8. 텍스트 분석

01. 텍스트 분석 이해

피처 벡터화(Feature Vectorization): 텍스트를 word(또는 word의 일부분) 기반의 다수의 피처로 추출하고 이 피처에 단어 빈도수와 같은 숫자 값을 부여하면 텍스트는 단어의 조합인 벡터값으로 표현될 수 있는데, 이렇게 텍스트를 변환하는 것. 피처 추출(Feature Extraction)이라고도 함.

텍스트를 피처 벡터화해서 변환하는 방법: BOW(Bag of Words)와 Word2Vec 방법

<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 통칭합니다.
2. 피처 벡터화/추출: 사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 여기에 벡터 값을 할당합니다. 대표적인 방법은 BOW와 Word2Vec이 있으며, BOW는 대표적으로 Count 기반과 TFHDF 기반 벡터화가 있습니다.
3. ML 모델 수립 및 학습/예측/평가: 피처 벡터화된 데이터 세트에 ML 모델을 적용해 학습/예측 및 평가를 수행합니다.

<파이썬 기반의 NLP, 텍스트 분석 패키지>

NLTK(Natural Language Toolkit for Python), Gensim, SpaCy

02. 텍스트 사전 준비 작업(텍스트 전처리) - 텍스트 정규화

텍스트 자체를 바로 피처로 만들 수는 없음. 사전에 텍스트를 가공하는 준비 작업이 필요함.

텍스트 정규화: 텍스트를 머신러닝 알고리즘이나 NLP 애플리케이션에 입력 데이터로 사용하기 위해 클렌징, 정제, 토큰화, 어근화 등의 다양한 텍스트 데이터의 사전 작업을 수행하는 것

- 클렌징 (Cleansing): 텍스트에서 분석에 오히려 방해가 되는 불필요한 문자, 기호 등을 사전에 제거하는 작업 예를 들어 HTML, XML 태그나 특정 기호 등을 사전에 제거
- 토큰화(Tokenization): 토큰화의 유형은 문서에서 문장을 분리하는 문장 토큰화와 문장에서 단어를 토큰으로 분리하는 단어 토큰화로 나눌 수 있음
- 필터링/스톱워드제거/철자수정
- Stemming
- Lemmatization

<문장 토큰화>

문장 토큰화(sentence tokenization)는 문장의 마침표(.), 개행문자(\n) 등 문장의 마지막을 뜻하는 기호에 따라 분리하는 것이 일반적
정규 표현식에 따른 문장 토큰화도 가능

<단어 토큰화>
단어 토큰화(Word Tokenization)는 문장을 단어로 토큰화하는 것.
기본적으로 공백, 콤마(,), 마침표(.), 개행문자 등으로 단어를 분리하지만, 정규 표현식을 이용해 다양한 유형으로 토큰화를 수행할 수 있음
마침표(.)나 개행문자와 같이 문장을 분리하는 구분자를 이용해 단어를 토큰화할 수 있으므로 Bag of Word와 같이 단어의 순서가 중요하지 않은 경우 문장 토큰화를 사용하지 않고 단어 토큰화만 사용해 도 충분

모든 단어를 토큰화-> 문서를 먼저 문장으로 나누고, 개별 문장을 다시 단어로 토큰화
문장을 단어별로 하나씩 토큰화 할 경우 문맥적인 의미는 무시됨
-> 해결방안 n-gram: 연속된 n개의 단어를 하나의 토큰화 단위로 분리해 내는 것
ex) 예를 들어 “Agent Smith knocks the door”를 2—gram(bigram)으로 만들면
(Agent, Smith), (Smith, knocks), (knocks, the), (the, door)와 같이 연속적으로 2개의 단어들을 순차적으로 이동하면서 단어들을 토큰화

스톱 워드 제거

스톱 워드(Stop word): 분석에 큰 의미가 없는 단어
ex) is, the, a, will 등 문장을 구성하는 필수 문법 요소지만 문맥적으로 큰 의미가 없는 단어 \

