KoNLPy / WordCloud

19기 DA 박수민

목차

1. Introduction to NLP

- NLP란?
- NLP의 과정 개요

2. NLP Processes

- 1) 토큰화 tokenization
- 2) 형태소 분석, 품사 태깅
- 3) 어간 추출, 표제어 추출 stemming & lemmatization
- 4) 불용어 제거 stopwords removal
- 5) 텍스트 표현 text representation
    - A) 정수 인코딩 integer encoding
    - B) 원-핫 인코딩 one-hot encoding
    - C) 워드 임베딩 word embedding

+) 한국어 텍스트 전처리 실습 !

1. Introduction to NLP - NLP란?

Natural Language Processing
자연어 처리

우리가 일상생활에서 사용하는 언어의 의미를 분석하여
컴퓨터가 처리할 수 있도록 하는 일

Ex) Sentiment Classification, Machine Translation …

1. Introduction to NLP - NLP 과정개요

Text Data
(Corpus)

토큰화

Stemming or
Lemmatization

불용어 제거

Text
Representation

Data Preprocessing
데이터 전처리

Train Model

Evaluate Model

Modeling 모델링

2. NLP Processes

사용할 패키지 ?

▶ NLTK
- 자연어 처리 및 문서 분석용 파이썬 패키지
- 주로 외국어 한정 사용

▶ KoNLPy
- 한국어를 위한 자연어처리 패키지

2. NLP Processes - 1) 토큰화 tokenization

▶ 1. 문장 토큰화

1. 문장 토큰화 (Sentence Tokenization)

- 문장 토큰화는 토큰의 단위를 문장으로 하여, 코퍼스 내 텍스트를 문장 단위로 구분하는 작업을 의미합니다.
- 영어의 경우 NLTK의 `sent_tokenize`를 사용하여 영어 문장 토큰화를 수행할 수 있습니다.

```
[ ] from nltk.tokenize import sent_tokenize
    text = 'His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mounta
    print(sent_tokenize(text))
```

```
['His barber kept his word.', 'But keeping such a huge secret to himself was driving him crazy.', 'Finally, the barber went up a mounta
```

```
[ ] # 문장 중간에 .이 있는 경우
    from nltk.tokenize import sent_tokenize
    text = 'I am actively looking for Ph.D. students. and you are a Ph.D student.'
    print(sent_tokenize(text))
```

```
['I am actively looking for Ph.D. students.', 'and you are a Ph.D student.']
```

2. NLP Processes - 1) 토큰화 tokenization

▶ 2. 단어 토큰화

2. 단어 토큰화 (Word Tokenization)

- 단어 토큰화는 토큰의 단위를 단어로 하여, 코퍼스 내 텍스트를 단어 단위로 구분하는 작업을 의미합니다.
- 영어의 경우 텍스트를 단어 단위로 구분할 때 보통 띄어쓰기 즉 공백(whitespace)을 기준으로 합니다.
- ex) text : Time is an illusion. Lunchtime double so!
  tokenized : "Time", "is", "an", "illustion", "Lunchtime", "double", "so"

```
[ ]  # 단어 토큰화 word_tokenize
     # (Don't => Do 와 n't 로 구분/ Jone's => Jone 와 's로 구분 )

     from nltk.tokenize import word_tokenize
     text = "Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."
     print(word_tokenize(text))
```

```
['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes',
```

2. NLP Processes - 1) 토큰화 tokenization

▶ 2. 단어 토큰화

```
[ ]  # WordPunctTokenizer: 구두점(punctuation)을 별도의 토큰으로 구분
     # (Don't => Don 와 ' 와 t 로 구분  / Jone's => Jone 와 ' 와 s 로 구분)

     from nltk.tokenize import WordPunctTokenizer
     text = "Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."
     print(WordPunctTokenizer().tokenize(text))
```

```
['Don', "'", 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone', "'", 's', 'Orphanage', 'is', 'as', 'cheery',
```

Penn Treebank Tokenization

Penn Treebank Tokenization은 표준으로 쓰이고 있는 토큰화 방법 중 하나입니다. Penn Treebank Tokenization의 규칙은 다음과 같습니다.

