[Week13] 파머완 8장(1)

CHAP 08 텍스트 분석

NLP vs 텍스트 분석

- NLP(National Language Processing) : 머신이 인간의 언어를 이해하고 해석하는 데 더 중점을 두고 발전한 기술
  → 기계 번역, 질의응답 시스템, 텍스트 분석을 향상하게 하는 기반 기술
- 텍스트 분석(Text Mining) : 비정형 텍스트에서 의미있는 정보를 추출하는 것에 중점을 두고 발전한 기술
  → 비즈니스 인텔리전스, 예측 분석 등 분석 작업
  - 텍스트 분류 : 문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법
  - 감성 분석 : 텍스트에서 나타나는 주관적인 요소 분석하는 기법
  - 텍스트 요약 : 텍스트 내에서 중요한 주제나 중심 사상을 추출하는 기법
  - 텍스트 군집화와 유사도 측정 : 비슷한 유형의 문서에 대해 군집화를 수행하는 기법

01 텍스트 분석 이해

텍스트 분석

- 비정형 데이터인 텍스트를 분석하는 것
- 비정형 텍스트 데이터를 어떻게 피처 형태로 추출하고 추출된 피처에 의미있는 값을 부여하는가 → 머신러닝 적용
- 피처 벡터화(Feature Vectorization) / 피처 추출(Feature Extraction) : 텍스트를 word 기반 다수 피처로 추출하고 단어 빈도수와 같은 숫자값을 부여해 단어의 조합인 벡터값으로 표현하는 것
  → BOW, Word2Vec

텍스트 분석 수행 프로세스

1. 텍스트 사전 준비작업(텍스트 전처리)
   - 클렌징, 대/소문자 변경, 특수문자 삭제, 등 클렌징 작업
   - 단어(Word) 등의 토큰화 작업
   - 의미 없는 단어 제거 작업
   - 어근 추출 등 텍스트 정규화 작업
2. 피처 벡터화/추출 : 가공된 텍스트에서 피처 추출, 벡터값 할당 → BOW(Count 기반/TF-IDF 기반), Word2Vec
3. ML 모델 수립 및 학습/예측/평가

파이썬 기반의 NLP, 텍스트 분석 패키지

- NLTK(Natural Language Toolkit for Python)
  - 파이썬의 가장 대표적인 NLP 패키지
  - 방대한 데이터 세트와 서브 모듈
  - 수행 속도 측면에서 아쉬운 부분 존재
- Gensim : 토픽 모델링 분야에서 가장 두각을 나타내는 패키지
- SpaCy : 뛰어난 수행성능을 가진 NLP 패키지

02 텍스트 사전 준비 작업(텍스트 전처리) - 텍스트 정규화

텍스트 정규화

- 텍스트를 머신러닝 알고리즘이나 NLP 어플리케이션에 입력 데이터로 사용하기 위해 다양한 텍스트 데이터 사전 작업을 수행하는 것
- 클렌징 & 토큰화 & 필터링/스톱 워드 제거/철자 수정 & Stemming & Lemmatization
- 클렌징(Cleansing) : 텍스트 분석에 오히려 방해가 되는 불필요한 문자, 기호 등을 사전에 제거

텍스트 토큰화

문장 토큰화(Sentence tokenization)

- 문장의 마침표, 개행문자(\n) 등 문장의 마지막을 뜻하는 기호에 따라 분리
- 정규표현식에 따라서도 가능
- `sent_tokenize()` : 각각의 문장으로 구성된 list 객체 반환
- `nltk.download('punkt')` : 마침표, 개행 문자 등 데이터셋 다운로드

단어 토큰화(Word tokenization)

- 문장을 단어로 토큰화
- 공백, 콤마, 마침표, 개행문자, 정규표현식 등으로 토큰화 수행
- `word_tokenize()` : 단어로 토큰화
- n-gram : 연속된 n개의 단어를 하나의 토큰화 단위로 분리

스톱 워드 제거

- 스톱 워드 : 분석에 큰 의미가 없는 단어
- `nltk.download('stopwords')` : stopwords 목록 다운로드

Stemming과 Lemmatization

- 문법적 또는 의미적으로 변화하는 단어의 원형을 찾는 것

- Stemming
  - 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향
  - Porter, Laucaster, Snowball Stemmer
  - `LancasterStemmer()` → `stem('단어')` : 원하는 단어 stemming
- Lemmatization
  - 정확한 철자로 된 어근 단어를 찾아줌
  - 더 정교하지만 오랜 시간을 필요로 함
  - WordNetLemmatizer
  - `WordNetLemmatizer()` → `lemmatize('단어','품사')` : 원하는 단어 Lemmatization

