5~6장(~ TF-IDF)

날짜	@2025년 5월 7일 → 2025년 5월 13일
선택	DL세션 과제
주차	10주차
진행 상태	완료

5장 토큰화

💡 **자연어 처리(Natural Language Processing, NLP)**

: 컴퓨터가 인간의 언어를 이해하고 해석 및 생성하기 위한 기술

- **모호성(Ambiguity)**: 사용되는 맥락에 따라 언어와 구가 여러 의미를 갖게 되어 모호한 경우, 다양한 의미를 이해하고 명확하게 구분할 수 있어야 함.
- **가변성(Variability)**: 사투리(Dialects), 강세(Accent), 신조어(Coined word), 작문 스타일로 인해 가변적인 특성을 처리할 수 있어야 하며, 사용중인 언어를 이해할 수 있어야 함.
- **구조(Structure):** 구문(Syntactic)을 파악하여 의미(Semantic)을 해석할 수 있어야 함

토큰화(Tokenization)

말뭉치(Corpus, 자연어 모델을 훈련하고 평가하는데 사용되는 대규모의 자연어)를 일정한 단위인 토큰(Token)으로 나눠야 함.

- 토큰화를 구축하는 방법
  - 공백 분할: 텍스트를 공백 단위로 분리해 개별 단어로 토큰화함
  - 정규표현식 적용: 정규 표현식으로 특정 패턴을 식별해 텍스트를 분할
  - 어휘 사전 적용: 사전에 정의된 단어 집합을 토큰으로 사용
    - 어휘사전(Vocabulary, Vocab): 사전에 정의된 단어를 활용해 토크나이저를 구축하는 방법

→ 직접 어휘 사전을 구축하기 때문에 없는 단어나 토큰이 존재할 수 있음 : OOV(Out of Vocab)

- 큰 어휘 사전 구축시 학습 비용 증대와 차원의 저주(Curse of Dimensionality)에 빠질 수 있음 : 모든 토큰이나 단어를 벡터화하면, 어휘 사전에 등장하는 토큰 개수만큼의 차원이 필요, 벡터값이 거의 모두 0을 가지는 희소(sparse) 데이터로 표현됨.
- 희소 데이터의 표현 방법은 어휘 사전의 출현 빈도만 고려 → 문장에서 발생한 토큰들의 순서 관계를 잘 파악하지 못함.

- 머신러닝 활용: 데이터세트를 기반으로 토큰화하는 방법을 학습한 머신러닝을 적용

단어 및 글자 토큰화

입력된 텍스트 데이터를 단어(word)나 글자(Character) 단위로 나누는 기법으로는 **단어 토큰화**와 **글자 토큰화**가 있다.

단어 토큰화(Word Tokenization)

자연어 처리 분야에서 핵심적인 전처리 작업 중 하나로 텍스트 데이터를 의미 있는 단위인 단어로 분리하는 작업

- 띄어쓰기, 문장 부호, 대소문자 등의 특정 구분자를 활용
- 품사 태깅, 개체명 인식, 기계 번역 등의 작업에서 사용
- 가장 일반적인 토큰화 방법
- 한국어 접사, 문장 부호, 오타, 띄어쓰기 오류 등에 취약

```
# 문자열 데이터 형태는 split 메서드로 토큰화
# 공백(Whitespace)을 기준으로
tokenized = review.split()
```

글자 토큰화(Character Tokenization)

띄어스기뿐만 아니라 글자 단위로 문장을 나누는 방식

- 비교적 작은 단어 사전을 구축할 수 있음 → 컴퓨터 자원을 아낌, 전체 말뭉치를 학습할 때 각 단어를 더 자주 학습할 수 있음.
- 언어 모델링과 같은 시퀀스 예측 작업에 활용

```
# 리스트 형태로 변환하면 쉽게 수행 가능
tokenized = list(review)
```

- 영어는 각 알파벳으로 나뉘지만, 한글은 여러 자음과 모음의 조합으로 이루어져 있음 → 한글 문자 및 자모 작업을 위한 한글 음절 분해 및 합성 라이브러리 사용
    - 자소 단위로 분해해 토큰화

```
# 자모 변환 함수
# 입력된 한글 문자열을 유니코드 U+1100 ~ U+11FE 사이 조합형 한글 자모로 변환
retval = jamo.h2j(hangul_string)
```

