🔟
자연어 전처리

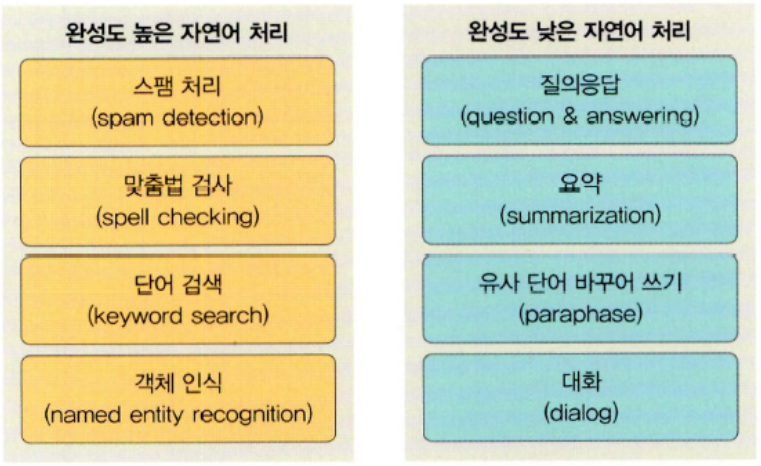
날짜	@2024년 12월 18일 → 2024년 12월 23일
범위	9장
상태	완료
주차	14주차

9장 자연어 전처리

9.1 자연어 처리란

- 자연어 처리 : 우리가 일상생활에서 사용하는 언어 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 과정.
    - 딥러닝에 대한 이해도 필요하지만, 그에 앞서 인간 언어에 대한 이해도 필요하기 때문에 접근하기 어려움.
    - 언어 종류가 다르고 그 형태가 다양하기 때문에 처리가 매우 어려움. ex) 영어는 명확한 띄어쓰기가 있지만, 중국어는 띄어쓰기가 없기 때문에 단어 단위의 임베딩이 어려움.
    - 자연어 처리를 위해 사용되는 용어들도 낯섬.
- 자연어 처리가 가능한 영역과 발전이 필요한 분야

❯ 그림 9-1 자연어 처리 완성도



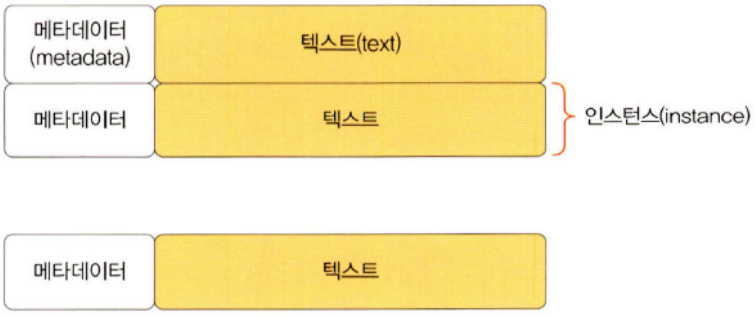
- 스팸 처리 및 맞춤법 검사는 완성도 높음.
- 질의응답 및 대화는 아직 발전이 더 필요한 분야.

9.1.1 자연어 처리 용어 및 과정

- 자연어 처리 관련 용어

1. 말뭉치(corpus(코퍼스)) : 자연어 처리에서 모델 학습시키기 위한 데이터, 자연어 연구를 위해 특정한 목적에서 표본을 추출한 집단.

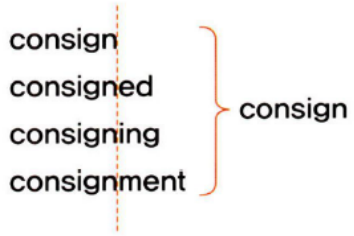
❯ 그림 9-2 말뭉치(corpus)



2. 토큰(token) : 자연어 처리를 위한 문서는 작은 단위로 나누어야 하는데, 이때 문서를 나누는 단위.
    - 문자열을 토큰으로 나누는 작업 → 토큰 생성(tokenizing)
    - 문자열을 토큰으로 분리하는 함수 → 토큰 생성 함수
3. 토큰화(tokenization) : 텍스트를 문장이나 단어로 분리하는 것. 토큰화 단계를 마치면 텍스트가 단어 단위로 분리됨.

