29장

진급 프로젝트: 문서 유사성 분석

29장 진급 프로젝트:문서 유사성 분석

29.1 문제를 하위 작업으로 나누기

29.2 파일 정보 읽기

29.3 파일에 있는 모든 단어 저장하기

29.4 단어의 빈도 구하기

29.5 유사도 점수를 사용해 두 문서 비교하기

29.6 하나로 합치기

29.7 이 프로그램을 확장할 방법 한 가지

29.8 요약

29 프로젝트 문제

» 텍스트가 들어 있는 두 파일이 주어졌다. 두 파일의 이름을 사용해 문서를 읽고, 두 문서의 내용이얼마나 비슷한지 판단하는 프로그램을 작성하라. 두 문서가 완전히 같으면 1점, 공통되는 단어가 전혀 없으면 0점을 받아야 한다.

» 문제를 보면 몇 가지 결정해야 할 사항이 있다.

- 파일에 있는 구두점을 고려할 것인가 구두점은 제외하고 단어만 고려할 것인가?
- 단어의 순서를 감안할 것인가? 두 파일이 단어는 일치하지만 순서가 다르다면 여전히 같은 파일인가?
- 유사성에 점수를 매길 때 어떤 지표를 사용해 수치를 계산할 것인가?

29.1 문제를 하위 작업으로 나누기

29.1 문제를 하위 작업으로 나누기

» 문제를 다시 읽어보면 각각 완결성이 있는 몇 가지 작업으로 자연스럽게 나눌 수 있다.

1. 파일 이름을 받아서 파일을 열고, 파일 내용을 읽는다.
2. 파일 내용 안에 있는 모든 단어를 얻는다.
3. 각 단어의 빈도를 계산한다. 현 시점에서 단어의 순서는 고려하지 않기로 하자.
4. 유사성을 계산한다.

29.2 파일 정보 읽기

29.2 파일 정보 읽기

» 파일 이름이 주어지면 파이썬 함수를 사용해 파일을 열고, 파일 내용 전체를 문자열에 저장한 다음 그 문자열을 반환한다.

```
def read_text(filename):
    """
    filename: string, 읽을 파일의 이름
    returns: string, 파일의 모든 내용이 들어 있음
    """
    inFile = open(filename, 'r') ---- 파일 이름으로 파일을 여는 파이썬 함수
    line = inFile.read() ---- 파일의 모든 내용을 문자열로 읽어 오는 파이썬 함수
    return line ---- 문자열 반환

text = read_text("sonnet18.txt") ---- 함수 호출
print(text)
```

--- 독스트링

코드 29-1 파일 읽기

29.3 파일에 있는 모든 단어 저장하기

29.3 파일에 있는 모든 단어 저장하기

» 문자열을 단어로 나누는 작업에는 함수를 사용한다. 이 코드는 새줄 문자를 공백으로 바꾸고, 특수 문자를 제거함으로써 입력 문자열을 정리한다.

```
import string ---- 문자열과 관련된 함수들을 가져온다

def find_words(text):
    """
    text: string
    returns: list. 입력 문자열인 text에 들어 있는 단어들이 들어 있다
    """
    text = text.replace("\n", " ") ---- 새줄 문자를 공백으로 바꾼다
    for char in string.punctuation: ---- string에 정의된 기호 문자들을 활용한다
        text = text.replace(char, "") ---- 기호를 빈 문자열로 바꾼다(따라서 기호를 제거하는 효과가 있다)
    words = text.split(" ") ---- 공백을 기준으로 문자열을 분리해 단어들로 이뤄진 리스트로 만든다

    return words ---- 단어의 리스트를 반환한다

words = find_words(text) ---- 함수 호출
```

코드 29-2 문자열에서 단어 찾기

29.4 단어의 빈도 구하기

29.4 단어의 빈도 구하기

» 단어 리스트에 있는 각 단어와 그 단어가 파일에 나타났던 횟수인 빈도를 짝지어 빈도 사전을 만든다.

```
def frequencies(words):
    """
    words: list. 단어들이 들어 있는 리스트
    returns: 입력으로 들어온 단어들의 빈도가 들어 있는 사전
    """
    freq_dict = {} ---- 빈 사전으로 초기화
    for word in words: ---- 리스트에 있는 각 단어를 처리
        if word in freq_dict: ---- 단어가 사전에 이미 들어 있다면…
            freq_dict[word] += 1 ---- 그 단어의 빈도에 1을 더해 사전 항목 갱신
        else: ---- 단어가 사전에 들어 있지 않다면
            freq_dict[word] = 1 ---- 그 단어와 빈도 1을 사전에 새로 추가

    return freq_dict ---- 사전 반환

freq_dict = frequencies(words) ---- 함수 호출
```

