2017 Database System Project #3

Text mining with MongoDB

20141494 강동욱

목차

[1. 문제 정의 3](#_Toc489289200)

[2. 요구사항 4](#_Toc489289201)

[2.1 형태소 분석 및 불용어 처리(10점) 5](#_Toc489289202)

[2.2 Word Count 구하기(15점) 5](#_Toc489289203)

[2.3 TF-IDF 수행(15점) 6](#_Toc489289204)

[2.4 문서 유사도 구하기(20점) 6](#_Toc489289205)

[3. 사용환경 7](#_Toc489289206)

[4. 프로그램 구현 7](#_Toc489289207)

[4.1 프로그램 메뉴 구성 관련 7](#_Toc489289208)

[4.2 모듈정의 8](#_Toc489289209)

[*4.2.1* MorpAnalysis 8](#_Toc489289210)

[*4.2.2* WordCount 9](#_Toc489289211)

[*4.2.3* Tf-idf 10](#_Toc489289212)

[*4.2.4* Simillarity 11](#_Toc489289213)

[*4.2.5* CopyData 11](#_Toc489289214)

[*4.2.6* printMenu 12](#_Toc489289215)

[*4.2.7* main 12](#_Toc489289216)

# 문제 정의

본 프로젝트에서는 텍스트 마이닝 기법 중 하나인 TF-IDF 가중치를 이용하여 제공된 뉴스

기사들을 분석하고 개별 기사들마다 핵심어를 추출함으로써 기사들 사이의 유사도를 구할 수

있는 프로그램을 작성한다. 또한 뉴스기사 분석을 위해 비정형 데이터를 다루기 쉬운 NoSQL

기반 데이터베이스인 MongoDB를 사용함으로써 NoSQL 데이터베이스의 사용법을 익힐 뿐

아니라 관계형 데이터베이스와 NoSQL 데이터베이스간의 차이점을 인식하는 것을 목적으로

한다.

텍스트 마이닝이란 비정형 데이터 마이닝의 유형 중 하나로 자연어 처리 기술과 문서 처리

기술을 적용하여 유용한 정보를 추출, 가공하는 것을 목적으로 하는 기술이다. 실생활에서 만

들어지는 대부분의 문서는 형태로 보관되며 제목, 저자, 출판날짜 등과 같은 구조적인 특징들

과 문서의 요약, 내용과 같은 크기가 일정하지 않은 비구조적 요소들을 포함하기에 반구조적

데이터로 분류된다. 응용분야로는 Risk management, Knowledge management, Cybercrime

prevention, Customer care service, Business intelligence, Spam filtering 등이 있다.

TF-IDF는 언어 자료 내의 특정 문서에서 어떤 단어의 중요도를 평가하기 위해 사용되는 통

계적인 수치로 문서의 핵심어 추출, 검색 엔진의 검색 결과 순위 결정, 문서들 사이의 유사도

등에 적용될 수 있다. 일반적으로 한 문서에서 중요한 단어일수록 해당 문서 내에서 반복적

으로 나타날 가능성이 높고, 그렇지 않은 단어의 등장 빈도는 낮을 것이다. 그러나 우리말의

조사나 부사와 같이 개별 문서 내에서 많이 등장하면서 또한 모든 문서 집합 내에서 반복적

으로 나타나는 단어의 경우 이런 단어들을 문서를 대표한다고 할 순 없다. TF-IDF 가중치는

이런 원리에 입각해 설정된 값으로 TF(단어 빈도, term frequency)는 특정한 단어가 문서 내에

얼마나 자주 등장하는지를 나타내고, 이 값이 높을수록 문서에서 중요하게 여겨진다. 하지만

단어 자체가 문서군 내에서 자주 사용되는 경우, 이것은 그 단어가 흔하게 등장한다는 것을

의미하고 이것을 DF(문서 빈도, document frequency)라고 한다. DF의 역수를 IDF(역문서 빈도,

inverse document frequency)라고 하며 TF-IDF는 TF와 IDF를 곱한 값이다. 전체 문서집합을 D,

개별 문서를 d ∈ D , 단어 t에 대해 문서 d에서 등장 횟수를 (d)라 하면 TF-IDF 식은 아래

와 같다.

