뉴스 키워드 분석기:

뉴스의 핵심 키워드 및 유사 키워드를 추출하여 시각화하는 인터랙티브 도구 개발

20200135 독일어문문화 임규빈
20241348 간호 이가영
20230056 영어영문 김수민

[목차]

[주제] 뉴스 키워드 분석기:

뉴스의 핵심 키워드 및 유사 키워드를 추출하여 시각화하는
인터랙티브 도구 개발

- 텍스트 분석
- 단어 중심의 데이터 처리
- 자연어 의미 파악

- 특정 뉴스 주제 or 기사 링크 입력
- 관련 뉴스 기사 자동 수집
- 핵심 키워드 및 유사 키워드 추출
- 그래프로 시각화

[배경]

단어 암기 + 문법 중심의 영어 학습

: 실생활 영어 사용 연결에 어려움

: 자기주도적 학습 동기 약화

실사용 영어 데이터 기반의

맥락 중심 학습

뉴스는 다양한 어휘와 표현을

접할 수 있는 유용한 자료

어휘의 의미와 쓰임, 관계를

함께 학습할 수 있는 도구 개발

[과정]

1. 뉴스 데이터 자동 수집

- 입력된 주제 or 링크 관련 뉴스를 실시간 수집

2. 자연어 기반 키워드 분석

- spaCy **spaCy**: 단어의 품사를 식별, 사전적 기본형으로 변환
- Google BERT **BERT**: 좌우 문맥 참고해 단어 이해, 주제 도출, 유사 단어 탐색

3. 추출된 결과의 시각화

- 입력된 단어의 유사 키워드를 그래프로

4. Gradio UI를 통한 사용자 인터랙션

- 기사 분석 실행
- 뉴스의 개수 조절
- 키워드가 포함 문장과 유사도 도출
- 빈도순으로 나열된 키워드 그래프 확인

[목표]

1. 주제 중심 어휘 습득

- 주제 관련 어휘를 반복적으로 익히며, 문맥 안에서 어휘를 이해함

2. 유의어 및 관련 표현 학습

- 의미가 유사한 다양한 형태의 영어 표현들을 자연스레 익힘

3. 실생활 사용 문서 기반 학습

- 실제 영어 데이터를 기반으로 언어의 사용 맥락 속에서 단어와 표현을 이해함

4. 자기주도적 학습 환경 조성

- 스스로 주제와 기사 개수를 설정함으로써, 자발적이고 지속적인 탐구의 기반이 됨

셀 #1: 패키지 설치

```
!pip install --upgrade googlesearch-python bs4 requests nltk spacy matplotlib
!python -m spacy download en_core_web_sm
```

프로젝트에 필요한 외부 라이브러리 설치

Googlesearch-python
파이썬 코드에서 구글 검색 결과를 간편하게 불러오기 위한 모듈
주제를 넣으면 관련 기사 URL을 가져와 리스트업할 수 있음 ⇒ 크롤링해서 저장하기

bs4(BeautifulSoup)
받아온 HTML (웹사이트 형식) 에서 본문 텍스트만 추출

requests
HTTP 요청을 보내 HTML(또는 기타 응답)을 받아오기: 호출하면 해당 URL의 서버에 요청을 보내 뭔가 응답을 얻어옴.

nltk
불용어 제거, WordNet 동의어 탐색, 토큰화 가능하도록 함

spaCy
문장 분리·토큰화·품사 분류·문장 임베딩·유사도 계산 *소형 버전.

matplotlib
단어 빈도 데이터를 시각화해 막대그래프(Bar chart)로 출력

셀#2: 라이브러리 임포트, SpaCy모델 다운로드

```
import os
import re
import requests
import nltk
import spacy
import pickle
import matplotlib.pyplot as plt

from googlesearch       import search
from urllib.robotparser import RobotFileParser
from urllib.parse       import urlparse
from bs4                import BeautifulSoup
from nltk.corpus        import wordnet


# NLTK 및 SpaCy 초기화

nltk.download('punkt')
nltk.download('wordnet')
nlp = spacy.load("en_core_web_sm")

nltk.download('stopwords')
```

표준 라이브러리 import

: 파일 입출력, 표현식, 그래프 출력에 필요한 모듈

외부 라이브러리 import

: 웹페이지 요청, HTML 파싱, 구글 검색 결과 불러오기, 웹 크롤링 가능여부 확인, WordNet으로부터 동의어 추출, 자연어 처리

Nltk 데이터 다운로드: 문장단위분할, 동의어, 불용어 확인

SpaCy모델 로드: 문장 분리, 토큰화, 품사 태깅 수행

셀 #3: URL 획득 & 크롤링 허용 여부 (1)

```
def fetch_urls(query: str, num_results: int = 10) -> list:
    """
    - 구글 검색을 이용해 query에 해당하는 URL 리스트를 가져온다.
    - 실패 시 빈 리스트 반환.
    """
    try:
        return list(search(query, num_results=num_results, lang="en"))
    except Exception as e:
        print(f"[fetch_urls] Error: {e}")
        return []
```

fetch_urls: 구글 검색어로 상위 N개 URL 리스트 반환

query (예: Climate Change)가 들어오면 구글 검색 실행
→검색 결과 상위 ('num_results')개의 URL리스트 반환
→`googlesearch.search()`를 호출하여 URL을 가져옴

셀 #3: URL 획득 & 크롤링 허용 여부(2)

```python
def is_crawlable(url: str) -> bool:
    """
    - robots.txt 확인 후, 해당 URL에 접근해도 되는지 체크.
    - 실패하거나 robots.txt가 없으면 False 반환.
    """
    try:
        parsed = urlparse(url)
        robots_url = f"{parsed.scheme}://{parsed.netloc}/robots.txt"
        rp = RobotFileParser()
        rp.set_url(robots_url)
        rp.read()
        return rp.can_fetch("*", url)
    except Exception as e:
        print(f"[is_crawlable] {url} -> Error: {e}")
        return False

def scrape_text_from_url(url: str) -> str or None:
    """
    - 요청 실패/raise_for_status() 오류 시 None 반환.
    - 성공 시, <p> 태그 안의 텍스트를 모아 하나의 문자열로 반환.
    """
    try:
        resp = requests.get(url, timeout=8)
        resp.raise_for_status()
        soup = BeautifulSoup(resp.text, "html.parser")
        paras = soup.find_all("p")
        text = "\n".join(p.get_text(strip=True) for p in paras)
        return text if text.strip() else None
    except Exception as e:
        print(f"[scrape_text_from_url] {url} -> Error: {e}")
        return None
```

is_crawlable: robots.txt 검사 후 크롤링 허용 여부 판단

→ urlparse로 URL의 도메인 추출해서 봇이 접근해도 되는 페이지인지 확인하고

→ 되면 True, 안되면 False를 return함.

