216 -0.05488994 0.02268943 0.09363872 -0.06431485
-0.03095681 -0.05316101 0.30823743 -0.14059968 -0.17805053 0.1678972
0.09701487 -0.09977413 -0.09444428 -0.2044433 0.32961005 -0.3144336
-0.12074087 -0.33073974 0.37865 0.0277428 -0.21067639 -0.09337255
0.52042174 0.35309136 0.2721743 0.048627 0.26896745 0.27406946
0.12525855 -0.05992164 -0.09017257 0.6374208 -0.13878061 -0.20493002
-0.19781367 0.3635268 -0.6092423 0.01466777 0.38520473 -0.20825289
-0.03021618 0.03222273 0.28114593 0.06910212 0.38893262 0.2515978
-0.03170713 -0.01703229 0.20119803 0.0328632 -0.38614103 -0.2269777
-0.19221091 -0.15128142 0.15788285 -0.13525599 -0.00141518 -0.26396286
-0.38499957 0.018236 0.14337663 0.07962994 0.14137611 -0.11330333
-0.03398293 -0.14878142]
일요일 = 월요일 0.91003597
안성기 = 배우 0.6970264
대기업 = 삼성 0.887688
일요일 != 삼성 0.6502185
히어로 != 삼성 0.4425709
[('설경구', 0.9500430226325989), ('최강희', 0.9497402310371399), ('지진희', 0.9482381939888), ('한석규', 0.94226837158
20312), ('박신양', 0.9418506026268005)]
[('엑스맨', 0.8177304267883301), ('포터', 0.7967832684516907), ('반지의 제왕', 0.7876511216163635), ('미이라', 0.78664
4458770752), ('스타워즈', 0.7854799032211304)]
```


텍스트 유사도

두 문장 간의 유사도를 계산하기 위해서는 문장 내에 존재하는 단어를 수치화해야 하며,
이 때 언어 모델에 따라 통계를 이용하는 방법과 인공 신경망을 이용하는 방법으로 나눌 수 있다.

앞에서 사용한 word2vec 모델은 인공 신경망을 사용한 경우이며,
이번에는 통계적인 방법을 이용해 유사도를 계산하는 방법을 알아본다.

둘 중 어느 것의 성능이 더 절대적으로 좋다고 할 수 없으며,
챗봇의 주제에 따라 방식을 고르는 것이 좋다.

예시)

대화형챗봇 – 인공 신경망

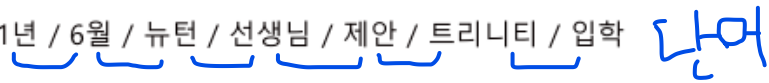
단순Q&A챗봇 – 통계적 방식

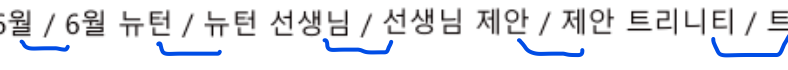
[통계적
수식적]

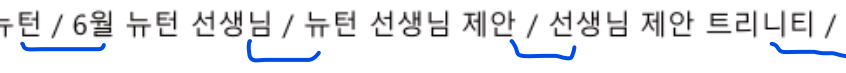
n-gram 유사도

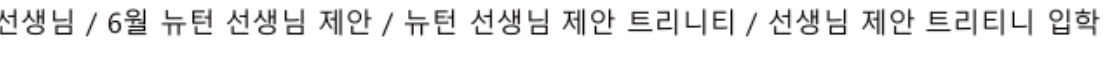
n-gram은 주어진 문장에서 n개의 연속적인 단어의 나열을 의미

1661년 6월 뉴턴은 선생님의 제안으로 트리니티에 입학하였다.

n=1 : 1661년 / 6월 / 뉴턴 / 선생님 / 제안 / 트리니티 / 입학
(unigram) 

n=2 : 1661년 6월 / 6월 뉴턴 / 뉴턴 선생님 / 선생님 제안 / 제안 트리니티 / 트리니티 입학
(bigram) 

n=3 : 1661년 6월 뉴턴 / 6월 뉴턴 선생님 / 뉴턴 선생님 제안 / 선생님 제안 트리니티 / 제안 트리니티 입학
(trigram) 

n=4 : 1661년 6월 뉴턴 선생님 / 6월 뉴턴 선생님 제안 / 뉴턴 선생님 제안 트리니티 / 선생님 제안 트리니티 입학
(4-gram) 

n에 따른 n-gram

(1-gram or 유니그램 / 2-gram or 바이그램 / 3-gram or 트라이그램 / 4이상 (4-gram, 5-gram))

2-gram 예시

1661년 6월 뉴턴은 선생님의 제안으로 트리니티에 입학하였다.

1661년	6월	뉴턴	선생님	제안	트리니티	입학
1661년	6월	뉴턴	선생님	제안	트리니티	입학
1661년	6월	뉴턴	선생님	제안	트리니티	입학
1661년	6월	뉴턴	선생님	제안	트리니티	입학
1661년	6월	뉴턴	선생님	제안	트리니티	입학
1661년	6월	뉴턴	선생님	제안	트리니티	입학

2-gram

n=2 : 1661년 6월 / 6월 뉴턴 / 뉴턴 선생님 / 선생님 제안 / 제안 트리니티 / 트리니티 입학
(bigram)

n-gram 유사도 공식

A : 6월에 뉴턴은 선생님의 제안으로 트리니티에 입학했다.

B : 6월에 뉴턴은 선생님의 제안으로 대학교에 입학했다.

$$similarity = \frac{tf(A, B)}{tokens(A)} \quad \dots\dots n\text{-gram 유사도}$$

문장 A의 2-gram 토큰

		(‘6월’, ‘뉴턴’)	(‘뉴턴’, ‘선생님’)	(‘선생님’, ‘제안’)	(‘제안’, ‘트리니티’)	(‘트리니티’, ‘입학’)	(‘입학’)	
문장 종류	A	1	1	1	1	1	1	6
	B	1	1	1	0	0	1	4

tokens(A) → 6
tf(A, B) → 4

$$tf(A, B) / token(A) = 4/6 = 0.66$$

즉, 두 문장의 유사도는 66%이다.

2-gram.py

