그리기이(가) 표시된 사진

자동 생성된 설명

그리기이(가) 표시된 사진

자동 생성된 설명

**학기말 프로젝트 표절 검사기 보고서**

컴퓨터학부 심화컴퓨터전공

학번 : 202011184 이름 : 신찬규

목차

1. 문제 정의하기 2

2. 순서도 그리기 4

3. 의사 코드 그리기 5

4. 코드 작성하기 8

1) 코드 작성하기 8

2) 프로그램 실행 결과 11

5. 프로그램 사용법 및 실행 시 유의 사항 12

6. 프로젝트를 진행하며 배운 점과 어려웠던 점 12

**1. 문제 정의하기**

입력으로 문서 파일을 최대 5개 받을 수 있고, 출력은 표절이 가장 의심되는 문서 2개와 문서 간의 유사도를 퍼센트로 계산하여 보여주어야 한다. 각 입력 파일은 최대 1,000자의 영문으로 이루어져 있다. 표절은 두산백과에 따르면 *다른 사람이 창작한 저작물의 일부 또는 전부를 도용하여 사용하여 자신의 창작물인 것 처럼 발표하는 것*을 말한다. 즉 두 문서의 유사도가 높으면 표절로 의심할 수 있다. 그러므로 문서들에 대한 유사도를 구하면 문제를 해결할 수 있다. 문서들에 대한 유사도를 구하는 방법으로 문서의 중요한 단어들을 추출하여 그 단어들의 비슷한 정도를 계산하는 방법을 사용할 것이다.

1) 분해

1. 문서들을 프로그램 내에 불러와야 한다.

2. 각 문서의 단어들을 추출해야 한다.

3. 추출한 단어들 중 중요한 단어들을 찾아야 한다.

4. 각 문서의 중요한 단어들에 대한 유사도를 구해야 한다.

5. 유사도가 가장 높은 문서 2개와 문서 간의 유사도를 출력해야 한다.

2) 패턴 인식

문서들을 프로그램 내로 불러온 후 각 문서에 대한 유사도를 구할 때 유사도를 구하는 방법은 모두 같으므로 유사도를 계산하는 함수를 만든 후 각 문서에 적용하면 쉽게 해결 할 수 있을 것이다.

3) 추상화

검사 정확도를 높이기 위해 문서의 단어들 중 의미가 없는 단어인 불용어를 제거해야 한다. 또한 대문자와 소문자로 의미가 구별되는 것을 막기 위해 모든 단어를 소문자로 바꾸어야 한다.

4) 알고리즘

문서의 중요한 단어를 추출하기 위해 이에 대한 알고리즘을 찾던 중 TF-IDF(Term Frequency-Inverse Document Frequency)이라는 알고리즘을 통해 문서를 대표하는 단어들의 중요도를 수치로 표현할 수 있다는 것을 알게 되어 이 알고리즘을 선택하게 되었다. *TF-IDF는 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다. TF(term frequency)는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값으로, 이 값이 높을수록 문서에서 중요하다고 생각할 수 있다. 하지만 단어 자체가 문서군 내에서 자주 사용되는 경우, 이것은 그 단어가 흔하게 등장한다는 것을 의미한다. 이것을 DF(문서 빈도, document frequency)라고 하며, 이 값의 역수를 IDF(역문서 빈도, inverse document frequency)라고 한다.*[[1]](#footnote-1)TF-IDF는 TF와 IDF를 곱한 값이다. TF와 IDF는 다음과 같은 식으로 표현할 수 있다.

TF-IDF의 유사도를 구하는 알고리즘 중 코사인 유사도를 선택했는데 코사인 유사도는 다른 유사도 알고리즘과 다르게 벡터의 크기가 아니라 벡터의 방향, 즉 패턴에 초점을 두기 때문에 문서의 길이가 다른 상황에서 비교적 공정하게 비교할 수 있어 이 알고리즘을 선택하게 되었다. *코사인 유사도는 내적 공간의 두 벡터 간 각도의 코사인값을 이용하여 측정된 벡터 간의 유사도 정도를 의미한다. 각도가 0°일 때의 코사인값은 1이며, 다른 모든 각도의 코사인값은 1보다 작다. 따라서 이 값은 벡터의 크기가 아닌 방향의 유사도를 판단하는 목적으로 사용되며, 두 벡터의 방향이 완전히 같을 경우 1, 90°의 각을 이룰 경우 0, 180°로 완전히 반대 방향인 경우 -1의 값을 갖는다. 이때 벡터의 크기는 값에 아무런 영향을 미치지 않는다*.[[2]](#footnote-2) 두 벡터 A, B에 대해서 코사인 유사도는 다음과 같은 식으로 표현할 수 있다.