tf(t,d) =

idf(t,D)=

MongoDB는 C++로 작성된 오픈소스의 문서지향(Document-Oriented)적 데이터베이스이며

기존의 RDBMS의 한계를 극복하기 위해 만들어진 새로운 형태의 데이터저장소이다. 관계형

데이터베이스가 아니므로 RDBMS처럼 고정된 스키마 및 JOIN 연산이 존재하지 않으며,

Document라고 불리는 기본 데이터 구조 단위로 이루어진다. 모든 데이터 구조는 한 개 이상

의 key-value 쌍으로 이루어진다.

{

"\_id": ObjectId("5099803df3f4948bd2f98391"),

"username": "ironman",

"name": { first: "Tony", last: "Stark" }

}

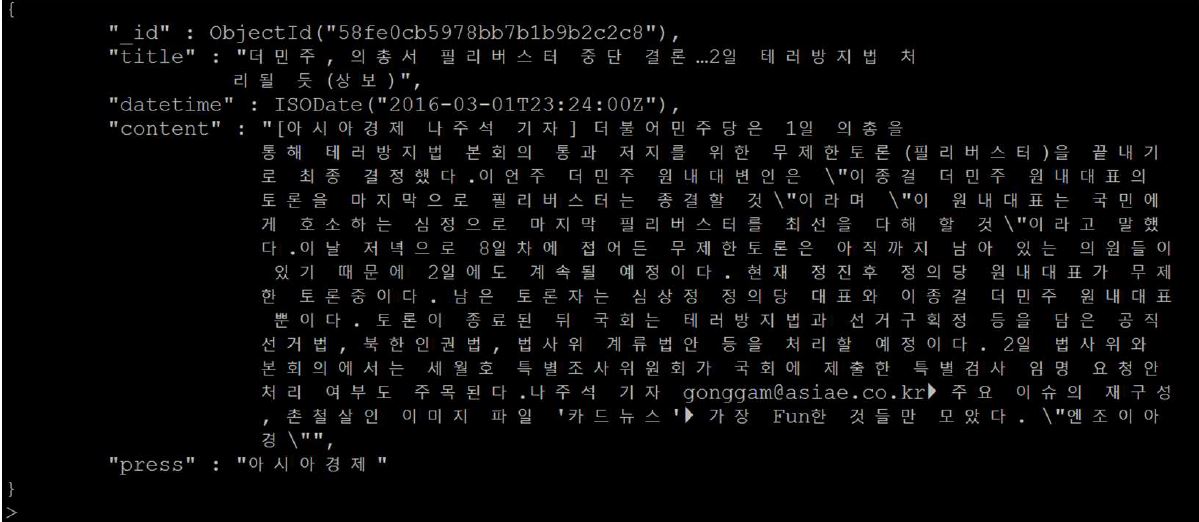
# 요구사항

본 프로젝트에서는 제공된 뉴스기사에 대해 형태소 분석을 통해 문서를 키워드의 집합으로

분할한 후 TF-IDF 작업을 수행한 후 기사들 간의 유사도를 구하는 것을 목적으로 한다. 다음

은 MongoDB에 저장된 뉴스기사에 대한 예제 화면이다. 아래 예제 화면을 참고하여 문서의

구조를 분석하고, 2.1 ~ 2.4까지의 작업을 진행한다.



## 형태소 분석 및 불용어 처리(10점)

형태소 분석은 텍스트 마이닝 시 자연어 처리의 첫번째 단계로 입력문자열을 형태소열로

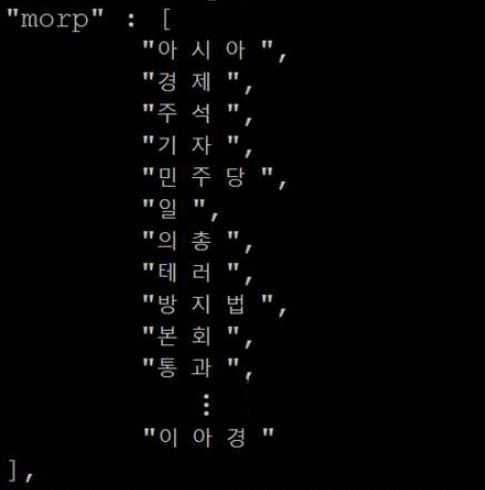
바꾸는 작업을 말한다. 형태소란 의미의 최소단위로써 더 이상 나눌 수 없는 가장 작은 단위

