

Data Science

#HW3



Text Mining, Document Vector Model
산업공학과 201911532 송용재 (2023.10)



Table of Contents



Combine Excel File

Text Mining

Document Vector Model

Conclusion

Preface

텍스트 마이닝 기법

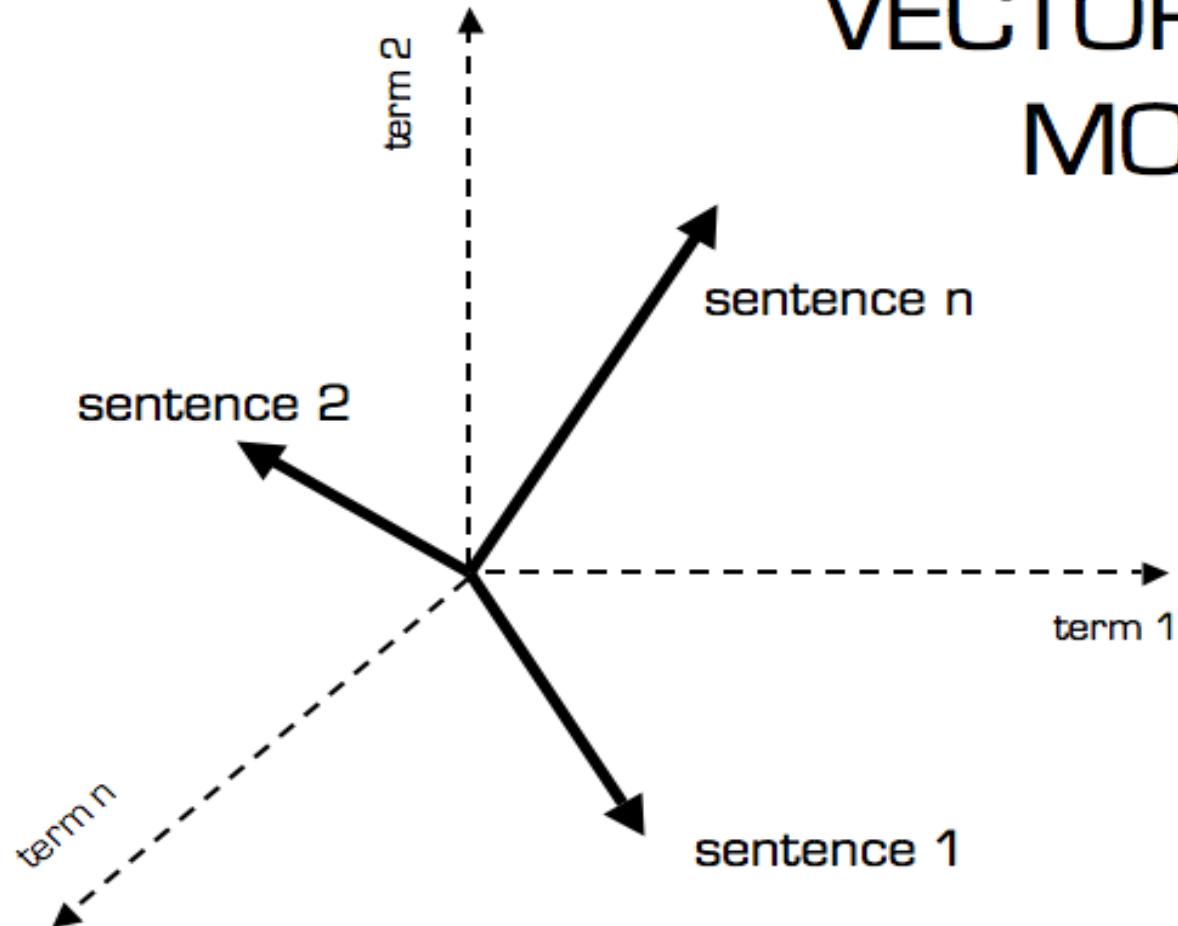
텍스트 마이닝 프로세스는 비정형 텍스트 데이터에서 정보를 추론하기 위한 몇 가지 활동으로 구성됩니다. 다양한 텍스트 마이닝 기법을 적용하기에 앞서 텍스트 전처리, 즉 텍스트 데이터를 정리하여 사용 가능한 형식으로 변환하는 프랙티스를 수행해야 합니다. 자연어 처리(NLP)의 핵심 요소 중 하나인 이 프랙티스에서는 대개 언어 식별, 토큰화(tokenization), 품사 태깅, 청킹, 구문 분석과 같은 기법을 활용하여 데이터를 분석에 적합한 형식으로 만듭니다. 텍스트 전처리를 완료하면, 텍스트 마이닝 알고리즘을 적용하여 데이터에서 인사이트를 발굴할 수 있습니다. 다음과 같은 텍스트 마이닝 기법이 자주 사용됩니다.

정보 검색

정보 검색(Information Retrieval, IR)은 사전 정의된 쿼리/구문 세트를 기반으로 연관 정보나 문서를 확보하는 기법입니다. IR 시스템에서는 각종 알고리즘을 활용하여 사용자 행동을 추적하고 관련 데이터를 식별합니다. 라이브러리 카탈로그 시스템, 그리고 Google과 같은 인기 검색 엔진에서 IR이 널리 활용됩니다. 특히 다음과 같은 IR 서브태스크(sub-task)가 자주 수행됩니다.

- **토큰화(Tokenization):** 긴 형태의 텍스트를 "토큰"이라는 문장 및 단어로 나누는 프로세스입니다. 그런 다음에 단어 가방 모형(Bag-of-Words, BoW)과 같은 모델을 적용하여 텍스트 클러스터링 및 문서 매칭 태스크를 수행합니다.
- **형태소 분석(Stemming):** 단어에서 접두사와 접미사를 분리하여 어근과 의미를 도출하는 프로세스입니다. 이 기법으로 인덱싱 파일의 크기를 줄여 정보 검색의 성능을 높일 수 있습니다.

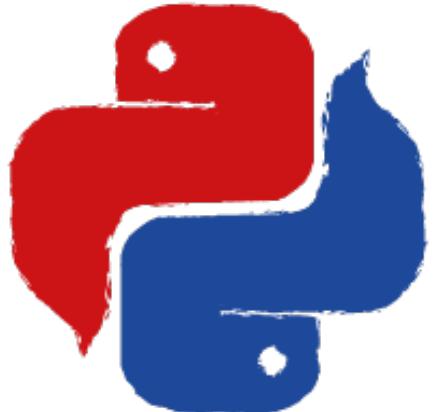
VECTOR SPACE MODEL



Preface

사용 라이브러리 : KoNLPy, Kkma

```
from konlpy.tag import *
import os
```



KoNLPy

꼬꼬마 세종 말뭉치 활용 시스템

국어와 관련된 연구를 수행할 때, 대량의 말뭉치를 필요로 하는 경우가 종종 있다. 세종 말뭉치는 질과 양 모든 면에서 매우 우수한 말뭉치이기는 하지만, 컴퓨터 프로그래밍 능력이 없는 사람은 이를 활용하기가 어렵다. 또한, 컴퓨터 프로그래밍에 익숙하다고 하더라도, 말뭉치의 구조를 파악하고 말뭉치를 처리할 수 있는 형태로 가공하는 과정이 필요하기 때문에 말뭉치를 활용하는데 어려움이 있다. 따라서 '꼬꼬마'팀에서는 세종 말뭉치를 다양한 용도로 활용할 수 있도록 1) 말뭉치를 구조화하여 데이터베이스에 저장하고, 2) 저장된 말뭉치로부터 다양한 통계 데이터를 생성하고, 3) 저장된 말뭉치 및 생성된 통계 정보를 다양한 방법으로 조회할 수 있는 시스템을 구현하였다.