scrape_text_from_url

→ BeautifulSoup로, <p>"태그에 있는 본문 텍스트만 긁어서 문자열로 합침

셀 #3: URL 획득 & 크롤링 허용 여부(3)

```
def crawl_and_save(query_or_urls, num_results: int = 10, save_dir: str = "corpus"):
    """
    1) query_or_urls:
      - 문자열 (예: "Climate Change") 면 구글 검색으로 URL 리스트 생성
      - 리스트 타입이면 URL 리스트를 그대로 사용
    2) num_results: 구글 검색 시 가져올 최대 URL 개수
    3) save_dir: 텍스트 파일(.txt) 과 Pickle(.pkl) 형태로 저장할 디렉토리
    ----------------------------------------
    - 각 URL에 대해 크롤링을 시도. 성공 시 순서대로 corpus/{idx}.txt 에 저장
    - 건너뛸 URL은 로그로 출력
    - 전체 저장된 파일(파일명 리스트 포함)을 corpus/index.pkl 에 pickle로 저장
    """
    # 저장 디렉토리 준비
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    # 1) URL 리스트 생성
    if isinstance(query_or_urls, str):
        urls = fetch_urls(query_or_urls, num_results)
    elif isinstance(query_or_urls, list):
        urls = query_or_urls[:num_results]
    else:
        raise ValueError("crawl_and_save: query_or_urls 은 str 또는 list 여야 합니다.")

    saved_files = []
    idx = 1
```

crawl_and_save

→Sources가 문자열이라면 URL리스트를 만들고, 리스트형이면 그대로 사용

→구글 검색 , 상위 결과 반환

셀 #3: URL 획득 & 크롤링 허용 여부(4)

```
for url in urls:
    if not is_crawlable(url):
        print(f"[crawl_and_save] Skipping (robots.txt disallow): {url}")
        continue

    txt = scrape_text_from_url(url)
    if txt is None:
        print(f"[crawl_and_save] Skipping (empty or request error): {url}")
        continue

    # 2) 텍스트 파일로 저장
    filename = os.path.join(save_dir, f"{idx:03d}.txt")
    with open(filename, "w", encoding="utf-8") as f:
        f.write(txt)

    saved_files.append(filename)
    print(f"[crawl_and_save] Saved: {filename}")
    idx += 1

# 3) pickle로 인덱스(파일명 리스트) 저장
idx_path = os.path.join(save_dir, "index.pkl")
with open(idx_path, "wb") as pf:
    pickle.dump(saved_files, pf)
print(f"[crawl_and_save] Index saved → {idx_path}")
return saved_files
```

→크롤링 가능한 내용에 한해 본문을 가져옴.

→Save_dir 폴더 안에 001.txt, 002.txt, ...
형태로 텍스트를 저장 (이후 분석에 사용할 수 있도록 저장해둠)

셀 #4: 동의어 추출 및 문장 필터링 함수 정의 (1)

```python
def get_synonyms(word: str) -> set:
    """
    - WordNet synsets 기반 동의어(lemma) 집합 생성
    - 실패 시 빈 집합 반환
    """
    out = set()
    try:
        for syn in wordnet.synsets(word):
            for lm in syn.lemmas():
                out.add(lm.name().replace("_", " "))
    except Exception as e:
        print(f"[get_synonyms] {word} -> Error: {e}")
    return out
```

get_synonyms(word)

NLTK의 WordNet 에 등록된 동의어 사전을 기반으로 입력한 word의 동의어 집합 (lemma)을 구하고, return.

예: caused → induce, lead 등...

셀 #4: 동의어 추출 및 문장 필터링 함수 정의 (2)

```python
def filter_corpus(corpus_dir: str, expr: str) -> dict:
    """
    - corpus_dir 아래 index.pkl을 읽고, 각 .txt 파일을 문장 단위로 분리
    - SpaCy로 문장 분리 후, expr 및 그 동의어를 포함한 문장만 추출
    - {파일경로: [매칭 문장 리스트]} 형태로 반환
    """
    idx_path = os.path.join(corpus_dir, "index.pkl")
    with open(idx_path, "rb") as pf:
        saved_files = pickle.load(pf)

    expr_lower = expr.lower()
    terms = {expr_lower} | get_synonyms(expr_lower)
    result = {}

    for filepath in saved_files:
        try:
            with open(filepath, "r", encoding="utf-8") as f:
                raw_text = f.read()
        except Exception as e:
            print(f"[filter_corpus] Cannot read {filepath}: {e}")
            continue

        doc = nlp(raw_text)
        sentences = [sent.text.strip() for sent in doc.sents]
        matched = [s for s in sentences
                   if any(re.search(rf"\b{re.escape(t)}\b", s, re.IGNORECASE)
                          for t in terms)]
        if matched:
            result[filepath] = matched

    return result
```

filter_corpus(corpus_dir, expr)

→ 앞서 구한 동의어 집합을 terms변수에 소문자로 통합해둔다.

→ 저장된 txt파일을 읽고, SpaCy를 통해 문장 단위로 분리

→ 입력한 단어 또는 동의어가 포함되어 있는 문장만 남겨서 **{파일경로: [문장1, 문장2, ...]}** 형태로 반환

*크롤링한 문서 중 사용자가 찾고자 하는 타겟 표현이 포함된 문장만 모아 보기 위함.

