파머완 8장(1)(Part 8.1~8.3, 8.5): 텍스트 분석

상태	진행 중
담당자	시현 이
마감일	@11/24/2025
작업 유형	개념정리 및 필사
설명	파이썬 머신러닝 완벽 가이드_개정2판_제8장 개념 정리+ 필사 링크
업데이트 시간	@November 22, 2025 12:56 AM

NLP이냐 텍스트 분석이냐?

NLP(National Language Processing)

- 머신이 인간의 언어를 이해하고 해석하는 것에 중점.
- 텍스트 분석을 향상하게 하는 기반 기술

텍스트 마이닝(Text Mining)

- 비정형 텍스트에서 의미 있는 정보를 추출하는 것에 중점
- 머신러닝, 언어 이해, 통계 등을 활용해 모델을 수립하고 정보를 추출해 비즈니스 인텔리전스나 예측 분석 등의 분석 작업을 주로 수행
    - 텍스트 분류(Text Classfication)(=Text Categorization): 문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법을 통칭. 지도학습 사용
        - ex) 특정 신문 기사 분류. 스팸메일 검출 등
    - 감성 분석(Sentiment Analysis): 텍스트에서 나타나는 감정/판단/믿음/의견/기분 등의 주관적인 요소를 분석하는 기법. Text Analytics에서 가장 활발하게 사용되고 있는 분야. 지도학습 + 비지도 학습

        - ex) 제품에 대한 리뷰, sns 감정 분석, 여론조사 분석
    - 텍스트 요약(Summerization): 텍스트 내에서 중요한 주제나 중심 사상을 추출하는 기법. 대표적으로는 토픽 모델링(Topic Modeling)
    - 텍스트 군집화와 유사도 측정: 비슷한 유형의 문서에 대해 군집화를 수행하는 기법. 텍스트 분류를 비지도학습으로 수행하는 방법의 일환으로도 사용
        - 유사도 특정: 문서들 간의 유사도 측정

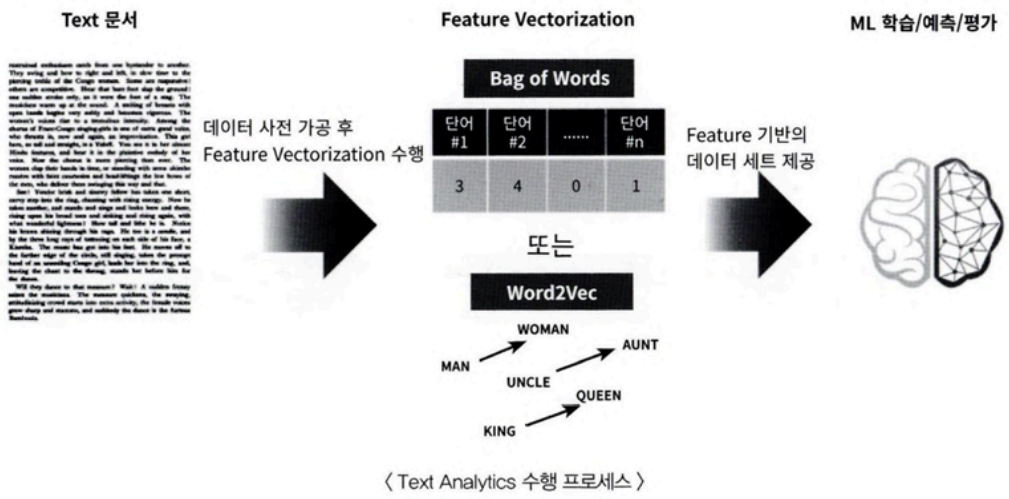
01. 텍스트 분석 이해

텍스트 분석: 비정형 데이터인 텍스트 분석

- 피처 벡터화(Feature Vectorization) or 피처 추출(Feature Extraction)
    - 텍스트를 word기반의 다수의 피처로 추출→ 피처에 단어 빈도수와 같은 숫자 값 부여 → 텍스트가 벡터값으로 표현됨
    - 대표적인 변환 방법: BOW(Bag of Words) 와 Word2Vec

텍스트 분석 수행 프로세스

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



〈 Text Analytics 수행 프로세스 〉

파이썬 기반의 NLP, 텍스트 분석 패키지

- NLTK(Natural Language Toolkit for Python):
  - 대표적인 NLP 패키지
  - 방대한 데이터 세트와 서브 모듈 가짐. NLP의 거의 모든 영역 커버
  - 수행 속도 측면에서 아쉬운 부분이 있어 실제 대량의 데이터 기반에는 제대로 활용X
- Gensim:
  - 토픽 모델링 구현 기능 제공해 토픽 모델링 분야에서 가장 두각을 나타내는 패키지
  - Word2Vec 구현 등의 다양한 신기능 제공
  - SpaCy와 함계 가장 많이 사용되는 NLP 패키지
- SpaCy:
  - 뛰어난 수행 성능을 보이는 NLP 패키지

*사이킷런은 머신러닝 위주의 라이브러리→ NLP를 위한 다양한 라이브러리('어근 처리' 등)는 가지고 있지X

But, 텍스트를 일정 수준으로 가공하고 텍스트 피처화 기능 제공해 텍스트 분석 기능 수행 가능.