규칙 1. 하이픈 (-)으로 구성된 단어는 하나로 유지한다.
규칙 2. 아포스트로피 (') 로 접어가 함께 하는 단어는 분리한다.

```
[ ]  from nltk.tokenize import TreebankWordTokenizer
     tokenizer = TreebankWordTokenizer()
     text = "Starting a home-based restaurant may be an ideal. it doesn't have a food chain or restaurant of their own."
     print(tokenizer.tokenize(text))
```

```
['Starting', 'a', 'home-based', 'restaurant', 'may', 'be', 'an', 'ideal.', 'it', 'does', "n't", 'have', 'a', 'food', 'chain', 'or', 'restaurant',
```

2. NLP Processes - 2) 형태소 분석, 품사 태깅

▶형태소 분석

3. 형태소 분석

- 영어의 경우 단어 토큰화를 수행할 때 띄어쓰기를 단어 구분 기준으로 하는데, 이를 어절 토큰화라고 합니다.
- 그런데 한국어의 경우 단어 토큰화를 수행할 때 어절 토큰화를 사용하는 것은 부적절합니다.
- 이는 한국어가 교착어((조사, 어미 등을 붙여서 말을 만드는 언어))라는 점에 기인합니다.

```
"그" + 조사 -> "그를", "그에게", "그가"
"즐겁다" + 어미 -> "즐거운", "즐거워서", "즐겁게"
하지만 이는 다른 단어가 아님! 따라서 조사와 어미 등을 분리해야 함.
```

- 대신, 한국어의 경우 단어 토큰화를 수행할 때 토큰의 단위를 형태소로 하는 **형태소 토큰화**를 사용합니다.

```
형태소(morpheme): 뜻을 가진 가장 작은 말의 단위
예) "아버지가 방에 들어가신다"
-> ['아버지', '가', '방', '에', '들어가', '시', 'ㄴ다']
```

- 한국어 텍스트 전처리 내용 -> "한국어_텍스트_전처리.ipynb" 파일 참고!

2. NLP Processes - 2) 형태소 분석, 품사 태깅

▶품사 태깅

4. 품사 태깅 (Part-Of-Speech Tagging ; POS Tagging)

- 때때로 단어는 표기는 같지만 품사에 따라 단어의 의미가 달라지는 경우가 발생합니다.
- 예) "fly" -> 날다, 파리
- 따라서, 단어의 의미를 제대로 파악하기 위해서는 해당 단어의 품사 정보가 필요합니다.
- 단어 토큰화 과정에서 각 단어가 어떤 품사로 쓰였는지 구분하는 것을 품사 태깅(Part-Of-Speech tagging ; POS Tagging)이라고 합니다.

Penn Treebank

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative

2. NLP Processes - 3) 어간 추출, 표제어 추출

▶ 어간 추출과 표제어 추출 ?
Stemming and lemmatization?

▾ 5. 어간 추출 (Stemming) & 원형 복원 (Lemmatization)

- 단어의 형태 변화(lexical variations of term ; term variation) 에 따라 같은 단어라도 다른 단어인 것처럼 취급되는 문제를 해결하기 위해 사용되는 보편적인 방법으로 어간 추출(Stemming)과 원형 복원(Lemmatization)이 있습니다.

예) automatic, automate, automation -> automat
예) watch, watches, watched -> watch

2. NLP Processes - 3) 어간 추출, 표제어 추출

▶어간 추출 stemming

Stemming (어간 추출)

- Stemming이란 어형이 변형된 단어로부터 접사 등을 제거하고 그 단어의 어간을 분리해내는 것을 의미합니다.

```
예) 'automate', 'automatic', 'automation' -> 'automat'
각각 모두 'automat' 어간 + 'e', 'ic', 'ion'이라는 접사
```

- 이러한 단어들에 대하여 접사를 제거하고 동일한 어간인 'automat'으로 매핑되도록 하는 작업이 stemming입니다.
- 대표적인 Stemming Algorithm으로 Martin Porter가 고안한 Porter Stemming Algorithm = Porter Stemmer 가 있습니다.

