8. 텍스트 분석 Part1

작성일시	@2024년 12월 21일 오전 9:57
강의 번호	Euron
유형	스터디 그룹
복습	☐

NLP & 텍스트 분석

- NLP (National Language Processing)
    - 머신이 인간의 언어(자연어)를 이해하고 해석하는데 중점을 둠
    - 텍스트 분석을 향상하게 하는 기반 기술
    - 활용 영역: 기계 번역, 질의응답 시스템
- 텍스트 분석
    - 비정형 텍스트에서 의미 있는 정보 추출
    - 활용 영역

텍스트 분류	문서가 특정 분류에 속하는 것을 예측 - 지도 학습 적용
감성 분석	텍스트의 감정, 판단, 믿음 등 주관적 요소를 분석 - 지도, 비지도 학습 모두 적용 가능

텍스트 요약	텍스트의 중요한 주제, 사상 추출 - ex. 토픽 모델링
텍스트 군집화 & 유사도 측정	비슷한 유형의 문서 군집화 수행 - 텍스트 분류를 비지도 학습으로 수행하는 방법 - 유사도 측정 → 비슷한 문서끼리 모으기

1. 텍스트 분석

- 비정형 데이터 (텍스트) 분석
  - **피처 벡터화** (피처 추출)
    1. 텍스트를 word 기반 피처로 추출
    2. 피처에 숫자값 (ex. 단어 빈도수) 부여
    3. 텍스트를 벡터값으로 표현 (= 변환)
       - ex. BOW (Bag of Words), Word2Vec
  - cf) ML 모델: 정형 데이터 기반 모델 수립, 예측

텍스트 분석 수행 프로세스

1. 텍스트 전처리
   - 클렌징
     - 대/소문자 변경, 특수문자 삭제
   - 토큰화
     - 토큰이라 불리는 단위 기준으로 나누는 작업
     - 단어 토큰화
   - 텍스트 정규화
     - 의미 없는 단어 제거, 어근 추출
2. 피처 벡터화 및 추출
3. ML 모델 수립 및 학습, 예측, 평가

파이썬 기반의 NLP, 텍스트 분석 패키지

패키지명	설명
NLTK	- 방대한 데이터 세트, 서브 모듈 포함 - 한계: 수행 속도 느림 ⇒ 대량 데이터에서 활용되지 못함

패키지명	설명
Gensim	- 토픽 모델링 분야에서 많이 쓰임
SpaCy	- 수행 성능 뛰어남

- 사이킷런
  - NLP 패키지에 특화된 라이브러리 갖고 있지 않음
  - 텍스트 가공, 데이터 피처 처리 등은 가능함

2. 텍스트 전처리 - 텍스트 정규화

클렌징

- 불필요한 문자, 기호 등 사전 제거 작업
- HTML, XML 태그, 기호 등 제거

텍스트 토큰화

문장 토큰화	일반적으로 문장 마지막 기호에 따라 분리 → 문장 마침표(.), 개행문자(\n) 등으로 분리 - 문맥상 의미가 중요한 요소 때 사용 - 정규 표현식에 따라 분리	- sent_tokenize( )
단어 토큰화	문장을 단어로 토큰화 → 공백, comma, 마침표 등으로 분리 - 정규 표현식에 따라 분리	- word_tokenize( )

- 단어 토큰화 → 문맥적 의미 무시하게 됨

  ⇒ **n-gram** 도입
  - n-gram
    - 연속적으로 n개의 단어를 하나의 토큰화 단위로 분리
    - ex. Agent Smith knocks the door. (2-gram)

      → (Agent, Smith) (Smith, knocks), (knocks, the) ...

