NLP (National Language Processing): 머신이 인간의 언어를 이해하고 해석하는 것

* 기계 번역, 질의 응답 시스템 등

텍스트 분석(텍스트 마이닝): 비정형 텍스트에서 의미 있는 정보 추출

* 머신러닝, 언어 이해, 통계 등을 활용해 모델 수립하고 정보 추출해 분석 작업 수행
* 텍스트 분류, 감성분석, 텍스트 요약, 텍스트 군집화와 유사도 측정

**텍스트 분석**

머신러닝에 적용하기 위해서 비정형 텍스트 데이터를 피처 형태로 추출하고 추출된 피처에 의미 있는 값을 부여해야 함

피처 벡터화, 피처 추출: 텍스트를 word 기반의 다수의 피처로 추출하고 이 피처에 단어 빈도수같은 숫자 값을 부여하면 텍스트는 단어의 조합인 벡터 값으로 표현될 수 있음

텍스트를 벡터값 가지는 피처로 변환하는 것은 머신러닝 모델 적용하기 전에 수행해야 할 중요한 요소

텍스트 분석 수행 프로세스

1. 텍스트 전처리: 텍스트를 피처로 만들기 전에 클렌징, 대소문자 변경, 특수문자 삭제, 단어 등 토큰화, stopword 제거, 어근 추출 등의 텍스트 정규화
2. 피처 벡터화/추출: 가공된 텍스트에서 피처 추출하고 벡터 값 할당

* BOW(Count 기반, TF-IDF 기반), Word2Vec

1. ML 모델 수립 및 학습/예측/평가

**텍스트 전처리 – 텍스트 정규화**

**클렌징**: 불필요한 문자, 기호 등을 제거 🡪 html, xml 태그나 특정 기호 등

**토큰화**

* 문장 토큰화: 문장의 마침표, 개행 문자 등 문장의 마지막을 뜻하는 기호에 따라 분리, 정규표현식에 따른 문장 토큰화도 가능

일반적으로 문장 토큰화는 각 문장이 가지는 시맨틱적인 의미가 중요한 요소로 사용될 때 사용

* 단어 토큰화: 문장을 단어로 토큰화, 공백/콤마/마침표/개행 문자 등으로 단어 분리, 정규 표현식 이용해 토큰화 가능

단어의 순서가 중요하지 않은 경우 문장 토큰화 사용하지 않고 단어 토큰화만 사용

문장을 단어 별로 토큰화 할 경우 문맥적인 의미는 무시될 수밖에 없음 🡪 n-gram을 통해 연속된 n개의 단어를 하나의 토큰화 단위로 분리, n개의 단어들을 순차적으로 이동하면서 토큰화

**필터링/스톱 워드 제거/철자 수정**

스톱워드: 분석에 큰 의미가 없는 단어 🡪 is, the, a 등 필수 문법 요소지만 문맥적으로 큰 의미가 없는 단어

**Stemming, Lemmatization:** 문법적 또는 의미적으로 변화하는 단어(진행형, 3인칭 단수, 과거형 동사, 비교/최상급에 따른 형용사의 변화)의 원형 찾는 것

Stemming: 일반적은 방법 적용하거나 더 단순화된 방법 적용해 원래 단어에서 일부 철자가 훼손된 어근 단어 추출하는 경향이 있음 🡪 Porter, Lancaster, Snowball Stemmer

Lemmatization: 품사와 같은 문법적 요소와 더 의미적인 부분을 감안해 정확한 철자로 된 어근 단어 찾음 🡪 더 정교하고 의미론적 기반에서 단어 원형 찾으며 더 오랜 시간 걸림 🡪 WordNetLemmatizer

**Bag of Words – BOW**

문서가 가지는 모든 단어를 문맥이나 순서 무시하고 일괄적으로 단어에 대해 빈도 값 부여해 피처 값을 추출하는 모델

쉽고 빠르며 단순히 단어의 발생 횟수에 기반하지만 문서 특징 어느정도 잘 나타냄

* 문맥의미 반영 부족, 희소 행렬 문제라는 단점이 존재

**BOW 피처 벡터화**

피처 벡터화: 텍스트를 특정 의미 가지는 숫자형 값인 벡터 값으로 변환

모든 문서에서 모든 단어를 칼럼 형태로 나열하고 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경

1. 카운트 기반의 벡터화

단어 피처에 값 부여할 때 각 문서에서 해당 단어가 나타나는 횟수 부여

Count 값이 높을수록 중요한 단어 인식

* 그 문서의 특징 나타내기보다는 언어의 특성상 문장에서 자주 사용되는 단어까지 높은 값 부여하는 단점

1. TF-IDF(Term Frequency – Inverse Document Frequency) 벡터화

개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 페널티 주는 방식으로 값을 부여

개별문서에서 단어i의 빈도 \* log(전체 문서 개수 / 단어i를 가지고 있는 문서 개수)