셀 #5: 단어 빈도 계산, 시각화 함수 정의 (1)

```python
from nltk.corpus import stopwords

# 위에서 파일경로:문장리스트를 입력받음

def count_term_frequencies(filtered_dict: dict) -> dict:
    """
    - filtered_dict: {파일경로: [매칭 문장 리스트]}
    - NLTK의 영어 불용어 전체 목록으로 단어를 걸러냄
    - 문장 내 모든 단어를 소문자로 추출한 뒤, stopwords에 없는 단어만 카운트
    - 결과: {"term1": count1, "term2": count2, ...}
    """
    # NLTK 영어 불용어 집합
    stop_words = set(stopwords.words('english'))

    freq = {}
    for filepath, sent_list in filtered_dict.items():
        for sentence in sent_list:
            # 단어만 추출 (소문자)
            words = re.findall(r"\b\w+\b", sentence.lower())
            for w in words:
                if w in stop_words:
                    continue
                freq[w] = freq.get(w, 0) + 1

    return freq
```

count_term_frequencies(filtered_dict)

→ 위에서 {파일경로: [문장 리스트]}를 입력받음

→ 문장을 단어 단위로 쪼갬

→숫자나 특수문자 없이 영문 소문자로 바꿈

→ NLTK의 불용어 목록에 포함되지 않는 단어만 카운트해서 빈도 딕셔너리를 만듦

셀 #5: 단어 빈도 계산, 시각화 함수 정의 (2)

```python
def plot_frequencies(freq_dict: dict, top_n: int = 20):
    """
    - freq_dict: {"term": count, ...}
    - 상위 top_n 개를 막대그래프로 출력 (NLTK 불용어 제외됨)
    """
    if not freq_dict:
        print("[plot_frequencies] 매칭된 용어가 없습니다.")
        return

    # 상위 top_n 추출
    items = sorted(freq_dict.items(), key=lambda x: x[1], reverse=True)[:top_n]
    terms, counts = zip(*items)

    plt.figure(figsize=(8, 4))
    plt.barh(list(terms)[::-1], list(counts)[::-1])  # 상위가 위로 오도록 뒤집기
    plt.xlabel("Frequency")
    plt.title(f"Top {top_n} term frequencies (stopwords 제외)")
    plt.tight_layout()
    plt.show()
```

plot_frequencies(freq_dict, top_n)

→위에서 만든 빈도수 단어 딕셔너리를 빈도수 (높은 순으로) 정렬해 상위 top_n개 (기본 20개) 까지 시각화함

*matplotlib.pyplot으로 가로 막대그래프(barh)를 그림: 빈도가 높은 단어가 위로 오도록 역순으로 배치([::-1]).

셀 #6: 예시 실행 (1) 크롤링

```
# 예시 1: "Climate Change" 라는 키워드로 구글 검색 후 상위 5개 URL을 크롤링해서 저장
saved_files = crawl_and_save("Climate Change", num_results=5, save_dir="corpus")

# 예시 2: 이미 알고 있는 사이트 URL 리스트를 직접 입력해서 크롤링
# urls = ["https://www.bbc.com/news/science-environment-56837908",
#         "https://en.wikipedia.org/wiki/Climate_change"]
# saved_files = crawl_and_save(urls, num_results=5, save_dir="corpus")

print("▶ 크롤링 및 저장 완료된 파일들:")
print("\n".join(saved_files))


# 결과:

# corpus/001.txt, corpus/002.txt, … 식으로 크롤된 본문이 저장됩니다.

# corpus/index.pkl 에는 이 파일들의 경로 리스트가 pickle로 저장됩니다.
```

crawl_and_save("Climate Change", num_results=5, save_dir="corpus")

→“Climate Change”로 구글 검색

→ 상위 5개 URL 크롤링,

→결과 텍스트 저장 (001.txt, 002.txt…)

셀 #6: 예시 실행 (2) 크롤링

```
# 예시 1: "Climate Change" 라는 키워드로 구글 검색 후 상위 5개 URL을 크롤링해서 저장
saved_files = crawl_and_save("Climate Change", num_results=5, save_dir="corpus")

# 예시 2: 이미 알고 있는 사이트 URL 리스트를 직접 입력해서 크롤링
# urls = ["https://www.bbc.com/news/science-environment-56837908",
#         "https://en.wikipedia.org/wiki/Climate_change"]
# saved_files = crawl_and_save(urls, num_results=5, save_dir="corpus")

print("▶ 크롤링 및 저장 완료된 파일들:")
print("\n".join(saved_files))


# 결과:

# corpus/001.txt, corpus/002.txt, … 식으로 크롤된 본문이 저장됩니다.

# corpus/index.pkl 에는 이 파일들의 경로 리스트가 pickle로 저장됩니다.
```

crawl_and_save("Climate Change", num_results=5, save_dir="corpus")

→“Climate Change”로 구글 검색

→ 상위 5개 URL 크롤링,

→결과 텍스트 저장 (001.txt, 002.txt…)

셀 #7: 예시 실행 (3) 필터링, 빈도계산, 시각화

```
# 1) 필터링할 표현(또는 단어)을 입력
expr = "caused"  # 예시

# 2) corpus 디렉토리에서 매칭 문장 뽑기
filtered = filter_corpus("corpus", expr)
print("▶ 매칭된 문장들 (파일별):")
for fpath, sents in filtered.items():
    print(f"\n--- {fpath} ---")
    for s in sents:
        print(" -", s)

# 3) 빈도 계산
freqs = count_term_frequencies(filtered)
print("\n▶ 각 단어 등장 빈도 TOP 10:")
for term, cnt in sorted(freqs.items(), key=lambda x: x[1], reverse=True)[:10]:
    print(f"{term}: {cnt}")


# 4) 빈도 시각화 (상위 20개)
plot_frequencies(freqs, top_n=20)
```

단어/구 지정

filter_corpus

→ 앞서 크롤링해 둔 txt 파일을 읽고

→각 문장 문장을 SpaCy로 분리

→ 타겟 표현이나 그 표현의 동의어가 포함된 문장만 골라 저장

→ 매칭된 문장을 파일별로 모아 출력

Count_term_frequencies(filtered)

불용어 제외 단어 딕셔너리를 빈도순으로 나열

Plot_frequencies

→ 상위 20개의 단어를 (20개가 기본) term:count 형태, 막대그래프로 시각화

셀 #8: Gradio 구현 (1) 필요 모델 로드

```
# 1) Gradio 설치 (Colab 환경)
!pip install --upgrade gradio

# Gradio 패키지를 최신 버전으로 설치하기

# 2) 필요한 라이브러리 임포트
import gradio as gr
import os
import pickle
import matplotlib.pyplot as plt
import re

# (셀 3~5에서 이미 정의된 함수들이 메모리에 로드되어 있다고 가정)
#   - 셀 3: fetch_urls, is_crawlable, scrape_text_from_url, search_and_scrape, crawl_and_save
#   - 셀 4: get_synonyms, filter_corpus
#   - 셀 5: count_term_frequencies, plot_frequencies

# SpaCy nlp 객체 (en_core_web_sm 또는 en_core_web_lg)
import spacy
nlp = spacy.load("en_core_web_sm")  # 벡터 모델을 원한다면 en_core_web_lg로 교체
```

Gradio 설치, 라이브러리 임포트

웹UI, 파일 읽는 라이브러리, 일치하는 단어 검색이나 동의어 매칭 검사에 쓰는 라이브러리 로드

이전 셀에서 만들어뒀던 크롤링, 필터링, 빈도 계산 함수가 이미 로드+실행되어 있다고 **가정함**.