<컴퓨터가 한글을 인코딩 하는 방식>

- 조합형: 글자를 자모 단위로 나눠 인코딩 → 조합해 한글 표현
    - h2j 함수는 완성형으로 입력된 한글을 조합형 한글로 변환
    - retval = jamo.j2hcj(jamo)
- 완성형: 조합된 글자 자체에 값을 부여해 인코딩
    - j2hcj 함수는 조합형 한글 문자열을 자소 단위로 나눠 반환
    - 초성, 중성, 종성으로 자소 단위로 나눔

```
from jamo impot h2j, jh2cj
review = “한글 문자열”
decomposed = j2hcj(h2j(review))
tokenized = list(decomposed)
print(tokenized)
```

→ 접사와 문장 부호의 의미 학습 가능, OOV 줄일 수 있음

형태소 토큰화(Morpheme Tokenization)

텍스트를 형태소 단위로 나누는 토큰화 방법

- 언어의 문법과 구조를 고려해 단어를 분리하고, 의미있는 단위로 분류
- 한국어와 같은 교착어(Agglutinative Language)인 언어에서 중요하게 수행됨

- 형태소
  - 자립 형태소(Free Morpheme): 스스로 의미를 가짐, 명사, 동사, 형용사
  - 의존 형태소(Bound Morpheme): 스스로 의미를 갖지 못하고 다른 형태소와 조합디어 사용됨, 조사, 어미, 접두사, 접미사

형태소 어휘 사전(Morpheme Vocabalary)

: 자연어 처리에서 사용되는 단어의 집합인 어휘사전 중에서도 각 단어의 형태소 정보를 포함하는 사전

- 각 형태소가 어떤 품사에 속하는지, 해당 품사의 뜻 등 정보를 제공

품사 태깅(POS Tagging)

: 텍스트 데이터를 형태소 분석하여 각 형태소에 해당하는 품사를 태깅하는 작업

- 그(명사) + 는(조사) + 나(명사) + 에게(조사) + 인사(명사) + 를(조사) + 했다(동사)

KoNLPy 256p

> 💡 한국어 처리를 위해 개발된 라이브러리로, 명사 추출, 형태소 분석, 품사 태깅 등의 기능을 제공
>
> - 텍스트 마이닝, 감성 분석, 토픽 모델링 등 다양한 NLP 작업에 사용

- 자바 언어 기반의 한국어 형태소 분석기 - JDK 기반
- Okt(Open Korean Text), 꼬꼬마(Kkma), 코모란(Komoran), 한나눔(Hannanum), 메캅(Mecab) 등의 형태소 분석기 제공

<대표적인 메서드>

- 명사 추출 okt. nouns
- 구문 추출 okt.phrases
- 형태소 추출 okt.morphs
- 품사 태깅 okt.pos

NLTK

💡 Natural Language Toolkit

자연어 처리를 위해 개발된 라이브러리로 토큰화, 형태소 분석, 구문 분석, 개체명 인식, 감성 분석 등 기능 제공

spaCy

💡 : 사이썬 기반으로 개발된 오픈 소스 라이브러리

NLTK와 마찬가지로 자연어 처리 기능 제공

빠른 속도와 높은 정확도를 목표로 하는 머신러닝 기반 자연어 처리 라이브러리
→ 크고 복잡, 많은 리소스 요구

↔ NLTK: 학습 목적으로 자연어 처리에 대한 다양한 알고리즘과 예제 제공

하위 단어 토큰화

- 형태소: 자연어의 최소 의미 단위로, 대부분의 자연어는 형태소의 조합으로 이루어짐
  - 외래어, 띄어쓰기 오류, 오탈자 등이 있는 문장 분석에서 문제 발생
  - 형태소 분석기: 전문어나 고유어가 많은 데이터를 처리할 때 약점을 보임

💡 **하위 단어 토큰화(Subword Tokenization)**

: 하나의 단어가 빈번하게 사용되는 하위단어의 조합으로 나누어 토큰화하는 장법

- Reinforcement 처리 시 Rein, force, ment 등으로 나눠서 처리
- 단어의 길이를 줄일 수 있어서 처리 속도가 빨라짐
- OOV 문제, 신조어, 은어, 고유어 등으로 인한 문제를 완화할 수 있음.
- 바이트 페어 인코딩, 워드피스, 유니그램 모델 등

바이트 페어 인코딩(Byte Pair Encoding, BPE)

💡 = 다이그램 코딩(Diagram Coding)