2. NLP Processes - 3) 어간 추출, 표제어 추출

▶어간 추출 stemming

```
[ ]  from nltk.tokenize import word_tokenize
     from nltk.stem import PorterStemmer

     text = "This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights and soundings--with the single exc

     # word tokenization
     words = word_tokenize(text)

     # stemming
     s = PorterStemmer()
     result = [s.stem(w) for w in words]

     # 결과 출력
     print('original text:', text)
     print('tokenized words:',words)
     print('stemmed words:',result)
```

```
original text: This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights and soundings--with the sir
tokenized words: ['This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate', 'copy', ',', 'complete
stemmed words: ['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi', ',', 'complet', 'in',
```

2. NLP Processes - 3) 어간 추출, 표제어 추출

▶어간 추출 stemming

```
[ ]  ## 두가지 종류의 stemmer 비교
     from nltk.stem import PorterStemmer
     from nltk.stem import LancasterStemmer

     words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']

     # stemming
     s = PorterStemmer() # 포터 스태머
     l = LancasterStemmer() # 랭커스터 스태머
     ss = [s.stem(w) for w in words]
     ll = [l.stem(w) for w in words]

     # 결과 출력
     print('original words:', words)
     print('porter stemmer:', ss)
     print('lancaster stemmer:',ll)
```

```
original words: ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
porter stemmer: ['polici', 'do', 'organ', 'have', 'go', 'love', 'live', 'fli', 'die', 'watch', 'ha', 'start']
lancaster stemmer: ['policy', 'doing', 'org', 'hav', 'going', 'lov', 'liv', 'fly', 'die', 'watch', 'has', 'start']
```

2. NLP Processes - 3) 어간 추출, 표제어 추출

▶표제어 추출 lemmatization

Lemmatization (원형 복원 ; 표제어 추출)

- Lemmatization은 한 단어가 여러 형식으로 표현되어 있는 것을 단일 형식으로 묶어주는 기법입니다.

```
ex) 'am', 'are', 'is' -> 'be'
```

- Lemmatization을 수행할 경우, 품사 정보가 남아있기 때문에 의미론적 관점에서 더 효과적입니다.
- 하지만, 여전히 품사정보를 가지고 있어 stemming만큼 DTM dimension reduction 측면에서 효과적이진 않습니다.

```
[ ]  from nltk.stem import WordNetLemmatizer

     words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']

     # lemmatization
     n = WordNetLemmatizer()
     result = [n.lemmatize(w) for w in words]

     # 결과 출력
     print('tokenized words:',words)
     print('lemmatized words:',result)
```

```
tokenized words: ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
lemmatized words: ['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 'starting']
```

2. NLP Processes - 3) 어간 추출, 표제어 추출

```
# 단어의 품사 정보를 알려주어 다시 출력 -> 더 정확한 lemmatization
print(n.lemmatize('dies', 'v'))
print(n.lemmatize('watched', 'v'))
print(n.lemmatize('has', 'v'))
```

```
die
watch
have
```

▶stemming vs. lemmatization

Stemming vs. Lemmatization

비교	Stemming	Lemmatization
의미	어간 추출	원형 복원
접근 방법	정보검색적	언어학적
DTM dimension reduction 관점	good	bad
의미론적 관점	bad (품사 X)	good (품사 O)

- 영어 텍스트의 경우에는 stemming과 lemmatization이 명확하게 구분되어 텍스트 전처리 과정에서 무엇을 사용할지를 결정해야 합니다.
- 반면 한글 텍스트의 경우에는 형태소 분석 과정에서 stemming과 lemmatization이 함께 이루어진다고 볼 수 있습니다.