SpaCy 모델 로드 : 토큰화·품사 태깅·문장 분리·벡터 임베딩(similarity 계산)

*소형 버전이므로 더 정교한 버전을 원한다면 en_core_web_lg로 교체할 수 있다

셀 #8: Gradio 구현 (2) – defining the pipeline

```
def gradio_pipeline(topic_or_urls: str, num_results: int, expr: str):
    """
    1) crawl_and_save: corpus 디렉토리 생성 및 텍스트 파일 저장
    2) filter_corpus: 저장된 말뭉치에서 표현/유사어 매칭 문장 추출
    3) 각 문장에 대해 입력 expr과의 최대 유사도 계산 → 유사도 내림차순 정렬
    4) count_term_frequencies: 매칭 문장에서 단어 빈도 계산
    5) plot_frequencies: matplotlib Figure 생성
    6) 크롤링 요약, (유사도 정렬된) 매칭문장, Figure 세 개를 반환
    """
```

Gradio_pipeline

Gradio 인터페이스에서 'submit'버튼을 클릭하면 벌어지는 일을 순서대로 설정 (크롤링 → 필터링 → 유사도 계산 → 빈도 계산 → 시각화)

(사용자가) 입력할 내용:

1. 검색, 크롤링할 주제 키워드 또는 URL 리스트
2. 검색, 크롤링할 최대 문서 수 설정 (20개)
3. 찾고싶은 단어, 표현

```
    # ── 1) 이전 corpus 초기화 & 새로 생성 ──
    corpus_dir = "corpus"
    if os.path.exists(corpus_dir):
        for fname in os.listdir(corpus_dir):
            os.remove(os.path.join(corpus_dir, fname))
        os.rmdir(corpus_dir)
```

corpus_dir = "corpus"

이전에 수행한 내용이 새로 실행할 때 남아 섞이지 않도록 매번 초기화함.

셀 #8: Gradio 구현 (3) 크롤링, 저장

```
# ── 2) 크롤링 & 저장 ──
if "," in topic_or_urls:
    urls = [u.strip() for u in topic_or_urls.split(",") if u.strip()]
    saved_files = crawl_and_save(urls, num_results, save_dir=corpus_dir)
else:
    saved_files = crawl_and_save(topic_or_urls, num_results, save_dir=corpus_dir)

# "크롤링 요약" 문자열 생성
if saved_files:
    summary_lines = [os.path.basename(fp) for fp in saved_files]
    crawl_summary = "저장된 말뭉치 파일:\n" + "\n".join(summary_lines)
else:
    crawl_summary = "🔍 크롤링된 파일이 없습니다."
```

Crawling_and_save

→ URL 목록에 바로 접근 또는 또는 키워드를 검색해서 URL 찾아옴

→ Robots.txt 확인해서 봇으로 크롤링 가능한지 확인

→ 가능하면 HTML을 가져와 본문만 뽑아내기

→ corpus/001.txt, corpus/002.txt... 와 같은 파일로 저장

```
# ── 3) 저장된 말뭉치 불러와서 "표현/유사어" 필터링 ──
try:
    filtered = filter_corpus(corpus_dir, expr)
except Exception as e:
    filtered = {}
    crawl_summary += f"\n\n🔴 필터링 에외 발생: {e}"
```

filter_corpus

→ 골라온 문장을 SpaCy를 사용해 토큰 단위로 나누기

→ 토큰끼리(단어, 표현끼리) 유사도 비교

→ 유사도 높은 순서대로 정리

셀 #8: Gradio 구현 (4) 유사도 계산, 정렬

```
# ── 4) 매칭된 문장들에 대해 유사도 계산 & 정렬 ──
matched_sentences = []  # (문장, 유사도) 튜플을 담을 리스트
expr_doc = nlp(expr.lower())

# filtered: {filepath: [문장 리스트]}
for fp, sentences in filtered.items():
    for sent in sentences:
        # SpaCy로 문장 전체를 파싱
        sent_doc = nlp(sent)
        # 문장 내에서 "expr"과 매칭된 단어만 뽑아냅니다.
        #  - 정규식으로 정확 단어 매칭
        matched_terms = []
        for token in sent_doc:
            if re.fullmatch(rf"\b{re.escape(expr.lower())}\b", token.text.lower()):
                matched_terms.append(token)
            else:
                # 동의어들( get_synonyms(expr) )과 일치하는 토큰도 추가
                for syn in get_synonyms(expr.lower()):
                    if re.fullmatch(rf"\b{re.escape(syn)}\b", token.text.lower()):
                        matched_terms.append(token)
                        break

        # matched_terms 중 하나라도 있다면, expr_doc 토큰과 매칭_term 토큰 간 유사도 계산
        if matched_terms:
            # expr_doc엔 통상 첫 번째 토큰만 있으면, expr_doc[0]으로 사용
            expr_token = expr_doc[0] if len(expr_doc) > 0 else None
            max_sim = 0.0
            if expr_token is not None:
                for m_t in matched_terms:
                    try:
                        sim = expr_token.similarity(m_t)
                    except:
                        sim = 0.0
                    if sim > max_sim:
                        max_sim = sim
            matched_sentences.append((sent, max_sim))
```

expr_doc 사용자가 입력한 내용을 기준으로 유사도를 계산

각 문장을 나눠서 sent_doc 모음을 만듦

→ 사용자 입력 단어와 정확히 일치하거나 그 동의어와 일치하는 표현만 찾아서 matched_terms에 보관

→ matched_terms 내의 유사도를 계산해서 가장 높은 값을 결정

셀 #8: Gradio 구현 (4) 유사도 계산, 정렬

```
# 유사도 순 내림차순으로 정렬 (유사도가 같다면 문장 순서 유지)
matched_sentences.sort(key=lambda x: x[1], reverse=True)
```