즉, 문서의 중요한 단어를 추출하여 TF-IDF 값을 구한 후 TF-IDF 벡터를 만들어 그 벡터들에 대한 코사인 유사도를 계산하는 알고리즘을 사용할 것이다.

**2. 순서도 그리기**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**3. 의사 코드 그리기**

def doc\_tokenize(doc\_name)

fp = open(doc\_name)

word\_list = word\_tokenize(string)

word\_list = [word for word in word\_list if word not in stop\_words]

for I in range(len(word\_list)

if word\_list[i].islower() = False

word\_list[i] = word\_list[i].lower()

return word\_list

def tf(list)

tf\_dict = {}

for word in word\_zip

if word in list

tf\_dict[word] = list.count[word]

else

tf\_dict[word] = 0

return rf\_dict

def tf\_idf(list):

tf\_idf\_dict = {}

tf\_dict = tf(list)

for word in tf\_dict.keys()

tf\_idf\_dict[word] = tf\_dict[word] \* idf\_dict[word]

return sorted(tf\_idf\_dict.items())

def cos\_similarity(doc\_1\_name, doc\_2\_name)

doc\_1 = tf\_idf(doc\_tokenize(doc\_1\_name))

doc\_2 = tf\_idf(doc\_tokenize(doc\_2\_name))

vector\_1 = []

vector\_2 = []

for value in doc\_1

vector\_1.append(value[1])

for value in doc\_2

vector\_2.append(value[1])

return round((dot(vector\_1, vector\_2) / norm(vector\_1) \* norm(vector\_2)\*100

while true

try

doc\_count = input()

if doc\_count % 1 != 0

raise NotIntegerError()

doc\_count = int(doc\_count)

doc\_name\_list = []

i = 0

while i < doc\_count

doc\_name = input()

if os.path.isfile(doc\_name)

doc\_name\_list.append(doc\_name)

i += 1

else

print(“Please enter the name of an existing document.”)

break

except ValueError

print(“Please enter the number.”)

except NotIntegerError

print(“Please enter the integer.”)

stop\_words = set(stopwords.words(‘english’)

doc\_zip = []

for name in range(len(doc\_zip))

for word in doc\_zip[i]

if word in word\_zip

continue

else

word\_zip.append(word)

idf\_dict = {}

for word in word\_zip

word\_count = 0

for doc in doc\_zip

if word in doc

word\_count += 1

idf\_dict[word] = np.log((1 + doc\_count) / word\_count)

similarity\_dict = {}

for i in range(len(doc\_name\_list)-1)

for j in range(i+1, doc\_count)

similarity\_dict[(doc\_name\_list[i], doc\_name\_list[j])] = cos\_similarity(doc\_name\_list[i], doc\_name\_list[j])

key\_min = max(similarity\_dict.keys(), key = lambda x : similarity\_dict[x])

value\_min = max(similarity\_dict.values())

print(f”The similarity between {key\_min[0]} and {key\_min[1]} is highest at {value\_min}%)”