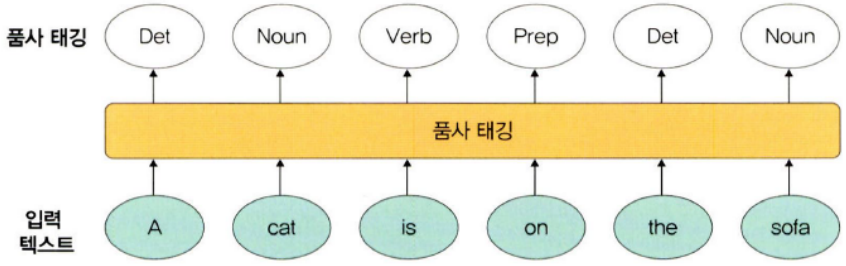
4. 불용어(stop words) : 문장 내에서 많이 등장하는 단어. 분석과 관계없으며, 자주 등장하는 빈도 때문에 성능에 영향을 미치므로 사전에 제거해 주어야 함.
   ex) “a”, “the”, “she”, “he” 등이 있음.
5. 어간 추출(stemming) : 단어를 기본 형태로 만드는 작업. ex) ‘consign’, ‘consigned’, ‘consigning’, ‘consignment’가 있을 때 기본 단어인 ‘consign’으로 통일하는 것.

그림 9-3 어간 추출



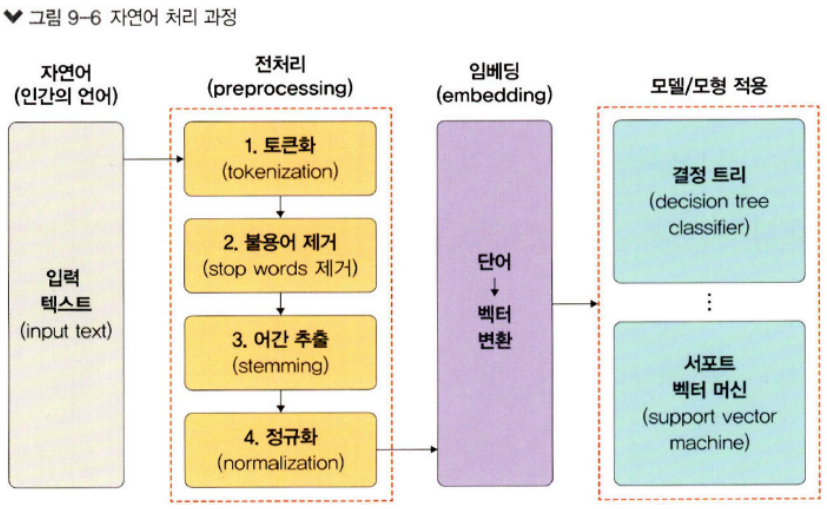
6. 품사 태깅(part-of-speech tagging) : 주어진 문장에서 품사 식별하기 위해 붙여주는 태그(식별 정보) 의미함.

그림 9-4 품사 태깅



- Det : 한정사
- Noun : 명사
- Verb : 동사
- Prep : 전치사
- 품사 태깅은 NLTK 이용할 수 있음.
  - 문장 토큰화(nltk)
  - 태깅에 필요한 자원 내려받기
  - 품사 태깅
    - VBZ : 동사, 동명사 또는 현재 분사

        - PRP : 인칭 대명사(PP)
        - JJ : 형용사
        - VBG : 동사, 동명사 또는 현재 분사
        - NNS : 명사, 복수형
        - CC : 등위 접속사
- 자연어 처리 과정
    - 자연어는 인간 언어, 인간 언어는 컴퓨터가 이해할 수 없기 때문에 컴퓨터가 이해할 수 있는 언어로 바꾸고 원하는 결과를 얻기 까지 크게 **4단계**를 거침.
        1. 인간 언어인 자연어가 입력 텍스트로 들어오게 됨. 이때 인간 언어가 다양하듯 처리 방식이 조금씩 다르며, 현재는 영어에 대한 처리 방법들이 잘 알려져 있음.
        2. 입력된 텍스트에 대한 전처리 과정 필요함.
        3. 전처리가 끝난 단어들을 임베딩함. 즉, 단어를 벡터로 변환하는 방법.
        4. 컴퓨터가 이해할 수 있는 데이터가 완성되었기 때문에 모델/모형(ex. 결정 트리)을 이용하여 데이터에 대한 분류 및 예측을 수행함. 이때 데이터 유형에 따라 분류와 예측에 대한 결과가 달라짐.