→ matched_terms 내의 유사도를 계산해서 가장 높은 값을 결정한 후, 유사도 점수를 기준으로 내림차순 정렬

```
# “매칭된 문장들” 문자열 생성 (유사도 표시: 소수점 3자리)
if matched_sentences:
    lines = []
    for sent, sim in matched_sentences:
        lines.append(f"{sent}    (유사도: {sim:.3f})")
    match_summary = "\n".join(lines)
else:
    match_summary = "❗ 매칭된 문장이 없습니다."
```

→ 정렬된 결과를 유사도 점수와 함께 긴 문자열로 만듦

*결과가 하나도 없으면 “매칭된 문장이 없습니다” 표시

셀 #8: Gradio 구현 (5) 단어 빈도 계산, 그래프로 시각화

```
# ── 5) 단어 빈도 계산 & 그래프 그리기 ──
try:
    # filtered 딕셔너리만 넘기면, 빈도 계산할 때는 UDF filtering에서 걸러진 문장들을 사용
    freqs = count_term_frequencies(filtered)

    # Figure 객체 생성
    fig = plt.figure(figsize=(6, 4))
    if freqs:
        items = sorted(freqs.items(), key=lambda x: x[1], reverse=True)[:20]
        terms, counts = zip(*items)
        plt.barh(list(terms)[::-1], list(counts)[::-1])
        plt.xlabel("Frequency")
        plt.title("Top Term Frequencies (stopwords excluded)")
        plt.tight_layout()
        plt.close()
    else:
        plt.text(0.5, 0.5, "빈도 데이터가 없습니다", ha="center", va="center")
        plt.axis("off")
        plt.close()
except Exception as e:
    fig = plt.figure(figsize=(4, 3))
    plt.text(0.5, 0.5, f"Plot Error:\n{e}", ha="center", va="center")
    plt.axis("off")
    plt.close()

return crawl_summary, match_summary, fig
```

→ 필터링된 문장들을 대상으로, 단어 등장 빈도를 세어 정리.

→ 상위 20개 단어를 빈도순으로 막대그래프 그리기

*그리는 도중 에러가 나면 에러 표시

셀 #8: Gradio 구현 (6) 인터페이스 정의

```python
# ── 6) Gradio 인터페이스 정의 및 실행 ──
iface = gr.Interface(
    fn=gradio_pipeline,
    inputs=[
        gr.Textbox(
            label="🔎 주제 또는 URL 리스트",
            placeholder="예) Climate Change  OR  https://site1.com, https://site2.com"
        ),
        gr.Slider(1, 20, step=1, value=5, label="검색/크롤링 URL 개수"),
        gr.Textbox(label="🖍️ 찾을 표현/단어", placeholder="예) caused"),
    ],
    outputs=[
        gr.Textbox(label="📋 크롤링 요약(저장된 파일 목록)", lines=5),
        gr.Textbox(label="✅ 유사도 순 매칭된 문장들", lines=10),
        gr.Plot(label="📊 단어 빈도 그래프"),
    ],
    title="Robust Topic Scraper + Similarity-Sorted Filter + Frequency",
    description=(
        "1) 주제 키워드 또는 쉼표로 구분된 URL 리스트를 입력\n"
        "2) 최대 N개 URL을 크롤링해 말뭉치를 저장\n"
        "3) 입력한 표현 및 동의어가 포함된 문장을 SpaCy 유사도로 정렬하여 출력\n"
        "4) 문장에서 등장한 단어 빈도를 그래프로 시각화\n\n"
        "→ 모든 단계는 예외 처리를 거쳐 안정적으로 동작합니다."
    ),
    allow_flagging="never"
)

iface.launch(share=True, debug=True)
```

Fn

Submit, clear 등 버튼 눌렀을 때 실행할 함수 지정

Inputs

1. 텍스트박스: 주제, URL
2. 슬라이더: 크롤링 개수
3. 텍스트박스: 찾을 단어 (타겟 표현)

Outputs

1. 텍스트박스: 크롤링 요약
2. 텍스트박스: 매칭 문장
3. 그래프: 단어 빈도

*UI제목, 사용법 비활성화

share=True: 외부에서도 접속 가능한 링크 **debug=True:** 콘솔에 상세 에러 로그 출력

```python
# 예시: 자동 테스트 실행
example_topic = "Peru earthquake"
example_expressions = "damage"

# 직접 함수 호출 (Gradio 외 테스트)
result_text, result_plot = analyze_news(example_topic, example_expressions)

# 결과 출력
print(result_text)
if result_plot:
    result_plot.show()
```

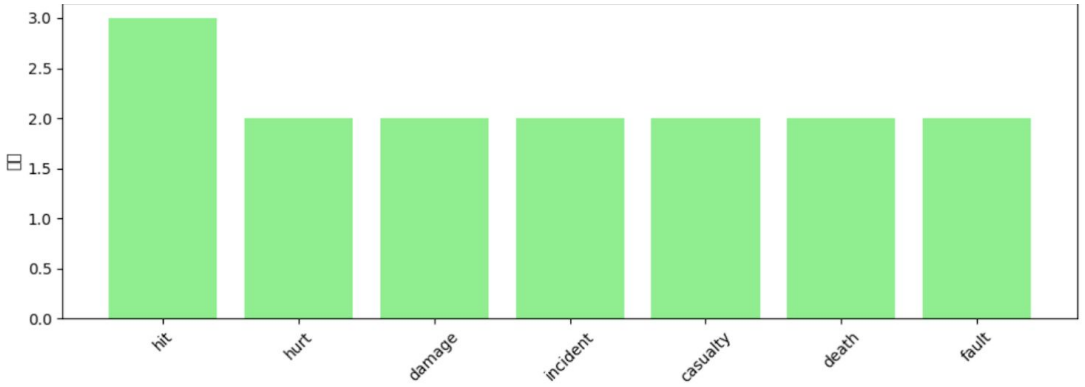
```
🔍 주제: Peru earthquake

📌 'damage' 와 유사한 단어:
 - hit (때리다) | 유사도: 0.53, 빈도: 3
 - hurt (아프다) | 유사도: 0.61, 빈도: 2
 - damage (손상) | 유사도: 1.00, 빈도: 2
 - incident (사건) | 유사도: 0.42, 빈도: 2
 - casualty (사상자 수) | 유사도: 0.45, 빈도: 2
 - death (죽음) | 유사도: 0.47, 빈도: 2
 - fault (잘못) | 유사도: 0.49, 빈도: 2
```

