TF-IDF

TF-IDF(Term Frequency-Inverse Document Frequency)

- 정보 검색과 텍스트 마이닝에서 이용하는 가중치
- 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치
- 문서의 핵심어 추출, 검색 엔진에서 검색 결과의 순위 결정, 문서들 사이의 비슷한 정도를 구하는 등의 용도로 사용 가능
- **TF(Term Frequency)**
  - 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값
  - 값이 높을수록 문서에서 중요하다고 생각할 수 있음
  - **But!** 단어 자체가 문서군 내에서 자주 사용되는 경우, 그 단어가 흔하게 등장한다는 것을 의미 = **DF(문서 빈도, document frequency) = 특정 단어 t가 등장한 문서의 수**
- **IDF(역문서 빈도, Inverse Document Frequency)**
  - DF의 역수 = IDF(역문서 빈도, Inverse Document Frequency)
  - **IDF는 문서군의 성격에 따라 결정됨**
    - ex) '원자'라는 낱말에 대한 IDF 값
      - **일반적인 문서들 사이에서**는 잘 나오지 않기 때문에 **IDF 값이 높아지고 문서의 핵심어가 될 수 있음**
      - **원자에 대한 문서를 모아놓은 경우**에는 '원자'는 상투어가 되어 각 문서들을 세분화하여 구분할 수 있는 다른 낱말들이 높은 가중치를 얻게 됨
  - 전체 문서의 수를 해당 단어를 포함한 문서의 수로 나눈 뒤 로그를 취하여 얻을 수 있음

$$\text{tfidf}(w, d) = \text{tf} \times \left(\log\left(\frac{N+1}{N_w+1}\right) + 1\right)$$

=> TF-IDF는 TF와 IDF를 곱한 값 (단어 빈도와 역문서 빈도의 곱)

> 특정 문서 내에서 단어 빈도가 높을수록, 전체 문서들 중 해당 단어를 포함한 문서가 적을수록 TF-IDF 값 커진다.
>
> **특정 단어를 포함하는 문서들이 많을수록**
>
> **IDF값과 TF-IDF값이 0에 가까워진다.**

파이썬으로 TF-IDF 직접 구현하기

```
import pandas as pd#데이터프레임 사용을 위해from math import log#IDF 계산을 위해from IPython.d
isplay import display

docs = [
    "먹고 싶은 사과",
    "먹고 싶은 바나나",
    "길고 노란 바나나 바나나",
    "저는 과일이 좋아요"
]

vocab = list(set(w for doc in docs for w in doc.split()))# set함수=중복 제거# print(voca
b) # ['노란', '과일이', '먹고', '저는', '싶은', '좋아요', '길고', '바나나', '사과']
vocab.sort()# 가나다 순으로 정렬# print(vocab) # ['노란', '과일이', '먹고', '저는', '싶은', '좋
아요', '길고', '바나나', '사과']

N = len(docs)# 총 문서의 수def tf(t,d):
    return d.count(t)

def idf(t):
    df = 0# 단어 t가 등장하는 문서 개수for doc in docs:
        df += t in doc
    return log(N/(df+1))

def tfidf(t,d):
    return tf(t,d)*idf(t)

# TF 구하기
result = []

# 각 문서에 대해서 아래 연산을 반복for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):# vocab = ['노란', '과일이', '먹고', '저는', '싶은', '좋아요',
'길고', '바나나', '사과']
        t = vocab[j]
        result[-1].append(tf(t,d))# result[-1] => result에서 마지막 요소
tf_ = pd.DataFrame(result, columns=vocab)
print(tf_)

# IDF 구하기
result = []
```

```
for j in range(len(vocab)):
    t = vocab[j]
    result.append(idf(t))

idf_ = pd.DataFrame(result, index=vocab, columns=["IDF"])
print(idf_)

# IF-IDF
result = []
for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tfidf(t,d))

tfidf_ = pd.DataFrame(result, columns=vocab)
display(tfidf_.to_string())
```

사이킷런(Scikit-learn)을 이용한 TF-IDF 실습

- 사이킷런(Scikit-learn) = 파이썬 프로그래밍 언어용 자유 소프트웨어 기계 학습 라이브러리
- 사이킷런은 TF-IDF 계산해주는 TfidfVectorizer 제공
- 사이킷런의 TF-IDF는 보편적인 TF-IDF 기본식에서 조정된 식 사용
- IDF의 로그항의 분자에 1을 더해주며, 로그항에 1을 더해주고 TF-IDF에 L2 정규화라는 방법으로 값을 조정하는 등의 차이로 TF-IDF가 가진 의도는 여전히 그대로 갖고 있음

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
    'you know I want your love',
    'I like you',
    'what should I do ',
]

tfidfv = TfidfVectorizer().fit(corpus)
print(tfidfv.transform(corpus).toarray())
print(tfidfv.vocabulary_)
```

TF-IDF의 단점

- 순서가 중요한 문제에는 쓰기 힘들다.
- vocabulary가 커지면 커질수록 쓰기 힘들다.

    - vocabulary가 커질수록 벡터 크기가 커지기 때문
- 단어 간의 관계를 표현하지 못한다.

참고글

https://ko.wikipedia.org/wiki/Tf-idf

https://wikidocs.net/31698

https://jiho-ml.com/weekly-nlp-2/