**4.**

1) 코드 작성하기

import os.path  
import numpy as np  
from nltk.corpus import stopwords  
from nltk.tokenize import word\_tokenize  
from numpy import dot  
from numpy.linalg import norm  
class NotIntegerError(Exception):  
 pass  
# 문서를 불러와 단어로 토큰화 후, 단어들을 word\_list에 저장후 word\_list 반환  
def doc\_tokenize(doc\_name):  
 with open(doc\_name, 'rt') as fp:  
 string = fp.read()  
 word\_list = word\_tokenize(string)  
 # 유사도 계산시 정확성을 높이기 위해 큰 의미가 없는 단어인 불용어를 word\_list에서 제거  
 word\_list = [word for word in word\_list if word not in stop\_words]  
 # 소문자와 대문자로 인해 의미 구별이 되는 것을 방지하기 위해, 모든 단어를 소문자화  
 word\_list = [word.lower() if word.islower() == False else word for word in word\_list]  
 return word\_list  
# list안 word의 term frequency 값 계산 후 dict 형태로 반환  
def tf(list):  
 tf\_dict = {word : list.count(word) if word in list else 0 for word in word\_zip}  
 return tf\_dict  
# list안 word의 tf 값과 idf 값을 곱하여 tf-idf 값 계산 후 알파벳 순으로 정렬하여 list 원소가 (word, tf-idf) 형식을 가진 list 형태로 반환  
def tf\_idf(list):  
 tf\_dict = tf(list)  
 tf\_idf\_dict = {word : tf\_dict[word] \* idf\_dict[word] for word in tf\_dict.keys()}  
 return sorted(tf\_idf\_dict.items())  
# doc\_1과 doc\_2 문서의 cosine 유사도를 계산 후 유사도 값을 반환  
def cos\_similarity(doc\_1\_name, doc\_2\_name):  
 # doc\_1과 doc\_2 문서의 tf-idf값 계산  
 doc\_1 = tf\_idf(doc\_tokenize(doc\_1\_name))  
 doc\_2 = tf\_idf(doc\_tokenize(doc\_2\_name))  
 # doc\_1의 word의 tf-idf 값을 vactor\_1에 할당  
 vector\_1 = [value[1] for value in doc\_1]  
 # doc\_2의 word의 tf-idf 값을 vactor\_2에 할당  
 vector\_2 = [value[1] for value in doc\_2]  
 # vector\_1과 vector\_2 사이의 각도를 구한후 100을 곱하여 % 수치로 반환, 소숫점 2자리까지 반올림  
 return round((dot(vector\_1, vector\_2) / (norm(vector\_1) \* norm(vector\_2)))\*100, 2)  
while True:  
 try:  
 # 문서 수 입력  
 doc\_count = float(input('Please enter the count of documents : '))  
 if doc\_count % 1 != 0:  
 raise NotIntegerError()  
 doc\_count = int(doc\_count)  
 doc\_name\_list = []  
 i = 0  
 while i < doc\_count:  
 doc\_name = input(f'Please enter the name of documents [{i + 1}{"/"}{doc\_count}] : ') + ".txt"  
 # 존재하지 않은 문서 이름을 입력시 다시 입력, 존재하는 문서 입력시 doc\_name\_list에 할당  
 if os.path.isfile(doc\_name):  
 doc\_name\_list.append(doc\_name)  
 i += 1  
 else:  
 print('Please enter the name of an existing document.')  
 break  
 except ValueError:  
 # 문서 수를 입력할 때 숫자를 입력하지 않으면 excpet 발생  
 print('Please enter the number.')  
 except NotIntegerError:  
 # 문서 수를 입력할 때 정수를 입력하지 않으면 excpet 발생  
 print('Please enter the integer.')  
stop\_words = set(stopwords.words('english'))  
# idf 값을 계산하기 위해 모든 문서를 doc\_zip에 할당  
doc\_zip = [doc\_tokenize(name) for name in doc\_name\_list]  
# tf-idf 값을 계산하기 위해 모든 문서의 단어를 중복되지 않게 word\_zip에 할당  
word\_zip = list(set([word for doc in doc\_zip for word in doc]))  
# 각 단어마다 inverse document frequency 값 계산 후 dict에 할당  
idf\_dict = {}  
for word in word\_zip:  
 word\_count = 0  
 for doc in doc\_zip:  
 if word in doc:  
 word\_count += 1  
 idf\_dict[word] = np.log((1 + doc\_count) / (word\_count))  
# 경로 상의 모든 문서의 서로 간의 유사도를 계산 후 similarity\_dict에 저장  
similarity\_dict = {(doc\_name\_list[i], doc\_name\_list[j]) : cos\_similarity(doc\_name\_list[i], doc\_name\_list[j]) for i in range(len(doc\_name\_list)-1) for j in range(i+1, doc\_count)}  
# 유사도가 가장 큰 문서 2개를 계산 후 출력  
key\_min = max(similarity\_dict.keys(), key = lambda x: similarity\_dict[x])  
value\_min = max(similarity\_dict.values())  
print(f"The similarity between {key\_min[0]} and {key\_min[1]} is highest at {value\_min}%")

(1) 비교할 문서의 개수와 이름들을 입력받는다. 이때 문서의 개수를 입력할 때 입력값이 글자이거나 정수가 아니면, 또한 문서의 이름을 입력할 때 입력값에 해당하는 문서가 폴더 내에 존재하지 않으면 오류 메시지를 출력하고 다시 입력을 받는다.

(2) 입력받은 문서의 이름들을 doc\_name\_list에 저장한다.

(3) 입력받은 문서들을 단어들로 토큰화한 후 불용어를 제거하고, 대문자와 소문자로 단어가 구별되지 않게 모든. 단어를 소문자로 바꾸어 단어가 중복되지 않게 word\_zip에 저장한다.

(4) 입력받은 문서들을 단어들로 토큰화한 후 불용어를 제거하고, 대문자와 소문자로 단어가 구별되지 않게 모든 단어를 소문자로 바꾸어 문서마다 하나의 리스트로 묶어 doc\_zip에 저장한다.

(5) word\_zip 안의 단어마다 그 단어가 들어간 doc\_zip 리스트 안의 문서의 개수를 세려 log ((1+문서의 개수)/ (단어가 들어간 문서의 개수))으로 idf 값을 계산하여 (word, idf) 쌍으로 idf\_dict를 만든다.

