8강 텍스트 분석 488p

NLP : 머신이 인간의 언어를 이해하고 해석하는 데 더 중점을 두고 기술이 발전

텍스트 마이닝(Text Mining): 텍스트 분석은 비정형 텍스트에서 의미 있는 정보를 추출

- 텍스트 분류 {Text Classification): 특정 신문 기사 내용이 연예/정치/사회/문화 중 어떤 카테고리에 속하는지
- 감성 분석(Sentiment Analysis): 텍스트에서 나타나는 감정/판단/믿음/의견/기분 등의 주관적인 요소를 분석
- 텍스트 요약(Summarization): 텍스트 내에서 중요한 주제나 중심 사상을 추출하는 기법
- 텍스트 군집호KClustering)와 유사도 측정: 비슷한 유형의 문서에 대해 군집화

---

1. 텍스트 분석 이해

❤️ **머신러닝 알고리즘은 숫자형의 피처 기반 데이터만 입력받을 수 있다**
→ **피처 형태로 추출하고 추출된 피처에 의미 있는 값을 부여**

- 비정형 데이터인 텍스트를 분석
- 피처 벡터화 & 피처 추출: 단어의 조합인 벡터값

✔️ 텍스트 분석 과정 요약

1. **텍스트 사전 준비(전처리)**
   - **클렌징**(특수문자 삭제, 대·소문자 통일 등)
   - **토큰화**: 문장을 단어 단위로 분리
   - **의미 없는 단어 삭제**
   - **정규화**: 어근 추출(Stemming) 또는 표제어 추출(Lemmatization)로 단어 형태 통일

2. **피처 벡터화/추출**
    - 전처리한 텍스트에서 중요한 특징을 수치 벡터로 변환
    - 대표 방법:
        - **BOW(Bag of Words)** → Count 기반, TF-IDF 기반
        - **Word Embedding** → Word2Vec 등
3. **머신러닝 모델 구축 및 평가**
    - 벡터화된 텍스트 데이터를 ML 모델에 입력
    - 모델 학습 → 예측 → 성능 평가 과정을 수행

✔️ 파이썬 NLP 패키지 비교 요약

- **NLTK**
    - 가장 오래된 대표 NLP 패키지, 방대한 데이터셋과 모듈 제공
    - 거의 모든 NLP 영역을 커버하지만 **속도·성능이 낮아 대량 데이터 처리엔 부적합**
- **Gensim**
    - **토픽 모델링에 강점**
    - Word2Vec 등 최신 기능 제공
    - 실무에서 SpaCy와 함께 가장 많이 사용됨
- **SpaCy**
    - **가장 빠르고 실무 친화적인 NLP 패키지**
    - 높은 성능과 정확도로 최근 활용도가 급증
- **Scikit-learn**
    - NLP 전문 패키지는 아니지만,

      텍스트 벡터화(BOW, TF-IDF 등)와 ML 모델링 기능이 강함
    - 고급 NLP 작업은 NLTK/Gensim/SpaCy와 **함께 사용**하는 경우 많음

---

2. 텍스트 전처리

❤️ 데이터 전처리 4단계

1. **클렌징**(특수문자 삭제, 대·소문자 통일 등)
2. **토큰화: 단어 vs 문장**
3. **스톱워드** = 분석에 의미가 거의 없는 단어 삭제 (예: *the, is, a, will* 등)
4. **Steamming vs Lemmatization**

- **Stemming:** 빠르고 단순하지만 부정확
- **Lemmatization:** 느리지만 의미 기반으로 가장 **정확한 단어 원형**을 제공

✔️ 텍스트 토큰화 요약

1. 토큰화 종류

- **문장 토큰화**: 문서를 문장 단위로 분리
- **단어 토큰화**: 문장을 단어 단위로 분리

2. NLTK에서의 문장 토큰화

- `sent_tokenize()` 함수로 쉽고 빠르게 문장 분리 가능
- 일반적으로 마침표(`.`), 개행문자(`\n`) 등 문장 끝나는 기호 기준으로 분할

✔️ 단어 토큰화 요약

- **단어 토큰화(Word Tokenization)**

  → 문장을 단어 단위로 분리하는 과정

  → 기본 기준: 공백, 콤마(,), 마침표(.), 개행문자 등
- **정규표현식 활용 가능**

  → 필요에 따라 특수 문자, 특정 패턴 등을 기준으로 더 정교한 토큰화 가능
- **문장 토큰화와의 관계**
  - Bag of Words처럼 **단어 순서가 중요하지 않은 모델**을 사용할 때는

