텍스트 분석

8.0 NLP vs 텍스트 마이닝(분석)

머신러닝이 발전 근간에 있음; 룰 기반 시스템(텍스트를 구성하는 언어적인 룰/업무의 룰을 따라 텍스트 분석) → 텍스트 데이터 기반으로 모델 학습 및 예측

NLP

- 머신이 인간의 언어하고 해석하는 데 중점
- 텍스트 분석을 향상하게 하는 기반 기술

텍스트 마이닝

- 비정형 텍스트에서 의미있는 정보를 추출하는 데 중점
- 모델 수립 by 머신러닝, 언어이해, 통계 → 정보 추출 → 분석 작업 (Business Inteligence, 예측 분석)
- 집중하고 있는 기술 영역
    - 텍스트 분류(Text Classification/Categorization): 문서가 특정 분류/카테고리에 속하는 것을 예측하는 기법. Supervised Learning 사용
    - 감성 분석(Sentiment Analysis): 텍스트에서 나타나는 감정/판단/믿음/의견/기분 등 주관적인 요소를 분석하는 기법이고 가장 활발하게 사용됨. Supervised, Unsupervised Learning 사용
    - 텍스트 요약(Summarization: 텍스트 내에서 중요한 주제/중심 사상을 추출하는 기법. Topic Modeling이 대표적
    - 텍스트 군집화(Clustering)와 유사도 측정: 비슷한 유형의 문서에 대해 clustering 수행/유사도 측정해서 비슷한 것 끼리 모으기. 텍스트 분류를 Unsupervised Learning으로 수행하는 방법..

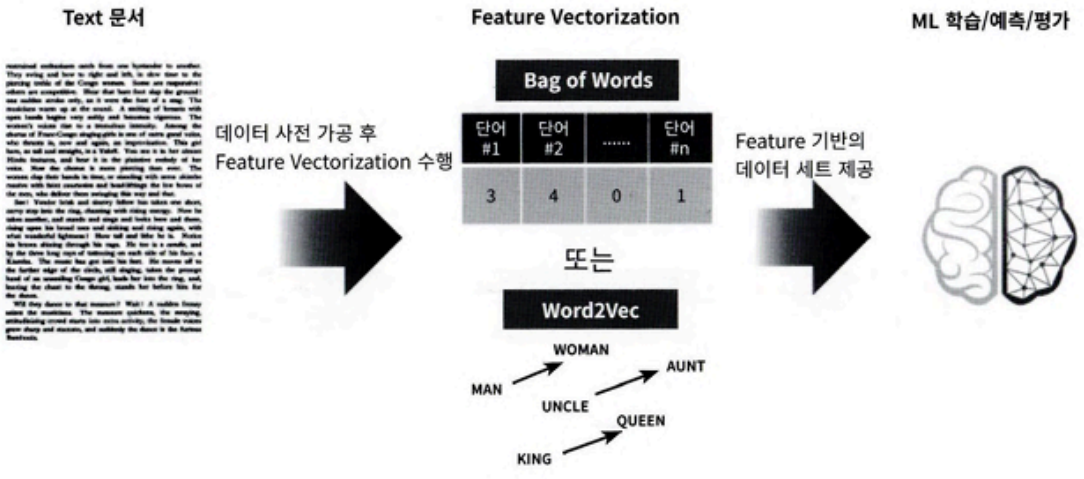
8.1 텍스트 분석 이해

- 텍스트 분석: 비정형 데이터인 텍스트를 분석하는 것
- ML은 숫자형 정형 데이터만 입력 받을 수 있음 ⇒ 텍스트를 벡터 형태의 피처로 변환하는 작업 필요