02. 텍스트 사전 준비 작업(텍스트 전처리)-텍스트 정규화

텍스트 자체를 바로 피처화X → 사전 가공 작업 필요

[텍스트 정규화 작업]

- 클렌징(Cleansing)
  - 텍스트에서 분석에 오히려 방해가 되는 불필요한 문자, 기호 등을 사전에 제거하는 작업
  - ex) HTML, XML 태그나 특정 기호 등을 사전에 제거
- 토큰화(Tokenization)
  - NLTK는 다양한 API 제공
  - 문서에서 문장을 분리하는 **문장 토큰화**
    - 문장의 마침표(.), 개행 문자(\n) 등 문장의 마지막을 뜻하는 기호에 따라 분리
    - 정규 표현식에 따른 문장 토큰화도 가능
  - 문장을 단어로 토큰화하는 **단어 토큰화**
    - 공백, 콤마(,), 마침표(.), 개행 문자 등으로 단어를 분리
    - 정규 표현식을 이용해 다양한 유형으로 토큰화 수행
    - BOW와 같이 단어의 순어가 중요하지 않은 경우 문장의 토큰화를 사용하지 않고 단어 토큰화만 사용해도 충분
  - 문장을 단어별로 하나씩 토큰화→ 문맥적 의미 무시됨 ⇒ 해결책: n-gram
    - n개의 단어를 하나의 토큰화 단위로 분리해 내는 것
    - ex) ‘Agent Smith knoxks the door” ⇒ 2-gram

    → (Agent, Smith), (Smith, knocks), (knocks, the), (the, door)
- 필터링/스톱 워드 제거/철자 수정
  - 스톱 워트(Stop word) : 분석에 큰 의미가 없는 단어 지칭(필수 문법 요소 but 의미X)
- Stemming과 Lemmatization
  - 문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 것

- Stemming
  - 원형 단어로 변환 시 일반적인 방법을 적용하거나 더 단순화된 방법을 적용해 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있음
  - NLTK에서 Porter, Lancaster, Snowball Stemmer 제공
- Lemmatization
  - 품사와 같은 문법적인 요소와 더 의미적인 부분을 감안해 정확한 철자로 된 어근 단어를 찾아줌
  - NLTK에서 WordNetLemmatizer 제
- Lemmatization이 Stemming 보다 정교하며 의미론적인 기반에서 단어의 원형을 찾음. But, 변환에 더 오랜 시간을 필요로 함

03.Bag of Words-BOW

BOW

: 문서가 가지는 모든 단어를 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피처 값을 추출하는 모델

〈Bag of Words 모델〉

- 2개의 문장을 가정하고 BOW 기반으로 추출
  - 문장1: 'My wife likes to watch baseball games and my daughter likes to watch baseball games too.'
  - 문장2: 'My wife likes to play baseball'

1. 문장1과 문장 2에 있는 모든 단어에서 중복을 제거하고 각 단어를 칼럼 형태로 나열 → 각 단어에 고유의 인덱스를 부여

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

2. 개별 문장에서 해당 단어가 나타나는 횟수를 각 단어에 기재.

	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

- BOW 모델의 장점은 쉽고 빠른 구축
- 단점
  - 문맥 의미(Semantic Context) 반영 부족: BOW는 단어의 순서 고려X. 문장 내에서 단어의 문맥적인 의미 무시됨. → n_gram 활용 가능 but, 제한적. 문맥적인 해석 처리 못함
  - 희소 행렬 문제(희소성, 희소 행렬): BOW로 피처 벡터화를 수행하면 희소 행렬 형태의 데이터 세트가 만들어지기 쉬움.
    - 많은 문서에서의 수만~수십만 개의 단어가 있는데, 하나의 문서에 있는 단어는 이 중 극히 일부분. 대부분의 데이터는 0값 ⇒ 희소행렬
    - 희소행렬과 반대로 대부분의 값이 0이 아닌 의미 있는 값으로 채워진 행렬 → 밀집 행렬
    - 희소행렬은 일반적으로 ML 알고리즘의 수행시간과 예측 성능을 떨어뜨리기에 희소행렬을 위한 특별한 기법 마련됨