    문장 토큰화 없이 단어 토큰화만으로 충분
  - 반대로 문장의 의미나 구조가 중요한 경우에는 문장 토큰화를 먼저 사용
- **NLTK의 단어 토큰화**

  → `word_tokenize()` 함수로 간단히 구현 가능

```
from nltk import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')

text_sample = "The Matrix is everywhere its all around us, here even in this room. \
You can see it out your window or on your television. \
You feel it when you go to work, or go to church or pay your taxes."

sentences = sent_tokenize(text=text_sample)
print(type(sentences), len(sentences))
print(sentences)
```

```
from nltk.tokenize import word_tokenize
import nltk

nltk.download('punkt')
nltk.download('punkt_tab')

sentence = "The Matrix is everywhere its all around us, here even in this ro om."
words = word_tokenize(sentence)

print(type(words), len(words))
print(words)
```

```
<class 'list'> 15
['The', 'Matrix', 'is', 'everywhere', 'its', 'all', 'around', 'us', ',', 'here', 'even', 'i n', 'this', 'room', '.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
from nltk import word_tokenize, sent_tokenize

def tokenize_text(text):
    sentences = sent_tokenize(text)
    word_tokens = [word_tokenize(sentence) for sentence in sentences]
    return word_tokens


word_tokens = tokenize_text(text_sample)
print(type(word_tokens), len(word_tokens))
print(word_tokens)
```

✔️ 스톱워드(Stop Word) 제거 요약

```
import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords

print("영어 stop words 개수: ", len(stopwords.words('english')))
print(stopwords.words('english')[:20])
```

- **스톱워드** = 분석에 의미가 거의 없는 단어

  (예: *the, is, a, will* 등)
- 문장에서 매우 자주 등장하지만 문맥적 의미가 약함
- 제거하지 않으면 빈도 기반 모델(BOW, TF-IDF)에서 *중요 단어로 잘못 인식*될 수 있음
- 그래서 **전처리 단계에서 반드시 제거하는 과정**이 포함됨

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')
all_tokens = []

for sentence in word_tokens:
  filtered_words=[]
  for word in sentence:
    word = word.lower()
    if word not in stopwords:
      filtered_words.append(word)
  all_tokens.append(filtered_words)
print(all_tokens)
```

✔️ Stemming vs Lemmatization 요약

1) **Stemming (어근 추출)**

- 단어의 **표면적 형태를 단순히 자르거나 변형**하여 어근을 얻는 방식
- 규칙 기반, 빠르지만 **정확도가 낮고 단어 형태가 훼손되기도 함**
  - 예:
    - *worked* → *work*
    - *studies* → *studi* (철자 훼손 예시)

```
from nltk.stem import LancasterStemmer
stemmer = LancasterStemmer()

print(stemmer.stem('working'), stemmer.stem('works'), stemmer.stem('worked'))
print(stemmer.stem('amusing'), stemmer.stem('amuses'), stemmer.stem('amused'))
print(stemmer.stem('happier'), stemmer.stem('happiest'))
print(stemmer.stem('fancier'), stemmer.stem('fanciest'))
```

```
work work work
amus amus amus
happy happiest
fant fancies
```

2) **Lemmatization (표제어 추출)**

- 단어의 **품사와 의미(semantic)를 고려**해 정확한 기본형을 찾는 방식
- 사전(WordNet 등)을 참조 → **정확하지만 속도가 느림**
  - 예:
    - *studies* → *study*
    - *better* → *good* (의미 기반 처리 가능)

```
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')

