9기_DL팀_김서영_요약본 (10주차 발표)

1) 토큰화 개요

- **자연어 vs. 자연어 처리(NLP)**: 사람이 쓰는 언어를 컴퓨터가 이해·해석·생성하도록 만드는 기술. 모델 개발 시 **모호성·가변성·구조** 문제를 다룸.
- **핵심 용어**: 말뭉치(corpus), 토큰(token), 토큰화(tokenization), 토크나이저(tokenizer).
- **토크나이저 구성 방식**: 공백 분할, 정규표현식, **어휘 사전 기반**(OOV 주의), **머신러닝 기반** 등. 토크나이저 선택은 **성능과 처리 결과**에 큰 영향.

<핵심 포인트>

- 토큰화는 모든 NLP 파이프라인의 출발점.
- OOV(사전에 없는 단어) 처리 전략이 품질에 결정적.

2) 단어/글자 토큰화

(a) 단어 토큰화

- 문장을 **단어 단위**로 분리. 품사 태깅, 개체명 인식, 번역 등에서 보편적 전처리.
- 간단 구현(split) 가능하나, 언어별 띄어쓰기·구두점 규칙 차이로 한계 존재.

(b) 글자 토큰화

- **문자 단위** 분해 → **작은 어휘**로 자원 효율 ↑, 학습 기회 ↑.
- 단점: 개별 글자의 **의미 빈약**, 긴 시퀀스 → 연산량 증가.

(c) 한글 **자모(jamo)** 토큰화

- 장점: 접사/문장부호 의미 학습 가능, **OOV 크게 감소**.
- 단점: 토큰 의미 조합을 모델이 전부 학습해야 함(다의어/동음이의어 구별 어려움), 시퀀스 길어져 연산량 ↑.

3) 형태소 토큰화

- **정의**: 문법·구조를 고려해 단어를 **형태소**(의미 최소 단위)로 분해. 한국어(교착어)에서 중요.
- **형태소 어휘 사전**: 단어의 **형태소 조합**과 **품사 정보**를 포함.
- **KoNLPy**: 한국어용 라이브러리.
  - **Okt**: `nouns/phrases/morphs/pos` 지원.
  - **Kkma**: 명사·문장·형태소·품사 추출(구문 추출은 미지원).
  - **Komoran** 등 다양한 분석기 존재. **분석기마다 결과가 달라** 목적에 맞는 선택 필요.
- **NLTK**: 다언어 지원이나 **영어 중심**, 토큰화·품사태깅에 **모델 다운로드** 필요(예: Punkt, Averaged Perceptron Tagger, 총 35 POS).
- **spaCy**: Cython 기반 **빠르고 정확**한 생산용 라이브러리. 모델이 크고 리소스 요구 ↑. **NLTK(학습용)**와 대비.
- **형태소 기반 한계:** 띄어쓰기/품사 오류 전파, 도메인·언어 편향, OOV, 다국어 확장 비용 등.

4) 하위 단어(Subword) 토큰화
(a) 아이디어
- **문자 ↔ 단어**의 중간 단위로 분해. **OOV 감소**, 어휘 축소, 어미·접사 변형 대응.

(b) BPE(Byte Pair Encoding)
- 절차: **문자 시작 → 가장 자주 등장하는 인접 쌍을 반복 병합** → 병합 규칙(merges) + 어휘(vocab) 산출.
- 특징: 빈도 기반, 간단·효율적, 신어 일반화 강함. (예: “lowest” → `low` + `est`).

(c) SentencePiece & Korpora
- **SentencePiece**: 공백 비의존, **raw text 바로 학습**, BPE/Unigram 지원, 특수토큰/정규화 포함, 재현성 우수.
- **Korpora**: 한국어 공개 코퍼스 수집 패키지(예: 네이버 영화평) — 실습용 말뭉치 준비를 쉽게.
- **학습 공통 프로세스**: 데이터 수집 → 정규화 → 프리토큰화 → 모델 학습(BPE/WordPiece/Unigram, vocab 크기) → 포스트프로세싱([CLS]/[SEP]) → 평가(토큰 길이·커버리지·OOV·Perplexity 대리지표) → 저장/배포.

(d) WordPiece (BERT 계열)
- **차이점**: BPE가 **빈도 최다 쌍** 병합인 데 비해, WordPiece는 **언어모델 우도/점수 최대**가 되는 쌍을 선택. 중간 조각은 `##` 접두사.
- 실습 포인트: 정규화(NFD, lowercase), 사전 토큰화(Whitespace), 인코딩/디코딩 일관성, **디코더 설정 필수**.

(e) Hugging Face **tokenizers**
- 러스트 기반 고성능, 파이썬 바인딩. BPE/WordPiece/Unigram/ByteLevel 지원.
- 정규화·프리토큰화·포스트프로세싱 **모듈화**, 대용량 코퍼스에서도 빠름. `transformers`와 호환.

5) 임베딩(Embedding) 및 벡터화
- **텍스트 벡터화**: 텍스트 → 숫자.
  - **원-핫**: 단순하나 차원 ↑, 희소, 의미 반영 X.
  - **Count(빈도)**: 출현 빈도 벡터. 희소/순서·의미 미반영.
- **워드 임베딩**: Word2Vec, fastText — 의미를 **연속 실수 벡터**로 표현, 유사성/관계 추론.
  - 한계: **고정 임베딩** → 문맥·다의어 처리 약함 → **동적 임베딩** 필요.

6) 언어 모델(Language Model)
- 목표: 문장의 **확률**을 추정하고 다음 토큰 예측.
- **자기회귀(AR) LM**: 이전 토큰 시퀀스 조건에서 다음 토큰 확률 예측.
  - **연쇄법칙**: 다음 토큰 **분류 문제**로 정의, 모델 출력이 입력으로 재귀.
- **통계적 LM**: 마르코프 체인 기반 **빈도 기반 조건부 확률**. 데이터 희소성 문제가 큼.

- **N-gram**: N개 연속 토큰을 하나의 단위로 보고 확률 추정(유니그램/바이그램/트라이그램/…); N 조절로 성능·복잡도 트레이드오프.

7) TF-IDF

- BoW에 **가중치**를 주는 방식. 단어의 **중요도** 평가.
  - **TF**: 특정 문서 내 단어 빈도.
  - **DF**: 몇 개 문서에 나타나는지.
  - **IDF**: $\log(\frac{\text{전체문서수}}{DF})$
  - **TF-IDF**: TF × IDF.
- **Vectorizer 주요 옵션**: `input`(content/file/filename), `encoding`, `lowercase`, `stop_words`, `ngram_range`, `max_df`, `min_df`, `vocabulary`, `smooth_idf` 등.
- **API 사용 흐름**: `fit`(사전 구축) → `transform`(벡터화) 또는 `fit_transform`. `toarray()`로 밀집 배열, `vocabulary_`로 단어-색인 조회.
- **한계**: 문맥/순서·의미 반영 X → 생성·의미 유사성 작업에는 부적합.