

## 07 데이터 분석 심화 - 자연어 처리(ver2.0)

AI 에이전트 개발

# 데이터 분석 심화

원티드랩

- [소개](#)
- [1. 토큰나이징](#)
  - [1\) 형태소 분석기](#)
  - [2\) 워드 클라우드](#)
- [2. 장바구니 분석](#)
- [3. 네트워크 분석](#)
- [4. 유사도 분석](#)
  - [1\) 인코딩\(Encoding\)](#)
  - [2\) 벡터화\(Vectorizing\)](#)
  - [3\) 임베딩\(Embedding\)](#)
  - [4\) 유사도 계산 방법](#)

## 소개

자연어 처리(NLP, Natural Language Processing)는 "언어"를 컴퓨터가 이해하고 분석할 수 있도록 만드는 기술로 기계 번역, 챗봇, 음성 인식, 요약 등 다양한 분야에서 활발하게 활용되고 있다.

이번 시간에는 자연어 처리 중에서도 텍스트 데이터를 분석하고 시각화 하는 것에 대해 설명한다.

### 🔑 핵심 Keywords

- 텍스트를 작은 단위로 나누는 토큰나이징
- 단어 빈도 기반 시각화 워드 클라우드
- 단어의 관계를 활용한 장바구니 분석
- 단어의 관계 시각화 네트워크 분석
- 문장의 유사성을 판단하는 유사도 분석

## 1. 토큰나이징

### 1) 형태소 분석기

토큰나이징(Tokenizing)은 문장을 작은 단위로 나누는 작업이다.

문장을 공백(" ")을 기준으로 쪼갠다고 생각해보자.

나는/오늘/잠실에서/밥을/먹었어  
 밥에/머리카락이/나와서/기분이/나빴어  
 그런데/너는/밥/먹었니?

3개의 문장에서 공통으로 등장하는 단어는 "밥"이다.

그런데 공백을 기준으로 나누게 되면 "밥을", "밥에", "밥"으로 인식되어 모두 다른 단어로 컴퓨터는 인식하게 된다. "먹었어", "먹었니" 또한 마찬가지이다.

이렇게 한국어는 조사, 어미, 합성어 등으로 인해 공백만으로는 핵심 키워드를 파악하기 어렵다. 그래서 문장을 의미 있는 최소 단위(형태소)로 나누는 형태소 분석기가 필요하다.

형태소 분석기	설명
Okt	트위터에서 개발한 형태소 분석기. 신조어에 강함. KoNLPy에서 사용 가능
Kkma/Komorani/Hannanum	Java 기반으로 품사를 세분화하여 나눔. KoNLPy에서 사용 가능
Kiwi	빠르고 정확하며, 오타 교정 및 사용자 사전 추가 기능이 편리함
Mecab-ko	빠르고 정확하지만 Windows에서 설치 과정 복잡함
Nori	ElasticSearch에서 플러그인으로 제공하여 편리함. 검색 시스템에 최적화 됨.

### 2) 워드 클라우드



워드 클라우드 는 토큰나이징을 통해 추출한 토큰들을 빈도수에 따라 시각화하는 방법이다.  
자주 등장할수록 크게 표현되어 텍스트의 주요 키워드나 흐름을 한눈에 파악할 수 있다.

문장 데이터가 있을 때 워드 클라우드를 그리는 과정은 다음과 같다.

1. 데이터 전처리: 특수문자, 숫자, 기호 등을 제거해 필요없는 표현들을 제거한다.
2. 토큰나이징: 형태소 분석기를 이용해 문장을 나눈다.
3. 불용어 제거: 자주 등장하지만 의미가 약한 조사나 접속사 등을 제거한다.
4. 단어 빈도 수 계산: `Counter` 를 이용하여 단어의 빈도수를 값으로 하는 딕셔너리를 만든다.
5. 워드 클라우드 생성: `WorldCloud` 를 이용하여 워드 클라우드를 그린다.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# 이미 만들어진 단어 빈도수 딕셔너리 (Counter 객체)
# 예시: counter => Counter({'밥': 10, '국': 7, '김치': 5})
# 여기서는 counter가 준비되었다고 가정