lemma = WordNetLemmatizer()
print(lemma.lemmatize('amusing', 'v'), lemma.lemmatize('amuses', 'v'), lem
ma.lemmatize('amused', 'v'))
print(lemma.lemmatize('happier', 'a'), lemma.lemmatize('happiest', 'a'))
print(lemma.lemmatize('fancier', 'a'), lemma.lemmatize('fanciest', 'a'))
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
amuse amuse amuse
happy happy
fancy fancy
```

---

3. Bag of Words(BOW) 모델 498p

1. BOW 개념

- 문서의 **모든 단어를 순서·문맥 없이 단어 등장 횟수(빈도)**로 표현하는 피처 벡터화 방식.
- 구현이 **매우 쉽고 빠르며**, 생각보다 문서 특징을 잘 반영함.

2. 장점

- **구현이 간단**, 속도가 빠름 ✔️
- **전통적 NLP에서 가장 널리 쓰인 방식**

3. 단점

1. **문맥(Semantic Context) 반영 불가**

- 단어 순서를 고려하지 않음 → 의미 파악의 한계
- 보완 방법: n-gram(그러나 한계 존재)

2. **희소 행렬(Sparse Matrix) 문제**

- 모든 단어를 열(feature)로 만들면 단어 수가 **수만~수십만**
- 대부분의 문서에는 그 단어가 없어서 **0이 매우 많음**
- 희소 행렬은 ML 알고리즘의 **속도·성능 저하**를 유발함

4. 왜 벡터화가 필요한가?

- 머신러닝 모델은 **숫자형 피처만 입력 가능**
- 텍스트를 단어 기반의 **수치 벡터**로 변환해야 함 → 이 과정이 **피처 벡터화**

5. BOW 벡터화 구조

- 문서 수 = M, 전체 단어 수 = N
- BOW 벡터화 후:
  - 각 문서 → N차원의 벡터
  - 전체 데이터 = **M × N 행렬**
- 대부분 0으로 구성된 **희소 행렬 형태**

```
from scipy import sparse

data = np.array([3, 1, 2])
row_pos = np.array([0, 0, 1])
col_pos = np.array([0, 2, 1])

sparse_coo = sparse.coo_matrix((data, (row_pos, col_pos)))
sparse_coo.toarray()
```

- BOW(Bag of Words)에서 단어를 숫자 벡터로 표현하는 방식은 두 가지

1) Count 기반 벡터화

- **각 단어가 문서 안에서 몇 번 등장했는지(Count)를 그대로 피처 값으로 사용**
- 등장 횟수가 높을수록 "중요 단어"로 판단됨
- 문제점:
  - **언어적으로 흔한 단어** 중요하게 잘못 인식
  - 문서의 특징 반영이 약함

2) TF-IDF 기반 벡터화

- **문서 내에서 자주 등장하면 가중치 ↑,**
  모든 문서에서 흔하게 나타나면 가중치 ↓

→ 즉, **각 문서의 '특징 단어'를 더 정확히 드러내는 방식**

- 문서 수가 많거나 텍스트가 긴 경우 **TF-IDF 성능이 Count보다 더 안정적·우수**

(1) CountVectorizer / TfidfVectorizer

- **텍스트 전처리 + 피처 벡터화**
- CountVectorizer는 **단어 Count**, TfidfVectorizer는 **TF-IDF 값**
- 소문자 변환, 토큰화, 스톱워드 제거 등의 **전처리 자동 수행**
- `fit()`, `transform()` 방식으로 **벡터화** 적용
- Stemming/Lemmatization은 **직접 지원 X → tokenizer에 커스텀 함수로 가능**

  ✔ **순서**

  1. 사전 데이터 가공: 문자를 소문자로 변환 등
  2. 토큰화
  3. 텍스트 정규화: Stop Words 필터링만
  4. 피처 벡터화: 토큰된 단어들을 feature extraction → vectorization

  ✔ **파라미터**

  1) max_df: 너무 자주 등장하는 단어 제외

  - `max_df=100` → 100번 이상 등장하는 단어 제외

- `max_df=0.95` → 전체 문서 중 95% 이상에 등장하는 단어 제외

2) min_df: 너무 드물게 등장하는 단어 제외

- `min_df=2` → 2번 이하 등장 단어 제거
- `min_df=0.02` → 전체 문서 중 하위 2%만 등장하는 단어 제거

3) max_features: 추출할 단어 피처 수 제한

- `max_features=2000` → 상위 2000개 단어만 사용

4) stop_words: 스톱워드 제거

- `stop_words='english'`

5) ngram_range: 단어 묶음 범위 지정

- `(1,1)` → unigram
- `(1,2)` → unigram + bigram

6) analyzer 기본: `'word'`

- `'char'`로 바꿔 문자 단위 n-gram도 가능

7) token_pattern: 정규식 기반 토큰 분리 패턴

- 기본값: `\b\w\w+\b` (2글자 이상 단어만 토큰)

8) tokenizer: 커스텀 **토큰화/어근변환 함수** 지정

- **Stemming/Lemmatization** 수행할 때 사용