2. NLP Processes -4) 불용어 제거 stopwords removal

▶불용어 (不用(불용)) 제거

6. Stopwords Removal (불용어 제거)

- 너무 자주 나타나는 단어들은 기능적인 역할을 하거나 문헌집단 전반에 걸쳐 나타나기 때문에 특정 문헌의 내용을 대표할 수 없습니다.
- 자연어 말뭉치 표현에 나타나는 단어들을 그 사용 빈도가 높은 순서대로 나열하였을 때, 왼쪽에 존재하는 고빈도 단어들을 **stopwords**라고 합니다.
- 영어: 정관사, 전치사 등/ 한글: 조사 등
- 불용어 제거는 단어 정제를 통해 보다 제대로 된 분석을 하기 위함이기도 하고, 차원을 축소하기 위함이기도 합니다.

2. NLP Processes -4) 불용어 제거 stopwords removal

▶불용어 (不用(불용)) 제거

NLTK에서 정의한 불용어 리스트 사용하기

```
[ ]  from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
```

```
[ ]  example = "Family is not an important thing. It's everything."

     # 불용어 리스트 생성
     stop_words = set(stopwords.words('english'))

     # 단어 토큰화 실시
     word_tokens = word_tokenize(example)

     # 단어 토큰화 결과로부터 불용어 제거 실시
     result = []
     for w in word_tokens:
         if w not in stop_words:
             result.append(w)

     # 결과 출력
     print(word_tokens)
     print(result)
```

```
['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
```

+) 한국어 텍스트 NLP Processes

▶KoNLPy 사용 !

```
[ ]  import konlpy
```

KoNLPy의 내장된 분석기 종류

1. Hannanum
2. Kkma
3. Komoran
4. Mecab
5. Okt(Twitter)

- 분석기마다 로딩시간과 실행시간이 다르다.
- 동일한 문장이라도 분석기에 따라 품사 태깅하는 결과가 다르다.
- KoNLPy의 분석기 중에서 사용할 것을 불러와서 인스턴스로 만들어준다.
  (인스턴스: 클래스의 정의를 통해 만들어진 객체)

```
[ ]  from konlpy.tag import Okt,Kkma
     okt = Okt()
     kkma = Kkma()
```

+) 한국어 텍스트 NLP Processes

▶KoNLPy 실습

사용할 수 있는 함수

- `morphs` : 형태소 분석
- `pos` : 품사 태깅 (Part-of-speech tagging)
- `nouns` : 명사 추출

분석기에 따라 사용할 수 있는 함수가 조금씩 다릅니다.
okt 함수의 기능들을 알아봅시다.

```
* morphs(phrase, norm=False, stem=False)
: phrase 를 형태소 단위로 나눔.

* nouns(phrase)
: phrase 의 형태소 중에서 noun 만 추출.

* phrases(phrase)
: phrase 에서 어절을 추출.

* pos(phrase, norm=False, stem=False)
: 품사(POS) 태깅.<br>
```

+) 한국어 텍스트 NLP Processes

▶KoNLPy 실습

```
* pos(phrase, norm=False, stem=False)
: 품사(POS) 태깅.<br>
```

- parameters?

```
1. norm - True 로 설정하면 문장을 정규화시킴(오류나 실수를 수정).
ex) 사릉해 -> 사랑해
2. stem - True 로 설정하면 어근화시킴(단어의 기본형으로 변환).
ex) 되나요 -> 되다
```

+) 한국어 텍스트 NLP Processes

▶토큰화

```
[ ]  # text 라는 변수에 실제 문장을 입력합니다.
     text = "아버지가 방에 들어가신다"
```

토큰화

```
[ ]  # 토큰화
     kkma.morphs(text)
```

```
['아버지', '가', '방', '에', '들어가', '시', 'ㄴ다']
```

```
[ ]  # 분석기에 따른 토큰화 결과 비교
     print("kkma :", kkma.morphs(text))
     print("okt :", okt.morphs(text))
```

```
kkma : ['아버지', '가', '방', '에', '들어가', '시', 'ㄴ다']
okt : ['아버지', '가', '방', '에', '들어가신다']
```

