개념정리 #14

딥러닝 파이토치 교과서 9장

9. 자연어 전처리
    9.1 자연어 처리란
    9.2 전처리

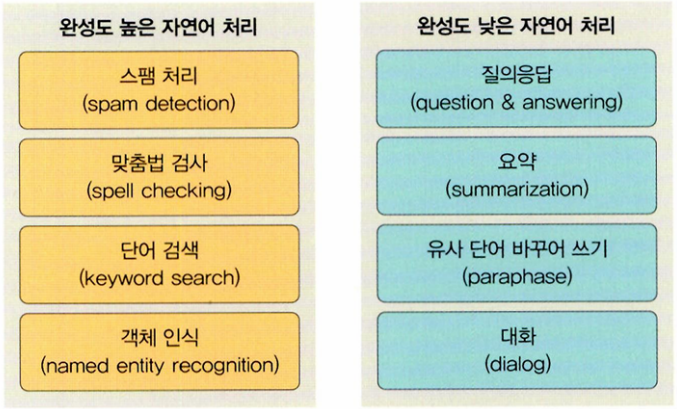
9. 자연어 전처리

9.1 자연어 처리란

자연어 처리 : 우리가 일상생활에서 사용하는 언어 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 과정

→ 인간 언어(자연어)에 대한 이해 + 딥러닝에 대한 이해가 필요

- 언어는 종류가 다르고 그 형태가 다양하기 때문에 처리가 매우 어려움
- 단어 단위의 임베딩이 어려운 언어 존재함
- 자연어 처리를 위해 사용되는 낯선 용어들



자연어 처리가 가능한 영역 / 발전이 필요한 분야

자연어 처리 용어 및 과정

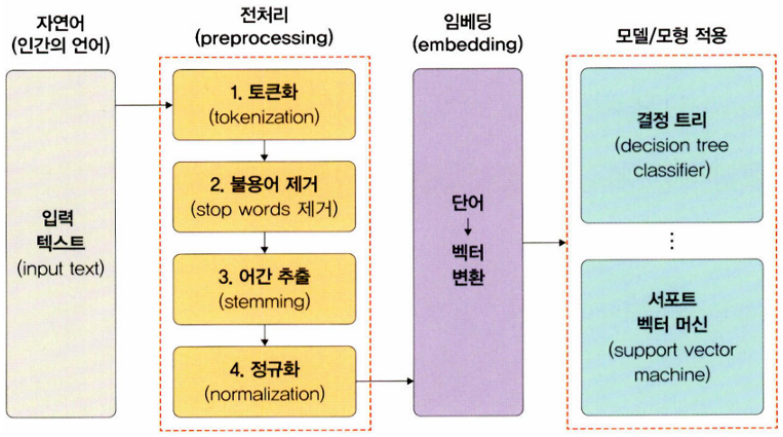
<자연어 처리 용어>

- **말뭉치 corpus** : 자연어 처리에서 모델을 학습시키기 위한 데이터이며, 자연어 연구를 위해 특정한 목적에서 표본을 추출한 집합
- **토큰 token** : 자연어 처리를 위한 document를 나누는 단위
  - 토큰 생성 tokenizing : 문자열을 토큰으로 나누는 작업
  - 토큰 생성 함수 : 문자열을 토큰으로 분리하는 함수
- **토큰화 tokenization** : 텍스트를 문장이나 단어로 분리하는 것
- **불용어 stop words** : 문장 내에서 많이 등장하는 단어. 중요도는 낮지만 빈도가 높게 나타나기 때문에 사전에 제거해 주어야 함.
- **어간 추출 stemming** : 단어를 기본 형태로 만드는 작업
- **품사 태깅 part-of-speech tagging** : 문장에서 품사를 식별하기 위해 붙여 주는 태그(식별 정보)

<자연어 처리 과정>

자연어를 컴퓨터가 이해할 수 있는 언어로 바꾸고 원하는 결과를 얻기까지의 과정

1. 입력 텍스트: **자연어**
2. 입력된 텍스트에 대한 **전처리** 과정
3. 전처리된 단어들을 **임베딩**
4. **분류 및 예측** 수행



자연어 처리를 위한 라이브러리

NLTK (Natural Language ToolKit)

- 교육용으로 개발된 자연어 처리 및 문서 분석용 파이썬 라이브러리
- 제공하는 주요 기능 : 말뭉치, 토큰 생성, 형태소 분석, 품사 태깅

KoNLPy

- 한국어 처리를 위한 파이썬 라이브러리
- 파이썬에서 사용할 수 있는 오픈 소스 형태소 분석기
- 기존에 공개된 KKma, Komoran, Hannanum, Twitter, Mecab 분석기를 한 번에 설치하고 동일한 방법으로 사용할 수 있도록 함
- 제공하는 주요 기능 : 형태소 분석, 품사 태깅

Gensim

- 파이썬에서 제공하는 워드투벡터(Word2Vec) 라이브러리
- 딥러닝 라이브러리는 아니지만 효율적이고 확장 가능함
- 제공하는 주요 기능 : 임베딩-Word2Vec, 토픽 모델링, LDA

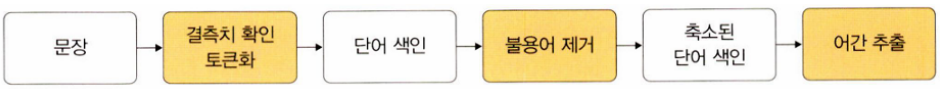
> 💡 토픽 모델링 : 텍스트 본문의 숨겨진 의미 구조를 발견하는 데 사용되는 텍스트 마이닝 기법. 각 주제별로 단어 표현을 묶어 주는 것.

