3장 말 잘하는 수학: TF-IDF 벡터

- 단어가 특정 문서에서 또는 말뭉치 전체에서 얼마나 중요한지 측정할 수 있어야 한다.
- 단어의 '중요도'는 검색 엔진에서 사용자가 입력한 검색어들과 관련성이 깊은 문서를 찾는 데 도움이 됨.
- 우리의 목표는 **단어의 중요도** 또는 단어의 정보 내용을 반영한 수치 표현을 찾아내는 것임

▼ 1. 단어 모음

- Bag of words
- 단어 빈도들의 벡터
- 단어 출현 횟수가 어떻게 도움이 되는지?

▼ 관련 코드

```
'A kite is traditionally a tethered heavier-than-air craft with wing surfaces that react against the air to create lift and drag. A kite consists of wings, tethers, and anchors. Kites often have a bridle to guide the face of the kite at the correct angle so the wind can lift it. A kite's wing also may be so designed so a bridle is not needed; when kiting a sailplane for launch, the tether meets the wing at a single point. A kite may have fixed or moving anchors. Untraditionally in technical kiting, a kite consists of tether-set-coupled wing sets; even in technical kiting, though, a wing in the system is still often called the kite.₩n₩nThe lift that sustains the kite in flight is generated when air flows around the kite's surface, producing low pressure above and high pressure below the wings. The interaction with the wind also generates horizontal drag along the direction of the wind. The resultant force vector from the lift and drag force components is opposed by the tension of one or ...'
```

```
stopwords = nltk.corpus.stopwords.words('english')
tokens = [x for x in tokens if x not in stopwords]
kite_counts = Counter(tokens)
kite_counts.most_common()
```

```
[('kite', 16),
 (',', 15),
 ('kites', 8),
 ('wing', 5),
 ('lift', 4),
 ('may', 4),
 ('also', 3),
 ('kiting', 3),
 ('flown', 3),
 ('tethered', 2),
 ('craft', 2),
 ('air', 2),
 ('consists', 2),
 ('tethers', 2),
 ('anchors.', 2),
 ('often', 2),
 ('bridle', 2),
```

- Kite, wing, lift 등이 많이 등장

- 단순한 단어 출현 횟수가 아니라 그것을 문서의 길이로 나눈 **'정규화된 용어 빈도'**로 문서를 표현하면 말뭉치 안에서의 문서의 특징을 좀 더 잘 표현할 수 있음.
- 예시

  TF("dog," documentA) = 3 / 30 = 0.1

  TF("dog," documentB) = 100 / 580,000 = 0.00017

▼ 2. 벡터화

- 자연어에서 벡터화는 왜 필요할까?
  - 사람은 문장에서 단어가 어떤 의미로 사용되었는지 문맥을 통해 바로 구분할 수 있지만 기계는 그렇게 할 수 없다. 그렇기 때문에 단어를 수치로 표현해서 기계가 이해할 수 있도록 해야 한다. **텍스트를 숫자로 표현하는 방법이고 벡터화**한다고 한다.
- 서로 가까운 두 벡터는 서로 "비슷하다"고 할 수 있다. 그리고 두 문서의 벡터 표현들이 비슷하면 두 문서는 비슷하다.
- 그러나 두 문서의 길이가 같다고 해서 그 두 문서가 비슷하다고 여기는 것은 바람직하지 못하다.
  - 대신, 두 문서가 비슷한 단어들을 비슷한 빈도로 사용했는지를 측정하는 것이 더 합리적이다.
- 종류
  - NLP, vector, One-hot encoding, Word-Embedding, BOW, Counter Vector, **TF-IDF**, Word2Vec, CBOW, Skip-gram, Sparse, Dense

▼ 코사인 유사도

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

- -1~1

- 두 벡터가 같은 방향을 가리킬수록 내적은 큰 양수가 되고 반대 방향을 가리킬수록 큰 음수가 된다.
    - 길이가 다를 수 있지만 방향이 정확히 일치하는 것이 중요
    - NLP에서 두 문서 표현 벡터의 코사인 유사도가 1에 가깝다는 것은 두 벡터가 비슷한 단어들을 비슷한 빈도로 사용한다는 뜻
    - 0이라는 것은 공통점이 전혀 없다.
    - 두 벡터와 코사인 유사도가 -1인 또 다른 벡터가 존재한다면, 원래의 두 벡터는 아주 비슷하다는 것.
- 언제 사용할까?
    - 벡터의 크기가 중요하지 않을때 거리를 측정하기 위한 메트릭으로 사용
    - 길이를 정규화해 비교하는 것과 유사하다고 할 수 있으며 이 때문에 텍스트 데이터를 처리하는 메트릭으로 주로 사용
    - 주로 데이터 마이닝이나 정보 검색에서 즐겨 사용

▼ 관련 코드

```
from numpy import dot
from numpy.linalg import norm
import numpy as np
def cos_sim(A, B):
       return dot(A, B)/(norm(A)*norm(B))

doc1=np.array([0,1,1,1])
doc2=np.array([1,0,1,1])
doc3=np.array([2,0,2,2])

print(cos_sim(doc1, doc2)) #문서1과 문서2의 코사인 유사도
print(cos_sim(doc1, doc3)) #문서1과 문서3의 코사인 유사도
print(cos_sim(doc2, doc3)) #문서2과 문서3의 코사인 유사도
```

```
0.6666666666666667
0.6666666666666667
1.0000000000000002
```