```
import numpy as np
from scipy import sparse

# Dense matrix 생성
dense2 = np.array([
    [0, 0, 1, 0, 0, 5],
    [1, 4, 0, 3, 2, 5],
    [0, 6, 0, 3, 0, 0],
    [2, 0, 0, 0, 0, 0],
    [0, 0, 0, 7, 0, 8],
    [1, 0, 0, 0, 0, 0]
])

# 0이 아닌 값들만 추출
```

```
data2 = np.array([1, 5, 1, 4, 3, 2, 5, 6, 3, 2, 7, 8, 1])

# 행 위치
row_pos = np.array([0, 0, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5])

# 열 위치
col_pos = np.array([2, 5, 0, 1, 3, 4, 5, 1, 3, 0, 3, 5, 0])

# COO 형식으로 변환
sparse_coo = sparse.coo_matrix((data2, (row_pos, col_pos)), shape=dense
2.shape)

#행위 치 배열의 고유한 값의 시작 위치 인덱스를 배열로 생성
row_pos_ind = np.array([0,2,7,9,10,12,13])

sparse_csr=sparse.csr_matrix((data2, col_pos, row_pos_ind))
print('c00 출력 다시 확인')
print(sparse_coo.toarray())
print('CSR 출력 다시 확인')
print(sparse_csr.toarray())
```

```
c00 출력 다시 확인
[[0 0 1 0 0 5]
 [1 4 0 3 2 5]
 [0 6 0 3 0 0]
 [2 0 0 0 0 0]
 [0 0 0 7 0 8]
 [1 0 0 0 0 0]]
CSR 출력 다시 확인
[[0 0 1 0 0 5]
 [1 4 0 3 2 5]
 [0 6 0 3 0 0]
 [2 0 0 0 0 0]
 [0 0 0 7 0 8]
 [1 0 0 0 0 0]]
```

(2) BOW 벡터화를 위한 희소 행렬 503p

- 단어 기반으로 벡터화한 뒤 **CSR(Compressed Sparse Row) 형태의 희소 행렬**을 반환
- 문서 전체에서 **중복을 제거한 단어들을 모두 피처로 만드는데**
- **각 문서에는 실제로 등장하는 단어가 매우 적기 때문에**, 문서-단어 행렬의 대부분 값이 **0**이 된다.
- 이렇게 **대부분이 0으로 채워진 행렬**을 **희소 행렬**
  - 이 0들을 그대로 저장하면 **메모리 낭비,** 연산 시 **비효율**
  - 0이 아닌 값만 효율적으로 저장하는 구조 必

✔️ **COO 형식 (Coordinate Format)**

- "0이 아닌 값"만 저장하는 간단한 방식
- **세 개의 배열**:
  1. **data**: 0이 아닌 실제 값
  2. **row**: 해당 값의 행 위치
  3. **col**: 해당 값의 열 위치

✔️ **CSR(Compressed Sparse Row) 형식**

- CSR 형식은 COO 형식보다 **더 적은 메모리**를 사용
- **행 단위 연산 속도가 매우 빠른** 희소 행렬 저장 방식
- **세 개의 배열**
  1. **data**: 0이 아닌 값
  2. **indices**: data에 대응하는 열(col) 위치
  3. **indptr: 각 "행(row)"이 data 배열에서 어디서 시작하는지 표시하는 배열**

---

4. 감성 분석 519-534pp

- 감성 분석은 문서나 문장 안에 담긴 감정(긍정·부정·중립)을 자동으로 판별하는 기술
- SNS 글, 리뷰, 댓글, 여론 분석 등에서 널리 활용
- 지도 학습/비지도 학습으로 나뉨

IMDB 영화평

- 영화평의 텍스트를 분석해 감성 분석 결과가 긍정 또는 부정인지를 예측하는 모델

```
import pandas as pd

review_df = pd.read_csv('/content/drive/MyDrive/kaggle/labeledTrainData.tsv', header=0, sep="\t", quoting = 3)
review_df.head(3)
```

```
import re

review_df['review'] = review_df['review'].str.replace('<br />', ' ')

review_df['review']= review_df['review'].apply(lambda x : re.sub("[^a-zA-Z]", " ", x))
```

```
from sklearn.model_selection import train_test_split

class_df = review_df['sentiment']
feature_df = review_df.drop(['id', 'sentiment'], axis=1, inplace=False)
X_train, X_test, y_train, y_test = train_test_split(feature_df, class_df, test_size=0.3, random_state=156)