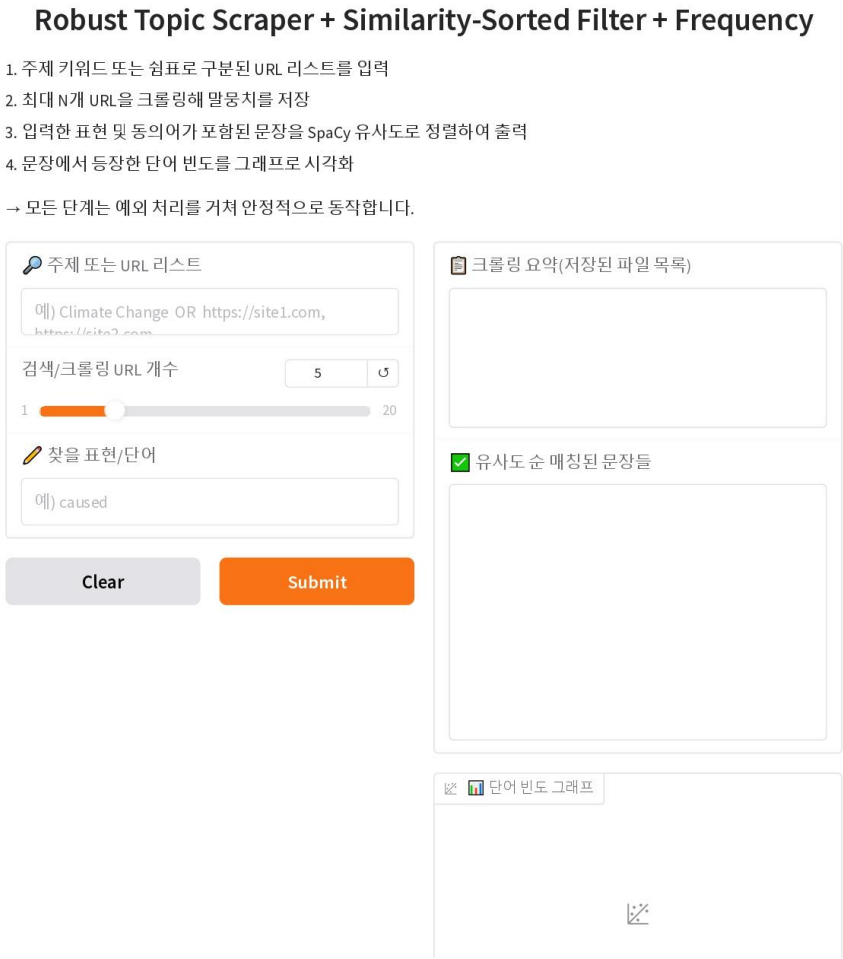
[과정 중]

최종적으로 gradio에는 구현하지 못했으나 10개의 기사를 수집했을 때,

입력한 단어와 비슷한 의미를 가진 단어의 유사도와 빈도를 번역과 함께 제공하기도 함



[인터페이스]


Robust Topic Scraper + Similarity-Sorted Filter + Frequency
1. 주제 키워드 또는 쉼표로 구분된 URL 리스트를 입력
2. 최대 N개 URL을 크롤링해 말뭉치를 저장
3. 입력한 표현 및 동의어가 포함된 문장을 SpaCy 유사도로 정렬하여 출력
4. 문장에서 등장한 단어 빈도를 그래프로 시각화
→ 모든 단계는 예외 처리를 거쳐 안정적으로 동작합니다.
주제 또는 URL 리스트
예) Climate Change OR https://site1.com,
검색/크롤링 URL 개수
5
1
20
찾을 표현/단어
예) caused
Clear
Submit
크롤링 요약(저장된 파일 목록)
유사도 순 매칭된 문장들
단어 빈도 그래프