**사이킷런의 Count 및 TF-IDF 벡터화 구현**

CountVectorizer 클래스는 카운트 기반의 벡터화 구현, 전처리도 함께 수행

1. Max\_df: 전체 문서에 걸쳐서 너무 높은 빈도수 가지는 단어 피처 제외하는 파라미터

정수 값 가지면 전체 문서에 걸쳐 그 정수값 이하로 나타나는 단어만 피처로, 부동소수점 값 가지면 그 퍼센트의 빈도수 가지는 단어만 피처로 추출

1. Min\_df: 전체 문서에 걸쳐서 너무 낮은 빈도수 가지는 단어 피처 제외, 이 파라미터에 설정된 값보다 적은 빈도수 가진다면 크게 중요하지 않거나 가비지성 단어일 확률 높음
2. Max\_features: 추출하는 피처의 개수 제한
3. Stop\_words
4. N\_gram\_range: 단어 순서를 어느정도 보강하기 위해 n\_gram 범위 튜플 형태로 설정
5. Analyzer: 피처 추출을 수행한 단위 지정, 디폴트는 word
6. Token\_pattern: 토큰화를 수행하는 정규 표현식 패턴 지정, 공백 또는 개행 문자 등으로 구분된 단어 분리자(\b) 사이의 2문자 이상의 단어를 토큰으로 분리
7. Tokenizer: 토큰화를 별도의 커스텀 함수로 이용 시 적용, 어근 변환 시 이를 수행하는 별도 함수를 이 파라미터에 적용

모든 문자를 소문자로 변환하는 등의 전처리 🡪 단어 기준으로 n\_gram\_range 반영하여 토큰화 🡪 stop\_word 필터링, stemming/lemmatization은 CountVectorizr에서 직접 지원하지는 않음 🡪 토큰화된 단어를 피처로 추출하고 단어 빈도수 벡터 값 적용

TF-IDF 벡터화는 TfidfVectorizer 클래스 이용, CountVectorizer와 동일

**BOW 벡터화를 위한 희소 행렬**

모든 문서에 있는 단어를 추출해 피처로 벡터화하면 많은 칼럼이 생성되어 대규모 행렬의 대부분의 값을 0이 차지하는 희소 행렬 생성됨

너무 많은 불필요한 0 값이 메모리 공간에 할당되어 많은 메모리 공간을 필요로 하며 데이터 엑세스를 위한 시간 많이 소모됨 🡪 COO, CSR 방식으로 희소 행렬을 물리적으로 적은 메모리 공간 차지할 수 있도록 변환

**희소 행렬 - COO 형식**

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

**희소 행렬 - CSR 형식**

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

COO 형식은 행 위치 배열에서 순차적인 같은 값이 반복적으로 나타날 수 있음

* 행 위치 배열 내에 있는 고유한 값의 시작 위치만 다시 별도의 위치 배열로 가지는 변환 방식
* 맨 마지막에는 데이터의 총 항목 개수를 배열에 추가

CountVectorizer, TfidfVectorizer 클래스로 변환된 피처 벡터화 행렬은 모두 scipy의 CSR 형태의 희소 행렬

**텍스트 분류**

* 문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법 – 연애/정치 등 어떤 카테고리에 속하는지 자동으로 분류하거나 스팸 메일 검출 등
* 지도학습 이용
* 특정 문서의 분류를 학습 데이터를 통해 학습해 모델 생성한 뒤 이 학습 모델 이용해 다른 문서의 분류 예측
* 희소 행렬에 분류를 효과적으로 잘 처리할 수 있는 알고리즘은 로지스틱 회귀, 선형 서포트 벡터 머신, 나이브 베이즈 등

**감성분석 (Sentiment Analysis)**

문서의 주관적 감성/의견/감정/기분 등을 파악하는 방법으로 문서 내 텍스트가 나타내는 여러 주관적인 단어와 문맥을 기반으로 감성 수치 계산하는 방법 이용 🡪 긍정/부정 감정 지수

* 지도학습: 학습 데이터와 타깃 레이블 값을 기반으로 수행한 뒤 이를 기반으로 다른 데이터의 감성 분석 예측하는 방법 🡪 텍스트 기반의 분류와 동일
* 비지도학습: lexicon이라는 감성 어휘 사전 이용, 감성 분석을 위한 용어와 문맥에 대한 다양한 정보를 가지며 이를 이용해 문서의 긍정적, 부정적 감성 여부 판단

많은 감성 분석용 데이터는 결정된 레이블 값 가지고 있지 않아서 Lexicon(감성 어휘 사전) 이용

* 감성 사전은 긍정 감성 또는 부정 감성의 정도를 의미하는 수치를 가지고 있으며 이 감정 지수는 단어의 위치나 주변 단어, 문맥, Part Of Speech(품사) 등으로 결정됨 🡪 NLTK 패키지