9.1.2 자연어 처리를 위한 라이브러리

- NLTK(Natural Language ToolKit)
    - 교육용으로 개발된 자연어 처리 및 문서 분석용 파이썬 라이브러리.
    - 다양한 기능 및 예제를 가지고 있으며 실무 및 연구에서도 많이 사용되고 있음.
    - 주요 기능

        - 말뭉치
        - 토큰 생성
        - 형태소 분석
        - 품사 태깅
    - NLTK 라이브러리 이용한 예제
        - nltk 라이브러리 호출 및 문장 정의
        - 단어 단위로 분리
- KoNLPy(코엔엘파이)
    - 한국어 처리를 위한 파이썬 라이브러리.
    - 파이썬에서 사용할 수 있는 오픈 소스 형태소 분석기, 기존 공개된 꼬꼬마(Kkma), 코모란(Komoran), 한나눔(Hannanum), 트위터(Twitter), 메카브(Mecab) 분석기를 한 번에 설치하고 동일한 방법으로 사용할 수 있도록 해 줌.
    - 윈도우 환경에서 KoNLPy 설치 방법
        1. Oracle JDK 설치
        2. JPype1 설치
        3. KoNLPy 설치
    - KoNLPy 라이브러리 이용한 예제
        - 라이브러리 호출 및 문장을 형태소로 변환
        - 품사 태깅

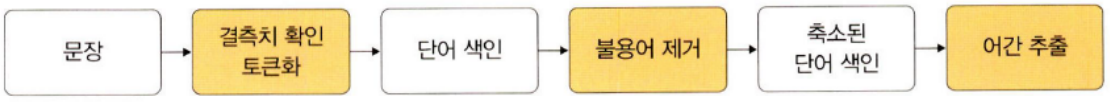
        ⇒ KoNLPy에서 제공하는 주요 기능 : 형태소 분석, 품사 태깅
- Gensim
    - 파이썬에서 제공하는 워드투벡터(Word2Vec) 라이브러리.
    - 딥러닝 라이브러리는 아니지만 효율적이고 확장 가능하기 때문에 폭넓게 사용하고 있음.
    - Gensim에서 제공하는 주요 기능
        - 임베딩 : 워드투벡터
        - 토픽 모델링 : 문서 집합의 추상적인 주제 발견하기 위한 통계적 모델 중 하나, 텍스트 본문의 숨겨진 의미 구조 발견하는 데 사용되는 텍스트 마이닝 기법. 각 주제별로 단어 표현 묶어 주는 것.

        - LDA(Latent Dirichlet Allocation)
- 사이킷런(scikit-learn)
    - 파이썬 이용하여 문서를 전처리할 수 있는 라이브러리 제공함.
    - 특히 자연어 처리에서 특성 추출 용도로 많이 사용됨.
    - 사이킷런에서 제공하는 주요 기능
        - CountVectorizer : 텍스트에서 단어의 등장 횟수를 기준으로 특성 추출함.
        - Tfidfvectierizer : TF-IDF 값을 사용해서 텍스트에서 특성 추출함.
        - HashingVectorizer : CountVectorizer와 방법이 동일하지만 텍스트 처리할 때 해시 함수 사용하기 때문에 실행 시간 감소함.

9.2 전처리

- 머신 러닝이나 딥러닝에서 텍스트 자체를 특성으로 사용할 수는 없음.
- 텍스트 데이터에 대한 전처리 작업이 필요한데, 이때 전처리를 위해 토큰화, 불용어 제거 및 어간 추출 등 작업이 필요함.

그림 9-15 전처리 과정



9.2.1 결측치 확인

⇒ 결측치 : 주어진 데이터셋에서 데이터가 없는(NaN) 것.

- 결측치 확인하기
    - 예제 파일 class2.csv 사용함.
    - 결측치 확인할 데이터 호출
    - 결측치 개수 확인
    - 결측치 비율
- 결측치 처리하기
    - 결측치 삭제 처리
    - 결측치를 0으로 채우기

- 결측치를 평균으로 채우기

+)

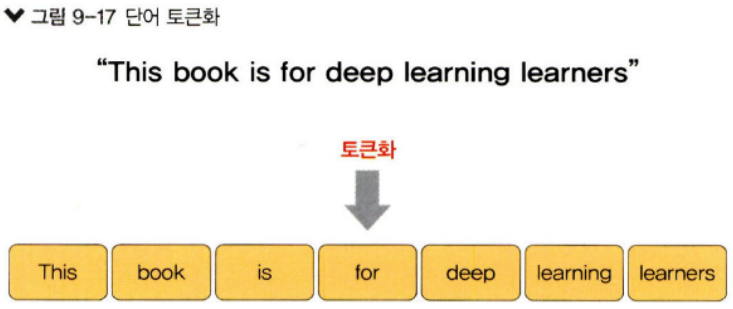
- 데이터에 하나라도 NaN 값이 있을 때 행 전체 삭제
- 데이터가 거의 없는 특성(열)은 특성(열) 자체를 삭제
- 최빈값 혹은 평균값으로 NaN 값을 대체

9.2.2 토큰화

⇒ 토큰화(tokenization) : 주어진 텍스트를 단어/문자 단위로 자르는 것.