하위 단어 토큰화의 한 종류

텍스트 데이터에서 가장 빈번하게 등장하는 글자 쌍의 조합을 찾아 부호화하는 압축 알고리즘

자연어 처리 분야에서 하위 단어 토큰화를 위한 방법

- 연속된 글자 쌍이 더이상 나타나지 않거나, 정해진 어휘 사전 크기에 도달할 때까지 조합 탐지와 부호화를 반복
- 이 과정에서 자주 등장하는 단어는 하나의 토큰으로 토큰화
- 덜 등장하는 단어는 여러 토큰의 조합으로 표현

👉 **원문: abracadabra**

Step 1: AracadAra

Step 2: ABcadAB

Step 3: CcadC

- 센텐스피스(Sentencepiece)
- 코포라(Kopora)
- 워드피스(WordPiece): 바이트페어인코딩 토크나이저와 유사한 방법으로 학습되지만, 빈도 기반이 아닌 확률 기반으로 글자 쌍을 병합함
  - 확률 정보 사용: 모델이 새로운 하위 단어를 생성할 때 이전 하위 단어와 함께 나타날 확률을 계산해 가장 높은 확률을 가진 하위 단어 선택
  - 글자 쌍 병합 점수 score = f(x,y) / {f(x),f(y)}
  - f는 빈도(frequency)를 나타내는 함수, x,y는 하위단어
- 토크나이저스(Tokenizers) 라이브러리
  - 정규화(Normalization)과 사전 토큰화(Pre-tokenization) 제공

6. 임베딩

👉 **텍스트 벡터와(Text Vectorization)**

: 컴퓨터가 이해할 수 있도록 텍스트를 숫자로 변환

- **원-핫 인코딩(One-Hot Encoding):** 문서에 등장하는 각 단어를 고유한 색인 값으로 매핑한 후, 해당 색인 위치를 1로 표시하고, 나머지 위치는 모두 0으로 표시
- **빈도 벡터화(Count Vectorization)**: 문서에서 단어의 빈도를 세어 해당 단어의 빈도로 표시함

<단점>

- 쉽고 간단하지만 벡터의 희소성(Sparsity)이 크다
- 말뭉치 내에 존재하는 토큰의 개수만큼 벡터 차원을 가져야 하지만, 입력 문자 내에 존재하는 토큰의 수가 현저히 적어 컴퓨팅 비용의 증가와 차원의 저주 등의 문제 발생
- 텍스트 벡터가 입력 텍스트의 의미 내표X → 두 문장이 의미적으로 유사하다고 해도 벡터가 유사하게 나타나지 않을 수 있음.

⇒ 이러한 문제를 해결하기 위해 워드 투 벡터(Word2Vec)나 패스트 텍스트(fastText) 등과 같이 단어의 의미를 학습해 표현하는 **워드 임베딩** 기법을 사용

👉 **워드 임베딩(Word Embedding)**

: 단어를 고정된 길이의 실수 벡터로 표현하는 방법으로, 단어의 의미를 벡터 공간에서 다른 단어와 상대적 위치로 표현해 단어 간의 관계를 추론함

- 고정된 임베딩을 학습하기 때문에 다의어나 문맥 정보를 다루기 어렵다는 단점이 있어, 인공 신경망을 활용해 동적 임베딩(Dynamic Embedding) 기법을 사용함

언어 모델(Language Model)

: 입력된 문자으로 각 문장을 생성할 수 있는 확률을 계산하는 모델

- 자동 번역, 음성 인식, 텍스트 요약 등 다양한 자연어 처리 분야에서 활용됨

📎 **자기회귀 언어 모델(Autoregressive Language Model)**

: 입력된 문장들의 조건부 확률을 이용해 다음에 올 단어를 예측

- 이를 위해 이저넹 등장한 모든 토큰의 정보를 고려하며, 문장의 문맥 정보를 파악하여 다음 단어를 생성함. 이 과정을 반복

조건부 확률의 연쇄법칙 (Chain rule for conditional probability를 적용한 언어 모델의 조건부 확률

수식 6.2 조건부 확률의 연쇄법칙을 적용한 언어 모델의 조건부 확률

$$P(w_t|w_1, w_2, \cdots, w_{t-1}) = P(w_1)P(w_2|w_1)\cdots P(w_t|w_1, w_2, \cdots, w_{t-1})$$