NLP 패키지의 WordNet: 방대한 영어 어휘 사전, 시맨틱(문맥상 의미) 분석 제공하는 어휘 사전

* WordNet은 다양한 상황에서 같은 어휘라도 다르게 사용되는 어휘의 시맨틱 정보 제공하며 이를 위해 각각의 품사로 구성된 개별 단어를 Synset이라는 개념을 이용해 표현, (단순한 하나의 어휘가 아니라 그 단어가 가지는 문맥, 시멘틱 정보 제공)

NLTK의 감성 사전은 예측 성능이 그리 좋지 않다는 단점이 있음

* SentiWordNet: NLTK의 WordNet과 유사하게 감성 단어 전용의 WordNet 구현, WordNet의 Synset 개념을 감성 분석에 적용한 것 🡪 긍정 감성/부정 감성/객관성 지수
* VADER: 소셜 미디어의 텍스트에 대한 감성 분석 제공
* Pattern: 파이썬 2.X 버전에서만 동작

SentimentWordNet Lexicon기반으로 감성 분석 수행하는 순서

1. 문서를 문장 단위로 분해
2. 문장을 단어 단위로 토큰화하고 품사 태깅
3. 품사 태깅된 단어 기반으로 synset 객체와 senti\_synset 객체 생성
4. Senti\_synset에서 긍정/부정 감성 지수 구하고 합산해 특정 임계치 값 이상일 때 긍정 감성으로, 그렇지 않으면 부정 감성으로 결정

VADER Lexicon: 소셜 미디어의 감성 분석 용도 🡪 NLTK 패키지의 서브 모듈 or 단독 패키지

**토픽 모델링 (Topic Modeling)**

문서 집합에 숨어 있는 주제를 찾아내는 것 – 중심 단어를 함축적으로 추출

* LSA (Latent Semantic Analysis)
* LDA (Latent Dirichlet Allocation)

🡪 사이킷런은 LDA 기반의 토픽 모델링을 LatentDirichletAllocation 클래스로 제공

🡪 n\_components 파라미터를 이용해 토픽 개수 조정

🡪 LatentDirichletAllocation.fit() 적용하면 LatentDirichletAllocation 객체는 components\_ 속성 가지게 됨, 개별 토픽별로 각 word 피처가 얼마나 많이 그 토픽에 할당되었는지에 대한 수치 가짐, 높은 값일수록 해당 word피처는 그 토픽의 중심 word가 됨

**문서 군집화 (Document Clustering)**

비슷한 텍스트 구성의 문서를 군집화하는 것

동일한 군집에 속하는 문서를 같은 카테고리 소속으로 분류할 수 있으므로 텍스트 분류 기반의 문서 분류와 유사 🡪 하지만 문서 군집화는 학습 데이터 세트가 필요 없는 비지도 학습

**문서 유사도**

문서와 문서 간의 유사도 비교는 일반적으로 코사인 유사도(Cosine Similarity) 사용

코사인 유사도: 두 벡터 간의 유사도 비교할 때 벡터의 크기보다는 벡터의 상호 방향성이 얼마나 유사한지에 기반함 🡪 두 벡터 사이의 사잇각 구해서 수치로 적용 🡪 내적 결과를 총 벡터 크기로 정규화

문서를 피처 벡터화 변환하면 희소 행렬이 되기 쉬움, 희소 행렬 기반에서 문서와 문서 벡터 간의 크기에 기반한 유사도 지표(ex)유클리드 거리 기반 지표)는 정확도 떨어지기 쉬움

**한글 텍스트 처리**

한글 NLP 처리의 어려움: 띄어쓰기와 다양한 조사 때문

KoNLPy: 파이썬의 대표적인 한글 형태소 패키지

* 형태소: 단어로서 의미를 가지는 최소 단위
* 형태소 분석: 말뭉치를 이러한 형태소 어근 단위로 쪼개고 각 형태소에 품사 태깅을 부착하는 작업
* Kkma, Hannanum, Komoran, Mecab(리눅스), Twitter 모듈을 KoNLPy에서 모두 사용 가능

Colab에서 konlpy 오류 날 경우

!apt-get update

!apt-get install g++ openjdk-8-jdk python-dev python3-dev

!pip3 install JPype1-py3

!pip3 install konlpy

!JAVA\_HOME=환경 변수 설정 경로

**정리**

텍스트 분석 프로세스: 텍스트 정규화 🡪 피처 벡터화 🡪 머신러닝 학습, 예측, 평가

텍스트 정규화: 텍스트 클렌징, 대소문자 변경, 단어 토큰화, 의미 없는 단어 필터링, 어근 추출 등 피처 벡터화 전 다양한 사전 작업

* 피처 벡터화: BOW의 대표 방식인 Count 기반과 TF-IDF 기반 피처 벡터화 (희소 행렬)

텍스트 분류: 문서들을 피처 벡터화한 후 로지스틱 회귀 적용해 지도 학습

감성 분석: 지도학습 기반으로 긍정/부정 이진 분류와 감성 사전 Lexicon

토픽 모델링: LDA 이용

텍스트 군집화: K-Means 군집화 기법 이용해 비슷한 문서 군집화

텍스트 유사도 측정: 코사인 유사도 이용