스톱워드 제거

- 분석에 큰 의미 없는 단어 제거
  - is, the, a, will ... (필수 문법 요소)
- 빈번하게 등장해서 중요 단어로 인지될 수 있음

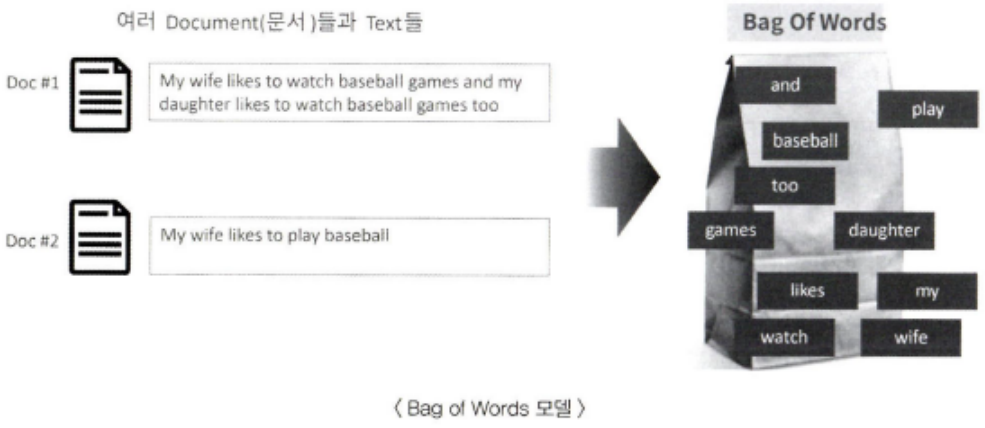
Stemming, Lemmatization

- 문법적, 의미적으로 변하는 단어의 원형 찾기

	Stemming	Lemmatization
설명	- 일반적인 방법, 단순화된 방법 적용 ⇒일부 철자 훼손된 어근 단어 추출하는 경향	- 상대적으로 정교함 - 의미론적인 기반 ⇒ 정확한 철자 어근 단어 추출
변환시간	짧음	오래 걸림
NLTK 제공 라이브러리	Porter, Lancaster, Snowball Stemmer	WordNetLemmatizer

3. BOW

- 문서의 모든 단어에 빈도 값 부여 → 피처값 추출 모델
  - 문맥, 순서 무시함



〈Bag of Words 모델〉

- 피처 추출 과정
  1. 문장 1, 2에 있는 중복된 단어 제거하고 단어를 칼럼 형태로 나열함
  2. 각 단어에 고유한 인덱스 부여

     'and':0, 'baseball':1, 'daughter':2, 'games':3, 'likes':4, 'my':5, 'play':6, 'to':7, 'too':8, 'watch':9, 'wife':10

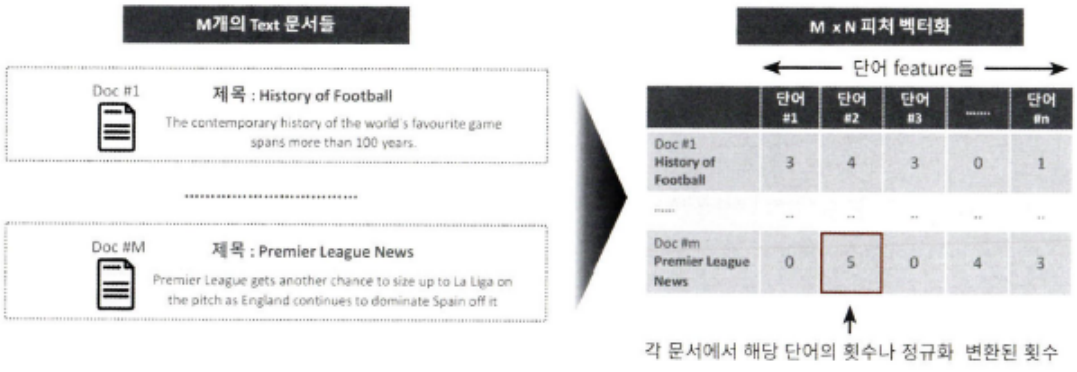
  3. 각 문장에서 단어 등장 횟수 기재함

	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6	Index 7	Index 8	Index 9	Index 10
	and	baseball	daughter	games	likes	my	play	to	too	watch	wife
문장 1	1	2	1	2	2	2		2	1	2	1
문장 2		1			1	1	1	1			1