Feature Vectorization/Extraction

1. 텍스트를 word 기반의 다수의 피처로 추출
2. 이 피처에 단어 빈도수 같은 숫자값 부여

텍스트 분석 프로세스



〈 Text Analytics 수행 프로세스 〉

1. 텍스트 사전 준비작업(텍스트 전처리)
    a. 클렌징 작업: 텍스트를 피처로 만들기 전에 미리 클렌징, 대/소문자 변경, 특수문자 삭제 등
    b. 토큰화 작업:단어(Word) 등
    c. 텍스트 정규화: 의미 없는 단어(Stop word) 제거 작업, 어근 추출 (Stemming/Lemmatization) 등
2. 피처 벡터화/추출
    a. 사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 여기에 벡터 값을 할당.
    b. 대표적인 방법: BOW(Count 기반, TFHDF 기반 벡터화), Word2Vec
3. ML 모델 수립 및 학습/예측/평가: 피처 벡터화된 데이터 세트에 ML 모델을 적용해 학습/예측 및 평가를 수행

파이썬 기반의 NLP, 텍스트 분석 패키지

NLTK

- Python 대표 NLP 패키지
- 광범위한 기능제공
- 속도, 실무 활용성 면에서 아쉬움

Gensim

- 토픽 모델링 특화
- Word2Vect 기능 강함

SpaCy

- 최신 기술
- 빠른 처리 속도
- 실무에서 가장 활발히 사용

Scikit-Learn

- 머신러닝 특화
- NLP 기능 부적
- 벡터와, Classificaton 등 텍스트 기반 ML 적용 가능

8.2 텍스트 사전 준비 작업(텍스트 전처리) - 텍스트 정규화

텍스트 정규화: 텍스트를 ML 알고리즘, NLP 어플리케이션에 입력 데이터로 사용하기 위해 클렌징, 정제, 토큰화, 어근화 등 다양한 텍스트 데이터 사전 작업 수행하는 것

클렌징

- 텍스트 분석에 방해되는 불필요한 문자, 기호 미리 제거
- ex) HTML 태그, XNL 태그, 특정 기호

텍스트 토큰화

- 문장 토근화: 문서에서 문장을 분리
- 단어 토근화: 문장에서 단어를 토큰으로 분리

⇒ NLTK가 다양한 API 제공

문장 토큰화

- `.`, `\n` 등 문장의 마지막을 뜻하는 기호에 따라 분리
- 정규 표현식에 따른 문장 토큰화도 가능
- 각 문장이 가지는 시맨틱적인 의미가 중요한 요소로 사용될 때 사

단어 토큰화

- 문장을 단어로 토큰화
- 문장을 분리하는 구분자(공백, `,`, `.`, 개행문자 등)으로 단어 분리
- 정규 표현식을 이용해 다양한 유형으로 토큰화 수행 가능
- BOW 처럼 단어의 순서가 중요하지 않을 경우 문장 토큰화 안쓰고 단어 토큰화만 써도 충분

n-gram

- 문장을 단어별로 하나씩 토큰화 → 문맥적인 의미 무시될 수밖에 없음⇒ `n-gram` 으로 해결
- 연속된 단어 n개를 하나의 토큰화 단위로 분리
- n개 단어 크기 윈도우를 만들어 문장의 처음부터 오른쪽으로 움직이면서 수행

스톱 워드 제거

- stop word: 분석에 큰 의미 없는 단어
- 문장을 구성하는 필수 문법 요소지만 문맥적으로 큰 의미 없는 단어
- 빈번하게 등장하면 중요한 단어로 인지될 위험 있음 ⇒ 제거 필요
- 언어별로 스톱워드가 목록화되어있고, NLTK가 제공해줌

Stemming과 Lemmatization

공통점

- 목적: 원형 단어 찾기

Stemming

- 원형 단어로 변환 시 일반적인 방법 적용/더 단순화된 방법 적용 ⇒ 원래 단어에서 일부 철자가 훼손된 어근 단어 추출
- NLTK가 다양한 Stemmer 제공함

Lemmatization

- 더 정교함
- 의미론적인 기반에서 단어 원형 찾음
- 품사같은 문법적인 요소, 더 의미적인 부분을 감안 ⇒ 정확한 철자로 된 어근 단어 찾아줌
- 변환하는데 시간이 더 오래걸림
- NLTK가 WordNetLemmatizer 제공함

8.3 BOW (Bag of Words)

- 문서가 가지는 모든 단어(Words)를 문맥이나 순서를 무시하고 일괄적으로 단어에 빈도값을 부여해 피처값 추출하는 모델
- 예시

  문장 1 : 'My wife likes to watch baseball games and my daughter likes to watch baseball games too'
  문장 2 : 'My wife likes to play baseball