의 의미 요소를 말한다. 모든 뉴스기사에 대해 제공된 형태소 분석 소스코드와 불용어 리스

트 파일을 이용해 텍스트 분석에 불필요한 단어(불용어)를 제거하고 형태소열을 데이터베이

스에 저장하라. 또한 사용자로부터 뉴스기사의 Object id를 입력 받아 해당하는 뉴스기사의

형태소들을 출력해주는 프로그램을 작성하라. 형태소 분석 결과를 데이터베이스에 저장한 결과 화면은 아래와 같다.



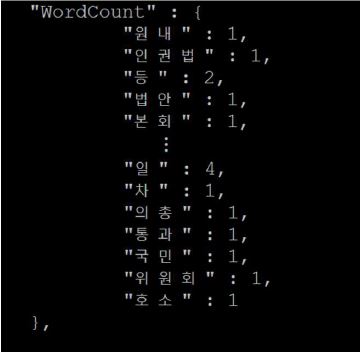
## Word Count 구하기(15점)

Word Count는 TF-IDF 작업을 수행하기 전 사전 단계로 각 문서들마다 포함되어 있는 고유

한 개별 단어들의 빈도수를 측정한다. 형태소 분석 단계에서와 마찬가지로 모든 뉴스 기사에

대해 고유한 단어들의 등장횟수를 측정하고, 그 값을 데이터베이스에 저장하라. 사용자로부터

뉴스기사의 Object id를 입력 받아 해당하는 뉴스기사의 WordCount 리스트를 출력해주는 프

로그램을 작성하라. 데이터베이스에 저장된 결과 화면은 아래와 같다. 

## TF-IDF 수행(15점)

본격적으로 텍스트 마이닝을 수행하기 위해 2.3에서 저장한 WordCount 값을 이용해 단어

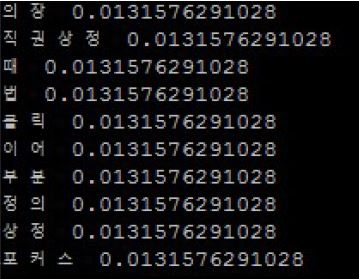
들마다 문제 정의 부분에서 설명한 TF-IDF 값을 계산한다. 계산한 TF-IDF 가중치를

WordCount를 저장한 방식과 마찬가지로 데이터베이스에 저장한다. 뉴스기사의 Object ID를

입력 받고 데이터베이스로부터 관련된 문서들을 찾은 후 TF-IDF 가중치를 출력하는 프로그램

을 작성하라. 출력은 TF-IDF 값이 가장 높은 단어 순서대로 상위 10개의 단어와 TF-IDF 값을

출력하라. 아래는 출력 예시이다.



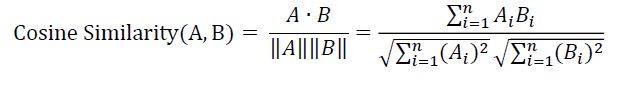
## 문서 유사도 구하기(20점)

두 뉴스기사의 Object ID를 입력 받고 데이터베이스로부터 관련된 문서들을 찾은 후 TFIDF

가중치를 이용해 두 문서의 유사도를 구하는 프로그램을 작성하라. 두 문서간의 유사도

는 주로 Cosine 유사도를 이용해 구해지는데, 문서 A,B가 주어졌을 때 문서를 단어 벡터형태

로 표현하고 벡터 성분을 tf-idf값으로 할당한 뒤 유사도를 다음과 같은 식에 의해 구한다.



주의: 위 식의 Ai,Bi 두 문서간의 공통된 단어의 TF-IDF 값을 곱한 것이다.