꼬꼬마 세종 말뭉치 활용 시스템의 기능은 크게 말뭉치 통계 정보 조회, 말뭉치 검색, 그리고 한국어 쓰기 학습의 세 가지로 구분된다. 말뭉치 통계 정보 조회 기능은 구축된 말뭉치에서 품사, 형태소, 문어 및 구어 등의 다양한 기준에 의한 출현빈도를 추출하여 조회할 수 있게 하는 기능이며, 말뭉치 검색은 형태소를 기준으로 형태소가 쓰인 문장을 조회하고 이에 대한 품사 부착, 의미 분석, 구문 분석 결과를 확인하는 기능이다. 한국어 쓰기 학습 기능은 한국어를 공부하는 학생이나 교수자가 사용할 수 있는 기능으로, 단어가 포함된 용례나 양식에 따른 용례를 조회할 수 있다.

Preface

1. “아버지가방에들어가신다”

이 예시를 통해 띄어쓰기 알고리즘의 성능을 확인해볼 수 있습니다. 이상적인 경우, 이 예시에 대해서는 아버지 + 가방에 + 들어가신다 보다는 아버지가 + 방에 + 들어가신다로 해석하는 것이 더 바람직하겠지요.

Hannanum	Kkma	Komoran	Mecab	Twitter
아버지가방에 들어가 / N	아버지 / NNG	아버지가방에 들어가신다 / NNP	아버지 / NNG	아버지 / Noun
이 / J	가방 / NNG		가 / JKS	가방 / Noun
시ㄴ다 / E	에 / JKM		방 / NNG	에 / Josa
	들어가 / VW		에 / JKB	들어가신 / Verb
	시 / EPH		들어가 / VW	다 / Eomi
	ㄴ다 / EFN		신다 / EP+EC	

Preface

QUERY: “march health awareness”

D1: “the Health Observances for March ”

D2: “the Health oriented Calendar”

D3: “the Awareness News for March Awareness”

$D=3$; ; $IDF = \log\left(\frac{D}{df_j}\right)$ df_j = number of documents containing term j

		COUNTS $TF_{I,J}$						WEIGHTS, $W_I = F_{Q,J} * IDF_J$	WEIGHTS $W_I = F_{I,J} * IDF_J$		
TERMS	Q	D1	D2	D3	df_j	D/df_j	IDF_J	Q	D1	D2	D3
Health	1	1	1	0	2	$3/2=1.5$	0.1761	0.1761	0.1761	0.1761	0
Observances	0	1	0	0	1	$3/1=3$	0.4771	0	0.4771	0	0
For	0	1	0	1	2	$3/2=1.5$	0.1761	0	0.1761	0	0.0881
March	1	1	0	1	2	$3/2=1.5$	0.1761	0.1761	0.1761	0	0.0881
Awareness	1	0	0	2	1	$3/1=3$	0.4771	0.4771	0	0	0.4771
Oriented	0	0	1	0	1	$3/1=3$	0.4771	0	0	0.4771	0
Calendar	0	0	1	0	1	$3/1=3$	0.4771	0	0	0.4771	0
News	0	0	0	1	1	$3/1=3$	0.4771	0	0	0	0.2285
the	0	1	1	1	3	$3/3=1$	0	0	0	0	0

Combine Excel File

파일 디렉토리의 존재하는 모든 .xlsx를 지정된 파일에 병합

```
class class_document_tfidf():
    def __init__(self):
        self.conn, self.cur = open_db()
        self.news_article_excel_file = 'combined_article1.xlsx'
        self.pos_tagger = Kkma()

    def combine_excel_file(self):
        directory_path = './'

        excel_files = [file for file in os.listdir(directory_path) if file.endswith('.xlsx')]

        combined_df = pd.DataFrame()
        for file in excel_files:
            try:
                file_path = os.path.join(directory_path, file)
                df = pd.read_excel(file_path)
                df = df[['url', 'title', 'content']]
                combined_df = pd.concat([combined_df, df], ignore_index=True)
            except Exception as e:
                print(file_path)
                print(e)
                continue

        combined_df.to_excel(self.news_article_excel_file, index=False)
```

Combine Excel File

병합이 된 모습



Text Mining

Text Mining

```
def 'import_news_article'
```

```
def import_news_article(self):
    drop_sql = """ drop table if exists news_article1;"""
    self.cur.execute(drop_sql)
    self.conn.commit()

    create_sql = """
        drop table if exists news_article1;

        CREATE TABLE news_article1 (
            id int auto_increment primary key,
            url varchar(500),
            title varchar(500),
            content TEXT,
            enter_date datetime default now()
        );
    """

    self.cur.execute(create_sql)
    self.conn.commit()

    file_name = self.news_article_excel_file
    news_article_data = pd.read_excel(file_name)
```

```
rows = []

insert_sql = """insert into news_article1(url, title, content)
               values(%s,%s,%s);"""

for _, t in news_article_data.iterrows():
    t = tuple(t)
    try:
        self.cur.execute(insert_sql, t)
    except:
        continue
    rows.append(tuple(t))

self.cur.executemany(insert_sql, rows)
self.conn.commit()
print("table created and data loaded")
```

table created and data loaded

doc_id=1

title res=['공식', '공식전', '전', '100', '100번짜리']

Table1 : News_Article1

news_article1

Don't Limit

1 • SELECT * FROM DS2023.news_article1;

Result Grid Filter Rows: Search Export/Import: Fetch rows:

id	url	title	content	enter_date
1423	https://n.news.naver.com/mnews/article/008/000...	[다도] 차기 유행여하하자 16의 경쟁 허이·조준하·소벽...	유행곡을 대표하는 차기 유행여하하자이 오는 16의 경...	2023-11-19 17:18:38
1424	https://view.asiae.co.kr/article/2023110092234...	전문인력 외국인 12.4% 불과...영주권 소득 학력 요건...	저출산·고령화 문제에 따른 경제활동인구 확충을 위해...	2023-11-19 17:18:38
1425	https://view.asiae.co.kr/article/2023110063046...	통신 3사 3분기 실적 들여다보니...수익성 절벽'에 몰렸다	대표적 수익성 지표 ARPU 지속 하락 SKT 3만원대 봉...	2023-11-19 17:18:38
1426	https://n.news.naver.com/article/005/000165111...	"커피에 수면제 타고 성폭행" 엠벌원서 호소하는 피해자	전 남자친구에게 성폭행을 당했다고 주장하는 여성이...	2023-11-19 17:18:38
1427	https://n.news.naver.com/article/008/000496000...	무작정 시포 던진 후 16년..."숨만 쉬어도 1000만원..."	"아무것도 하지 않아도 한달(현금흐름) 1000만원이 넘..."	2023-11-19 17:18:38
1428	https://n.news.naver.com/article/310/000011186...	"맞아도 되는 여성은 없다"...여대생들, '웃컷 여성폭행'...	20대 남성이 '여자가 짧은 걸 보니 페미니스트다. 페미...'...	2023-11-19 17:18:38
1429	https://n.news.naver.com/article/366/000094649...	리터당 0.3㏊ 모자라 稅 혜택 못 받는 카니발 하이브리드	리터당 14.3㏊ 나와야 하는데 14㏊ 측정	2023-11-19 17:18:38
1430	https://n.news.naver.com/article/655/000001425...	외연받는 '왕의 온천 수안보'..국내 첫 온천도시 지정 재...	<앵커> 전국 최초, 왕의 온천으로 불리는 수안보 온천...	2023-11-19 17:18:38
1431	https://n.news.naver.com/article/654/000005727...	[속보] 전철조 사기 피해 23명·28억으로 늘어..."피해자...	전철조(27)씨가 전 펜싱 국가대표 남현희(42)씨의 재...	2023-11-19 17:18:38
1432	https://n.news.naver.com/mnews/article/003/001...	한국 생산 결정한 폴스타, '중국차' 이미지 벗을까?	[서울=뉴시스] 안경무 기자 = 폴스타가 2025년 하반기...	2023-11-19 17:18:38
1433	https://n.news.naver.com/mnews/article/138/000...	요기요, '빼로데이' 기념 리방 진행 ... "1시간 내 배송"	[디지털데일리 이안나 기자] 배달앱 요기요는 매장 방문...	2023-11-19 17:18:38
1434	https://n.news.naver.com/mnews/article/057/000...	조선왕조실록, 110년 만에 고향 오대산으로 돌아온다	[앵커멘트]	2023-11-19 17:18:38
1435	https://n.news.naver.com/article/031/000078690...	귀가하는 여중생 따라가 성폭행하고 부모에 돈까지 뜯...	집으로 들어가던 여학생을 따라가 강간하고 부모를 협...	2023-11-19 17:18:38
1436	https://n.news.naver.com/mnews/article/015/000...	SK바이오팜, 올 4분기 흐자전환 전망...제2의 세노바...	SK바이오팜이 지난 1~3분기 계속해서 영업적자 폭을...	2023-11-19 17:18:38
1437	https://n.news.naver.com/mnews/article/003/001...	영화서나 보던 'AI 비서' 나왔다..."내 손비단이 화면으로"	후배인 옷에 물어 쓰는 'AI 팬' 공개... 699달러에 16일...	2023-11-19 17:18:38
1438	https://n.news.naver.com/mnews/article/023/000...	비트코인 18개월 만에 3만7000달러 돌파..."현물 ETF..."	가상화폐 대장주인 비트코인 가격이 지난밤 3만7000달...	2023-11-19 17:18:38
1439	https://n.news.naver.com/mnews/article/092/000...	[이기자의 게임펍] 게임사 3분기 실적, 신작에 엇갈려...	넥슨 크래프톤·위메이드·네오위즈 등 게임 부문 호실적....	2023-11-19 17:18:38
1440	https://n.news.naver.com/mnews/article/138/000...	오픈 AI "챗GPT '90분 미통' 원인은 디도스 공격"	생성형 인공지능(AI) 챗GPT, ©OpenAI [디지털데일...	2023-11-19 17:18:38
1441	https://n.news.naver.com/mnews/article/023/000...	美 중국 수출 규제 강화에...엔비디아, '중국용 반도체'...	미국 캘리포니아 산타클라라에 있는 엔비디아 본사. /로...	2023-11-19 17:18:38
1442	https://n.news.naver.com/mnews/article/018/000...	LG화학, 사상 첫 연매출 1조 달성이 남다른 까닭	제품 매출 비중 95%로 질적 성장 이뤄... 국내 5대 제약...	2023-11-19 17:18:38
1443	https://n.news.naver.com/mnews/article/138/000...	올해 3분기, 3N-2K→NKE로...넥슨 독주 속 크래프톤...	매출 텁 5 게임사, 2023년 3분기 실적 회복...글로벌 IP...	2023-11-19 17:18:38
1444	https://n.news.naver.com/mnews/article/092/000...	"애플의 생성 AI 기능, 아이폰16에만 독점 제공"	IT립스터 레베그너스·전망 애플이 한창 개발 중인 생성...	2023-11-19 17:18:38
1445	https://n.news.naver.com/mnews/article/092/000...	"삼성 갤럭시S3, 이렇게 나온다"	삼성전자의 차세대 스마트밴드 신제품 '갤럭시핏 3'의...	2023-11-19 17:18:38
1446	https://n.news.naver.com/mnews/article/092/000...	中 애국소비 덕쳤나...샤오미 신형 스마트폰 100만대...	스마트폰 수요 침체 회복 조짐... 4분기 반등 전망 중국...	2023-11-19 17:18:38
1447	https://n.news.naver.com/mnews/article/015/000...	셀트리온, 브라질 정부 물량 수주... 램시마 점유율 80%...	셀트리온헬스케어, 브라질 연방정부 입찰 3년 연속 수...	2023-11-19 17:18:38
1448	https://n.news.naver.com/mnews/article/092/000...	マイクロ, 삼성 SK에 HBM3E 도전장...내년 1분기 대...	대만 타이중 신규 공장 가동 시작... 내년 HBM 7억 달...	2023-11-19 17:18:38
1449	https://n.news.naver.com/mnews/article/277/000...	40만원대 갤럭시, KT가 단독 출시..."개인·영상 재생도...	오늘부터 온오프라인 매장서 판매 통신비 완화... 출고...	2023-11-19 17:18:38
1450	https://n.news.naver.com/mnews/article/138/000...	네이버·카카오, 논란 일으킨 인수 기업이 3분기 최대 매...	[디지털데일리 최민지 기자] 네이버와 카카오가 올해 3...	2023-11-19 17:18:38
NULL	NUL...	NUL...	NUL...	NUL...

news_article1 1

Apply

def 'Extract_Nouns', Table 2, 3

Title과 Contents의 'Noun'을 추출하고 DB화

```
self.pos_tagger = Kkma()
```

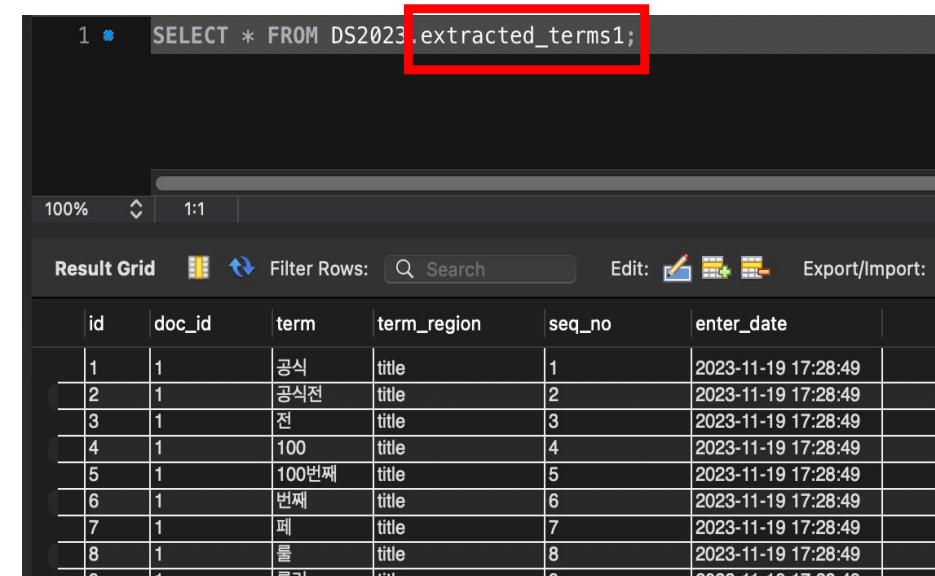
```
def extract_nouns(self):

    create_sql = """
        drop table if exists extracted_terms1;
        drop table if exists term_dict1;

        create table extracted_terms1 (
            id int auto_increment primary key,
            doc_id int,
            term varchar(30),
            term_region varchar(10),
            seq_no int,
            enter_date datetime default now() ,
            index(term)
        );

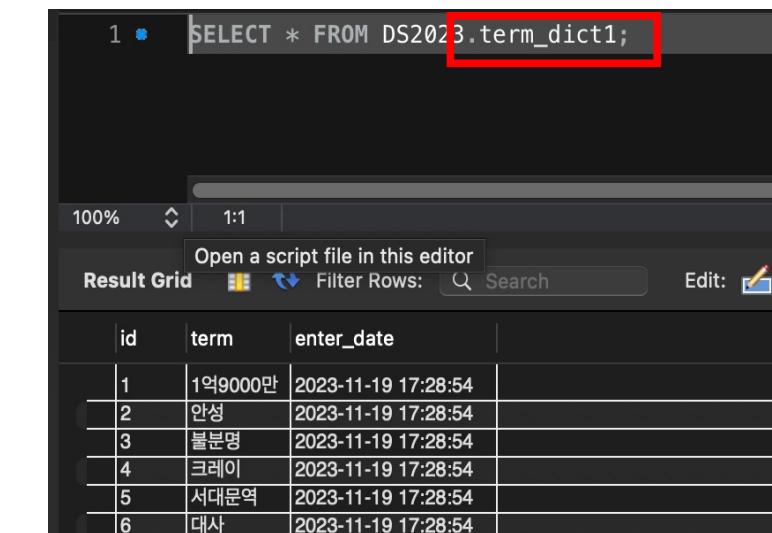
        create table term_dict1 (
            id int auto_increment primary key,
            term varchar(30),
            enter_date datetime default now(),
            index(term)
        );
    .....

    self.cur.execute(create_sql)
    self.conn.commit()
```