(6) doc\_name\_list에서 문서 2개를 가져와 각 문서를 단어들로 토큰화한 후 불용어를 제거하고, 대문자와 소문자로 단어가 구별되지 않게 모든 단어를 소문자로 바꾸어 각 문서의 단어마다 그 단어가 속한 문서 내에 같은 단어가 몇 개 존재하는지 세려 TF 값을 계산하여 (word, tf) 쌍으로 tf\_dict를 만든다.

(7) 각 문서의 단어의 TF 값과 IDF 값을 곱하여 TF-IDF 값을 계산하여 (word, tf\_idf) 쌍으로 tf\_idf\_dict를 만든다.

(8) 각 문서의 tf\_idf\_dict의 value들을 vector\_1, vector\_2 리스트에 넣은 후, vector\_1과 vector\_2의 코사인 유사도를 계산한다. 이 값에 100을 곱한 후 소수점 2자리까지 반올림하여 퍼센트 화 하여 similarity\_dict에 (doc\_name\_1, doc\_name\_2, similarity) 쌍으로 저장한다.

(9) 이 과정을 doc\_name\_list 안의 문서들을 combination을 통해 반복하고 similarity\_dict 안의 유사도 중 가장 큰 유사도를 가진 문서 2개와 유사도를 출력한다.

2) 프로그램 실행 결과 스크린샷, 모니터, 앉아있는, 화면이(가) 표시된 사진

자동 생성된 설명

교수님이 올려주신 문서의 표절 검사 결과, 1.txt와 5.txt의 유사도가 약 50%였다. 실행 결과, 1.txt와 5.txt의 유사도가 49.7%로 비슷한 결과를 보여주었다.

실내, 모니터, 스크린샷, 노트북이(가) 표시된 사진

자동 생성된 설명

다른 문서로 bad.txt, good.txt, blinding lights.txt,, circles.txt, honey pie.txt는 팝송의 가사들이다. 이 중 good은 bad의 가사 중 ‘bad’ 단어를 모두 ‘good’ 단어로 교체한 문서이다. 다른 문서는 서로 독립적인 관계이다. 검사 결과 bad.txt와 good.txt의 유사도가 36.68%로 가장 높은 유사도를 보여준다.

**5. 프로그램 사용법 및 실행 시 유의사항**

(1) 표절을 검사할 문서의 수를 입력한다.

(2) 표절을 검사할 문서의 이름들을 입력한다. (이때, 문서의 확장자는 입력하지 않도록 유의한다.)

(3) 문서의 길이가 길면 검사할 때까지 잠시 시간이 소요될 수 있음으로 기다린다.

(4) 가장 표절이 의심되는 문서 2개의 이름과 유사도가 출력된다.

**6. 프로젝트를 진행하며 배운 점과 어려웠던 점**

처음 코드를 짜는 과정에서 단어의 IDF 값을 구할 때 log(문서의 개수/ 단어가 들어가있는 문서의 개수)로 구하였는데 이때 문서의 개수와 단어가 들어가 있는 문서의 개수가 같은 경우, 즉 그 단어가 모든 문서에 있는 경우 log 값이 log1이 되기 때문에 IDF 값이 0이 되어 2개의 문서를 비교할 때 코사인 유사도를 구하는 과정에서 유사도가 0으로 나와 2개의 문서를 비교할 수 없었다. 이를 해결하기 IDF 값을 구할 때 문서의 개수에 1을 더해 값이 0이 되는 것을 방지했다. 이를 통해 버그가 발생했을 시 문제 되는 코드를 찾고 어떠한 방식으로 수정할지 고민해봄으로써 버그를 고치는 과정과 방법을 배울 수 있었다. 처음에는 단어가 들어가 있는 문서의 개수에 1을 더해 보았는데, 검사 결과가 예상 결과와 매우 다른 양상을 띠어 당황했다. 다양한 방법을 시도해 보면서 예상 결과와 비슷한 결과가 나올 때까지 시행착오를 겪은 과정에서 어려움을 느꼈다. 하지만 버그를 고쳤을 때 예상대로 작동하는 프로그램의 모습을 보면서 재미와 뿌듯함을 느꼈다. 여태까지 짧은 코드나 간단한 알고리즘만을 사용했는데, 이 프로젝트를 진행하면서 프로그램의 개발 과정을 직접 체험해 봄으로써 좋은 경험을 했다고 생각하고 한 프로그램을 만들었다는 자신감과 더 다양한 지식을 가지고 있으면 프로그램의 성능을 더 높일 수 있었을 것이라는 아쉬움을 느꼈다. 앞으로 컴퓨터 구조와 알고리즘 및 프로그래밍 언어에 관한 공부를 더욱 열심히 하여 프로그램의 정확도를 높여보고 싶다.

1. 위키백과 참조. [↑](#footnote-ref-1)
2. 위키백과 참조. [↑](#footnote-ref-2)