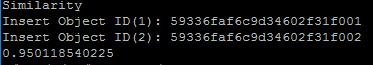
예를 들어, { jazz(0.7), classic(0.1), music(0.5)}라는 단어(괄호 안은 tf-idf 가중치)로 이루어진

문서 A와 {music(0.7), pop(0.3)}라는 단어로 이루어진 문서 B가 있다면, 전체 문서의 총 단어

집합은 {jazz, classic, music, pop}이 된다. 이에 따라 문서 A를 단어 벡터로 표현하면

{0.7,0.1,0.5,0.0}이 되고, 문서 B를 단어 벡터로 표현할 경우 {0.0,0.0,0.7,0.3}이 된다.

아래는 실행 예시이다.



2.5 프로그램 메뉴 구성도(필수)

처음 프로그램을 실행하면 아래와 같이 전체 메뉴 구성도를 구성한다. 그 후 사용자로부터 세부

프로그램 번호를 입력 받아 해당하는 프로그램을 수행하도록 한다.

1. WordCount

2. TF-IDF

3. Simirality

# 사용환경

서버: Host - dbpurple.sogang.ac.kr / Port - 22

운영체제: Ubuntu 14.04.5 LTS

데이터베이스: Mongodb 3.0.14

사용언어: python 2.7.6

라이브러리: pymongo, MeCab

# 프로그램 구현

사용하는 라이브러리는 다음과 같다.

import datetime

import time

import sys

import MeCab

from pymongo import MongoClient

import math

MongoDB를 제어하기 위해 MongoClient를 import하고, 형태소 분석을 위해 MeCab을 import한다.

MeCab에 대한 정보는 https://bitbucket.org/eunjeon/mecab-ko-dic/overview

## 프로그램 메뉴 구성 관련

   - 기본적으로 프로그램 흐름은 다음과 같습니다.

1) 메뉴를 선택다.

2) 뉴스 기사의 object\_id를 입력받는다.

3) 전체 뉴스 기사를 불러온다.

4) 불러온 기사를 분석해 해당 작업을 시행한다.(,MorpAnalysis, WordCount, TF-IDF)

5) 입력받은 object\_id에 대한 시행결과를 출력한다.

- WordCount는 MorpAnalysis의 결과물을 필요로 하고, TF-IDF는 WordCount의 결과를 필요로 한다.

프로그램은 각 작업을 한번 수행하고 종료되도록 요구되었기 때문에, 필요한 결과물이 존재하는지 확

하기 어렵다.

- 매번 object\_id를 입력 받을 때 마다 분석작업을 시행하는 것은 수행시간이 상당히 오래 걸리는 비정형데이터 분석 특성상 좋지 않기 때문에, 분석작업을 시행과 결과 검색을 따로 만드는 것이 훨씬 효율적이지만, 요구사항에서 번호를 입력 받고 id를 입력 받는 과정만을 명시하고 있기 때문에 비효율적으로 만들어졌다.

## 모듈정의

### MorpAnalysis

#### 기능

기사를 읽어 형태소 단위로 나눠 데이터베이스에 저장한다.

#### 코드 분석

def MorpAnalysis(docs,col\_tfidf):

print "MorpAnalysis"

t=MeCab.Tagger('-d/usr/local/lib/mecab/dic/mecab-ko-dic')

**// 외부라이브러리MeCab에서 한국어 형태소 분석을 위한 Tagger를 불러온다.**

**// MeCab에 대한 정보는 https://bitbucket.org/eunjeon/mecab-ko-dic/overview**

stop\_word={}

f=open("wordList.txt",'r')

while True:

line = f.readline()

if not line:break

stop\_word[line.strip('\n')]=line.strip('\n')

f.close()

**//wordlist.txt에서 추가하지 않을 단어들의 리스트를 불러와 dictionary형태로 저장한다.( key와 value가 같은 dictionary 형태로 저장할 이유가 없기 때문에 List 형태로 작성하는 것이 더 나을 것이라 생각하는데 조교가 제공한 코드여서 수정하지 못했다. )**

obj\_id=raw\_input("Input Object ID to start MorpAnalysis: ")

**//obj\_id 입력**

for doc in docs:

**// docs는 기사들의 리스트이므로 doc에는 단일 기사에 대한 object가 배정된다**

content=doc['content']

nodes = t.parseToNode(content.encode('utf-8'))

**//MeCab 라이브러리에서 만든 Tagger를 사용하여, 현재 기사의 내용을 나눠 nodes에 저장한다.**

MorpList=[]

while nodes:

if nodes.feature[0]== 'N' and nodes.feature[1]=='N':

w= nodes.surface

if not w in stop\_word:

MorpList.append(w)

nodes = nodes.next

**//형태소들의 리스트를 리스트형태로 저장한다. 나눠진 기사의 내용중 저장할 형태소를 선택하는데,**

**// if nodes.feature[0]== 'N' and nodes.feature[1]=='N': 의 조건은 MeCab에서 일반명사, 고유명사, 의존명사, 단위를 나타내는 명사만을 선택한다는 것을 의미하고, 그중에서도 stop\_word에 저장되지 않은 형태소만을 저장한다.**

|  |  |
| --- | --- |
| NNG | 일반 명사 |
| NNP | 고유 명사 |
| NNB | 의존 명사 |
| NNBC | 단위를 나타내는 명사 |

contentDic={}

for key in doc.keys():

contentDic[key]=doc[key]

contentDic['morp'] =MorpList

col\_tfidf.update({'\_id':contentDic['\_id']},contentDic,True)

print "MorpAnalysis success"

docs= col\_tfidf.find()

for doc in docs:

if(obj\_id== str(doc['\_id']) ):

for i in doc['morp']:

print(i)

return 0;

print "There is no Object ID %s"%(obj\_id)

return -1

**// 현재 기사 doc에 MorpList를 morp의 이름으로 추가한후 데이터베이스에 업데이트한다.**

### WordCount

#### 기능

각 문서들마다 포함되어 있는 고유한 개별 단어들의 빈도수를 측정한다.

#### 코드 분석

def WordCount(docs,col\_tfidf):

print "WordCount"

obj\_id=raw\_input("After MorpAnalysis, Input Object ID to start WordCount: ")

**//obj\_id를 입력받는다**

for doc in docs:

countList={}

MorpList=doc['morp']

**//MorpAnalysis에서 생성한 MorpList가 doc에 doc[‘morp’]로 저장되어 있으니 불러온다.**

for Morp in MorpList:

countList[Morp]=countList.get(Morp,0)+1 #make WordCountList

**//저장된 형태소들을 key를 형태소, value를 횟수로 하는 dictionary로 묶는다.**

contentDic={}

for key in doc.keys():

contentDic[key]=doc[key]

contentDic['WordCount'] =countList #add WordCount

col\_tfidf.update({'\_id':contentDic['\_id']},contentDic,True)

print "WordCount success"

**// 현재 기사 doc에 countList를 WordCount의 이름으로 추가한후 데이터베이스에 업데이트한다.**

#----print WordCount of input id

docs= col\_tfidf.find()

for doc in docs:

if(obj\_id== str(doc['\_id']) ):

for i in doc['WordCount']:

print "%s : %d"%(i,doc['WordCount'][i])

return 0;

print "There is no Object ID %s"%(obj\_id)

return -1

**//입력받은 obj\_id의 WordCount를 출력한다.**

### Tf-idf

#### 기능

저장한 WordCount 값을 이용해 단어

들마다 문제 정의 부분에서 설명한 TF-IDF 값을 계산한다.

#### 전체 코드 분석

def TfIdf(docs,col\_tfidf):

print "TF-IDF"

obj\_id=raw\_input("After WordCount,Input Object ID to start TF-IDF: ")

**//obj\_id를 입력받는다**

#----------------------find idf-----

docslen=0

idf\_divider={}

idf={}

for doc in docs:

docslen+=1

**//총 기사의 수를 카운트한다.**

for key in doc['WordCount']:

idf\_divider[key]=idf\_divider.get(key,0)+1

**//idf\_divider에 각 key에 대한 WordCount의 횟수를 합산한 값을 저장한다.**

for key in idf\_divider:

idf[key]=math.log(float(docslen)/idf\_divider[key])