03 Bag of Words - BOW

BOW

- 문서가 가지는 모든 단어를 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피처 값을 추출하는 모델
- 피처 추출 방식
  1. 단어의 중복을 제거하고, 각 단어를 칼럼 형태로 나열
  2. 각 단어에 고유 인덱스 부여
  3. 해당 단어가 나타나는 횟수를 단어 인덱스에 기재
- 장점 : 쉽고 빠른 구축
- 단점 : 문맥 의미 반영 부족, 희소 행렬 문제

BOW 피처 벡터화

- 모든 문서에서 모든 단어를 칼럼 형태로 나열 → 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터세트 모델로 변경
- N개의 단어 가정 → M개의 문서는 각각 N개의 값이 할당된 피처 벡터 세트 → M x N 행렬
- 카운트 벡터화
  : 피처값을 count로 부여하는 경우
  : Count 값이 높을수록 중요한 단어
  : 언어 특성 상 자주 사용될 수밖에 없는 단어까지 높은 값을 부여하는 문제
- TF-IDF(Term Frequency Inverse Document Frequency) 벡터화
  : 자주 나타나는 단어에 높은 가중치
  : 모든 문서에서 전반적으로 자주 나타나는 단어에는 페널티 부여

사이킷런의 Count 및 TF-IDF 벡터화 구현

- CountVectorizer : 카운트 기반 벡터화 구현, 텍스트 전처리도 함께 수행
  - max_df : 너무 높은 빈도수를 가지는 단어 피처 제외
    - 정수값 → 정수 이하로 나타나는 단어만 추출

- 부동소수점값 → 빈도 퍼센트까지만 추출

  - min_df : 너무 낮은 빈도수를 가지는 단어 피처 제외
    - 정수값 → 정수 이하로 나타나는 단어 제외
    - 부동소수점값 → 하위 퍼센트 단어 제외
  - max_features : 추출하는 피처 개수 제한
  - stop_words : 'english'로 지정할 경우 영어의 스톱 워드 추출 제외
  - n_gram_range : BOW 모델 단어 순서를 보강하기 위한 n_gram 범위 설정 (범위 최솟값, 최댓값)
  - analyzer = 'word' : 피처 추출을 수행한 단위 지정
  - token_pattern : 토큰화를 수행하는 정규 표현식 패턴 지정
  - tokenizer : 토큰화를 별도의 커스텀 함수로 이용시 적용

  1. 전처리 작업 수행
  2. n_gram_range를 반영해 각 단어 토큰화
  3. 텍스트 정규화 수행
  4. 피처 벡터화 수행

- TfidfVectorizer : TF-IDF 벡터화

BOW 벡터화를 위한 희소 행렬

- CSR 형태의 희소 행렬 반환
- 희소 행렬 : 대부분의 값을 0이 차지하는 행렬
- 너무 많은 불필요한 0값이 메모리 공간에 할당 → 변환 필요

희소 행렬

- COO(Coordinate) : 좌표 형식, 0이 아닌 데이터만 별도의 데이터 배열에 저장 → 행과 열의 위치를 별도의 배열로 저장
- Scipy의 sparse 패키지 → `coo_matrix()`
- CSR(Compressed Sparse Row)
- `csr_matrix()`

04 텍스트 분류 실습 - 20 뉴스그룹 분류

텍스트 정규화

- `fetch_20newsgroups()` : 뉴스그룹 분류를 수행해볼 수 있는 예제 데이터 제공

```
from sklearn.datasets import fetch_20newsgroups

#subset으로 학습용 데이터만 추출, removed로 내용만 추출
```

```
train_news = fetch_20newsgroups(subset='train', remove=('headers','footers','quotes'), random_state=156)
X_train = train_news.data
y_train = train_news.target

test_news = fetch_20newsgroups(subset='test', remove=('headers','footers','quotes'), random_state=156)
X_test = test_news.data
y_test = test_news.target
print('학습 데이터 크기 {0}, 테스트 데이터 크기 {1}'.format(len(train_news.data), len(test_news.data)))
```