MySQL Workbench screenshot showing the results of the query:

id	doc_id	term	term_region	seq_no	enter_date
1	1	공식	title	1	2023-11-19 17:28:49
2	1	공식전	title	2	2023-11-19 17:28:49
3	1	전	title	3	2023-11-19 17:28:49
4	1	100	title	4	2023-11-19 17:28:49
5	1	100번째	title	5	2023-11-19 17:28:49
6	1	번째	title	6	2023-11-19 17:28:49
7	1	페	title	7	2023-11-19 17:28:49
8	1	룰	title	8	2023-11-19 17:28:49
9	1	룰고	title	9	2023-11-19 17:28:49



MySQL Workbench screenshot showing the results of the query:

id	term	enter_date
1	1억9000만	2023-11-19 17:28:54
2	안성	2023-11-19 17:28:54
3	불분명	2023-11-19 17:28:54
4	크레이	2023-11-19 17:28:54
5	서대문역	2023-11-19 17:28:54
6	대사	2023-11-19 17:28:54

def 'Extract_Nouns'

총 1450개의 doc의 'title과 content의 res'가 load 되었다.

```
sql = """select * from news_article1;"""
self.cur.execute(sql)

r = self.cur.fetchone()

noun_terms = set()
rows = []

while r:
    print(f"doc_id={r['id']}")
    i = 0
    title_res = self.pos_tagger.nouns(r['title'])
    content_res = self.pos_tagger.nouns(r['content'])

    title_rows = [ (r['id'], t, 'title', i+1) for i, t in enumerate(title_res) ]
    content_rows = [ (r['id'], c, 'content', i+1) for i, c in enumerate(content_res) ]

    rows += title_rows
    rows += content_rows

    print(f"title_res={title_res}")
    print(f"content_res={content_res}")

    noun_terms.update(title_res)
    noun_terms.update(content_res)

    r = self.cur.fetchone()
```

```
...
content_res=['매출', '톱', '톱5', '5', '게임', '게임사', '사', '2023', '2023년', '년', '3', '3분기', '분기', '실적', '회비', '엔'
doc_id=1444
title_res=['애플', '생성', '기능', '아이', '아이폰16', '폰', '16', '독점', '제공']
content_res=['팁', '팁스터', '스터', '레', '너스', '전망', '애플', '한창', '개발', '증인', '생성', '인공', '인공지능', '지능', '기능',
doc_id=1445
title_res=['삼성', '럭', '럭시핏3', '시', '핏', '3']
content_res=['삼성', '삼성전자', '전자', '차세대', '스마트', '스마트밴드', '밴드', '신제품', '럭', '럭시핏', '시', '핏', '3', '의', '제
doc_id=1446
title_res=['애국', '애국소비', '소비', '덕', '미', '신형', '스마트', '스마트폰', '폰', '100', '100만대', '만', '대', '판매']
content_res=['스마트', '스마트폰', '폰', '수요', '침체', '회복', '조짐', '4', '4분기', '분기', '반등', '전망', '중국', '시장', '기미',
doc_id=1447
title_res=['셀', '셀트리온', '트리', '온', '브라질', '정부', '정부물량', '물량', '수주', '랩', '램시마', '시마', '점유율', '80', '예상'
content_res=['셀', '셀트리온헬스', '트리', '온', '헬스', '케어', '브라질', '연방', '연방정부', '정부', '입찰', '3', '3년', '년', '연속',
doc_id=1448
title_res=['마이크', '삼성', '3', '도전장', '내년', '1', '1분기', '분기', '대량', '생산']
content_res=['대', '타이', '타이충', '증', '신규', '공장', '가동', '시작', '내년', '7', '7억', '억', '달러', '매출', '목표', '미국',
doc_id=1449
title_res=['40', '40만원대', '만', '원', '대', '럭', '럭시', '시', '단독', '출시', '게임', '영상', '재생']
content_res=['온', '온', '오프라인', '매장', '판매', '통신비', '완화', '출고', '43', '43만8900원', '만', '8900', '원', '제휴', '자
doc_id=1450
title_res=['네이버', '카오', '논란', '인수', '기업', '3', '3분기', '분기', '최대', '매출']
content_res=['디지털', '디지털데일리', '데일리', '민지', '기자', '네이버', '카오', '올해', '3', '3분기', '분기', '역대', '최대', '매출
{'1억9000만', '안성', '분명', '크레이', '서대문역', '대사', '사회악', '경계심', '소개', '수만', '외상값', '지역별', '트라이아웃', '범죄단체']
```

def 'Extract_Nouns'

```

if rows:
    insert_sql = """insert into extracted_terms1(doc_id, term, term_region, seq_no)
    | | | | values(%s,%s,%s,%s);"""
    self.cur.executemany(insert_sql, rows)
    self.conn.commit()

print(noun_terms)
print(f"\nnumber of terms = {len(noun_terms)}") 

```

Document Vector's Dimension
= **25428**

number of terms = 25428

Table : extracted_terms1

doc_id	Term의 어느 doc에서 추출되었는지를 표현
term	Term, noun() 단위로 분류
term_region	Title에 존재하는지, Content에 존재하는지 분류
seq_no	Term 배열의 순서를 표현

def 'gen_idf'

Document Frequency를 만들기 위해 SQL 'count(*)' 구문 활용

```
def gen_idf(self):  
  
    create_sql = """"  
        drop table if exists idf1;  
  
        create table idf1 (  
            term_id int primary key,  
            df int,  
            idf float,  
            enter_date datetime default now()  
        );  
    """  
  
    self.cur.execute(create_sql)  
    self.conn.commit()  
  
    sql = "select count(*) as doc_count from news_article1;"  
    self.cur.execute(sql)  
    self.num_of_docs = self.cur.fetchone()['doc_count']  
  
    idf_sql = f""" insert into idf1(term_id, df, idf)  
        select ntd.id, count(distinct doc_id) as df, log({self.num_of_docs}/count(distinct doc_id)) as idf  
        from extracted_terms1 ent, term_dict1 ntd  
        where ent.term = ntd.term  
        group by ntd.id;  
    """  
  
    self.cur.execute(idf_sql)  
    self.conn.commit()
```

def 'gen_idf'