코드 29-3 단어의 빈도 사전 만들기

29.5 유사도 점수를 사용해 두 문서 비교하기

29.5 유사도 점수를 사용해 두 문서 비교하기

문서를 비교할 수 있는 간단한 지표를 사용해서 어떤 결과가 나오는지 살펴보자. 다음과 같은 단계를 거치면서 누적 계산을 통해 점수를 계산한다.

- 두 빈도 사전에서 단어를 하나 가져온다(한 문서에 하나씩 가져옴).
- 그 단어가 두 사전에 모두 존재한다면 빈도수의 차이를 누적 값에 더한다. 단어가 한 문서에만 들어 있다면 그 단어의 빈도를 누적 값에 더한다(결국 이는 그 단어가 들어 있는 사전의빈도와 들어 있지 않은 사전의 빈도(한 번도 단어가 나타나지 않았으므로 0) 차이를 더하는것과 같다).
- 이렇게 계산한 누적 값(각 단어의 빈도 차이의 합)을 두 문서의 전체 단어 수로 나눈 값이 차이를 표현하는 점수다

지표를 만들었으므로 그 지표가 올바른지 간단히 검증해봐야 한다.

- 문서가 완전히 같다면 모든 단어의 빈도 차이가 없으므로 두 사전의 빈도수 차이의 합은 0이다. 이를 두 문서의 모든 단어 수로 나누면 0이다.
- 두 문서에 공통인 단어가 하나도 없다면 양 사전의 빈도수 차이의 합이 전체 단어 수와 같을 것이다. 겹치는 단어가 없어서 두 사전의 빈도수 차이의 합이 '한 문서에 나타난 단어의빈도수 합 + 다른 문서에 나타난 단어의 빈도수 합'과 같기 때문이다.
- 이 결과를 두 문서의 모든 단어 수로 나누면 1이 된다. 이러한 결과가 타당하다 할 수 있지만, 문제는 우리가 바라는 것이 두문서가 완전히 같으면 1, 완전히 다르면 0이라는 점이다. 이를 만족시키려면 1에서 결과를 빼면 된다.

29.5 유사도 점수를 사용해 두 문서 비교하기

코드 29-4는 두 사전을 입력으로 받아서 유사도를 계산한다.

- 이 코드는 한 사전의 모든 키에 대해 이터레이션한다. 이때 어떤 사전의 키에 대해 루프를 돌든 관계 없다. 어차피 나중에는 다른 사전의 키에 대해서도 루프를 돌 것이기 때문이다.

한 사전의 키를 쭉 살펴보면서, 그 키가 다른 사전에도 들어가 있는지 검사한다.

- 여기서 각 키와 대응하는 값을 찾는데, 그 값은 단어가 문서에 나온 횟수라는 점을 기억하자. 단어가 두 사전에 모두 들어 있으면 두 빈도 사이의 차이를 계산한다. 단어가 다른 사전에 들어 있지 않으면, 현재 사전에서 빈도를 찾아서 사용한다.

한 사전의 키에 대해 이터레이션을 끝낸 후에는 다른 사전을 검사한다.

- 이미 두 사전에 공통으로 들어간 단어들을 검사했기 때문에, 이번에는 검사 대상 사전에만 들어 있고 반대쪽 사전에는 들어있지 않은 단어들을 찾아야 한다. 그런 단어를 찾으면 그 단어의 빈도를 모두 더한다.

마지막으로 빈도수 차이의 합계를 두 사전의 모든 단어 빈도수 합으로 나눈다. 그리고 문제가 요구하는 유사도 점수의 조건에 맞추기 위해 그 결과를 1에서 뺀다.