+) 한국어 텍스트 NLP Processes

대분류	태그	설명
체언	NNG	일반 명사
체언	NNP	고유 명사
체언	NNB	의존 명사
체언	NR	수사
체언	NP	대명사
용언	VV	동사
용언	VA	형용사
용언	VX	보조 용언
용언	VCP	긍정 지정사
용언	VCN	부정 지정사
관형사	MM	관형사

▶품사태깅

▾ 품사태깅

```
[ ]  # 분석기에 따른 품사 태깅 결과 비교
     print(kkma.pos(text))
     print(okt.pos(text))
```

```
[('아버지', 'NNG'), ('가', 'JKS'), ('방', 'NNG'), ('에', 'JKM'), ('들어가', 'VV'), ('시', 'EPH'), ('ㄴ다', 'EFN')]
[('아버지', 'Noun'), ('가', 'Josa'), ('방', 'Noun'), ('에', 'Josa'), ('들어가신다', 'Verb')]
```

```
[ ]  # 품사가 명사인 단어들만 남기기
     nouns = []
     pos_text = okt.pos(text)

     for each_tuple in pos_text:
         if each_tuple[1] == "Noun":
             nouns.append(each_tuple[0])

     nouns
```

```
['아버지', '방']
```

+) 한국어 텍스트 NLP Processes

▶정규화

▾ 정규화

```
[ ]  text = "안녕하세욬ㅋㅋㅋ 반가워요 샤릉해"
```

```
[ ]  print("정규화 적용 X:", okt.pos(text, norm=False))
     print("정규화 적용 O:", okt.pos(text, norm=True))
```

```
정규화 적용 X: [('안녕하세', 'Adjective'), ('욬', 'Noun'), ('ㅋㅋㅋ', 'KoreanParticle'), ('반가워요', 'Adjective'), ('샤릉해', 'Noun')]
정규화 적용 O: [('안녕하세요', 'Adjective'), ('ㅋㅋㅋ', 'KoreanParticle'), ('반가워요', 'Adjective'), ('사랑', 'Noun'), ('해', 'Verb')]
```

+) 한국어 텍스트 NLP Processes

▶어근화

어근화

```
[ ]  text = "달콤한 너의 러시안 룰렛"
```

```
[ ]  print("어근화 적용 X:", okt.pos(text, stem=False))
     print("어근화 적용 O:", okt.pos(text, stem=True))
```

```
어근화 적용 X: [('달콤한', 'Adjective'), ('너', 'Noun'), ('의', 'Josa'), ('러시안', 'Noun'), ('룰렛', 'Noun')]
어근화 적용 O: [('달콤하다', 'Adjective'), ('너', 'Noun'), ('의', 'Josa'), ('러시안', 'Noun'), ('룰렛', 'Noun')]
```

2. NLP Processes -5) 텍스트 표현 -A) 정수인코딩

▶정수 인코딩

1. 정수 인코딩 (Integer Encoding)

- 인코딩: 텍스트 -> 숫자로 변환하여 컴퓨터가 데이터를 처리할 수 있게끔 하는 절차

- 정수 인코딩: 각 단어를 고유한 정수에 매핑하는 인코딩 방법.
- 예를 들어 갖고 있는 텍스트에 단어가 5000개가 있다면, 5000개의 단어들 각각에 1번부터 5000번까지 인덱스, 즉 단어와 매핑되는 고유한 정수를 부여합니다. 가령, book은 150번, dog는 171번, love는 192번, books는 212번과 같이 숫자가 부여됩니다.
- 단어에 인덱스를 부여하는 방법에 따라 다양한 정수 인코딩 방법이 존재합니다.