→ 문장 1에서 baseball은 2회 나타남

장점	단점
모델을 쉽고 빠르게 구축함	순서, 문맥 의미 반영하지 못함 ⇒ 문맥적인 해석이 어려움
	희소 행렬 문제 - 희소 행렬로 인해 수행시간, 예측 성능 떨어짐 - **희소 행렬** : 대부분의 데이터가 0으로 채워지는 행렬 (단어의 종류가 매우 많고, 문서마다 등장하지 않는 단어 생김) - 밀집 행렬 : 대부분의 값이 0이 아닌 값으로 채워지는 행렬

BOW 피처 벡터화



〈 BOW 피처 벡터화 〉

1. 카운트 기반 벡터화
    - 각 문서에서 해당 단어 나타나는 횟수를 부여함
    - count 값 높을수록 중요한 단어로 인식
        - 문장에서 자주 사용할 수 밖에 없는 단어 (ex. is, will, a ...)를 중요하게 인식
        - ⇒ **TF-IDF 벡터화**
2. TF-IDF 기반 벡터화
    - 개별 문서에 자주 나타나는 단어에 가중치

- 모든 문서에 전반적으로 자주 나타나는 단어에 패널티 부여
- 텍스트 길고 문서 개수 많은 경우 주로 사용함

Term Frequency

The	Matrix	is	nothing	but	an	advertising	gimmick
40	5	50	12	20	45	3	2

Document Frequency

The	Matrix	is	nothing	but	an	advertising	gimmick
2000	190	2300	500	1200	3000	52	12

$$TFIDF_i = TF_i * \log\frac{N}{DF_i}$$

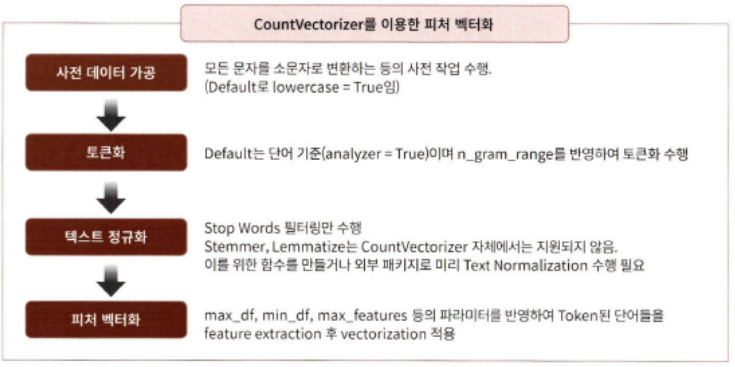
TF_i = 개별 문서에서의 단어 i 빈도
DF_i = 단어 i를 가지고 있는 문서 개수
N = 전체 문서 개수

사이킷런 카운트 및 TF-IDF 벡터화 구현

- TfidVectorizer 클래스
  - TF-IDF 벡터화 구현
- CountVectorizer 클래스
  - 카운트 기반 벡터화 구현
  - 텍스트 전처리 & 피처 벡터화
    - 객체 반환: fit( ), transform( )
  - 입력 파라미터

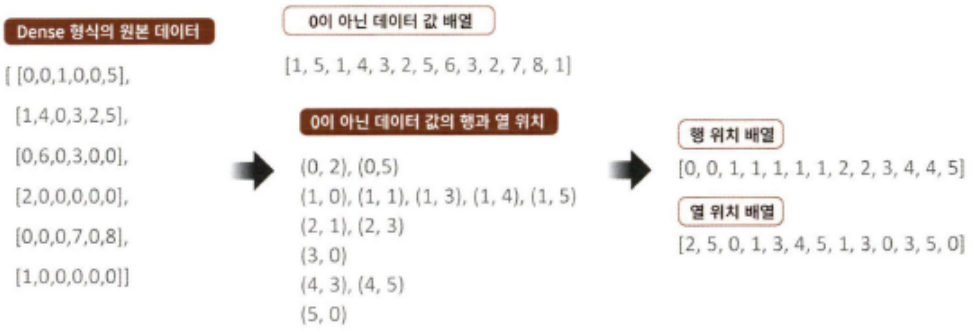
max_df	전체 문서에서 너무 높은 빈도수 갖는 단어 피처 제외
min_df	전체 문서에서 너무 낮은 빈도수 갖는 단어 피처 제외 - 중요하지 않거나 garbage 성 단어일 가능성 있음
max_features	추출 피처 개수 제한 - 가장 높은 빈도 갖는 단어 순 정렬 → 개수만큼 제한
stop_words	‘english’로 지정할 때 → 영어의 스톱워드로 지정된 단어 추출하지 않음
n_gram_range	n_gram 범위 설정 - ex. (1,2) 토큰화된 단어 1개씩, 순서대로 2개씩 묶음
analyzer	피처 추출 수행 단위 지정
token_pattern	정규 표현식 패턴
tokenizer	토큰화를 별도의 커스텀 함수로 이용할 때 적용함 - 어근 변환할 때 수행하는 별도의 함수를 적용