🔎 주제 또는 URL 리스트

Gender equality

검색/크롤링 URL 개수

8 ↺

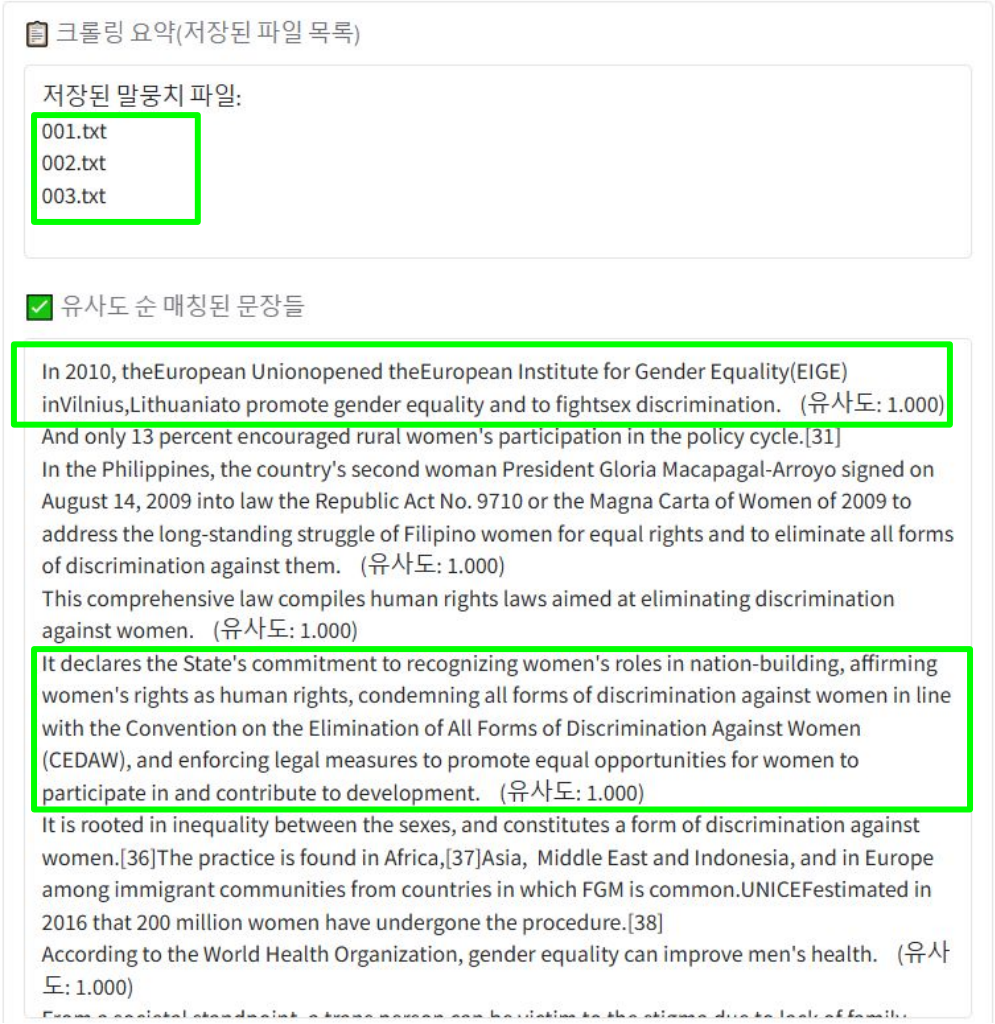
1 20

🖉 찾을 표현/단어

discrimination

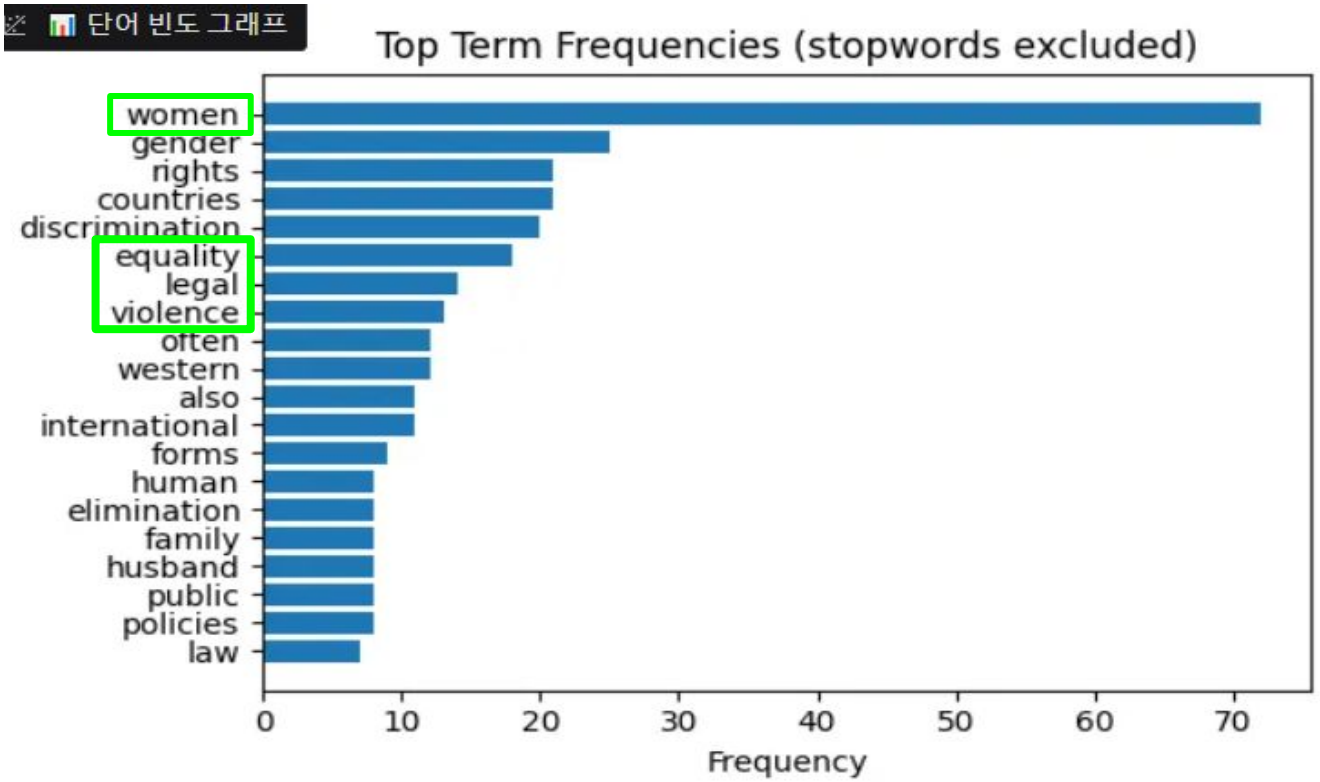
Clear

Submit

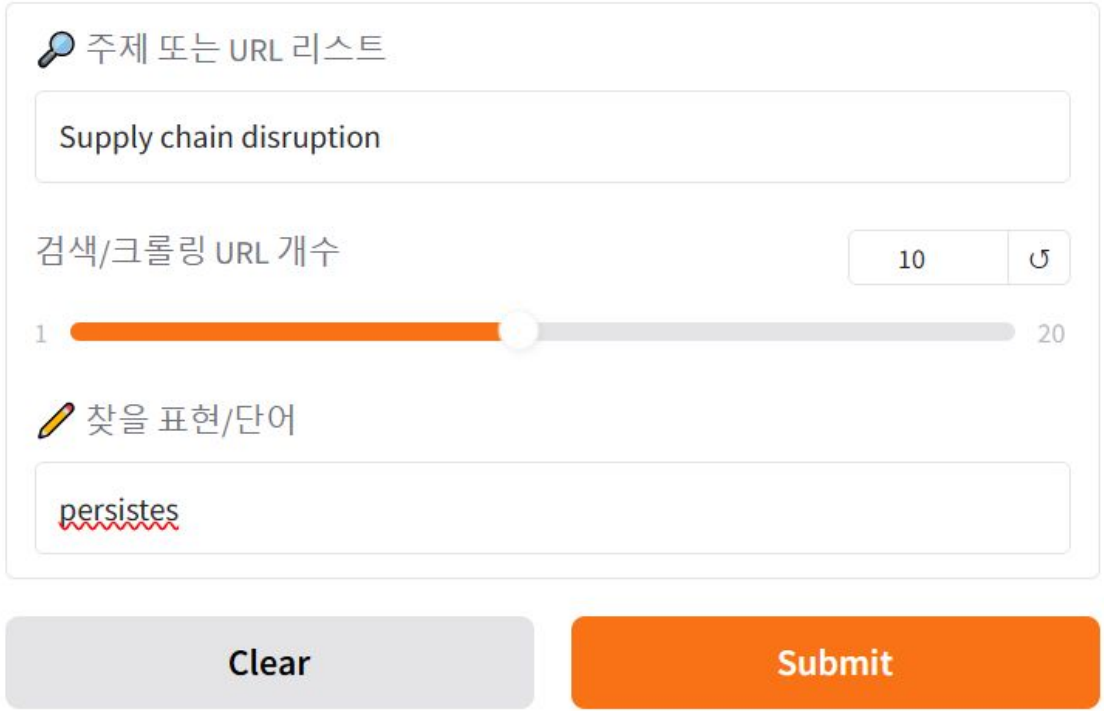


1. In 2010, the European Union opened the European Institute for Gender Equality(EIGE) in Vilnius, Lithuaniato promote gender equality and to fight sex discrimination. (유사도: 1.000)
2. It declares the State's commitment to recognizing women's roles in nation-building, affirming women's rights as human rights, condemning all forms of discrimination against women in line with the Convention on the Elimination of All Forms of Discrimination Against Women (CEDAW) ….. (유사도: 1.000)

Gender equality - discrimination



Supply chain disruption – persists



✅ 유사도 순 매칭된 문장들

With investors seeking information on a company's ESG performance, employees wanting to work for companies with sustainability built into their mission statements, increased customer expectations for sustainability and increasing regulation from various countries, sustainable supply chain practices no doubt are here to stay. (유사도: 0.153)

The effects brought about by the pandemic in the economy, various industries, government bodies, and societies are continually materializing — so it won't be far-fetched to say that they are here to stay. (유사도: 0.152)

To stay ahead, decision-makers need a complete and data-driven view of the market. (유사도: 0.067)