피처 벡터화 변환과 머신러닝 모델 학습/예측/평가

❗ 테스트 데이터에서 CountVectorizer를 적용할 때는 반드시 학습 데이터를 이용해 fit( )이 수행된 CountVectorizer 객체를 이용해 테스트 데이터를 변환해야 함 ❗

```
#count 벡터화
from sklearn.feature_extraction.text import CountVectorizer

cnt_vect = CountVectorizer()
cnt_vect.fit(X_train)
X_train_cnt_vect = cnt_vect.transform(X_train)

#학습 데이터로 fit()된 CountVectorizer를 이용해 테스트 데이터를 피처 벡터화 변환 수행
X_test_cnt_vect = cnt_vect.transform(X_test)
print('학습 데이터 텍스트의 CountVectorizer Shape : ', X_train_cnt_vect.shape)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=300 )
tfidf_vect.fit(X_train)
X_train_tfidf_vect = tfidf_vect.transform(X_train)
X_test_tfidf_vect = tfidf_vect.transform(X_test)

lr_clf = LogisticRegression(solver='liblinear')
lr_clf.fit(X_train_tfidf_vect , y_train)
pred = lr_clf.predict(X_test_tfidf_vect)
print('TF-IDF Vectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred)))

from sklearn.model_selection import GridSearchCV

params = {'C':[0.01,0.1,1,5,10]}
grid_cv_lr = GridSearchCV(lr_clf, param_grid=params, cv=3, scoring='accuracy',verbose=1)
grid_cv_lr.fit(X_train_tfidf_vect, y_train)
print('Logistic Regression best C parameter : ', grid_cv_lr.best_params_)

pred = grid_cv_lr.predict(X_test_tfidf_vect)
print('TF-IDF Vectorized Logistic Regression의 예측 정확ㄷ노는 {0:.3f}'.format(accuracy_score(y_test, pred)))
```

사이킷런 파이프라인 사용 및 GridSearchCV와의 결합

- Pipeline : 데이터의 가공, 변환 등의 전처리와 알고리즘 적용을 한꺼번에 스트림 기반으로 처리

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english')),
    ('lr_clf', LogisticRegression(solver='liblinear'))
```

```
])

params = { 'tfidf_vect__ngram_range': [(1,1), (1,2), (1,3)],
           'tfidf_vect__max_df': [100, 300, 700],
           'lr_clf__C': [1, 5, 10]
}

# GridSearchCV의 생성자에 Estimator가 아닌 Pipeline 객체 입력
grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3 , scoring='accuracy',verbose=1)
grid_cv_pipe.fit(X_train , y_train)
print(grid_cv_pipe.best_params_ , grid_cv_pipe.best_score_)

pred = grid_cv_pipe.predict(X_test)
print('Pipeline을 통한 Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred)))
```

05 감성 분석

감성 분석(Sentiment Analysis)

- 문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
- 문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와 문맥을 기반으로 감성 수치를 계산하는 방법
- 긍정 감성 지수 / 부정 감성 지수

비지도학습 기반 감성 분석

- Lexicon 기반 → 감성 어휘 사전
- 감성 지수 : 긍정 감성 또는 부정 감성의 정도를 의미하는 수치 → NLTK 패키지 Lexicon 모듈
- 시맨틱(semantic) : 문맥상 의미
- NLP의 WordNet 모듈 : 방대한 영어 어휘의 시맨틱 정보 제공 → Synset(Sets of cognitive synonyms)
- 감성 사전 종류
  - SentiWordNet : WordNet의 Synset 개념을 감성 분석에 적용
  - VADER : 소셜 미디어 텍스트에 대한 감성 분석을 제공하기 위한 패키지
  - Pattern : 예측 성능 측면에서 가장 주목받는 패키지

06 토픽 모델링

토픽 모델링(Topic Modeling)

- 문서 집합에 숨어있는 주제를 찾아내는 것
- 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출
- LSA와 LDA 기법 → LDA는 count 기반 벡터화만 적용
- 사이킷런 LDA(Latent Dirichlet Allocation) `LatentDirichletAllocation` 클래스
  - n_components : 토픽 개수 조정

- components_ : 개별 토픽별로 각 word 피처가 얼마나 많이 그 토픽에 할당됐는지에 대한 수치
  → 높은 값일수록 해당 word 피처가 중심 word