- 문장 토큰화와 단어 토큰화로 구분됨.

- 문장 토큰화
  - 마침표(.), 느낌표(!), 물음표(?) 등 문장의 마지막을 뜻하는 기호에 따라 분리하는 것.
  - 문장 단위로 구분.
- 단어 토큰화
  - 띄어쓰기 기준으로 문장을 구분함.



  - 한국어의 경우 띄어쓰기만으로 토큰을 구분하기 어려운 단점(한글 토큰화는 KoNLPy 사용.)
  - 어퍼스트로피에 대한 분류 → NLTK에서 제공하는 WordPunctTokenizer 이용함.
- 한글 토큰화 예제
  - KoNLPy 라이브러리 사용
  - 한글 형태소 분석을 위해 오픈 소스 한글 형태소 분석기(Twitter(Okt)) 사용함.
  - 앞서 생성했던 형태소를 별도 파일로 저장함.

- Word2Vec 모델을 생성한 후 저장함.

9.2.3 불용어 제거

⇒ 불용어(stop word) : 문장 내에서 빈번하게 발생하여 의미를 부여하기 어려운 단어들을 의미함. ex) 'a', 'the' 같은 단어들은 모든 구문에 매우 많이 등장. → 아무런 의미 없음.

- 불용어는 자연어 처리에 있어 효율성 감소시키고 처리 시간이 길어지는 단점. 반드시 제거 필요.
- `stopwords` (NLTK 라이브러리) 통해 불용어 제거

9.2.4 어간 추출

- 어간 추출(stemming) & 표제어 추출(lemmatization) ⇒ 단어 원형을 찾아 주는 것.
- 어간 추출은 단어 그 자체만 고려하기 때문에 품사가 달라고 사용 가능함.
  - Automates, automatic, automation ⇒ automat
- 표제어 추출은 단어가 문장 속에서 어떤 품사로 쓰였는지 고려하기 때문에 품사가 같아야 사용 가능함.
  - am, are, is ⇒ be
  - car, cars, car's, cars' ⇒ car
- 즉, 어간 추출과 표제어 추출은 둘 다 어근 추출이 목적이지만, 어간 추출은 사전에 없는 단어도 추출할 수 있고 표제어 추출은 사전에 있는 단어만 추출할 수 있다는 점.
- 어간 추출 ⇒ 포터(porter) & 랭커스터(lancaster)
  - 포터 : 단어 원형이 비교적 잘 보존되어 있음.
  - 랭커스터 : 단어 원형을 알아볼 수 없을 정도로 축소시키기 때문에 정확도 낮음.
- 표제어 추출 ⇒ 어간 추출보다 추출 성능이 좋음. 품사와 같은 문법뿐만 아니라 문장 내에서 단어 의미도 고려하기 때문에 성능 좋음. 하지만 시간이 더 오래 걸림.
  - WordNetLemmatizer

9.2.5 정규화

- 정규화(normalization) : 데이터셋이 가진 특성(혹은 칼럼)의 모든 데이터가 동일한 정도의 범위(스케일 혹은 중요도)를 갖도록 하는 것.
  - 머신 러닝 / 딥러닝은 데이터 특성들을 비교하여 패턴을 분석함.
  - 이때 각각의 데이터가 갖는 스케일 차이가 크면?

❯ 표 9-2 정규화

Monthly Income	Age	PercentSalary Hike	Relationship Satisfaction	TrainingTimes LastYear	YearsInCurrent Role
5993	23	11	1	0	4
5130	55	23	4	3	7
2090	45	15	2	3	0
2909	60	11	3	3	7
3468	47	12	4	3	2
3068	51	13	3	2	7
2670	19	20	1	3	0
2693	33	22	2	2	0
9526	37	21	2	2	7
5237	59	13	2	3	7

    - MonthltIncome : 0~10000 범위
    - RelationshipSatisfaction : 0~5 범위
    - 상당히 다른 값의 범위를 갖는데, 이 상태에서 데이터 분석하면 MonthlyIncome 값이 더 크기 때문에 상대적으로 더 많은 영향을 미치게 됨.
  - 예제
    - 라이브러리 호출
    - 데이터셋 경로 지정 및 훈련과 테스트 용도로 분리
      - https://www.kaggle.com/saurabh00007/diabetescsv
      - 전체 데이터셋 중 67%는 훈련 용도, 33%는 테스트 용도로 사용함.
    - 훈련과 테스트용 데이터를 정규화
      - 훈련용 데이터는 `StandardScaler()`, 테스트용 데이터는 `MinMaxScaler()` 이용하여 정규화함.