▼ 3. 지프의 법칙

- 자연어 말뭉치 표현에 나타나는 단어들을 그 사용 빈도가 높은 순서대로 나열하였을 때, 모든 단어의 사용 빈도는 해당 단어의 순위에 반비례

    - 1위 단어는 2위 단어보다 두 배 더 자주 쓰이고, 3위 단어보다는 세 배 더 자주 쓰인다.
- 브라운 말뭉치는 1961년 브라운 대학교가 만든 영어 말뭉치
    - 1위 단어 “the”는 2위 단어 “of”보다 약 두 배 자주 쓰이고, 3위 단어인 “and”보다는 약 세 배 자주 쓰임.
    - 즉, 말뭉치가 충분히 크다고 할 때 주어진 한 단어가 임의의 한 문서에 몇 번이나 출현할지 예측할 수 있음.

▼ 관련 코드

```
from collections import Counter
puncs = set((',', '.', '--', '-', '!', '?', ':', ';',  '(', ')', '[',']' ))

word_list = (x.lower() for x in brown.words() if x not in puncs)
token_counts = Counter(word_list)
token_counts.most_common(20)
```

```
[('the', 69971),
 ('of', 36412),
 ('and', 28853),
 ('to', 26158),
 ('a', 23195),
 ('in', 21337),
 ('that', 10594),
 ('is', 10109),
 ('was', 9815),
 ('he', 9548),
 ('for', 9489),
 ('``', 8837),
 ("''", 8789),
 ('it', 8760),
 ('with', 7289),
 ('as', 7253),
 ('his', 6996),
 ('on', 6741),
 ('be', 6377),
```

▼ 4. 주제 모형화

- 단어 출현 횟수의 한계는 말뭉치의 나머지 문서들에 상대적인 중요도를 가늠하기 어려움.
- 주어진 단어가 문서에서 얼마나 중요한지 보려면 IDF가 필요
- IDF를 보는 관점은 “이 토큰이 이 문서에 등장한다는 것이 얼마나 이상한 일인가?”로 보자.
    - 만일 어떤 용어가 이상하게도 한 문서에만 자주 등장하고 말뭉치의 나머지 문서들에는 거의 나오지 않는다면, 그 용어는 그 문서에 아주 중요한 단어일 것임.

- IDF는 전체 문서 수를 그 용어가 출현한 문서 수로 나눈 것

▼ **TF-IDF(Term Frequency-Inverse Document Frequency)**

- 특정한 단어 또는 토큰을 특정 말뭉치의 특정 문서에 연관시키는 측도

  좀 더 구체적으로 주어진 문서에서 그 단어가 얼마나 중요한지를 말뭉치 전체에서의 그 단어의 사용 빈도에 기초해서 추정하는 값

- 왜 사용?
  - TF-IDF는 특징 추출, Feature Extraction 기법
  - 머신러닝, 딥러닝 학습시에 변수들을 만들 때, 텍스트 데이터의 특징을 담아서 학습을 시켜줘야 하기 때문에 사용됨.
- tf(t,d) = count(t) / count(d)
  - 한 단어가 문서에 자주 나올수록 그 단어의 tf가 올라간다.
- idf(t,D) = log (전체 문서 수 / t가 나오는 문서 수)
  - 그 단어를 포함한 문서가 많을수록 그 단어의 idf가 내려간다.
- 즉, **특정 문서 내에서 단어 빈도가 높을** 수록, 그리고 **전체 문서들 중 그 단어를 포함한 문서가 적을** 수록 TF-IDF값이 높아진다.
- tfidf(t,d,D) = tf(t,d) * idf(t,D)
- https://chan-lab.tistory.com/24
  - TF-IDF 개념 해부

▼ **챗봇 적용**

- 우리의 최종 목표는 챗봇이다. 대부분의 챗봇은 검색 기능에 크게 의존한다.
- 사용자가 텍스트를 입력하면 챗봇은 그것에 가장 가까운 제시문을 TF-IDF로 찾는다.
- 검색 엔진이라면 그 제시문 자체를 결과로 돌려주겠지만, **챗봇은 그 제시문과 짝을 이루는 응답문을 돌려주어야 한다**
- **이처럼 문장 부호들을 생략하는 등의 정규화를 수행해서 TF-IDF 모형을 미리 최적화하면 파이프라인 이후 단계들의 계산량을 줄일 수 있다. 말뭉치가 크다면 절약 효과도 더욱 커질 것임**

▼ **이후**

- 이후 장들에서는 단순히 질의문에 있는 단어들이 많이 나온 문서들을 찾는 것이 아니라 질의문의 단어들에 담긴 '의미'를 담고 있을 가능성이 큰 문서들을 찾는 방식의 의미론적 검색 엔진을 구현하는 방법을 살펴볼 것임.
- TF-IDF 벡터를 아무리 세심하게 정규화하고 평활화해도 주제 벡터보다 단어 모음의 의미를 잘 표현하지는 못한다.
    - word2vec 벡터와 그 이후 장들에서 논의하는 단어 및 문서 의미의 신경망 내장은 문서의 의미를 주제 벡터보다 더 잘 표현