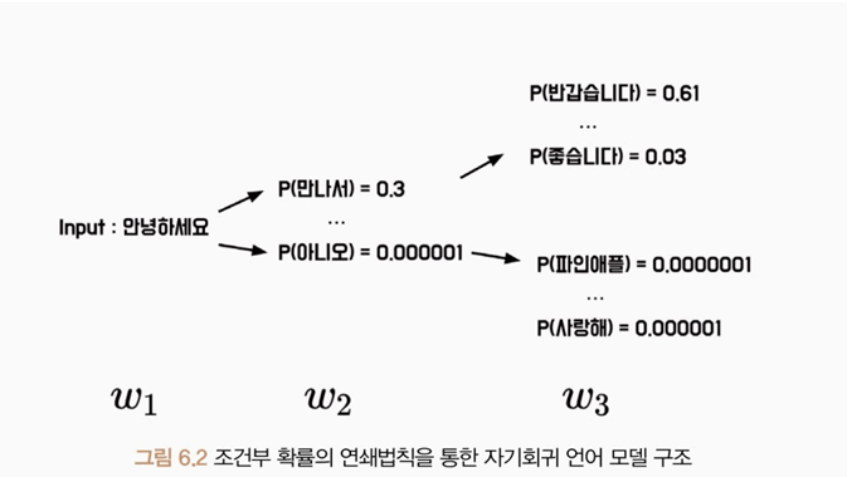

그림 6.2 조건부 확률의 연쇄법칙을 통한 자기회귀 언어 모델 구조

자기회귀 언어 모델에서는 각 시점에서 다음에 올 토큰을 예측하는 것이 중요하다. 이를 위해 w1을 바탕으로 w2가 등장할 확률 P(w2|w1)을 예측 → P(w3|w2)를 예측

→ 모델의 출력값이 모델의 입력값으로 사용되는 특징

📎 **통계적 언어 모델(Statistical Language Model)**

: 언어의 통계적 구조를 이용해 문장이나 단어의 시퀀스를 생성하거나 분석함

시퀀스에 대한 확률 분포를 추정해 문자으이 문맥을 파악 → 다음 단어의 확률을 예측

- 마르코프 체인(Markov Chain)을 이ㅛㅇ해 구현

- 단어의 순서와 빈도에만 기초해 문장의 확률을 예측 → 문맥을 제대로 파악하지 못하면 불완전/부적절한 결과를 생성할 수 있음
- 한 번도 등장한 적이 없는 단어나 문장에 대해 정확한 확률을 예측하기 어려움

📌 **데이터 희소성(Data Sparsity):** 관측한 적이 없는 데이터를 예측하지 못하는 문제

- 통계적 언어 모델은 기존에 학습한 텍스트 데이터에서 패턴을 찾아 확률 분포를 생성 → 이를 이용해 문장을 생성할 수 있으며, 다양한 종류의 텍스트 데이터를 학습할 수 있음
- 대규모 자연어 데이터를 처리하는 데 효과적이며, 딥러닝 등을 통해 더욱 강력한 모델을 구현할 수 있게 됨

⚙️ **GPT(Generative Pre-trained Transformer)**

- Raw Sentence: GPT는 OpenAI에서 개발한 인과적 언어모델입니다.
- Input: GPT는 OpenAI에서 개발한
- Preidiction-1: 인과적
- Prediction-2: 언어 모델 입니다.

BERT(Bidirectional Encoder Representations from Transformers)

- BERT는 Google에서 개발한 마스킹된 언어모델입니다.
- Input: Bert는 [MASK]에서 개발한 [MASK] 언어 모델 입니다.
- Prediction: Google, 마스킹된

→ GPT와 BERT 모델은 자연어 처리 분야에서 혁신적 성과를 거둠

N-gram

: 가장 기초적인 통계적 언어 모델로, 텍스트에서 N개의 연속된 단어 시퀀스를 하나의 단위로 취급하여 특정 단어 시퀀스가 등장할 확률을 추정함

👉 입력 텍스트를 하나의 토큰 단위로 분석하지 않고 N개의 토큰을 묶어서 분석, 연속된 N개의 단어를 하나의 단위로 취급하여 추론

- N이 1일 때는 유니그램(Unigram)
- N이 2일 때는 바이그램(Bigram)
- N이 3일 때는 트라이그램(Trigram)
- N≥4 N-gram