X_train.shape, X_test.shape
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score

pipeline = Pipeline([
  ('cnt_vect', CountVectorizer(stop_words='english', ngram_range=(1,2))),
  ('lr_clf', LogisticRegression(C=10))])

pipeline.fit(X_train['review'], y_train)
```

```
pred = pipeline.predict(X_test['review'])
pred_probs = pipeline.predict_proba(X_test['review'])[:, 1]

print("예측정확도는 {0:.4f}, ROC-AUC는 {1:.4f}".format(accuracy_score(y_tes
t, pred), roc_auc_score(y_test, pred_probs)))

pipeline = Pipeline([
  ('tfidf_vect', TfidfVectorizer(stop_words='english', ngram_range=(1,2))),
  ('lr_clf', LogisticRegression(C=10))])

pipeline.fit(X_train['review'], y_train)
pred = pipeline.predict(X_test['review'])
pred_probs = pipeline.predict_proba(X_test['review'])[:, 1]

print("예측정확도는 {0:.4f}, ROC-AUC는 {1:.4f}".format(accuracy_score(y_tes
t, pred), roc_auc_score(y_test, pred_probs)))
```

비지도 감성 분석 523p

- **레이블이 없는 데이터**에서 감성을 판단할 때 사용
- 감성 어휘 사전(Lexicon)에 기반하여 문서의 긍정·부정을 계산
  - Lexicon은 단어마다 **감성 점수(Polarity score)** 를 가지고 있으며,
  - 문맥·품사(POS)·주변 단어를 참고해 단어의 긍정/부정 정도를 정의
- NLTK의 WordNet
  - WordNet은 단순한 단어 사전이 아니라, 단어의 **시맨틱(semantic, 문맥 의미)** 표현 구조 제공
  - 단어를 Synset(동의어 집합) 형태로 묶어 의미를 표현한다.
  - NLTK의 감성 사전은 개념적으로 중요하지만 예측 성능이 높지 않은 단점으로 실제 분석에서는 주로 VADER와 같은 다른 Lexicon이 많이 사용됨
- SentiWordNet
  - WordNet의 Synset마다 **긍정 점수, 부정 점수, 객관성 점수**를 부여한 감성 사전.
  - 문장 내 단어들의 감성 점수를 합산하여 전체 감성을 판단
- VADER

    - SNS 텍스트 분석에 최적화된 감성 사전으로 **빠르고 정확도가 높아** 널리 사용
- Pattern
    - 예측 성능이 좋지만 파이썬 3.X에서 지원되지 않아 현재는 활용이 제한적

SentiWordNet을 이용한 감성 분석

1. 먼저 WordNet을 이용하기 위해서 NLTK 셋업 → WordNet 서브패키지와 데이터 세트 내려 받기

- NLTK 모든 데이터 세트와 패키지 내려받기

```
import nltk
nltk.download('all')
```

- WordNet 모듈을 임포트
- present’ 단어 synsets 추출
- `synsets()`는 파라미터로 지정된 단어에 대해 WordNet에 등재된 모든 Synset 객체를 반환
    - 여러 개의 Synset 객체를 가지는 리스트 반환
    - ‘present.n.01’에서 present는 의미, n은 명사품사', 01은 present가 명사로서 가지는 의미가 여러 가지 있어서 이를 구분하는 인덱스

2. synset 객체가 가지는 여러 가지 속성

- Synset은 POS(Part of Speech로 우리말로 바꾸면 품사입니다), 정의(Definition), 부명제(Lemma) 등으로 시맨틱적인 요소를 표현

```
from nltk.corpus import wordnet as wn
term = 'present'

synsets = wn.synsets(term)
print('synsets() 반환 type :', type(synsets))
print('synsets() 반환 값 갯수:', len(synsets))
print('synsets() 반환 값 :', synsets)
```

```
for synset in synsets:
  print('##### Synset name : ', synset.name(), '#####')
```

```
  print('POS :', synset.lexname())
  print('정의 :', synset.definition())
  print('표제어 :', synset.lemma_names())
```