# 워드클라우드 객체 생성
wc = WordCloud(
    font_path='NanumGothic.ttf', # 한글 폰트 경로 (없으면 깨짐)
    background_color='white',
    width=800,
    height=400
)

# 빈도수 기반으로 워드클라우드 생성
wc.generate_from_frequencies(counter)

# 시각화
plt.figure(figsize=(10, 5))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud", fontsize=20)
plt.show()
```

## 2. 장바구니 분석

장바구니 분석은 데이터마ining 기법 중 하나로 "어떤 상품이 함께 구매되는가?"를 분석할 때 활용된다.  
이를 텍스트 분야에 응용하면 어떤 단어들이 함께 등장하는가? 를 분석할 수 있다.

**Apriori** 알고리즘 자주 함께 등장하는 단어 조합(frequent itemsets)를 찾아주는 알고리즘이다.

**Association Rule Analysis**는 Apriori 알고리즘의 결과로 나온 단어 조합(frequent itemsets)를 이용하여 연관 규칙(ex.  $A \rightarrow B$ )을 만들고 그 규칙에 대한 여러 지표들을 계산하여 연관성을 평가한다.

지표	의미	예시
지지도 (Support)	전체 문서 중 A와 B가 함께 등장한 비율	$support(A \rightarrow B) = P(A \cup B)$
신뢰도 (Confidence)	A가 등장했을 때 B도 등장할 조건부 확률	$confidence(A \rightarrow B) = P(B A)$
향상도 (Lift)	A와 B가 우연이 아닌 실제 연관이 있는지를 측정	$lift(A \rightarrow B) = \frac{P(B A)}{P(B)}$

④ 향상도(Lift)의 경우  $lift(A \rightarrow B)$ 와  $lift(B \rightarrow A)$ 가 같다.

⚠ 단어 A가 등장하고 B가 등장하는 순서를 의미하지 않는다.

예를 들어  $A \rightarrow B$ 는 A가 등장할 때 B도 등장하는 경향이 있는가를 파악하기 위한 것이다.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# 1. TransactionEncoder를 사용해 one-hot 인코딩
te = TransactionEncoder()
te_ary = te.fit(tokenized_sentences).transform(tokenized_sentences)
df = pd.DataFrame(te_ary, columns=te.columns_)

# 2. 자주 등장하는 항목 집합 추출 (min_support 조절 가능)
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)

# 3. 연관 규칙 생성
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
```

### 3. 네트워크 분석

네트워크 분석(Network Analysis)은 텍스트 내 단어들의 관계를 노드(점)와 엣지(선)로 시각화하는 기법이다. 텍스트의 흐름, 주제 중심 단어, 클러스터를 한눈에 파악할 수 있다.

용어	설명
노드(Node)	네트워크 내의 개체, 여기서는 단어 하나를 의미
엣지(Edge)	단어 간 연결선, 두 단어가 함께 등장했을 때 연결
가중치(Weight)	단어 간의 연결 강도. 함께 등장한 횟수를 의미

```
import networkx as nx
import matplotlib.pyplot as plt

# 1. 그래프 객체 생성
G = nx.DiGraph() # 방향 있는 그래프 (A → B)

# 2. 엣지 추가
for _, row in rules.iterrows():
    # 단어 추출 (frozenset → str)
    antecedent = ','.join(row['antecedents'])
    consequent = ','.join(row['consequents'])
    weight = row['lift'] # 또는 row['confidence']

    # 엣지 추가 (노드 간 연결 + 가중치 포함)
    G.add_edge(antecedent, consequent, weight=weight)

# 노드 위치 자동 배치
pos = nx.spring_layout(G, seed=42) # seed 고정으로 레이아웃 재현 가능

# 엣지 가중치 추출
edge_weights = [G[u][v]['weight'] for u, v in G.edges()]

# 노드 스타일 지정
```

```
plt.figure(figsize=(5, 3))
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=1000)
nx.draw_networkx_edges(G, pos, edge_color='gray', width=edge_weights)
nx.draw_networkx_labels(G, pos, font_size=12, font_family='Malgun Gothic')

# 엣지 라벨 (lift 값)
edge_labels = {(u, v): f"{d['weight']:.2f}" for u, v, d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=10)

plt.title("단어 간 연관 규칙 기반 네트워크 그래프 (Lift 기준)")
plt.axis('off')
plt.tight_layout()
plt.show()
```