idf : 특정 단어가 나타내는 문서의 갯수 'DF'를 **역수**를 취하고 정규성을 높이기 위한 **Log**처리, Table 4

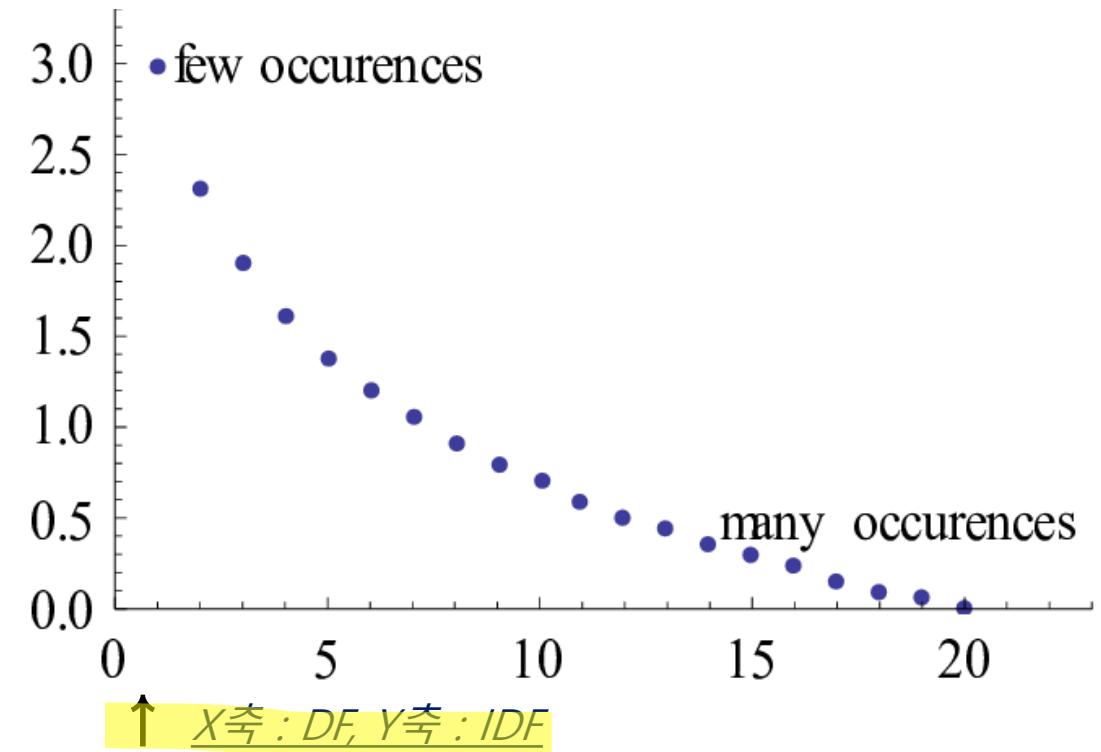
SELECT * FROM DS2023.idf1;				
	term_id	df	idf	enter_date
1	1	4	5.89302	2023-11-19 17:28:55
2	2	2	6.58617	2023-11-19 17:28:55
3	3	4	5.89302	2023-11-19 17:28:55
4	4	8	5.19988	2023-11-19 17:28:55
5	5	2	6.58617	2023-11-19 17:28:55
6	6	32	3.81358	2023-11-19 17:28:55

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

, where n = no. of documents, df_i = document frequency of i

Document frequency: **df**

- The number of document in the collection that contain a term



Visualization

matplotlib.pyplot을 활용한 df, idf의 movement 시각화, frequency에 따른 값의 variants.

```
def show_top_df_terms(self):
    sql = """select * from idf1 idf, term_dict1 td
             where idf.term_id = td.id
             order by df desc
             ;"""
    self.cur.execute(sql)

    res = [ (r['term'], r['df'], r['idf']) for r in self.cur.fetchall() ]

    print("\nTop 10 DF terms:\n")

    for r in res[:10]:
        print(f"{r[0]}: df={r[1]}, idf={r[2]}")

    df_list = [ r[1] for r in res]
    plt.figure(figsize=(10, 5))
    plt.hist(df_list, bins=100, alpha=0.7, color='blue')
    plt.title('Histogram of DF')
    plt.xlabel('Document Frequency')
    plt.ylabel('Number of Terms')
    plt.grid(axis='y', alpha=0.75)
    plt.show()
```

```
def show_top_idf_terms(self):
    sql = """select * from idf1 idf, term_dict1 td
             where idf.term_id = td.id
             order by idf desc
             ;"""
    self.cur.execute(sql)

    res = [ (r['term'], r['df'], r['idf']) for r in self.cur.fetchall() ]

    print("\nTop 10 IDF terms:\n")

    for r in res[:10]:
        print(f"{r[0]}: df={r[1]}, idf={r[2]}")

    idf_list = [ r[2] for r in res]
    plt.figure(figsize=(10, 5))
    plt.hist(idf_list, bins=100, alpha=0.7, color='red')
    plt.title('Histogram of IDF')
    plt.xlabel('IDF')
    plt.ylabel('Number of Terms')
    plt.grid(axis='y', alpha=0.75)
    plt.show()
```

Visualization

Top 10 DF / IDF terms (term / df / idf)

DF는 가장 많은, IDF는 가장 적은 frequency를 가진 term을 각각 나타냄.

Top 10 DF terms:

일: df=1314, idf=0.0984876
등: df=1044, idf=0.328504
수: df=1032, idf=0.340065
이: df=864, idf=0.517746
10: df=834, idf=0.553085
3: df=820, idf=0.570014
고: df=814, idf=0.577358
년: df=808, idf=0.584757
1: df=770, idf=0.632928
2: df=764, idf=0.640751

Top 10 IDF terms:

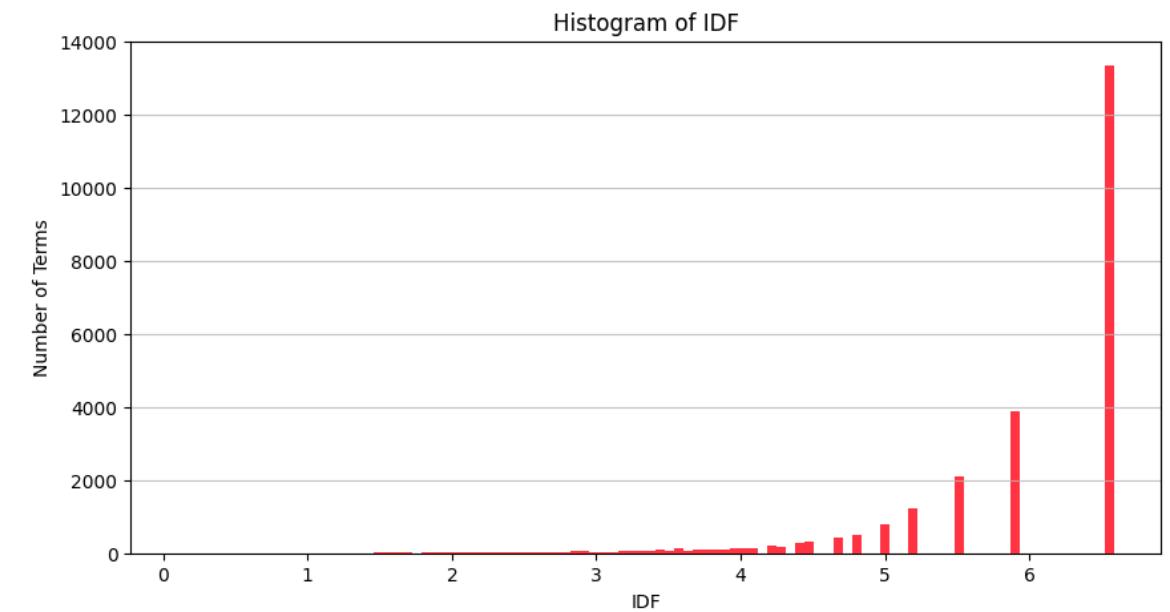
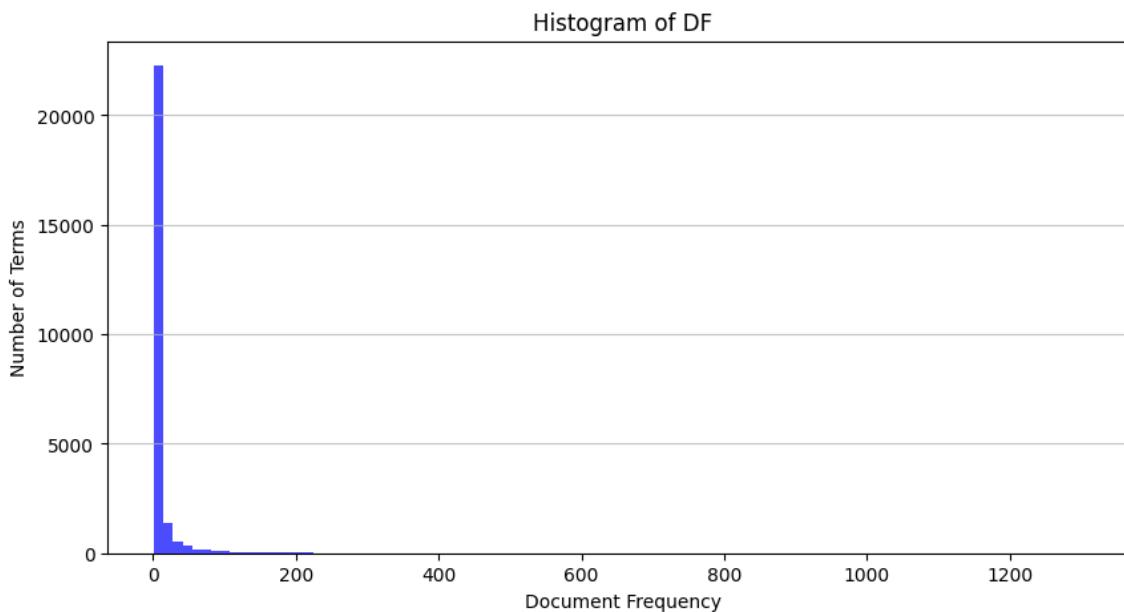
-2.5: df=2, idf=6.58617
회생제동: df=2, idf=6.58617
제공량: df=2, idf=6.58617
김학철: df=2, idf=6.58617
방재: df=2, idf=6.58617
질식: df=2, idf=6.58617
우승후보: df=2, idf=6.58617
애착: df=2, idf=6.58617
뒷골목: df=2, idf=6.58617
깃발: df=2, idf=6.58617

Visualization

DF : DF 값이 0에 근접한 term의 개수가 가장 많음, 전체적으로 term의 분포가 independent한 속성을 가짐.

IDF : IDF 값이 6 이상인 term의 개수가 가장 많음, IDF는 dissimilarity로 인식 -> 이 또한 independent한 속성의 분포.

Summary : 결론적으로, Log 값을 취한 IDF에서 더욱 정규성을 지닌 분포를 얻게 됨, DF-IDF의 역수 관계 또한 확인 가능.



Term Frequency

Definition 2.1 TF Variants: $\text{TF}(t, d)$. $\text{TF}(t, d)$ is a quantification of the within-document term frequency, tf_d . The main variants are:

$$\text{tf}_d := \text{TF}_{\text{total}}(t, d) := \text{lf}_{\text{total}}(t, d) := n_L(t, d) \quad (2.1)$$

$$\text{TF}_{\text{sum}}(t, d) := \text{lf}_{\text{sum}}(t, d) := \frac{n_L(t, d)}{N_L(d)} \quad \left(= \frac{\text{tf}_d}{\text{dl}} \right) \quad (2.2)$$

$$\text{TF}_{\text{max}}(t, d) := \text{lf}_{\text{max}}(t, d) := \frac{n_L(t, d)}{n_L(t_{\text{max}}, d)} \quad (2.3)$$

$$\text{TF}_{\text{log}}(t, d) := \text{lf}_{\text{log}}(t, d) := \log(1 + n_L(t, d)) \quad \left(= \log(1 + \text{tf}_d) \right) \quad (2.4)$$

$$\text{TF}_{\text{frac}, K}(t, d) := \text{lf}_{\text{frac}, K}(t, d) := \frac{n_L(t, d)}{n_L(t, d) + K_d} \quad \left(= \frac{\text{tf}_d}{\text{tf}_d + K_d} \right) \quad (2.5)$$

$$\text{TF}_{\text{BM25}, k_1, b}(t, d) := \text{lf}_{\text{BM25}, k_1, b}(t, d) := \frac{n_L(t, d)}{n_L(t, d) + k_1 \cdot (b \cdot \text{pivdl}(d, c) + (1 - b))} \quad (2.6)$$

def 'gen_tfidf'

$$tf\ idf\ (t, d, D) = tf\ (t, d) \cdot idf\ (t, D)$$

```
def gen_tfidf(self):

    create_sql = """
        drop table if exists tfidf1;

        create table tfidf1 (
            id int auto_increment primary key,
            doc_id int,
            term_id int,
            tf float,
            tfidf float,
            enter_date datetime default now()
        );
        """

    self.cur.execute(create_sql)
    self.conn.commit()

    tfidf_sql = """ insert into tfidf1(doc_id, term_id, tf, tfidf )
                    select ent.doc_id, ntd.id, count(*) as tf, count(*) * idf.idf as tfidf
                    from extracted_terms1 ent, term_dict1 ntd, idf1 idf
                    where ent.term = ntd.term and ntd.id = idf.term_id
                    group by ent.doc_id, ntd.id;
        """

    self.cur.execute(tfidf_sql)
    self.conn.commit()
```

Table 5

1 • SELECT * FROM DS2023.tfidf1;

100% 1:1

Result Grid Filter Rows: Search Edit:

	id	doc_id	term_id	tf	tfidf	enter_date
1	1	163	1	3.40812	2023-11-19 17:28:58	
2	1	497	1	4.28359	2023-11-19 17:28:58	
3	1	745	1	0.905999	2023-11-19 17:28:58	
4	1	774	1	3.49513	2023-11-19 17:28:58	
5	1	829	2	7.3916	2023-11-19 17:28:58	
6	1	962	1	3.6958	2023-11-19 17:28:58	
7	1	1463	1	1.77399	2023-11-19 17:28:58	
8	1	1794	1	0.328504	2023-11-19 17:28:58	
9	1	1906	1	2.97525	2023-11-19 17:28:58	
10	1	2133	1	3.05981	2023-11-19 17:28:58	
11	1	2384	1	3.59044	2023-11-19 17:28:58	
12	1	2514	1	4.10126	2023-11-19 17:28:58	
13	1	2700	1	3.59044	2023-11-19 17:28:58	
14	1	2710	2	4.64698	2023-11-19 17:28:58	
15	1	3076	1	2.75753	2023-11-19 17:28:58	
16	1	3111	1	3.6958	2023-11-19 17:28:58	
17	1	3264	1	3.81358	2023-11-19 17:28:58	
18	1	3328	1	2.05357	2023-11-19 17:28:58	