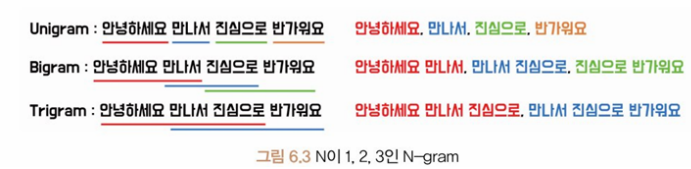

그림 6.3 N이 1, 2, 3인 N-gram

N-gram 언어 모델은 모두 토큰을 사용하지 않고 N-1개의 토큰만을 고려해 확률을 계산함

수식 6.4 N-gram에서의 조건부 확률

$$P(w_t \mid w_{t-1}, w_{t-2}, \cdots, w_{t-N+1})$$

- 이전 단어들의 개수를 결정하는 N의 값을 조정하여 N-gram 모델의 성능을 조절할 수 있음
- 작은 규모의 데이터세트에서 연속된 문자열 패턴을 분석하는 데 큰 효과를 보임
- 관용적 표현 분석에 활용됨
- 연속된 n개의 단어를 추출하므로, 단어의 순서가 중요한 자연어 처리 작업 및 문자열 패턴 분석에 활용됨

TF-IDF

👉 **Term Frequency-Inverse Document Frequency**

: 텍스트 문서에서 특정 단어의 중요도를 계산하는 방법으로, 문서 내에서 단어의 중요도를 평가하는 데 사용되는 통계적인 가중치를 의미

- BoW(Bag-of-Word)에 가중치를 부여하는 방법
  - BoW는 문서나 문장을 단어의 집합으로 표현하는 방법
  - 문서나 문장에 등장하는 단어의 중복을 허용해 빈도를 기록
  - 등장 빈도 고려 ↔ 원핫 인코딩: 단어의 등장 여부

표 6.2 BoW 벡터화

	I	like	this	movie	don't	famous	is
This movie is famous movie	0	0	1	2	0	1	1
I like this movie	1	1	1	1	0	0	0
I don't like this movie	1	1	1	1	1	0	0

BoW를 이용해 벡터화하는 경우 모든 단어는 동일한 가중치를 가짐 → 영화 리뷰의 긍/부정 모델을 만든다고 가정하면 성능이 높지 않을 것임

👉 **단어 빈도(Term Frequency, TF)**

: 특정 단어의 빈도수를 나타내는 값

TF(t,d) = count(t,d)

- TF가 높을수록 해당 단어가 특정 문서에서 중요한 역할을 한다고 생각할 수도 있지만, 단어 자체가 특정 문서 내에서 자주 사용되는 단어이므로 전문 용어나 관용어로 간주할 수 있음.

문서 빈도(Document Frequency, DF)

: 한 단어가 얼마나 많은 문서에 나타나는지

수식 6.6 문서 빈도

$$DF(t,D) = count(t \in d : d \in D)$$

- DF 값이 높으면 특정 단어가 많은 문서에서 등장한다고 볼 수 있음 - 일반적으로 널리 사용, 낮은 중요도
- DF 값이 낮으면 특정한 문맥에서만 사용되는 단어일 가능성이 있으며, 중요도가 높을 수 있다.

역문서 빈도(Inverse Document Frequency, IDF)

: 전체 문서 수를 문서 빈도로 나눈 다음에 로그를 취한 값

수식 6.7 역문서 빈도

$$IDF(t,D) = \log\left(\frac{count(D)}{1 + DF(t,D)}\right)$$

- 문서 빈도가 높을수록 해당 단어가 일반적이고, 상대적으로 중요하지 않다는 의미가 됨 → 문서 빈도의 역수를 취하면 단어의 빈도수가 적을수록 IDF 값이 커지게 보정하는 역할을 함
- IDF는 분모의 DF 값에 1을 더한 값을 사용함

  특정단어가 한 번도 등장하지 않아 분모가 0이 되느 경우를 방지
- IDF는 로그를 취해 너무 큰 값이 나오는 것을 방지하고 정교한 가중치를 얻음

TF-IDF는 문서 빈도와 역문서 빈도를 곱한 값으로 사용

수식 6.8 TF–IDF

$$TF-IDF(t,d,D) = TF(t,d) \times IDF(t,d)$$

: 문서 내에 단어가 자주 등장하지만 전체 문서에 해당 단어가 적게 등장 → TF-IDF 값은 커짐

→ 전체 문서에서 자주 등장할 확률이 높은 관사나 관용어 등의 가중치는 낮아짐