BOW 피처 벡터화

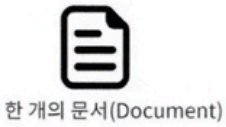
피처 벡터화는 기존 텍스트 데이터를 또 다른 형태의 피처의 조합으로 변경하기 때문에 넓은 범위의 피처 추출에 포함(Text Analysis에서는 피처 벡터화와 피처 추출을 같은 의미로 사용됨)

BOW 모델에서 피처 벡터화를 수행→ 모든 문서에서 모든 단어를 칼럼 형태로 나열→각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경하는 것

〈 BOW 피처 벡터화 〉

[BOW의 피처 벡터화]

- 카운트 기반의 벡터화
  - 단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수, 즉 Count를 부여하는 경우
  - 카운트 값이 높을수록 중요한 단어로 인식
  - 그러나 카운트만 부여할 경우, 그 문서의 특징을 나타내기보다는 언어의 특성상 문장에서 자주 사용될 수 밖에 없는 단어까지 높은 값을 부여
- TF-IDF(Term Frequency - inverse Document Frequency) 기반의 벡터화
  - 카운트 벡터화 보완
  - 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 패널티를 주는 방식으로 값을 부여.
  - 단순히 단어의 반복 횟수에 따라 주요도 평가하면X. → 모든 문서에 반복적으로 자주 발생하는 단어에 대해서는 페널티를 부여하는 방식으로 단어에 대한 가중치의 균형을 맞춤



한 개의 문서(Document)

모든 문서들(Corpus)

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

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

- CountVectorizer 클래스는 카운트 기반의 벡터화를 구현한 클래스
  - 피처 벡터화 수행 + 소문자 일괄 변환, 토큰화, 스톱 워드 필터링 등 텍스트 전처리도 함께 수행

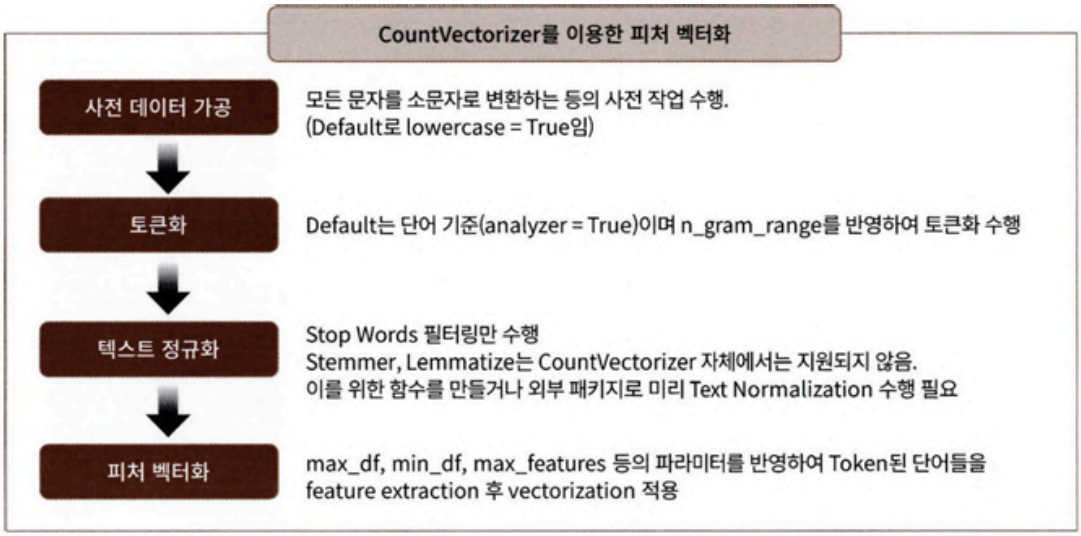
파라미터 명	파라미터 설명
max_df	전체 문서에 걸쳐서 너무 높은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 너무 높은 빈도수를 가지는 단어는 스톱 워드와 비슷한 문법적인 특성으로 반복적인 단어일 가능성이 높기에 이를 제거하기 위해 사용됩니다. max_df = 100과 같이 정수 값을 가지면 전체 문서에 걸쳐 100개 이하로 나타나는 단어만 피처로 추출합니다. Max_df = 0.95와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐 빈도수 0~95%까지의 단어만 피처로 추출하고 나머지 상위 5%는 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.