  1. 문장 1 과 문장 2에 있는 모든 단어에서 중복을 제거하고 그! 단어(feature 또는 term)를 칼럼 형태로 나열
  2. 각 단어에 고유의 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. 개별 문장에서 해당 단어가 나타나는 횟수(Occurrence)를 각 단어(단어 인덱스)에 기재함

     ex) base-ball은 문장 1, 2에서 총 2번 나타나며, daughter는 문장 1에서만 1 번 나타남

장점

- 단순하고 빠름
- 특정 문서의 주요 단어 특징을 효과적으로 표현 가능

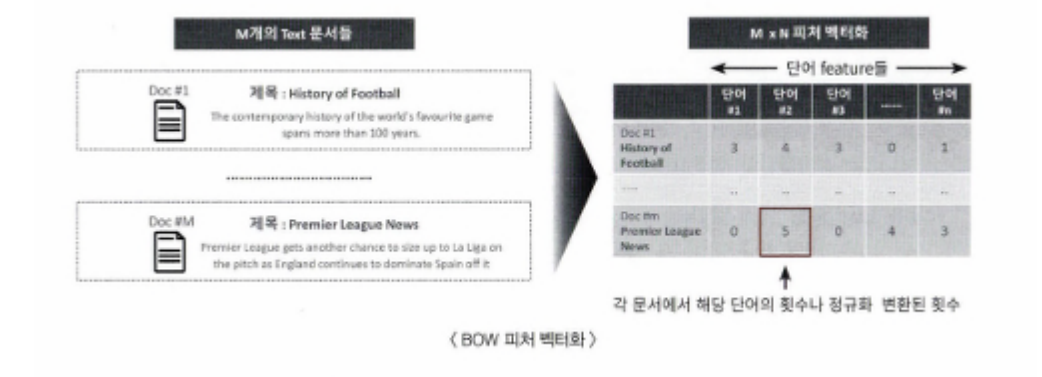
단점

- 문맥 의미(Semantic Context) 반영 부족
  - BOW는 단어의 순서를 고려X ⇒ 문장 내에서 단어의 문맥적인 의미가 무시됨
  - 보완하기 위해 n_gram 기법을 활용할 수 있지만, 제한적

    ⇒ 언어의 많은 부분을 차지하는 문맥적인 해석을 처리하지 못함
- 희소 행렬 문제(희소성, 희소 행렬)
  - BOW로 피처 벡터화를 수행하면 희소 행렬 형태의 데이터 세트가 만들어지기 쉬움
  - 많은 문서에서 단어를 추출하면 매우 많은 단어가 칼럼으로 만들어짐
  - 문서마다 서로 다른 단어로 구성됨 ⇒ 단어가 문서마다 나타나지 않는 경우가 훨씬 더 많음
  - 희소 행렬(Sparse Matrix): 대규모의 칼럼으로 구성된 행렬에서 대부분의 값이 0으로 채워자는 행렬. 일반적으로 ML 알고리즘의 수행 시간과 예측 성능을 떨어뜨리기 때문에 희소 행렬을 위한 특별한 기법이 있음
  - 밀집 행렬(Dense Matrix): 대부분의 값이 0이 아닌 의미 있는 값으로 채워져 있는 행렬