> 💡 LDA (Latent Dirichlet Allocation) : 주어진 문서에 대해 각 문서에 어떤 주제들이 존재하는지를 서술하는 확률적 토픽 모델 기법

Scikit-learn

- 파이썬을 이용해 문서를 전처리할 수 있는 라이브러리를 제공함
- 특히 자연어 처리에서 특성 추출(feature extraction) 용도로 많이 사용됨
- 제공하는 주요 기능
  - `CountVectorizer` : 텍스트에서 단어 빈도를 기준으로 특성 추출
  - `Tfidfvectorizer` : TF-IDF 값을 사용해 특성 추출
  - `HashingVectorizer` : 빈도 기준 특성 추출 but 해시 함수 사용해 실행 시간 감소

9.2 전처리



전처리 과정

결측치 확인

결측치 : NaN. 데이터가 없는 것

- 결측치 확인

```
df.isnull().sum()
```

- 결측치 처리
  - 모든 값이 NaN인 행 삭제

```
df = df.dropna(how='all')
df.dropna(how='all', inplace='True')
```

  - 결측치가 존재하는 행 삭제

```
df = df.dropna()
df = df.dropna(axis=0)
```

  - 결측치를 0으로 채우기

```
df = df.fillna(0)
```

  - 결측치를 해당 열의 평균값으로 채우기

```
df['x'].fillna(df['x'].mean(), inplace=True)
```

  - 데이터가 거의 없는 feature(열)은 feature(열) 자체를 삭제
  - 최빈값 혹은 평균값으로 결측치 대체

토큰화

Tokenization : 주어진 텍스트를 단어/문자 단위로 자르는 것

- **문장 토큰화**

  문장의 종결 부호에 따라 문장을 분리

- **단어 토큰화**

  띄어쓰기를 기준으로 문장을 구분

  (But 한국어는 띄어쓰기만으로는 토큰을 구분하기 어려움. KoNLPy를 사용함.)

불용어 제거

불용어 stop word : 문장 내에서 빈번하게 발생하여 의미를 부여하기 어려운 단어들 (e.g. 'a', 'the', 'of')

→ 자연어 처리에 있어 효율성을 감소시키고 처리 시간이 길어지는 단점이 있기 때문에 제거 必

어간 추출

어간 추출 stemming, 표제어 추출 lemmatization : 단어 원형(어근)을 찾아 주는 것

- 어간 추출 : 단어 그 자체만 고려하기 때문에 품사가 달라도 사용 가능함

  → 사전에 없는 단어도 추출 가능함
- 표제어 추출 : 단어가 문장 속에서 어떤 품사로 쓰였는지 고려하기 때문에 품사가 같아야 사용 가능함

  → 사전에 있는 단어만 추출할 수 있음

NLTK의 어간 추출

- **포터 알고리즘** : 단어 원형이 비교적 잘 보존됨
- **랭커스터 알고리즘** : 단어 원형을 알아볼 수 없을 정도로 축소시키기 때문에 정확도가 낮음 → 일반적인 상황보다는 데이터셋을 축소시켜야 하는 특정 상황에서나 유용함

표제어 추출

👍 일반적으로 어간 추출보다 성능이 더 좋음. 품사와 같은 문법뿐만 아니라 문장 내에서 단어 의미도 고려하기 때문

👎 시간이 오래 걸림

`WordNetLemmatizer`를 주로 사용함

정규화 Normalization

: 데이터셋이 가진 특성(칼럼)의 모든 데이터가 동일한 정도의 스케일을 갖도록 하는 것

값이 크다고 해서 분석에 더 중요한 요소라고 간주할 수 없기 때문에 정규화가 필요함

- `MinMaxScaler()` : 0과 1 사이의 값으로 스케일링. 이때 특정 범위에서 많이 벗어난 데이터의 경우 좁은 범위로 압축될 수 있음 → 아웃라이어에 민감함
- `StandardScaler()` : 각 특성의 평균을 0, 분산을 1로 변경해 칼럼 값의 범위를 조정함
- `RobustScaler()` : 평균과 분산 대신 median과 IQR을 사용함
- `MaxAbsScaler()` : 절댓값이 0~1 사이 (값이 -1~1 사이) 가 되도록 스케일링. 큰 아웃라이어에 민감함