의미 없는 단어를 제거하는 것이 중요한 전처리 작업

Stemming과 Lemmatization Stemming, Lemmatization:문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 것

- Stemming은 원형 단어로 변환 시 일반적인 방법을 적용하거나 더 단순화된 방법을 적용해 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있음
- Lemmatization은 품사와 같은 문법적인 요소와 더 의미적인 부분을 감안해 정확한 철자로 된 어근 단어를 찾아줌

NLTK
Stemmer - Porter, Lancaster, Snowball Stemmer 제공
Lemmatization - WordNetLemmatizer를 제공

- Stemming 수행
  work의 경우 -> 진행형(working), 3인칭 단수(works), 과거형(worked) 모두 기본 단어인 work에 ing, s, ed가 붙는 단순한 변화이므로 원형 단어로 work를 제대로 인식

amuse의 경우 각 변화가 am니se가 아닌 am니으에 ing, s, ed가 붙으므로 정확한 단어 인 amuse가 아닌 amus를 원형 단어로 인식

- Lemmatization수행
  Stemmer보다 정확하게 원형 단어를 추출해줌

03. Bag of Words — BOW

Bag of Words 모델: 문서가 가지는 모든 단어(Words)를 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피처 값을 추출하는 모델

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 각 단어(feature 또는 term)를 칼럼 형태로 나열합니다. 그러고 나서 각 단어에 고유의 인덱스를 다음과 같이 부여합니다.
   '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 횟수(Occurrence)를 각 단어(단어 인덱스)에 기재합니다.
   예를 들어 base-ball은 문장 1, 2에서 총 2번 나타나며, daughter는 문장 1에서만 1 번 나타납니다.

BOW 모델의 장점: 쉽고 빠른 구축에 있음.
단순히 단어의 발생 횟수에 기반하고 있지만, 예상보다 문서의 특징을 잘 나타낼 수 있는 모델이어서 전통적으로 여러 분야에서 활용도가 높음

BOW 모델의 단점: 문맥 의미(Semantic Context) 반영 부족: BOW는 단어의 순서를 고려하지 않기 때문에 문장 내에서 단어의 문맥적인 의미가 무시됨
희소 행렬 문제(희소성, 희소 행렬): BOW로 피처 벡터화를 수행하면 희소 행렬(대규모의 칼럼으로 구성된 행렬에서 대부분의 값이 0으로 채워자는 행렬)형태의 데이터 세트가 만들어지기 쉬움

BOW 피처 벡터화
머신러닝 알고리즘은 일반적으로 숫자형 피처를 데이터로 입력받아 동작하기 때문에 텍스트와 같은 데이터는 머신러닝 알고리즘에 바로 입력할 수가 없음
-> 스트는 특정 의미를 가지는 숫자형 값인 벡터 값으로 변환해야 하는데, 이러한 변환이 *피처 벡터화*

<피처 벡터화>
각 문서(Document)의 텍스트를 단어로 추출해 피처로 할당하고, 각 단어의 발생 빈도와 같은 값을 이 피처에 값으로 부여해 각 문서를 이 단어 피처의 발생 빈도 값으로 구성된 벡터로 만드는 기법

BOW 모델에서 피처 벡터화: 모든 문서에서 모든 단어를 칼럼 형태로 나열하고 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경하는 것

- 방식1. 카운트 기반의 벡터화: 단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수, 즉 Count를 부여하는 경우
- 방식2. TF-IDF(Term Frequency — Inverse Document Frequency) 기반의 벡터화: 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 노든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 페널 티를 주는 방식으로 값을 부여

-> 문서마다 텍스트가 길고 문서의 개수가 많은 경우 카운트 방식보다는 TF-IDF 방식을 사용하는 것이 더 좋은 예측 성능을 보장할 수 있음

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

사이킷런의 CountVectorizer 클래스: 카운트 기반의 벡터화를 구현한 클래스.

