8

08 _ 텍스트 분석

01. 텍스트 분석 이해

- **텍스트 분석**은 비정형 데이터인 텍스트를 분석하는 것
- 머신러닝 알고리즘은 숫자형의 피처기반 데이터만 입력받을 수 있기 때문에 **'비정형 텍스트 데이터를 어떻게 피처 형태로 추출하고 추출된 피처에 의미있는 값을 부여하는가'** 하는 것이 매우 중요한 요소이다.
- 텍스트를 word 기반의 다수의 피처로 추출하고 이 피처에 단어 빈도수와 같은 숫자 값을 부여하면 텍스트는 단어의 조합인 벡터값으로 표현될 수 있는데, 이렇게 텍스트를 변환하는 것을 **피처 벡터화** 또는 **피처 추출**이라고 한다.

💡 **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 통칭한다.
2. 피처 벡터화/추출: 사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 여기에 벡터 값을 할당한다. 대표적 인 방법은 BOW와 Word2Vec이 있으며, BOW는 대표적으로 Count 기반과 TFHDF 기반 벡터화가 있다.
3. ML 모델 수립 및 학습/예측/평가: 피처 벡터화된 데이터 세트에 ML 모델을 적용해 학습/예측 및 평가를 수행한다.

파이썬 기반의 NLP, 텍스트 분석 패키지

- NLTK
- Genism
- SPaCy
-

02. 텍스트 사전 준비 작업 - 텍스트 정규화

클렌징

- 불필요한 문자, 기호 등을 사전에 제거 (ex. HTML, XML 태그나 특정기호)

텍스트 토큰화

문장 토큰화

- 마침표(.), 개행문자(\n) 등의 문장의 마지막을 뜻하는 기호에 따라 분리
- NLTK 에서는 sent_tokenize()를 일반적으로 많이 사용
- sent_tokenize() 가 반환하는 것은 각각의 문장으로 구성된 list 객체

단어 토큰화

- 문장을 단어로 토큰화
- NLTK 에서 word_tokenize()를 이용
- sent_tokenize와 word_tokenize의 조합으로 단어 단위로 토큰화 가능
- 문장의 개수만큼의 개별 리스트를 내포하는 리스트를 반환함 (list in list)
- 내포된 개별 리스트 객체는 각각 문장별로 토큰화된 단어를 요소로 가짐

스톱 워드 제거

- 스톱워드(stop word) : 분석에 큰 의미가 없는 단어 (ex. is , the, a, will..)
- NLTK 에는 다양한 스톱 워드가 목록화 되어있음
- 영어의 경우 스톱워드 개수가 179개

Stemming과 Lemmatization

- 문법적, 의미적으로 변화하는 단어의 원형을 찾는 작업
- Lemmatization이 Stemming 보다 정교하여 의미론적인 기반에서 단어의 원형을 찾음
- 따라서 Lemmatization이 변환에 더 오랜시간을 필요로 함
- NLTK에서의 대표적인 Stemmer 클래스 : Porter, Lancaster, Snowball stemmer
- NLTK에서의 대표적인 Lemmiatization 클래스 : WordNetLemmatizer

💡 <LancasterStemmer 를 이용한 Stemming>

- Stemmer 객체를 생성한뒤, stem('단어') 메서드를 통해 원하는 단어 stemming
- amuse와 비교형, 최상급형으로 변형된 단어의 정확한 원형을 찾지 못하고 철자가 다른 어근단어로 인식하는 경우 발생

<WordNetLemmatizer를 이용한 Lemmatization>

- Lemmatization 객체를 생성한 뒤 **lemmatize() 메서드의 인자로 품사를 입력**해주어야함
- Stemmer 보다 정확하게 원형 단어 추출

03. Bag of Words - BOW

- Bag of words : 문서가 가지는 모든 단어의 문맥이나 순서를 무시하고 단어에 대해 빈도 값을 부여해 피처 값을 추출하는 모델
- 장점 : 쉽고 빠른 구축
- 단점:
  1. 문맥 의미 반영 부족 : 단어의 문맥, 순서를 반영하지 않기 때문. n_gram 기법을 활용할 수 있지만 제한적임