Example

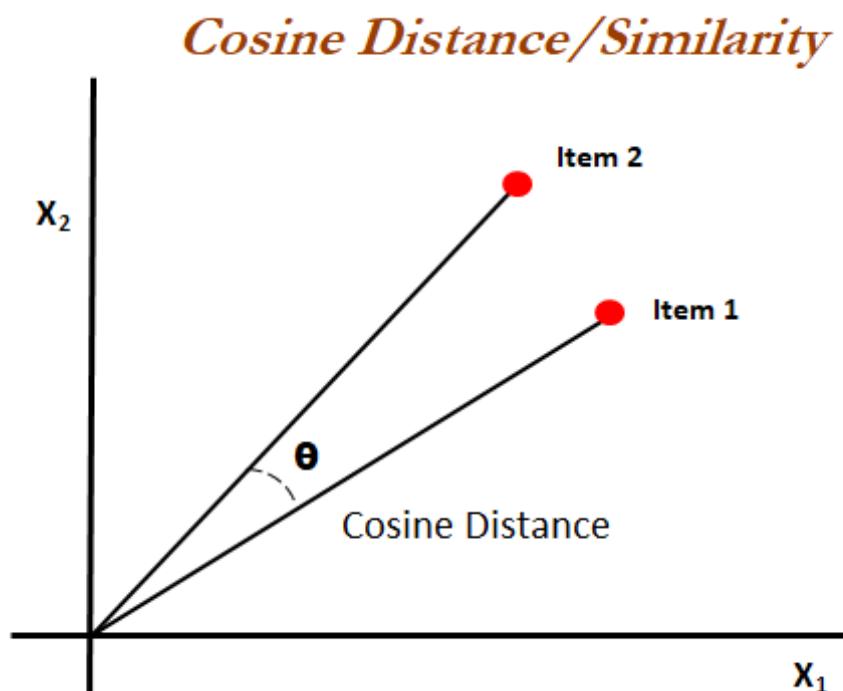
Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	$\log(2/2) = 0$	0	0
Car	1/7	0	$\log(2/1) = 0.3$	0.043	0
Truck	0	1/7	$\log(2/1) = 0.3$	0	0.043
Is	1/7	1/7	$\log(2/2) = 0$	0	0
Driven	1/7	1/7	$\log(2/2) = 0$	0	0
On	1/7	1/7	$\log(2/2) = 0$	0	0
The	1/7	1/7	$\log(2/2) = 0$	0	0
Road	1/7	0	$\log(2/1) = 0.3$	0.043	0
Highway	0	1/7	$\log(2/1) = 0.3$	0	0.043

Document Vector Model

Document Vector Model

각 Document들을 ‘Vector’화 했을 때, 수행 가능한 measure

- 문서 번호를 지정할 때, 해당 문서의 Top 5 키워드를 보여줄 것.
- 문서 번호를 지정할 때, 해당 문서와 가장 가까운 문서 top 3를 보여할 것.
- 사용자의 쿼리를 입력으로 받아, 이 쿼리에 적합한 상위문서 top3를 보여줄 것.



$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Document Vector's Dimension
= 25428

number of terms = 25428

Document Vector Model

1. 지정된 문서의 Top 5 Keywords

```
cdb.get_keywords_of_document(250)
```

```
def get_keywords_of_document(self, doc):
    sql = f"""
        select *
        from tfidf1 tfidf, term_dict1 td
        where tfidf.doc_id = {doc}
        and tfidf.term_id = td.id
        order by tfidf.tfidf desc
        limit 5;
    """

    self.cur.execute(sql)
    result = self.cur.fetchall()

    print(f"\nTop 5 키워드 for 문서 {doc}:")
    for row in result:
        print(f"{row['term']}: {row['tfidf']}")
```

Top 5 키워드 for 문서 300:
 전우: 13.1723
 6사단: 13.1723
 사단: 13.1723
 30잔: 13.1723
 커피: 9.58882

**tf-idf를 기준으로,
 term_id 선정 후,
 term을 추적**

id	doc_id	term_id	tf	tfidf	ent
42368	250	2990	1	1.97105	202
42369	250	3048	1	3.25397	202
42370	250	3107	1	4.10126	202
42371	250	3151	2	1.812	202
42372	250	3184	1	3.87812	202
42373	250	3492	1	2.35207	202
42374	250	3575	1	2.97525	202
42375	250	3609	1	2.26868	202

Top 5 키워드 for 문서 250:
 케이: 7.62717
 기차: 6.99026
 수급난: 5.89302
 페이스리프트: 5.89302
 케이카: 5.89302

Document Vector Model

2. 지정된 문서와 가장 가까운 Top 3 Documents

```
cdb.sort_similar_docs(50)
```

```
def sort_similar_docs(self, doc):
    sim_vector = []

    for i in range(1,1450):
        if i == doc:
            continue
        sim = self.doc_similarity(doc, i)
        sim_vector.append((i, sim))
    sorted_sim_vector = sorted(sim_vector, key=lambda x: x[1], reverse=True)

    print(f"\n문서 {doc}와 가장 가까운 문서 Top 3:")
    for idx, similarity in sorted_sim_vector[:3]:
        print(f"문서 {idx}: 유사도 {similarity}")
```

```
def doc_similarity(self, doc1, doc2):
    def cosine_similarity(vec1, vec2):

        dict1 = dict(vec1)
        dict2 = dict(vec2)

        common_terms = set(dict1.keys()) & set(dict2.keys())
        dot_product = sum([dict1[term] * dict2[term] for term in common_terms])

        vec1_magnitude = sum([val**2 for val in dict1.values()])**0.5
        vec2_magnitude = sum([val**2 for val in dict2.values()])**0.5

        if vec1_magnitude == 0 or vec2_magnitude == 0:
            return 0
        else:
            return dot_product / (vec1_magnitude * vec2_magnitude)

    sql1 = f"""select term_id, tfidf from tfidfl where doc_id = {doc1};"""
    self.cur.execute(sql1)
    doc1_vector = [(t['term_id'], t['tfidf']) for t in self.cur.fetchall()]

    sql2 = f"""select term_id, tfidf from tfidfl where doc_id = {doc2};"""
    self.cur.execute(sql2)
    doc2_vector = [(t['term_id'], t['tfidf']) for t in self.cur.fetchall()]

    return cosine_similarity(doc1_vector, doc2_vector)
```

Document Vector Model

2. 지정된 문서와 가장 가까운 Top 3 Documents

문서 50와 가장 가까운 문서 Top 3:

문서 775: 유사도 0.9999999999999997
 문서 67: 유사도 0.6118009710149699
 문서 792: 유사도 0.6118009710149699

문서 80와 가장 가까운 문서 Top 3:

문서 805: 유사도 1.0
 문서 430: 유사도 0.8887090714047272
 문서 1155: 유사도 0.8887090714047272

- *angle* between two nonzero vectors a, b defined as

$$\angle(a, b) = \arccos\left(\frac{a^T b}{\|a\| \|b\|}\right)$$

- $\angle(a, b)$ is the number in $[0, \pi]$ that satisfies

$$a^T b = \|a\| \|b\| \cos(\angle(a, b))$$

but, Vector의 Sparse 문제로 인한 결과 왜곡 또한 염두해야 함.