- 단지 피처 벡터화만 수행하지는 않으며 소문자 일괄 변환, 토큰화, 스톱 워드 필터링 등의 텍스트 전처리도 함께 수행
- CountVectorizer에 이러한 텍스트 전처리 및 피처벡터화를 위한 입력 파라미터를 설정해 동작
- CountVectorizer 역시 사이킷런의 다른 피처 변환 클래스와 마찬가지로 fit()과 transform()을 통해 피처 벡터화된 객체를 반환

<CountVectorizer 클래스를 이용해 카운트 기반의 피처 여러 개의 문서로 구성된 텍스트의 피처 벡터화 방법>

1. 영어의 경우 모든 문자를 소문자로 변경하는 등의 전처리 작업을 수행
2. 디폴트로 단어 기준으로 n_gram_range를 반영해 각 단어를 토큰화

3. 텍스트 정규화를 수행
4. max_df, min_df, max_features 등의 파라미터를 이용해 토큰화된 단어를 피처로 추출하고 단어 빈도수 벡터 값을 적용

BOW 벡터화를 위한 희소 행렬

사이킷런의 CountVectorizer/TfidfVectorizer를 이용해 텍스트를 피처 단위로 벡터화해 변환하고 CSR 형태의 희소 행렬을 반환

희소 행렬: 대규모 행렬의 대부분의 값을 0이 차지하는 행렬
BOW 형태를 가진 언어 모델의 피처 벡터화는 대부분 희소 행렬 희소 행렬을 물리적으로 적은 메모리 공간을 차지할 수 있도록 변환해야 함
-> 대표적인 방법: COO 형식과 CSR 형식

희소 행렬 — COO 형식

COO(Coordinate: 좌표) 형식: 0이 아닌 데이터만 별도의 데이터 배열(Array)에 저장하고, 그 데이터가 가리키는 행과 열의 위치를 별도의 배열로 저장하는 방식

희소 행렬 - CSR 형식

CSR(Compressed Sparse Row) 형식: COO 형식이 행과 열의 위치를 나타내기 위해서 반복적인 위치 데이터를 사용해야 하는 문제점을 해결한 방식
고유 값의 시작 위치만 알고 있으면 얼마든지 행 위치 배열을 다시 만들 수 있기에 COO 방식보다 메모리가 적게 들고 빠른 연산이 가능

05. 감성 분석

감성 분석: 감성 분석(Sentiment Analysis)은 문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법으로 소셜 미디어, 여론조사, 온라인 리뷰, 피드백 등 다양한 분야에서 활용

- 문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와문맥을 기반으로 감성(Sentiment) 수치를 계산하는 방법을 이용
- 긍정 감성 지수와 부정 감성 지수로 구성되며 이들 지수를 합산해 긍정 감성 또는 부정 감성을 결정
- 머신러닝 관점에서 지도학습과 비지도학습 방식으로 나뉨
  • 지도학습은 학습 데이터와 타깃 레이블 값을 기반으로 감성 분석 학습을 수행한 뒤 이를 기반으로 다른 데이터의 감성 분석을 예측하는 방법으로 일반적인 텍스트 기반의 분류와 거의 동일합니다.
  • 비지도학습은 'Lexicon'이라는 일종의 감성 어휘 사전을 이용합니다. Lexicon은 감성 분석을 위한 용어와 문맥에 대한 다양한 정보를 가지고 있으며. 이를 이용해 문서의 긍정적, 부정적 감성 여부를 판단합니다.

지도학습 기반 감성 분석 실습 - IMDB 영화평

영화평의 텍스트를 분석해 감성 분석 결과가 긍정 또는 부정인지를 예측하는 모델을 만들기

https://www.kaggle.com/competitions/word2vec-nlp-tutorial/data 데이터 사용
labeledTrainData.tsv

- id : 각데이터의 id
- sentiment : 영화평(review)의 Sentiment 결과 값(Target Label). 1은 긍정적 평가, 0은 부정적 평가를 의미합니다.