- 벡터화 절차



희소 행렬 - COO 형식

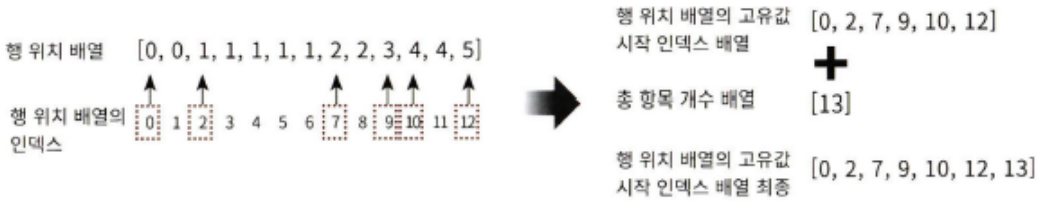
- 0 아닌 데이터만 별도 배열에 저장 → 데이터의 행,열 위치를 별도의 배열로 저장하는 방식



  - 행 위치 배열에서 같은 값 반복됨 (0: 2회, 1: 5회, 2: 2회, 4: 2회)
- 반복적인 위치 데이터 사용하는 문제 ⇒ **CSR 형식** 사용

희소 행렬 - CSR 형식

- 행 위치 배열에서 고유한 값의 시작 위치만 배열로 갖는 변환 방식
- 시작 위치 배열 + 데이터 총 항목 개수



- COO 형식에 비해 메모리 적고 연산 빠름

5. 감성 분석

감성 분석

- 문서의 주관적 감성, 의견, 기분 등 파악
- 텍스트의 단어, 문맥 → 감성 수치 계산
    - 긍정 감성 지수, 부정 감성 지수 합산 → 긍정 감성, 부정 감성 결정

지도 학습	학습데이터, 타깃 레이블 값 기반 감성 분석 학습 수행 → 다른 데이터의 감성 분석 예측
비지도 학습	감성 어휘 사전 (Lexicon) 이용

비지도 학습 기반 감성 분석

- 결정된 레이블 값 없음
- Lexicon 사용
    - 감성 지수
        - 긍정 감성, 부정 감성의 정도 의미하는 수치
        - 단어 위치, 주변 단어, 문맥 등 참고하여 결정됨
    - NLTK 패키지로 구현
        - 단점: 예측 성능 떨어짐
- NLTK 패키지 - WordNet
    - 시맨틱 분석 제공하는 어휘 사전
        - 시맨틱 (semantic): 문맥상 의미
        - Synset (Sets of cognitive synonyms)
            - 동의어 집합, 연어 관계
- 감성 사전 종류

SentiWordNet	감성 단어 전용 WordNet 구현 - Synset 개념 적용 1. Synset 별로 긍정 감성 지수, 부정 감성 지수, 객관성 지수 할당 2. 최종 감성 지수 = 긍정 감성 지수 + 부정 감성 지수 3. 긍정 감성, 부정 감성 결정
VADER	소셜 미디어 텍스트 감성 분석 제공 - 뛰어난 성능, 빠른 수행 시간 - 대용량 데이터에 사용됨
Pattern	예측 성능 뛰어남

SentiWordNet 이용한 감성 분석

- 영화 감상평 감성 분석
    1. 문서를 문장 단위로 분해
    2. 문자를 단어 단위로 분해(토큰화)하고 품사 태깅
    3. 품사 태깅된 단어 기반으로 synset 객체, senti_synset 객체 생성
    4. Senti_synset에서 긍정/부정 감성 지수 구하고 합산

       → 임계치 이상: 긍정감성, 임계치 미만: 부정감성