Document Vector Model

3. 사용자의 쿼리를 입력으로 받아, 이 쿼리에 적합한 상위문서 Top 3 산출

```

def process_user_query(self, user_query):
    def calculate_tf(document, words):
        tf = {}
        for w in words:
            tf[w] = document.count(w)
        return list(tf.values())

    drop_sql1 = """ drop table if exists query1;"""
    self.cur.execute(drop_sql1)
    self.conn.commit()

    create_sql1 = """
        CREATE TABLE query1 (
            id int auto_increment primary key,
            term varchar(30),
            tf float
        );
    """
    self.cur.execute(create_sql1)
    self.conn.commit()

    drop_sql2 = """ drop table if exists query_tfidf1;"""
    self.cur.execute(drop_sql2)
    self.conn.commit()

    create_sql2 = """
        CREATE TABLE query_tfidf1 (
            id int auto_increment primary key,
            term_id int,
            tfidf float
        );
    """
    self.cur.execute(create_sql2)
    self.conn.commit()

```

```

rows = []
query_terms = self.pos_tagger.nouns(user_query)
tf_values = calculate_tf(user_query, query_terms)
query_rows = list(zip(query_terms, tf_values))
rows += query_rows

if rows:
    insert_sql = """insert into query1(term, tf)
                    values(%s, %s);"""
    self.cur.executemany(insert_sql, rows)
    self.conn.commit()

    tfidf_sql = """ insert into query_tfidf1(term_id, tfidf )
                    select ntd.id, AVG(que.tf * idf.idf) as tfidf
                    from extracted_terms ent, query1 que, idf1 idf, term_dict1 ntd
                    where ent.term = que.term and que.term = ntd.term and ntd.id = idf.term_id
                    group by ntd.id
    """
    self.cur.execute(tfidf_sql)
    self.conn.commit()

    sql2 = f"""select term_id, tfidf from query_tfidf1;"""
    self.cur.execute(sql2)
    query_vector = [(t['term_id'], t['tfidf']) for t in self.cur.fetchall()]

    print(f"\n쿼리에 대한 상위 문서 Top 3:")
    self.sort_similar_docs2(query_vector)

```

Document Vector Model

추가 테이블, query1 & query_tfidf1

1 • SELECT * FROM DS2023.query1;

100% 1:1

Result Grid Filter Rows: Search

id	term	tf
1	공매도	2
2	금지	2
3	이후	1
4	에코	1
5	에코프로	1
6	프로	1
7	등	1
8	증시	1
9	상승	1
10	주도	2
11	이차	1
12	이차전지	1
13	전지	1
14	관련	1
15	관련주도	1
16	첫날	1
17	상한가	1
18	도로	1
19	하락	1
NULL	NULL	NUL

1 • SELECT * FROM DS2023.query_tfidf1;

100% 1:1

Result Grid Filter Rows: Search

id	term_id	tfidf
1	15267	7.89423
2	23151	5.7452
3	3241	1.15245
4	7687	5.19988
5	6697	5.48756
6	2048	2.57884
7	18458	0.328504
8	1548	3.45068
9	6522	2.12026
10	14250	6.99026
11	13499	5.19988
12	2835	5.19988
13	1139	4.18828
14	11065	1.36042
15	17547	4.38895
16	9791	5.89302
17	23914	3.15218
18	20285	2.41178
NULL	NULL	NUL

Document Vector Model

def 'sort_similar_docs2', 'doc_similarity2' & 'cosine_similarity'

```
def sort_similar_docs2(self, query_vector):
    sim_vector = []

    for i in range(1,1451):
        sim = self.doc_similarity2(query_vector, i)
        sim_vector.append((i, sim))
    sorted_sim_vector = sorted(sim_vector, key=lambda x: x[1], reverse=True)

    print(f"\nQuery와 가장 가까운 문서 Top 3:")
    for idx, similarity in sorted_sim_vector[:3]:
        print(f"문서 {idx}: 유사도 {similarity}")
```

기존 소스코드에서,
약간의 변형을 준 형태.

query_vector를 그대로 받고,
tfidf1 table에서 searching

```
def doc_similarity2(self, query_vector, doc):
    def cosine_similarity(vec1, vec2):

        dict1 = dict(vec1)
        dict2 = dict(vec2)

        common_terms = set(dict1.keys()) & set(dict2.keys())
        dot_product = sum([dict1[term] * dict2[term] for term in common_terms])

        vec1_magnitude = sum([val**2 for val in dict1.values()])**0.5
        vec2_magnitude = sum([val**2 for val in dict2.values()])**0.5

        if vec1_magnitude == 0 or vec2_magnitude == 0:
            return 0
        else:
            return dot_product / (vec1_magnitude * vec2_magnitude)

    doc1_vector = query_vector

    sql2 = f"""select term_id, tfidf from tfidf1 where doc_id = {doc};"""
    self.cur.execute(sql2)
    doc2_vector = [(t['term_id'], t['tfidf']) for t in self.cur.fetchall()]

    return cosine_similarity(doc1_vector, doc2_vector)
```

Document Vector Model

Query 1 : “공매도 금지 이후 에코프로 등 증시 상승을 주도한 이차전지 관련주도
공매도 금지 첫날만 상한가로 치솟았다가 도로 하락했다”

Query 2 : “특히 미국 금리가 떨어지니까 당연히 달러화도 약해지고, 그럼 미국
밖의 자산에 대한 어떤 외국인의 매수세도 늘어나는 경향이 있는데
여러 가지 이유가 있습니다”

Query 3 : “특정 종목의 주식을 보유하고 있지 않은 상태에서 주식을 빌려 매도
주문을 내는 투자 전략으로, 주로 초단기 매매차익을 노리는 데
사용되는 기법이다”

쿼리에 대한 상위 문서 Top 3:

Query와 가장 가까운 문서 Top 3:

문서 284: 유사도 0.3346000965897693
문서 1009: 유사도 0.3346000965897693
문서 338: 유사도 0.22384043786226684

쿼리에 대한 상위 문서 Top 3:

Query와 가장 가까운 문서 Top 3:

문서 688: 유사도 0.1250395402210623
문서 1413: 유사도 0.1250395402210623
문서 696: 유사도 0.10914310611091362

쿼리에 대한 상위 문서 Top 3:

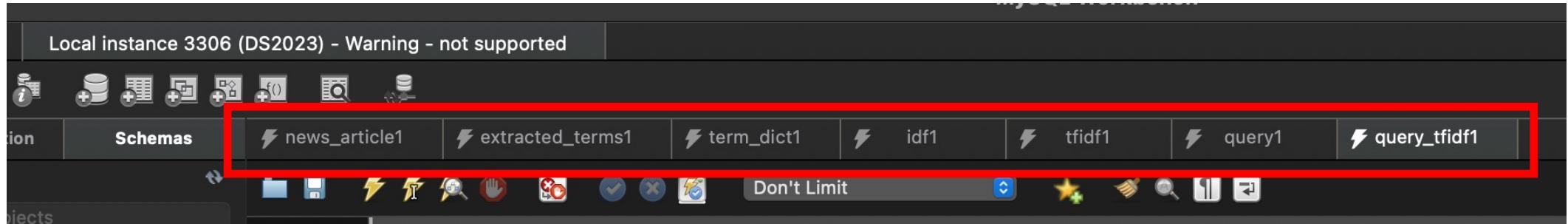
Query와 가장 가까운 문서 Top 3:

문서 70: 유사도 0.10138432583807706
문서 795: 유사도 0.10138432583807706
문서 681: 유사도 0.0961983032978339

Conclusion

Conclusion

총 7개의 MySQL Workbench Table을 Create하고, 서로 interaction이 가능한 환경을 구축함.



Problem Solving의 핵심은, 바로 Documents의 ‘Vectorization’이었으며,
Factor들을 연산 및 처리 하는데 있어서 ‘Algebra’ 관점에서 Matrix를 Design 하는 것이 Key Points.
SQL 구문의 정교한 Syntax를 통해, tf-idf 연산을 실행.

한계점 : ‘꼬꼬마(Kkma)’를 통해 noun을 분해할 때, 분해된 noun과 더 작은 범위에서의 형태소를
구분하지 않고, 무차별적으로 insert하여, 연산의 정확도가 떨어짐.
i.e) 공식전 -> “공식, 공식전, 전” 모두 inserted

id	doc_id	term
1	1	공식
2	1	공식전
3	1	전