```
from konlpy.tag import Komoran

def word_ngram(bow, num_gram):
    text = tuple(bow)
    ngrams = [text[x:x + num_gram] for x in range(0, len(text))]
    return tuple(ngrams)

def phoneme_ngram(bow, num_gram):
    sentence = ' '.join(bow)
    text = tuple(sentence)
    slen = len(text)
    ngrams = [text[x:x + num_gram] for x in range(0, slen)]
    return ngrams
```

```
def similarity(doc1, doc2):  
    cnt = 0  
    for token in doc1:  
        if token in doc2:  
            cnt = cnt + 1  
  
    return cnt/len(doc1)
```

```
sentence1 = '6월에 뉴턴은 선생님의 제안으로 트리니티에 입학하였다'  
sentence2 = '6월에 뉴턴은 선생님의 제안으로 대학교에 입학하였다'  
sentence3 = '나는 맛있는 밥을 뉴턴 선생님과 함께 먹었습니다.'
```

```
komoran = Komoran()  
bow1 = komoran.nouns(sentence1)  
bow2 = komoran.nouns(sentence2)  
bow3 = komoran.nouns(sentence3)
```

```
doc1 = word_ngram(bow1, 2)
doc2 = word_ngram(bow2, 2)
doc3 = word_ngram(bow3, 2)

print(doc1)
print(doc2)
print(doc3)

r1 = similarity(doc1, doc2)
r2 = similarity(doc3, doc1)
print(r1)
print(r2)
```

실행 결과