```
##### Synset name :  present.n.01 #####
POS : noun.time
정의 : the period of time that is happening now; any continuous stretch of ti
me including the moment of speech
표제어 : ['present', 'nowadays']
##### Synset name :  present.n.02 #####
POS : noun.possession
정의 : something presented as a gift
표제어 : ['present']
##### Synset name :  present.n.03 #####
POS : noun.communication
정의 : a verb tense that expresses actions or states at the time of speaking
표제어 : ['present', 'present_tense']
##### Synset name :  show.v.01 #####
POS : verb.perception
정의 : give an exhibition of to an interested audience
표제어 : ['show', 'demo', 'exhibit', 'present', 'demonstrate']
##### Synset name :  present.v.02 #####
POS : verb.communication
정의 : bring forward and present to the mind
표제어 : ['present', 'represent', 'lay_out']
##### Synset name :  stage.v.01 #####
POS : verb.creation
정의 : perform (a play), especially on a stage
표제어 : ['stage', 'present', 'represent']
##### Synset name :  present.v.04 #####
POS : verb.possession
정의 : hand over formally
표제어 : ['present', 'submit']
##### Synset name :  present.v.05 #####
POS : verb.stative
정의 : introduce
표제어 : ['present', 'pose']
##### Synset name :  award.v.01 #####
```

```
POS : verb.possession
정의 : give, especially as an honor or reward
표제어 : ['award', 'present']
##### Synset name :  give.v.08 #####
POS : verb.possession
정의 : give as a present; make a gift of
표제어 : ['give', 'gift', 'present']
##### Synset name :  deliver.v.01 #####
POS : verb.communication
정의 : deliver (a speech, oration, or idea)
표제어 : ['deliver', 'present']
##### Synset name :  introduce.v.01 #####
POS : verb.communication
정의 : cause to come to know personally
표제어 : ['introduce', 'present', 'acquaint']
##### Synset name :  portray.v.04 #####
POS : verb.creation
정의 : represent abstractly, for example in a painting, drawing, or sculpture
표제어 : ['portray', 'present']
##### Synset name :  confront.v.03 #####
POS : verb.communication
정의 : present somebody with something, usually to accuse or criticize
표제어 : ['confront', 'face', 'present']
##### Synset name :  present.v.12 #####
POS : verb.communication
정의 : formally present a debutante, a representative of a country, etc.
표제어 : ['present']
##### Synset name :  salute.v.06 #####
POS : verb.communication
정의 : recognize with a gesture prescribed by a military regulation; assume
a prescribed position
표제어 : ['salute', 'present']
##### Synset name :  present.a.01 #####
POS : adj.all
정의 : temporal sense; intermediate between past and future; now existing o
r happening or in consideration
표제어 : ['present']
##### Synset name :  present.a.02 #####
```

```
POS : adj.all
정의 : being or existing in a specified place
표제어 : ['present']
```

3. `path_similarity( ) 메서드`

- 단어의 상호 유사도

```
import nltk
from nltk.corpus import wordnet as wn
import pandas as pd

# synset 객체 생성
tree = wn.synset('tree.n.01')
lion = wn.synset('lion.n.01')
tiger = wn.synset('tiger.n.02')
cat = wn.synset('cat.n.01')
dog = wn.synset('dog.n.01')

entities = [tree, lion, tiger, cat, dog]

# 단어 이름 추출
entity_names = [entity.name().split('.')[0] for entity in entities]

# 유사도 계산
similarities = []
for entity in entities:
    similarity = [
        round(entity.path_similarity(compared_entity), 2)
        for compared_entity in entities
    ]
    similarities.append(similarity)

# DataFrame 생성
similarity_df = pd.DataFrame(similarities, columns=entity_names, index=ent
ity_names)
similarity_df
```

4. SentiWbrdNet - Senti_Synset 클래스

- `senti_synsets()` : Senti_Synset 클래스를 리스트 형태로 반환
- 감성 지수와 객관성을(감성과 반대) 나타내는 객관성 지수

```
import nltk
nltk.download('sentiwordnet')
from nltk.corpus import sentiwordnet as swn


senti_synsets = swn.synsets('slow')
print('senti_synsets() 반환 type: ', type(senti_synsets))
print('senti_synsets() 반환 값 개수: ', len(senti_synsets))
print('senti_synsets() 반환 값: ', senti_synsets)
```