- review : 영화평의 텍스트입니다.

감상평(Review) 텍스트를 피처 벡터화한 후에 ML 분류 알고리즘을 적용해 예측 성능을 측정

비지도학습 기반 감성 분석 소개

- 비지도 감성 분석은 Lexicon을 기반으로 하는 것
- 지도 감성 분석은 데이터 세트가 레이블 값을 가지고 있지만 많은 감성 분석용 데이터는 이러한 결정된 레이블 값을 가지고 있지 않기에 Lexicon이 유용
- Lexicon은 일반적으로 어휘집을 의미하지만 여기서는 주로 감성만을 분석하기 위해 지원하는 감성 어휘 사전

<감성 사전>
긍정(Positive) 감성 또는 부정 (Negative) 감성의 정도를 의미하는 수치를 가지고 있으며 이를 감성 지수 (Polarity score)라고 함 단어의 위치나 주변 단어, 문맥, POS(Part of Speech) 등을 참고해 결정됨
감성 사전을 구현한 대표격은 NLTK 패키지

<NLTK 패키지의 WordNet>
WordNet: 단순한 어휘 사전이 아닌 시맨틱('문맥상 의미') 분석을 제공하는 어휘 사전
다양한 상황에서 같은 어휘라도 다르게 사용되는 어휘의 시맨틱 정보를 제공
이를 위해 각각의 품사(명사, 동사, 형용사, 부사 등)로 구성된 개별 단어를 Synset(Sets of cognitive synonyms)이라는 개념을 이용해 표현
Synset: 단순한 하나의 단어가 아니라 그 단어가 가지는문맥, 시맨틱 정보를 제공하는 WordNet의 핵심 개념

NLTK의 감성 사전의 단점: 아쉽게도 예측 성능은 그리 좋지 못함

<대표적 감성 사전>

- sendtiWordNet
- VADER
- Pattern

SentiWordNet 을 이용한 감성 분석

Synset:POS(Part of Speech로 우리말로 바꾸면 품사), 정의(Definition), 부명제(Lemma) 등으로 시맨틱적인 요소를 표현할 수 있음

WordNet: 어떤 어휘와 다른 어휘 간의 관계를 유사도로 나타낼 수 있음.
synset 객체: 단어 간의 유사도를 나타내기 위해서 path_similarity( ) 메서드를 제공

SentiWbrdNet: WbrdNet의 Synset과 유사한 Senti_Synset 클래스를 가지고 있음
SentiWordNet 모듈의 senti_synsets(): WordNet 모듈이라서 synsets()와 비슷하게 Senti_Synset 클래스를 리스트 형태로 반환

SentiSynset 객체: 단어의 감성을 나타내는 감성 지수와 객관성을(감성과 반대) 나타내는 객관성 지수를 가지고 있음
감성 지수: 긍정 감성 지수와 부정 감성 지수로 나뉨.
어떤 단어가 전혀 감성적이지 않으면 객관성 지수는 1이 되고, 감성 지수는 모두 0이 됨

SentiWordNet을 이용한 영화 감상평 감성 분석

1. 문서(Document)를 문장(Sentence) 단위로 분해

2. 다시 문장을 단어(Word) 단위로 토큰화하고 품사 태깅
3. 품사 태깅된 단어 기반으로 synset 객체와 senti_synset 객체를 생성
4. senti_synset 객체에서 긍정 감성/부정 감성 지수를 구하고 이를 모두 합산해 특정 임계치 이상일 때 긍정 감성으로, 그렇지 않을 때는 부정 감성으로 결정

VADER 를이용한 감성분석

또 다른 Lexicon인 VADER Lexicon

VADER: 소셜 미디어의 감성 분석 용도로 만들어진 룰 기반의 Lexicon
SentimentIntensityAnalyzer 클래스를 이용해 쉽게 감성 분석을 제공