```
(chatbot2) C:\Users\yubeen\python-chatbot>python 2-gram.py
(('6월', '뉴턴'), ('뉴턴', '선생님'), ('선생님', '제안'), ('제안', '트리니티'), ('트리니티', '입학'), ('입학',))
(('6월', '뉴턴'), ('뉴턴', '선생님'), ('선생님', '제안'), ('제안', '대학교'), ('대학교', '입학'), ('입학',))
(('맛', '밥'), ('밥', '뉴턴'), ('뉴턴', '선생'), ('선생', '님과 함께'), ('님과 함께',))
0.6666666666666666
0.0
```

A : 6월에 뉴턴은 선생님의 제안으로 트리니티에 입학했다. 단점
B : 6월에 뉴턴은 선생님의 제안으로 대학교에 입학했다. 단어 개리가 길면 분석X
C : 나는 맛있는 밥을 뉴턴 선생님과 함께 먹었다.

1. A, B에 대한 유사도 (0.666..) 출력.
2. A, C에 대한 유사도 (0) 출력

`word_ngram(bow1, {n})` <- n부분 숫자 수정 시 3-gram, 4-gram 등 변경 가능

코사인 유사도

코사인 유사도는 두 벡터 간 코사인 각도를 이용해 유사도를 측정하는 방법이다.
벡터의 크기가 중요하지 않을 때 텍스트 간의 거리를 측정하기 위해 사용된다.

n-gram의 경우 동일한 단어가 문장 내에 자주 등장하면 유사도 결과에 부정적,

코사인 유사도는 단어 출현 빈도를 통해 유사도를 계산 한다면 동일한 단어가 많이 포함되어 있을수록 벡터의 크기가 커지지만, 이 벡터의 크기와 상관없이 결과가 안정적이다.

A : 6월에 뉴턴은 선생님의 제안으로 트리니티에 입학했다.

B : 6월에 뉴턴은 선생님의 제안으로 대학교에 입학했다.

문장 A, B의 단어 토큰							
문장 종류		'6월'	'뉴턴'	'선생님'	'제안'	'트리니티'	'대학'
	A	1	1	1	1	1	0
	B	1	1	1	1	0	1

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

..... 코사인 유사도

$$A = [1, 1, 1, 1, 1, 1, 0]$$

$$B = [1, 1, 1, 1, 0, 1, 1]$$

분자 계산

$$A \cdot B = \sum_{i=1}^n A_i \times B_i$$

$$= (1 \times 1) + (1 \times 1) + (1 \times 1) + (1 \times 1) + (1 \times 0) + (1 \times 1) + (0 \times 1)$$

$$= 1 + 1 + 1 + 1 + 0 + 1 + 0$$

$$= 5$$

분모 계산

$$\|A\| \|B\| = \sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}$$

$$= \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2} \times \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2}$$

$$= \sqrt{6} \times \sqrt{6}$$

$$= \sqrt{36}$$

$$= 6$$

코사인 유사도 계산 결과

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\| \|B\|} = \frac{5}{6} = 0.8333333333$$

cosine.py

```
from konlpy.tag import Komoran
import numpy as np
from numpy import dot
from numpy.linalg import norm

def cos_sim(vec1, vec2):
    return dot(vec1, vec2) / (norm(vec1) * norm(vec2))

def make_term_doc_mat(sentence_bow, word_dics):
    freq_mat = {}

    for word in word_dics:
        freq_mat[word] = 0

    for word in word_dics:
        if word in sentence_bow:
            freq_mat[word] += 1

    return freq_mat
```

```
def make_vector(tdm):  
    vec = []  
    for key in tdm:  
        vec.append(tdm[key])  
    return vec
```

```
sentence1 = '6월에 뉴턴은 선생님의 제안으로 트리니티에 입학하였다'  
sentence2 = '6월에 뉴턴은 선생님의 제안으로 대학교에 입학하였다'  
sentence3 = '나는 맛있는 밥을 뉴턴 선생님과 함께 먹었습니다.'
```

```
komoran = Komoran()  
bow1 = komoran.nouns(sentence1)  
bow2 = komoran.nouns(sentence2)  
bow3 = komoran.nouns(sentence3)
```

```
def make_vector(tdm):  
    vec = []  
    for key in tdm:  
        vec.append(tdm[key])  
    return vec
```

```
sentence1 = '6월에 뉴턴은 선생님의 제안으로 트리니티에 입학하였다'  
sentence2 = '6월에 뉴턴은 선생님의 제안으로 대학교에 입학하였다'  
sentence3 = '나는 맛있는 밥을 뉴턴 선생님과 함께 먹었습니다.'
```

```
komoran = Komoran()  
bow1 = komoran.nouns(sentence1)  
bow2 = komoran.nouns(sentence2)  
bow3 = komoran.nouns(sentence3)  
bow = bow1 + bow2 + bow3
```

```
word_dics = []  
for token in bow:  
    if token not in word_dics:  
        word_dics.append(token)
```

```
freq_list1 = make_term_doc_mat(bow1, word_dics)
freq_list2 = make_term_doc_mat(bow2, word_dics)
freq_list3 = make_term_doc_mat(bow3, word_dics)
print(freq_list1)
print(freq_list2)
print(freq_list3)
```