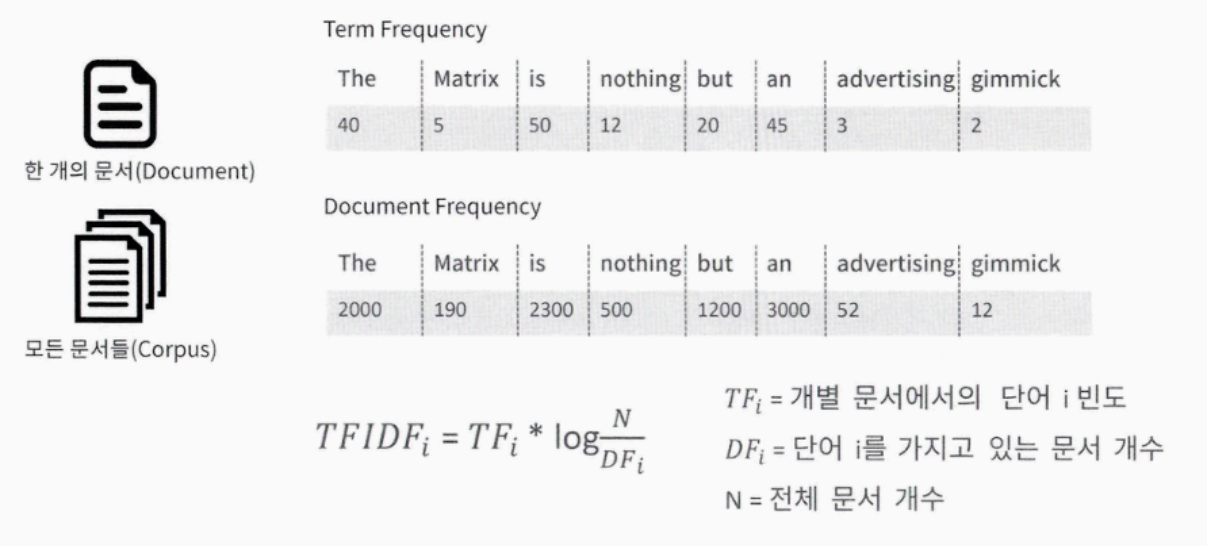
BOW 피처 벡터화

- 숫자형 피처를 데이터로 입력받아 동작 ⇒ 텍스트 같은 데이터는 ML에 바로 입력 불가해서 피처 벡터화 필요
- 피처 벡터화: 텍스트를 특정 의미를 가지는 숫자형 값인 벡터 값으로 변환하는 것



〈BOW 피처 벡터화〉

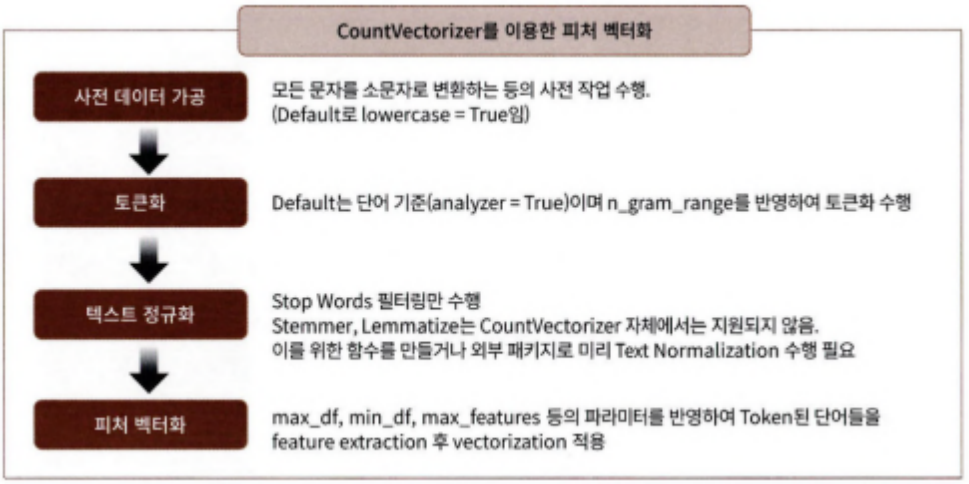
- 방식
  - 카운트 벡터화 (Count Vectorization)방식
    - 단어가 문서에 등장한 횟수(count)를 그대로 값으로 부여
    - count값이 높을수록 중요한 단어로 인식
    - 언어 특성상 문장에서 자주 사용될 수 밖에 없는 단어까지 높은 값 부여 ⇒ TF-IDF 벡터화 사용
  - TF-IDF 벡터화 방식
    - TF (Term Frequency): 문서 내 단어의 빈도
    - IDF (Inverse Document Frequency): 전체 문서 중 해당 단어가 얼마나 희귀한지
      자주 등장하지만 흔한 단어는 패널티, 특정 문서에만 자주 등장하는 단어는 가중치 부여
    - 텍스트가 길고 문서 개수 많을 때 사용