**//각 key에대한 idf값을 저장한다.**

#------------------------------find tf and tf-idf and save in db

docs=col\_tfidf.find()

for doc in docs:

countList=doc['WordCount']

tfidfList={}

countSum=sum(countList.values())

for key,Count in countList.items():

tf=float(Count)/countSum

tfidfList[key]=tf\*idf[key]

contentDic={}

for key in doc.keys():

contentDic[key]=doc[key]

contentDic['TF-IDF'] =tfidfList

col\_tfidf.update({'\_id':contentDic['\_id']},contentDic,True)

**//WordCount를 불러와 이전에 저장한idf와 현재 기사의 tf값을 계산하여 tf-idf값을 저장한 후, 그 값을 데이터베이스에 업데이트한다.**

#-------------print------------------

docs= col\_tfidf.find()

for doc in docs:

if(obj\_id== str(doc['\_id']) ):

count=0

sorted\_tfidf=sorted(doc['TF-IDF'].items(),key=lambda t:t[1],reverse=True)

count=0

for i in sorted\_tfidf:

if(count<10):

print "%s : %.12f"%(i[0],i[1]);count+=1

else:

break

return 0;

print "There is no Object ID %s"%(obj\_id)

return -1

**// 상위 10개단어의 tf-idf를 출력한다.**

### Simillarity

#### 기능

두 뉴스기사의 Object ID를 입력 받고 데이터베이스로부터 관련된 문서들을 찾은 후 TFIDF

가중치를 이용해 두 문서의 유사도를 구하는 프로그램

#### 코드 분석

def Similarity(docs,col\_tfidf):

print "Similarity"

id\_1=raw\_input("After TF-IDF,Input First ObjectID :")

id\_2=raw\_input("Input Second ObjectID to start calculate Similarity:")

**//비교할 두 obj\_id를 입력받는다**

for doc in docs:

if(id\_1==str(doc['\_id'])):

tfidf\_1=doc['TF-IDF']

if(id\_2==str(doc['\_id'])):

tfidf\_2=doc['TF-IDF']

**//입력받은 두 obj\_id에 대한 기사를 찾아 저장한다.**

#----------------make list

for t in tfidf\_1:

if t not in tfidf\_2 :

tfidf\_2[t]=0.0

for t in tfidf\_2:

if t not in tfidf\_1 :

tfidf\_1[t]=0.0

**//두 기사에서 겹치지 않은 tf-idf를 0.0으로 설정하여 저장한다.**

#----------------calculate

ab=0.0

a=0.0

b=0.0

for t in tfidf\_1:

ab+=tfidf\_1[t]\*tfidf\_2[t]

a+=tfidf\_1[t]\*\*2

b+=tfidf\_2[t]\*\*2

a=math.sqrt(a)

b=math.sqrt(b)

similarity=ab/(a\*b)

print "Similarity is %f"%(similarity)

**//유사도를 요구사항에서 설명한 식으로 계산하여 출력한다.**

### CopyData

#### 기능

Data를 복사.

#### 코드 분석

def CopyData(docs,col\_tfidf):

col\_tfidf.drop()

for doc in docs:

contentDic={}

for key in doc.keys():

if key!="\_id":

contentDic[key]=doc[key]

col\_tfidf.insert(contentDic)

conn=MongoClient('localhost')

db=conn['db20141494']

db.authenticate('db20141494','db20141494')

col= db['news']

col\_tfidf=db['news\_tfidf']

### printMenu

#### 기능

메뉴 출력

#### 코드 분석

def printMenu():

print "1. WordCount"

print "2. TF-IDF"

print "3. Similarity"

print "4. MorpAnalysis"

print "5. CopyData"

### main

#### 기능

입력 받은 메뉴를 실행

#### 코드 분석

**/\*메인\*/**

printMenu()

selector=input()

docs=col\_tfidf.find()

if selector==1:

WordCount(docs,col\_tfidf);

elif selector==2:

TfIdf(docs,col\_tfidf);

elif selector==3:

Similarity(docs,col\_tfidf);

elif selector==4:

MorpAnalysis(docs,col\_tfidf);

elif selector==5:

docs=col.find()

CopyData(docs,col\_tfidf)