      1. `MinMaxScaler()` : 모든 칼럼이 0과 1 사이에 위치하도록 값의 범위 조정함. 이때 특정 범위에서 많이 벗어난 데이터(이상치)의 경우 좁은 범위로 압축될 수 있음. 이상치에 매우 민감할 수 있기 때문에 주의해야 함.

$$MinMaxScaler() = \frac{x - x_{min}}{x_{max} - x_{min}}$$

(x: 입력 데이터)

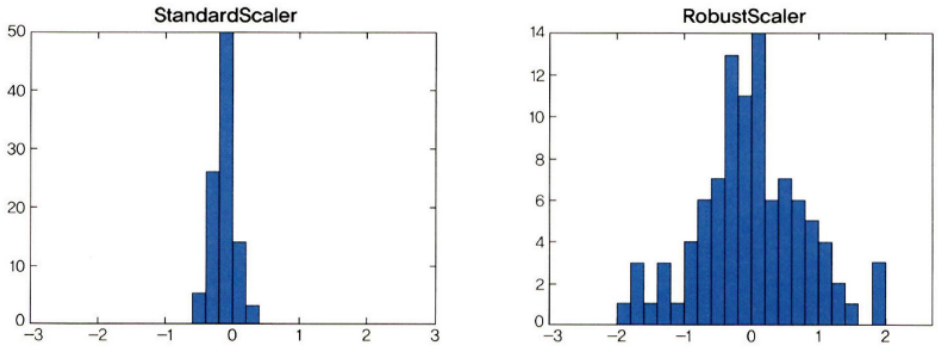
2. `StandardScaler()` : 각 특성의 평균을 0, 분산을 1로 변경하여 칼럼 값의 범위 조정함.

$$StandardScaler() = \frac{x - \mu}{\sigma}$$

(x: 입력 데이터, μ: 평균, σ: 표준편차)

3. `RobustScaler()` : 평균과 분산 대신 중간 값(median)과 사분위수 범위(InterQuartile Range, IQR) 사용함.

그림 9-18 StandardScaler와 RobustScaler 비교



4. `MaxAbsScaler()` : 절댓값이 0~1 사이가 되도록 조정함. 즉, 모든 데이터가 -1~1의 사이가 되도록 조정하기 때문에 양의 수로만 구성된 데이터는 MinMaxScaler()와 유사하게 동작함. 또한, 큰 이상치에 민감하다는 단점.

- 커스텀 데이터셋 생성
  - 데이터셋을 좀 더 다루기 쉽도록 커스컴 데이터셋(`customdataset()`) 생성함.
  - 미니 배치나 데이터를 무작위로 섞는(shuffle) 등의 용도로 사용할 수 있음.
- 데이터로더에 데이터 담기
  - 파이토치 데이터셋과 데이터로더 이용하면 방대한 양의 데이터를 배치 단위로 쪼개서 처리할 수 있고, 데이터 무작위로 섞을 수 있기 때문에 효율적

으로 데이터 처리 가능함.

- 여러 개의 GPU 사용하여 데이터를 병렬로 학습시킬 수도 있음. (데이터양 많을 때 주로 사용, 데이터양 많지 않다면 꼭 사용할 필요 없음.)

- 네트워크 생성
- 손실 함수와 옵티마이저 지정
  1. 이진 분류에서 사용하는 손실 함수로는 이진 교차 엔트로피(Binary Cross Entropy Loss, BCELoss)와 BCEWithLogitsLoss 손실 함수

     BCELoss 손실 함수에 시그모이드(sigmoid) 함수가 함께 결합된 것이 BCEWithLogitsLoss 손실 함수.

     `torch.nn.BCEWithLogitsLoss = torch.nn.BCELoss + torch.sigmoid`
- 모델 성능 측정 함수 정의
  1. torch.round() : 반올림할 때 사용.
- 모델 학습
  - 학습 진행될수록 훈련과 테스트 데이터셋에 대한 성능이 모두 좋아지고 있음.
  - 오차는 줄어들고 정확도는 높아지고 있지만 테스트 데이터셋의 경우 오차가 획기적으로 줄어들지는 않았음.