2. 희소 행렬 문제 : 매우 많은 문서에 단어의 개수는 수만~수십만개 인 것에 비해 하나의 문서에 있는 단어는 이 중 극히 일부분임. 따라서 대규모의 칼럼으로 구성된 행렬에서 대부분의 값이 0으로 채워져 ML 알고리즘의 수행시간, 예측성능을 떨어트림

BOW 피처 벡터화

- BOW 피처 벡터화 모델은 대부분 희소 행렬
- 불필요한 0값이 많이 할당되어 많은 메모리 공간 필요, 연산 수행 시간 오래걸림
- 희소행렬을 물리적으로 적은 메모리 공간을 차지할 수 있도록 변환해주는 방법이 **COO, CSR**

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

💡 countvectorizer 은 카운트 기반의 피처 벡터화 방법, 텍스트 전처리 수행 후 fit(), transform()을 통해 피처 벡터화된 객체 반환

1. 사전 데이터 가공 : 모든 문자를 소문자로 변환하는 등의 사전 작업 수행
2. 토큰화 : Default 는 단어 기준, n_gram_range 를 사용하여 토큰화 수행
3. 텍스트 정규화 : Stop words 필터링만 수행, Stemmer, lemmatize를 수행하기 위해서는 tokenizer 파라미터에 해당 함수 적용
4. 피처 벡터화 : max_df, min_df, max_features 등의 파라미터를 반영하여 token 된 단어들을 피처 추출한 후 벡터화 진행

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.

💡 TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer

: 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 패털티를 주는 방식으로 값을 부여한다.

- 문서마다 텍스트가 길고 문서의 개수가 많을 경우 카운트 방식보다 TF-IDF 방식을 사용하는 게 좋다.

BOW 벡터화를 위한 희소 행렬

- BOW 피처 벡터화 모델은 대부분 희소 행렬
- 불필요한 0값이 많이 할당되어 많은 메모리 공간 필요, 연산 수행 시간 오래걸림
- 희소행렬을 물리적으로 적은 메모리 공간을 차지할 수 있도록 변환해주는 방법이 **COO, CSR**

희소행렬 - COO 형식

- 0이 아닌 데이터만 별도의 배열에 저장하고, 그 데이터가 가르키는 행, 열 위치를 별도의 배열로 저장하는 방식
- 희소행렬 변환은 scipy의 coo_matrix 클래스를 이용
-

희소행렬-CSR 형식

- COO 형식이 반복적인 행,열 위치 데이터를 사용해야하는 문제점 해결
- 행 위치 배열 대신 **행 위치 배열의 고유값 시작 인덱스 배열 + 총 항목 개수 배열로 변환** → COO 보다 메모리가 적게들고 빠른 연산 가능
- 희소행렬 변환은 scipy 의 csr_matrix 클래스를 이용
- 사이킷런의 CountVectorizer, TfidfVectorizer 클래스로 변환된 피처 벡터화 행렬은 모두 CSR 형태의 희소 행렬

05. 감정 분석

감정 분석 소개

- 감성분석은 텍스트가 나타내는 주관적인 단어와 문맥을 기반으로 긍정, 부정 감성(sentiment) 수치를 계산하는 방법을 이용
- 지도학습과 비지도학습 방식이 있음
  - 지도학습은 텍스트 기반의 이진분류와 거의 동일
  - 비지도학습은 'Lexicon' 감성어휘사전을 이용하여 긍정적, 부정적 감성여부 판단

비지도 학습 기반 감성 분석 소개

- 감성 어휘 사전 'Lexicon' 기반
- 감성 사전은 긍정, 부정 감성의 정도를 나타내는 감성 지수를 가지고 있음
- NLP의 WordNet 은 **시맨틱 (문맥상 의미) 분석**을 제공하는 어휘 사전
- WordNet 은 각각의 품사로 구성된 개별 단어를 *Synset* 이라는 단어의 문맥, 시맨틱 정보를 제공하는 개념을 이용해 표현
- WordNet 예측 성능이 좋지 못하다는 단점으로 인해 실무에서는 VADER, Pattern 등 다른 감성사전을 사용