CountVectorizer 사용법

1. 영어의 경우 모든 문자 소문자로 변경(전처리 작업)
2. 디폴트로 단어 기준으로 n_gram_range를 반영해 각 단어를 토큰화
3. 텍스트 정규화를 수행.
    a. 단, stop_words = 'english'와 같이 stop_words 파라미터가 주어진 경우 스톱 워트 필터링만 가능

b. Steeming과 Lemmatization같은 어근 변환은 CountVectorizer에서 직접 지원X
→ tokenizer 파라미터에 커스텀 어근 변환 함수 사용해 어근 변환 수행 가능.

4. max_df, min_df, max_features 등 파라미터 이용해 토큰화된 단어를 피처로 추출하고 단어의 빈도수 벡터값을 적용



*TF-IDF 벡터화는 TfidVectorizer 클래스를 이용

BOW 벡터화를 위한 희소행렬

사이킷런의 CountVectorizer/TfidVectorizer를 이용해 텍스트를 피처 단위로 벡터화해 변환하고 CSR형태의 희소 행렬을 반환

- BOW 형태를 가진 언어 모델의 피처 벡터화는 대부분 희소 행렬

	단어 1	단어 2	단어 3	....	단어 1000	....	단어 2000	....	단어 10000	....	단어 20000	....	단어 100000
문서1	1	2	2	0	0	0	0	0	0	0	0	0	0
문서2	0	0	1	0	0	1	0	0	1	0	0	0	1
문서 ...	....	....	....	....	....	....	....	....	....	....	....	....	....
문서 10000	0	1	3	0	0	0	0	0	0	0	0	0	0

BOW의 Vectorization 모델은 너무 많은 0값이 메모리 공간에 할당되어 많은 메모리 공간이 필요하며
연산 시에도 데이터 액세스를 위한 많은 시간이 소모됩니다.

- 불필요한 0 값들이 메모리 공간에 할당돼 메모리 공간이 많이 필요 & 데이터 액세스 위한 시간이 많이 소모
- COO 형식과 CSR 형식으로 변환하는 방법
  - 일반적으로 큰 희소 행렬 저장하고 계산 수행하는 능력이 CSR 형식이 더 뛰어나기에 CSR을 많이 사용

희소 행렬-COO 형식

COO(Coordinate: 좌표) 형식은 0이 아닌 데이터만 별도의 데이터 배열에 저장하고, 그 데이터가 가리키는 행과 열의 위치를 별도의 배열로 저장하는 방식.

ex) [[3,0,1], [0,2,0]]과 같은 2차원 데이터

→ 0이 아닌 데이터: [3,1,2] → 위치: (0,0), (0,2), (1,1)

→ 로우와 칼럼을 별도의 배열로 저장하면 로우는 [0,0,1], 칼럼은 [0,2,1]

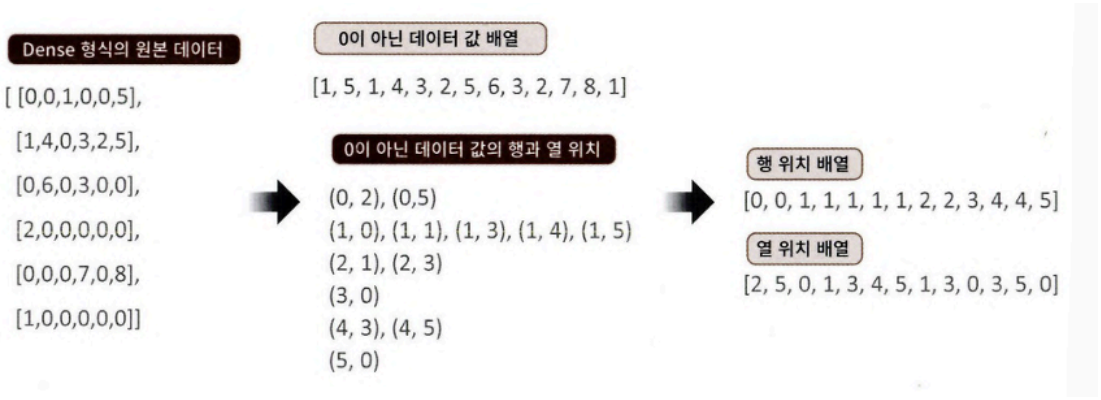
희소 행렬 - CSR 형식

CSR(Compressed Sparse Row) 형식은 COO 형식이 행과 열의 위치를 나타내기 위해서 반복적인 위치 데이터를 사용해야하는 문제점 해결한 방식