사이킷런의 Count 및 TF-IDF 벡터화 구현: CountVectorizer, TfidfVectorizer

- CountVectorizer 클래스
  - 카운트 기반의 벡터화를 구현한 클래스
  - 전처리 + 토큰화 + 스톱워드 제거 가능
- 주요 파라미터

파라미터 명	파라미터 설명
max_df	전체 문서에 걸쳐서 너무 높은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 너무 높은 빈도수를 가지는 단어는 스톱 워드와 비슷한 문법적인 특성으로 반복적인 단어일 가능성이 높기에 이를 제거하기 위해 사용됩니다. max_df = 100과 같이 정수 값을 가지면 전체 문서에 걸쳐 100개 이하로 나타나는 단어만 피처로 추출합니다. Max_df = 0.95와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐 빈도수 0~95%까지의 단어만 피처로 추출하고 나머지 상위 5%는 피처로 추출하지 않습니다.
min_df	전체 문서에 걸쳐서 너무 낮은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 수백~수천 개의 전체 문서에서 특정 단어가 min_df에 설정된 값보다 적은 빈도수를 가진다면 이 단어는 크게 중요하지 않거나 가비지(garbage)성 단어일 확률이 높습니다. min_df = 2와 같이 정수 값을 가지면 전체 문서에 걸쳐서 2번 이하로 나타나는 단어는 피처로 추출하지 않습니다. min_df = 0.02와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐서 하위 2% 이하의 빈도수를 가지는 단어는 피처로 추출하지 않습니다.
max_features	추출하는 피처의 개수를 제한하며 정수로 값을 지정합니다. 가령 max_features = 2000으로 지정할 경우 가장 높은 빈도를 가지는 단어 순으로 정렬해 2000개까지만 피처로 추출합니다.
stop_words	'english'로 지정하면 영어의 스톱 워드로 지정된 단어는 추출에서 제외합니다.
n_gram_range	Bag of Words 모델의 단어 순서를 어느 정도 보강하기 위한 n_gram 범위를 설정합니다. 튜플 형태로 (범위 최솟값, 범위 최댓값)을 지정합니다. 예를 들어 (1, 1)로 지정하면 토큰화된 단어를 1개씩 피처로 추출합니다. (1, 2)로 지정하면 토큰화된 단어를 1개씩(minimum 1), 그리고 순서대로 2개씩(maximum 2) 묶어서 피처로 추출합니다.
analyzer	피처 추출을 수행한 단위를 지정합니다. 당연히 디폴트는 'word'입니다. Word가 아니라 character의 특정 범위를 피처로 만드는 특정한 경우 등을 적용할 때 사용됩니다.
token_pattern	토큰화를 수행하는 정규 표현식 패턴을 지정합니다. 디폴트 값은 '\b\w\w+\b로, 공백 또는 개행 문자 등으로 구분된 단어 분리자(\b) 사이의 2문자(문자 또는 숫자, 즉 영숫자) 이상의 단어(word)를 토큰으로 분리합니다. analyzer= 'word'로 설정했을 때만 변경 가능하나 디폴트 값을 변경할 경우는 거의 발생하지 않습니다.
tokenizer	토큰화를 별도의 커스텀 함수로 이용시 적용합니다. 일반적으로 CountTokenizer 클래스에서 어근 변환 시 이를 수행하는 별도의 함수를 tokenizer 파라미터에 적용하면 됩니다.