2. NLP Processes -5) 텍스트 표현 -A) 정수인코딩

▶정수 인코딩

```
[ ]  text = "A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret
     print(text)
```

```
A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He Kept
```

```
[ ]  print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

```
[ ]  print(encoded)
```

```
[[1, 5], [1, 6, 5], [1, 3, 5], [6, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [6, 6, 3, 2, 6, 1, 6], [1, 6, 3, 6]]
```

자세한 설명은 실습 파일에…

2. NLP Processes -5) 텍스트 표현 -B) 원-핫 인코딩

▶원-핫 인코딩

원-핫 인코딩(One-Hot Encoding)은 단어 집합의 크기를 벡터의 차원으로 하고, 표현하고 싶은 단어의 인덱스에 1의 값을 부여하고, 나머지 단어들의 인덱스에는 0을 부여하는 단어의 벡터 표현 방식입니다. 원-핫 인코딩을 통해 표현한 벡터를 원-핫 벡터(One-Hot vector)라고 합니다.

원-핫 인코딩의 과정은 다음과 같습니다.

1. 코퍼스에 대하여 단어 집합을 생성합니다.
2. 단어 집합에 존재하는 각 단어에 고유한 정수 즉 인덱스를 부여합니다. (정수 인코딩)
3. 표현하고 싶은 단어의 인덱스 위치에는 1을 값으로 부여하고, 나머지 단어들의 인덱스 위치에는 0을 값으로 부여합니다.

2. NLP Processes -5) 텍스트 표현 -B) 원-핫 인코딩

▶원-핫 인코딩

```
print(token)
```

```
['나', '는', '자연어', '처리', '를', '배운다']
```

```
print(word2index)
```

```
{'나': 0, '는': 1, '자연어': 2, '처리': 3, '를': 4, '배운다': 5}
```

```
one_hot_encoding('자연어', word2index)
```

```
[0, 0, 1, 0, 0, 0]
```

```
one_hot_encoding('배운다', word2index)
```

```
[0, 0, 0, 0, 0, 1]
```

자세한 설명은 실습 파일에…

2. NLP Processes -5) 텍스트 표현 -C) 워드 임베딩

▶워드 임베딩

임베딩 (Embedding)

- 단순 숫자가 아닌, 단어를 밀집 벡터 (dense vector)로 표현하는 것을 워드 임베딩이라고 합니다.
- 단어 간 유사성을 알 수 없다는 인코딩의 치명적인 단점을 해결하고자, 단어의 잠재 의미를 반영하여 다차원 공간에 단어를 벡터화 하는 방법을 이용합니다.
- 다양한 임베딩 방법이 존재합니다.

```
1. 카운트 기반 벡터화 방법 : LSA, HAL 등
2. 예측 기반 벡터화 방법 : NNLM, RNNLM, Word2Vec, FastText 등
3. 카운트 기반 & 예측 기반 방법 : GloVe
```

2. NLP Processes -5) 텍스트 표현 -C) 워드 임베딩

▶워드 임베딩
→ CountVectorizer 사용

▾ CountVectorizer

- CountVectorizer는 입력된 문장을 토큰화 하여, 해당 문장을 토큰의 등장 빈도 벡터로 표현하는 기법입니다.
- sklearn 패키지에 있는 CountVectorizer를 이용합니다.

2. NLP Processes -5) 텍스트 표현 -C) 워드 임베딩

▶워드 임베딩

CountVectorizer 학습

- 어떤 단어들을 사용할지, 어떤 단어가 중요한지 학습하는 과정입니다.
- 토큰의 출현 빈도를 기준삼아, 문당을 벡터로 표현합니다.

```
[3]  sample_text1 = ["hello, my name is dacon and I am a data scientist!"]
```

```
[4]  #CountVectorizer 학습
     sample_vectorizer.fit(sample_text1)
```

```
CountVectorizer()
```

```
[5]  # 학습한 단어 목록 (Vocabulary) 확인
     # 이 Vocabulary를 기준으로 새로운 문장을 벡터로 표현
     print(sample_vectorizer.vocabulary_)
```

```
{'hello': 4, 'my': 6, 'name': 7, 'is': 5, 'dacon': 2, 'and': 1, 'am': 0, 'data': 3, 'scientist': 8}
```

```
[13] vocab = sample_vectorizer.vocabulary_

     sorted_vocab = sorted(vocab.items())
     print(sorted_vocab)
```

```
[('am', 0), ('and', 1), ('dacon', 2), ('data', 3), ('hello', 4), ('is', 5), ('my', 6), ('name', 7), ('scientist', 8)]
```

2. NLP Processes -5) 텍스트 표현 -C) 워드 임베딩

▶워드 임베딩

새로운 문장에 대해 CountVectorizer 적용