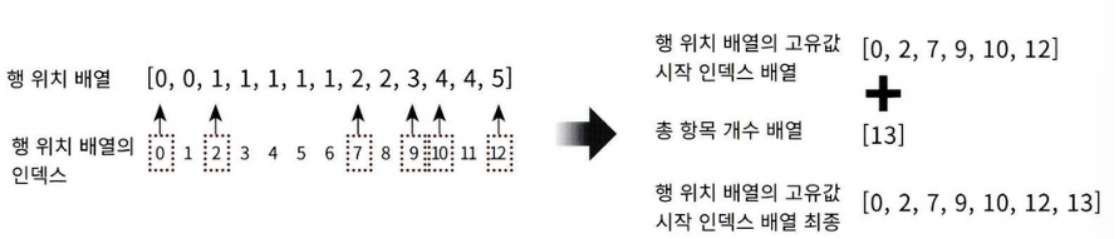
ex) COO 변환 형식의 문제점

[[0,0,1,0,0,5], [1,4,0,3,2,5], [0,6,0,3,0,0], [2,0,0,0,0,0], [0,0,0,7,0,8], [1,0,0,0,0,0]] 의 배열이 있다고 할 때

→변환→ 데이터 배열은 [1,5,1,4,3,2,5,6,3,2,7,8,1] / 행 위치 배열은 [0,0,1,1,1,1,1,2,2,3,4,4,5] / 열 위치 배열은 [2,5,0,1,3,4,5,1,3,0,3,5,0]



- CSR는 행 위치 배열 내에 있는 고유한 값의 시작 위치만 다시 별도의 위치 배열로 가지는 변환 방식
  - [0,0,1,1,1,1,1,2,2,3,4,4,5] →CSR 변환→ [0,2,7,9,10,12]
  - 맨 마지막에는 데이터의 총 항목 개수를 배열에 추가
  - 최종적으로 [0,2,7,9,10,12,13]



05.감성 분석

감성분석 소개

감성 분석(Sentiment Analysis)은 문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와 문맥을 기반으로 감성 수치를 계산하는 방법을 이용

- 감성 지수

    - 긍정 감성 지수
    - 부정 감성 지수
- 감정 분석의 지도학습 방식
    - 학습/타깃 데이터 값 기반으로 감성분석 후, 다른 데이터의 감성 분석을 예측
- 감성 분석의 비지도학습 방식
    - ‘Lexicon’이라는 일종의 감성 어휘 사전을 이용. Lexicon은 감성 분석을 위한 용어와 문맥에 대한 다양한 정보 가지고 있으며, 이를 이용해 문서의 긍/부정적 감성 여부를 판단

비지도학습 기반 감성 분석 소개

많은 감성 분석용 데이터는 결정된 레이블 값을 가지고 있지 X

- Lexicon 기반으로 함
    - 일반적으로 어휘집을 의미. 여기서는 주로 감성만을 분석하기 위해 지원하는 감성 어휘 사전. →감성지수(Polarity score)를 가짐
    - 감성 지수는 단어의 위치나 주변 단어, 문맥, POS(Part of Speech) 등을 참고해 결정됨
        - 감성 사전을 구현한 대표격: NLTK 패키지
- NLTK 패키지
    - WordNet (NLTK에서 제공하는 모듈)
        - 시맨틱(=문맥상 의미) 분석을 제공하는 어휘 사전
        - 같은 어휘라도 다르게 사용되는 어휘의 시맨틱 정보를 제공하며, 이를 위해 각각의 품사로 구성된 개별 단어를 Synset(Sets of congnitive synonyms)이라는 개념을 이용해 표현함
    - 감성에 대한 훌륭한 사전 역할 but, 예측 성능은 좋지 않음
    - NLTK를 포함한 대표적인 감성 사전
        - **SentiWordNet**: 감성 단어 전용의 WordNet을 구현. WordNet의 Synset 개념을 감성 분석에 적용
            - Synset별 감성 정수: 긍정/부정/객관성 지수

    - 긍정+부정 감성 지수 합산 = 최종 감성 지수 ⇒ 이에 기반해 감성이 긍/부인지 결정
- **VADER**: SNS 텍스트에 대한 감성 분석 제공 패키지. 뛰어난 감성 분석 결과 제공. 비교적 빠른 수행 시간 보장(대용량 텍스트 데이터에 사용됨)
- **Pattern**: 예측 성능 측면에서 가장 주목 받는 패키지. 파이썬 2.X 버전에서만 동작.

지원 파일

https://colab.research.google.com/drive/1avkmCG0Lm2eIIdREd5jrFkrGiig1A6Fn?usp=sharing