BOW 벡터화를 위한 희소 행렬

- 대부분의 값이 0으로 채워진 행렬
- 문서가 가진 단어 수는 적고 전체 단어 수(N)는 많아서 발생
- BOW 형태를 가진 언어 모델의 피처 벡터화는 대부분 희소 행렬
- 형태
  - Dense Matrix: 거의 모든 값이 존재 (0이 적음)
  - Sparse Matrix: 대부분의 값이 0 (BOW 피처 벡터화의 전형적 결과)
- 적은 메모리 공간 사용할 수 있도록 변환 필요 ⇒ CSR, COO 형식

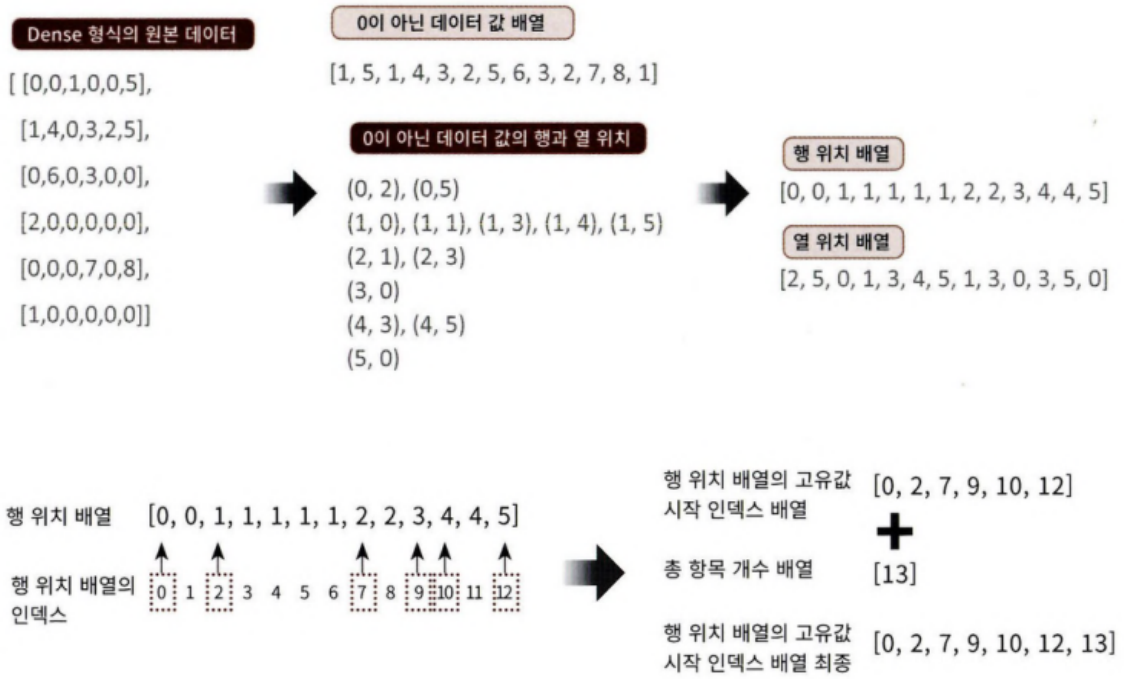
COO 형식

- 0이 아닌 값만 저장하고, 그 위치(row, col)를 별도의 배열 저장
- ex)
  - 값 배열: [3, 1, 2]
  - 행 위치: [0, 0, 1]
  - 열 위치: [0, 2, 1]
- Scipy 이용

CSR 형식

- COO에서 행 위치 반복 문제를 해결

- 행의 시작 위치 인덱스만 따로 저장
- 더 빠르고 메모리 효율적 → 기본 형식



8.5 감성 분석

감성 분석 소개

- 문서의 감정, 의견, 기분, 태도 등의 주관적인 요소를 분석하는 기법
- 예: 영화 리뷰, 상품 후기, SNS 게시글 등에서 긍정/부정 감성을 분류
- 문서 내 텍스트가 나타내는여러 가지 주관적인 단어, 문맥을 기반으로 감성 수치 계산
- 활용 분야: 소셜 미디어, 여론 분석, 온라인 리뷰, 피드백 등

방식

- Supervised Learning
  - 감성 레이블(긍정/부정)이 명시된 학습 데이터를 바탕으로 감성 예측 모델 학습
  - 일반적인 텍스트 기반의 분류와 거의 동일

- Unsupervised Learning
  - Lexicon(감성 어휘 사전) 을 기반으로 단어의 긍정/부정 감성을 점수화하여 분석
  - 일반적인 텍스트 기반의 분류와 거의 동일
  - 대표적인 사전: SentiWordNet, VADER, Pattern

실습은 .ipynb 파일에 있습니다