29.5 유사도 점수를 사용해 두 문서 비교하기


```
def calculate_similarity(dict1, dict2):
    """
    dict1: 한 문서의 단어 빈도 사전
    dict2: 다른 문서의 단어 빈도 사전
    returns: float, 두 문서의 유사도를 표현하는 값(1이면 같음. 0이면 완전히 다름)
    """
    diff = 0
    total = 0
    for word in dict1.keys(): ---- 한 사전에 들어 있는 모든 단어에 대해 이터레이션
        if word in dict2.keys(): ---- 단어가 두 사전에 모두 들어 있음
            diff += abs(dict1[word] - dict2[word]) ---- 빈도 차이를 더함
        else: ---- 단어가 반대쪽 사전에는 들어 있지 않음
            diff += dict1[word] ---- 단어의 빈도를 더함

    for word in dict2.keys(): ---- 나머지 사전의 모든 단어에 대해 이터레이션
        if word not in dict1.keys(): ---------- 두 사전에 모두 포함된 단어는 이미 검사했으므로,
                                              반대쪽 사전에 들어 있지 않은 단어만 검사
            diff += dict2[word] ---- 단어의
                                    빈도를 더함

    total = sum(dict1.values()) + sum(dict2.values()) ---- 두 사전의 모든 단어 수를 더함
    difference = diff / total ---- 빈도 차이의 합을 전체 단어 수로 나눔
    similar = 1.0 - difference ---- 1에서 difference를 뺌

    return round(similar, 2) ---- 소수점 이하 2자리까지 반올림하고 0부터 1 사이의 점수를 반환
```

코드 29-4
주어진 두 사전의 유사도 계산하기

29.6 하나로 합치기

29.6 하나로 합치기

서로 다른 두 파일에 프로그램을 적용시켜 보기 전에, 먼저 프로그램이 올바른지 간이 검사를 실행해 보자.

- 먼저 같은 파일을 두 문서로 사용해 점수가 1.00이 나오는지 본다.
- 빈 파일과 소네트 파일을 사용해 점수가 0.00이 나오는지 살펴본다.
- 그 후 두 문서로 세익스피어의 'sonnet 18'과 'sonnet 19'를 사용해 본다.

```
text_1 = read_text("sonnet18.txt")
text_2 = read_text("sonnet19.txt")
words_1 = find_words(text_1)
words_2 = find_words(text_2)
freq_dict_1 = frequencies(words_1)
freq_dict_2 = frequencies(words_2)
print(calculate_similarity(freq_dict_1, freq_dict_2))
```

코드 29-5
문서의 유사도를 계산하는 프로그램

- sonnet 18'과 'sonnet 19'에 대해 프로그램을 실행하면 유사도가 0.24로 나온다. 이 값이 0에 가까운 것은 타당하다. 두 문서는 서로 다른 문서이기 때문이다.
- 이 프로그램을 'sonnet 18'과 '변경(summer를 winter로)한 sonnet 18'에 대해 실행하면 유사도가 0.97이 나온다. 두 문서가 거의 같기 때문에 이 결과 또한 타당하다.

29.6 하나로 합치기

» 추가

한글파일(애국가) korea_1.txt 와 korea_2.txt 에 대해서도 비교해보자.

29.7 이 프로그램을 확장할 방법 한 가지

29.7 이 프로그램을 확장할 방법 한 가지

» 이 프로그램을 좀 더 튼튼하게 만들려면 한 단어씩 살펴보는 대신 단어의 쌍을 살펴보게 만들면 된다.

- 파일 내용을 읽은 다음에, 문서에 연속으로 나타나는 단어 쌍(이를 바이그램(bigram)이라 한다)을 찾아서 리스트에 등록한다.
- 단어 대신 바이그램을 살펴보도록 바꾸면 프로그램을 좀더 개선 할 수 있다.
- 한 단어의 빈도를 검사하는 것보다 연달아 나타나는 두 단어의 조합이 어떤지를 비교 하면 문서가 사용하는 언어의 특징을 좀 더 잘 표현할 수 있기 때문이다.
- 이렇게 하면 문서에 대해 좀 더 정확하고 나은 모델을 제공할 수 있다.
- 원한다면 바이그램과 단어를 조합해 유사도를 계산할 수도 있을 것이다.

29.8 요약

29.8 요약

- 재사용할 수 있는 함수를 작성함으로써 코드를 모듈화할 수 있다.
- 리스트를 사용해 여러 원소를 저장할 수 있다.
- 사전을 사용해 어떤 단어와 그 단어의 빈도를 저장할 수 있다.