To stay ahead of future supply chain challenges, companies must continue their ongoing efforts to build resilience and take new actions to address blind spots in their systems, processes, and capabilities. (유사도: 0.055)

In no particular order, here are the top ten risks your company needs to monitor this year, and some steps you can take to stay ahead of them. (유사도: 0.003)

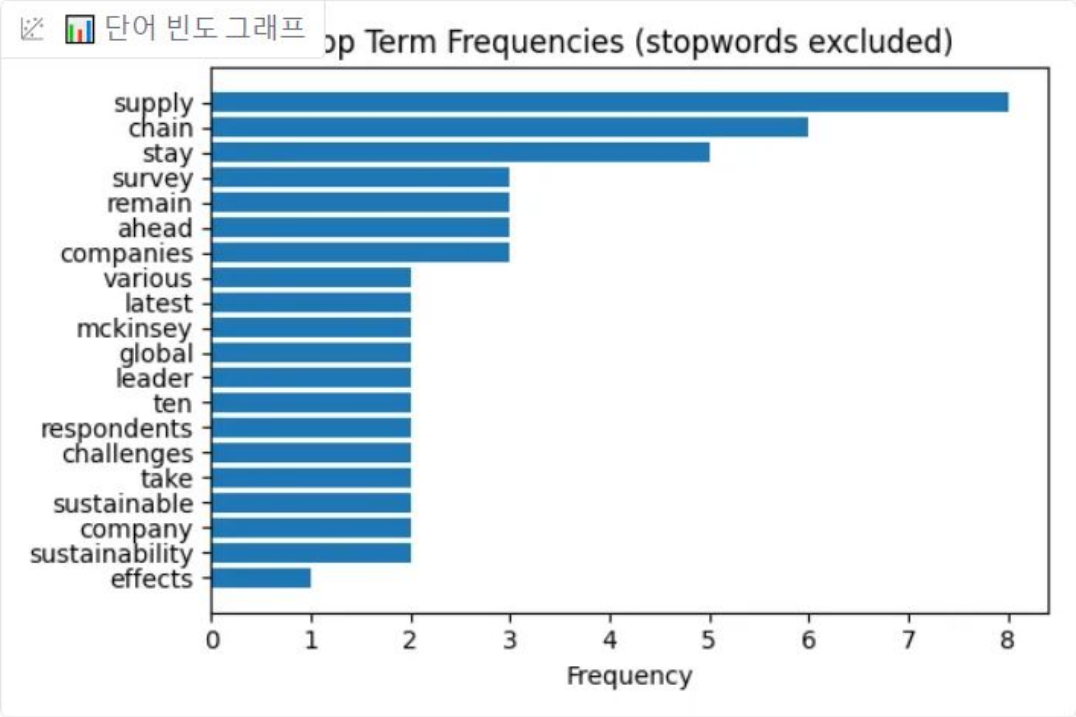
The latest McKinsey Global Supply Chain Leader Survey suggests that

1. With investors seeking information on a company's ESG performance, employees wanting to work for companies with sustainability built into their mission statements, increased customer expectations for sustainability and increasing regulation from various countries, sustainable supply chain practices no doubt are **here to stay.**

2. The effects brought about by the pandemic in the economy, various industries, government bodies, and societies are **continually materializing** — so it won't be far-fetched to say that they are **here to stay**.

3. Survey respondents also **remain** concerned that their senior management teams have a limited knowledge of supply chain issues.

4. While companies have made strides in strengthening their supply chains, the latest McKinsey Global Supply Chain Leader Survey shows that substantial vulnerabilities **remain**.



공급망 혼란, 즉 원자재, 부품, 완제품 등이 생산지에서 소비자에게 전달되는 과정에서 발생하는 지연, 차질, 중단 등의 문제와 관련된 주제와 연관된 단어가 빈도 높게 등장

[결과]

“뉴스 키워드 분석기”의 기대와 효과

- 사용자가 관심있는 흥미로운 주제
- 실제 뉴스 기사의 **실생활 용어** 학습 효과
- 다양한 문맥, **어휘력 향상**
- **시각화** 된 그래프 제공

감사합니다!