## 4. 유사도 분석

텍스트를 숫자로 바꿀 수 있다면,  
문장과 문장 간 유사도를 계산해 “이 문장과 저 문장이 얼마나 비슷한가?”를 수치로 비교할 수 있다.

### 1) 인코딩(Encoding)

단어에 고유한 숫자 ID를 부여하거나, 단어마다 1 또는 0으로 나타내는 기초적 표현 방식이다.

방법	예시	특징
<b>Label Encoding</b>	"밥", "국", "김치" → [1, 2, 3]	고유 숫자 부여. 단어 간 관계 없음
<b>One-Hot Encoding</b>	"국" → [0,1,0]	위치만 1. 의미 없음. 벡터 길이 급증

하지만 이 방식은 단어 간 의미나 관계를 전혀 반영하지 못하고, 단순히 “존재 여부”만을 알려준다.  
그래서 문장 간 유사도나 의미 분석 같은 고급 작업이 어렵다.

### 2) 벡터화(Vectorizing)

텍스트 데이터를 계산 가능한 형태로 바꾸는 벡터화 단계다.  
여기서 벡터란, 단어들의 빈도나 중요도를 수치로 나타낸 수학적 표현이다.

방법	예시	특징
<b>CountVectorizer</b>	"밥 먹고 국 먹음" → {"밥":1, "국":1, ...}	단순 빈도. 빠르고 직관적
<b>TfidfVectorizer</b>	중요 단어에 가중치 부여	전체 문서 중 희귀 단어에 더 높은 점수

### 3) 임베딩(Embedding)

벡터화는 단어의 “진짜 의미”나 “문맥”은 모른다.  
임베딩(Embedding)은 이 한계를 넘어서기 위한 방식으로 문장을 의미 기반의 벡터로 표현한다.

임베딩(Embedding)은 의미나 문맥이 비슷한 단어/문장이 비슷한 벡터로 표현되도록 학습됩니다.  
예를 들어 "나는 밥을 먹었다"와 "나는 식사를 했다"는 서로 다른 단어지만,  
비슷한 의미이기 때문에 임베딩 벡터도 가까워집니다.

## 4) 유사도 계산 방법

### 코사인 유사도(Cosine Similarity)

- 벡터 간 각도를 기준으로 유사도를 계산
- 1에 가까울수록 유사, 0이면 관련 없음

- 음수는 반대 의미지만 TF-IDF에서는 거의 안 나옴

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity(vec1, vec2)
```

## 유클리디안 거리 (Euclidean Distance)

- 두 벡터 사이의 직선 거리를 계산
- 값이 작을수록 유사, 크면 다름
- 단어 수나 문장 길이 차이가 큰 경우엔 민감할 수 있음

```
from sklearn.metrics.pairwise import euclidean_distances
euclidean_distances(vec1, vec2)
```

## 자카드 유사도 (Jaccard Similarity)

- 두 집합 간 교집합 / 합집합의 비율
- 값이 1이면 완전히 동일, 0이면 완전히 다름
- One-hot encoding 또는 단어 집합 기반으로 사용할 때 유용

```
def jaccard_similarity(set1, set2):
    return len(set1 & set2) / len(set1 | set2)

# 예시
jaccard_similarity(set(["밥", "국"]), set(["밥", "김치"]))
```