```
import nltk
from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn

# WordNet synset
father = wn.synset('father.n.01')
print(f"WordNet synset: {father.definition()}\n")

# SentiWordNet synset — 감성 점수는 swn으로 불러야 함
father_swn = swn.senti_synset('father.n.01')

print('father 긍정감성지수:', father_swn.pos_score())
print('father 부정감성지수:', father_swn.neg_score())
print('father 객관감성지수:', father_swn.obj_score())
print('\n')

# 형용사 'fabulous'
fabulous = swn.senti_synset('fabulous.a.01')

print('fabulous 긍정감성지수:', fabulous.pos_score())
print('fabulous 부정감성지수:', fabulous.neg_score())
print('fabulous 객관감성지수:', fabulous.obj_score())
```

SentiWordNet을 이용한 영화 감상평 감성 분석

1. 문서를 문장 단위로 분해
2. 다시 문장을 단어 단위로 토큰화하고 품사 태깅
3. 품사 태깅된 단어 기반으로 snysets 객체와 senti_synset 객체 생성
4. 긍부정 지수 구하고 임계치로 긍/부정 감정 결정

```
from nltk.corpus import wordnet as wn
from nltk.stem import WordNetLemmatizer
from nltk.corpus import sentiwordnet as swn
from nltk import sent_tokenize, word_tokenize, pos_tag

def penn_to_wn(tag):
    if tag.startswith('J'):
        return wn.ADJ
    if tag.startswith('V'):
        return wn.VERB
    if tag.startswith('N'):
        return wn.NOUN
    if tag.startswith('R'):
        return wn.ADV
    return None

def swn_polarity(text):
    sentiment = 0.0
    tokens_count = 0
    lemmatizer = WordNetLemmatizer()
    raw_sentences = sent_tokenize(text)

    for raw_sentence in raw_sentences:
        tagged_sentence = pos_tag(word_tokenize(raw_sentence))

        for word, tag in tagged_sentence:
            wn_tag = penn_to_wn(tag)
            if wn_tag not in (wn.NOUN, wn.ADJ, wn.ADV, wn.VERB):
                continue
```

```
            lemma = lemmatizer.lemmatize(word, pos=wn_tag)
            if not lemma:
                continue

            synsets = wn.synsets(lemma, pos=wn_tag)
            if not synsets:
                continue

            synset = synsets[0]
            swn_synset = swn.senti_synset(synset.name())

            sentiment += (swn_synset.pos_score() - swn_synset.neg_score())
            tokens_count += 1

    if tokens_count == 0:
        return 0

    if sentiment >= 0:
        return 1
    return 0
```

```
review_df['preds'] = review_df['review'].apply(lambda x: swn_polarity(x))
y_target = review_df['sentiment'].values
preds = review_df['preds'].values
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_s
core
from sklearn.metrics import recall_score, f1_score, roc_auc_score
import numpy as np

print(confusion_matrix(y_target, preds))
print("정확도: ", accuracy_score(y_target, preds))
print("정밀도: ", precision_score(y_target, preds))
print("재현율: ", recall_score(y_target, preds))
```

VADER를이용한 감성분석

- **VADER**는 소셜 미디어(트위터, 댓글 등)의 문장 감성 분석을 위해 만들어진 **룰 기반 Lexicon**
- **긍정/부정/중립 점수**를 제공하는 간단하고 빠른 감성 분석 도구
- **SentimentIntensityAnalyzer** 클래스를 사용해 아주 쉽게 감성 점수를 계산

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

senti_analyzer = SentimentIntensityAnalyzer()
senti_scores = senti_analyzer.polarity_scores(review_df['review'][0])
print(senti_scores)
```

- `polarity_scores()` 메서드를 호출
  - 딕셔너리 형태의 감성 점수를 반환
  - compound는 neg, neu, pos score를 적절히 조합해 -1에서 1 사이의 감성 지수를 표현한 값

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score, precision_s
core, recall_score

def vader_polarity(review, threshold=0.1):
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review)
    agg_score = scores['compound']
    final_sentiment = 1 if agg_score >= threshold else 0
    return final_sentiment

review_df['vader_preds'] = review_df['review'].apply(lambda x: vader_polar
ity(x, 0.1))

y_target = review_df['sentiment'].values
vader_preds = review_df['vader_preds'].values

print(confusion_matrix(y_target, vader_preds))
print("정확도:", np.round(accuracy_score(y_target, vader_preds), 4))
```

```
print("정밀도:", np.round(precision_score(y_target, vader_preds), 4))
print("재현율:", np.round(recall_score(y_target, vader_preds), 4))
```