```
[6]  sample_text2 = ["you are learning dacon data science"]
```

```
[7]  sample_vector = sample_vectorizer.transform(sample_text2)
     print(sample_vector.toarray())
```

```
[[0 0 1 1 0 0 0 0 0]]
```

- "you are learning dacon data science" 문장을 sample_vectorizer 를 활용해 transform한 결과
- 단어들의 출현 빈도로 이루어진 크기 9의 백터가 출력됨
- dacon, data -> 한 번씩 출현 하여 '1'/ 나머지는 '0'

```
[8]  sample_text3 = ["you are learning dacon data science with movie data"]
```

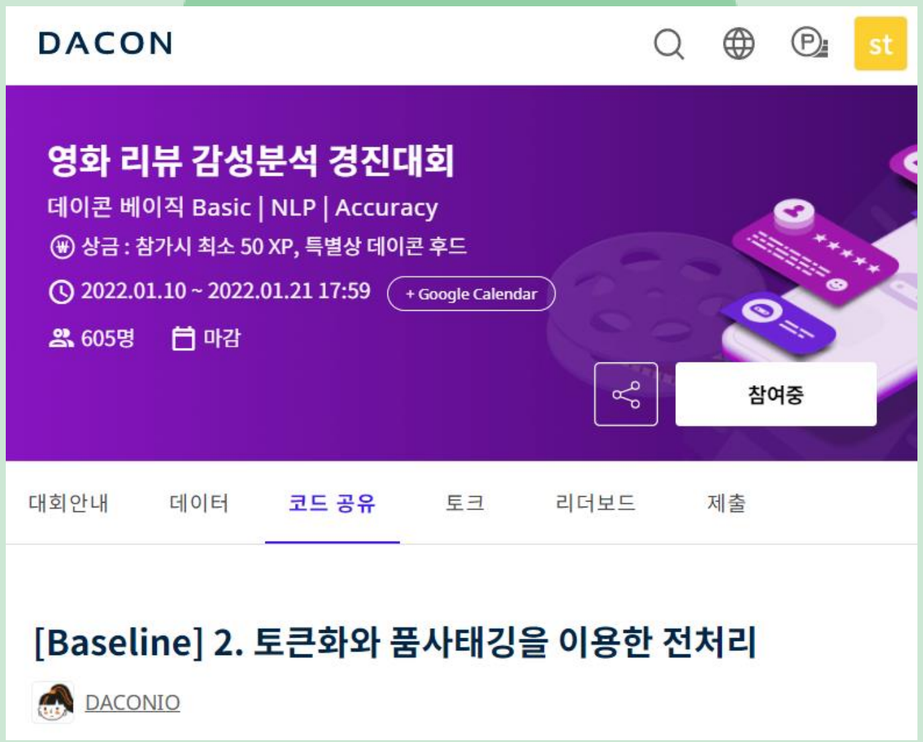
```
[9]  sample_vector2 = sample_vectorizer.transform(sample_text3)
     print(sample_vector2.toarray())
```

```
[[0 0 1 2 0 0 0 0 0]]
```

- dacon -> 한 번 출현하여 '1'/ data -> 두 번 출현하여 '2' / 나머지는 '0'

과제 안내 !

NLP 전 과정 경험해보기 !



해당 링크에 있는 코드 그대로 돌리면서,
가장 마지막 단계인,
Test dataset에 대한 예측 값이 산출되면,
그 화면을 캡처해서
구글 드라이브에 업로드 해주세요 ! :)

https://dacon.io/competitions/official/235864/codeshare/4201?page=1&dtype=recent

예시:

	id	label
0	1	0
1	2	1
2	3	0
3	4	1
4	5	1
...	...	...
4995	4996	0
4996	4997	0
4997	4998	1
4998	4999	0
4999	5000	0

감사합니다