```
doc1 = np.array(make_vector(freq_list1))
doc2 = np.array(make_vector(freq_list2))
doc3 = np.array(make_vector(freq_list3))
r1 = cos_sim(doc1, doc2)
r2 = cos_sim(doc3, doc1)
print(r1)
print(r2)
```

실행 결과

```
(chatbot2) C:\Users\yubeen\python-chatbot>python cosine.py
{'6월': 1, '뉴턴': 1, '선생님': 1, '제안': 1, '트리니티': 1, '입학': 1, '대학교': 0, '맛': 0, '밥': 0, '선생': 0, '님과 함께': 0}
{'6월': 1, '뉴턴': 1, '선생님': 1, '제안': 1, '트리니티': 0, '입학': 1, '대학교': 1, '맛': 0, '밥': 0, '선생': 0, '님과 함께': 0}
{'6월': 0, '뉴턴': 1, '선생님': 0, '제안': 0, '트리니티': 0, '입학': 0, '대학교': 0, '맛': 1, '밥': 1, '선생': 1, '님과 함께': 1}
0.8333333333333335
0.18257418583505536
```

A : 6월에 뉴턴은 선생님의 제안으로 트리니티에 입학했다.

B : 6월에 뉴턴은 선생님의 제안으로 대학교에 입학했다.

C : 나는 맛있는 밥을 뉴턴 선생님과 함께 먹었다.

1. A, B에 대한 유사도 (0.833..) 출력

2. A, C에 대한 유사도 (0.182..) 출력

아까보단 ↑
(n-gram)

중간 정리

토큰나이징 : 주어진 문장에서 최소한의 의미를 가지는 단어로 토큰화 시키는 과정
(추출된 토큰은 아직 자연어의 형태이기 때문에 컴퓨터가 처리할 수 없음)

이 토큰들을 원-핫, word2vec 등의 방식으로 임베딩 처리를 진행한 후,

2-gram, 코사인 유사도 공식 등의 방식을 이용하여 두 문장 간의 유사도를 계산.

자연어 처리 과정을 순차적으로 진행

머신러닝(딥러닝)

Artificial Intelligence

인공지능

사고나 학습등 인간이 가진
지적 능력을 컴퓨터를 통해
구현하는 기술



Machine Learning

머신러닝

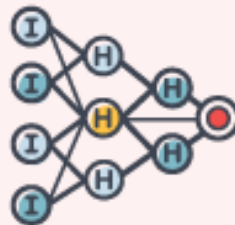
컴퓨터가 스스로 학습하여
인공지능의 성능을
향상 시키는 기술 방법



Deep Learning

딥러닝

인간의 뉴런과 비슷한
인공신경망 방식으로
정보를 처리



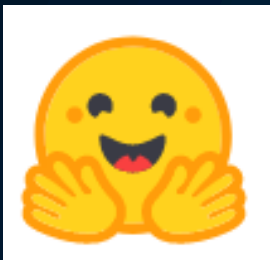
Transformer

2017년 구글에서 발표한 논문을 통해 공개된 딥러닝 모델.
GPT(Generative Pre-trained Transformer)의 기반이 된 모델이기도 하다.

자연어로 프롬프트를 입력하면 답을 내는 지금의 AI는 모두 여기서 파생됨
(ChatGPT, Midjourney, Stable Diffusion, Dall-E..)

기존의 순환신경망(RNN) 방식은 순차적으로 연산을 한다는 단점을
Transformer 모델은 이를 병렬로 처리함으로써
단어간의 연관성 파악, 이해 능력을 획기적으로 높임.

또한 Transformer 모델은 순차적 텍스트, 이미지, 비디오 데이터 등을 사용하는
모든 어플리케이션에 적용될 수 있다.



https://www.youtube.com/watch?v=qEjIMHKmcvA&ab_channel=AINetwork

라이브러리 설치

텐서플로우 2.1 – 딥러닝 모델 실습

```
pip install tensorflow==2.1
```

사이킷런 – 머신러닝 도구 제공

```
pip install sklearn ( 혹은, pip install scikit-learn)
```

Segeval – 모델의 평가를 위한 라이브러리

```
pip install segeval
```

판다스 – 데이터 분석 및 처리

```
pip install pandas xlr
```

Matplotlib – 데이터 시각화 도구 제공

```
pip install matplotlib
```