기본

04. 카운트 기반의 단어표현®

Count word based Representation

12기 이나윤 발표

차례®

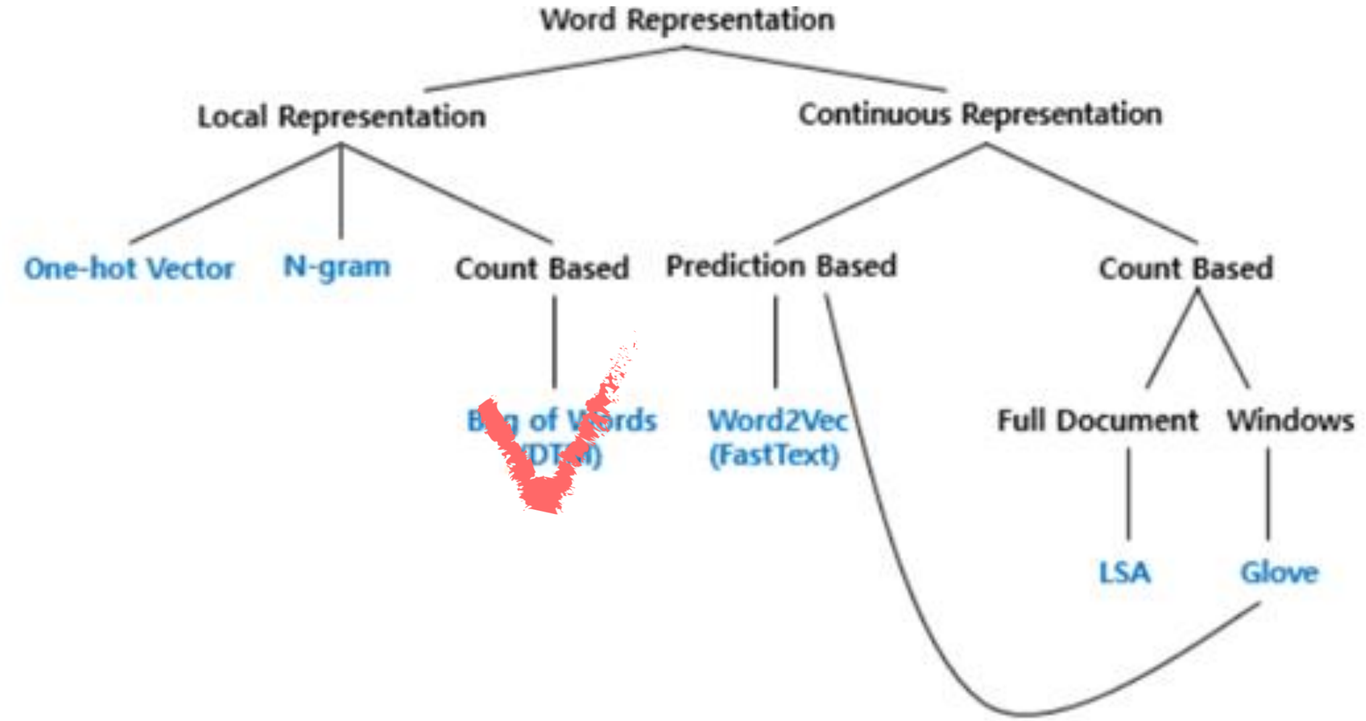
04. 카운트 기반의 단어표현방법 no.

01

다양한 단어의 표현 방법

다양한 단어의 표현 방법

- **국소(이산) 표현 (Local Representation)** :해당 단어 자체만 보고 단어표현.
- **분산 표현(Distributed Representation):** 주변을 참고하여 단어 표현. 뉘앙스 표현 가능 o



02

Bag of Words(BoW)

Bag of Words

: 단어의 등장 순서는 고려하지 않고, **출현 빈도(카운트)**를 중심으로 텍스트 데이터를 **수치화**하는 방법

: 분류 문제, 여러 문서 간의 유사도에 활용.

BoW 만드는 과정

1) 각 단어에 고유한 **정수 인덱스** 부여

2) 각 인덱스의 위치에 단어 토큰의 **등장 횟수**를 기록한 벡터를 생성.

02

Bag of Words(BoW)

```
from konlpy.tag import Okt
import re
okt=Okt()

token=re.sub("(ㅜ?)","","우리 집 강아지는 동생을 좋아할까, 간식을 더 좋아할까?")
token=okt.morphs(token)

word2index={}
bow=[]
for voca in token:
         if voca not in word2index.keys():
             word2index[voca]=len(word2index)
             bow.insert(len(word2index)-1,1)
         else:
             index=word2index.get(voca)
             bow[index]=bow[index]+1

print(word2index)
bow
```

```
{'우리': 0, '집': 1, '강아지': 2, '는': 3, '동생': 4, '을': 5, '좋아할까': 6,
',': 7, '간식': 8, '더': 9}

[1, 1, 1, 1, 1, 2, 2, 1, 1, 1]
```

02

Bag of Words(BoW)

CountVectorizer

: 길이가 2 이상인 문자들에 대해서만, 띄어쓰기 기준으로 단어를 자름.

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ['Let it go, let it go. Can not hold it back anymore.']
vector = CountVectorizer()
print(vector.fit_transform(corpus).toarray())
print(vector.vocabulary_)
```

```
[[1 1 1 2 1 3 2 1]]
{'let': 6, 'it': 5, 'go': 3, 'can': 2, 'not': 7, 'hold': 4, 'back': 1, 'anymor
e': 0}
```

```
from sklearn.feature_extraction.text import CountVectorizer

text= ['Let it go, let it go. Can not hold it back anymore.']
vect = CountVectorizer(stop_words="english")   NLTK
print(vect.fit_transform(text).toarray())
print(vect.vocabulary_)
```

```
[[1 1 2]]
{'let': 2, 'hold': 1, 'anymore': 0}
```

03

DTM(Document-Term Matrix)

문서 단어 행렬(DTM)

BoW 표현 방법을 바탕으로, 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것.

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

이를 문서 단어 행렬로 표현하면 다음과 같습니다.

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

03

DTM(Document-Term Matrix)

문서 단어 행렬(DTM)의 한계점

1) 희소표현

: 대부분의 값이 0인 표현을 희소 벡터, 희소 행렬

: 많은 양의 저장 공간, 계산을 위한 리소스를 필요로 함.

-> DTM 전, **텍스트 전처리**를 통해 단어 정규화 필요.

2) 단순 빈도수 기반 접근

Ex) The, a 공통적으로 많더라도, 유사한 문서 아닐 수 있음.

-> **TF-IDF** 중요한 단어에 대해 가중치를 부여.

04

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF(Term Frequency-Inverse Document Frequency)

: DTM 내 각 단어에 대한 중요도를 가중치로 주는 방법.
: 모든 문서에서 자주 등장하는 단어는 중요도가 낮고,
특정 문서에서만 자주 등장하는 단어의 중요도가 높다고 본다.

-tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장횟수 (DTM이 각 단어의 tf가 된다.)

-df(t): 특정 단어 t가 등장한 문서의 수

-idf(d,t): df(t)에 반비례하는 수

$$idf(d,t) = log(\frac{n}{1+df(t)}) \text{ +1}$$

04 TF-IDF (Term Frequency-Inverse Document Frequency)

- **파이썬으로 TF-IDF 구현**

```
import pandas as pd
from math import log
docs = [
  '유튜브 접속 끊겼다',
  '유튜브 서버 터졌다',
  '유튜브 강의 못 듣는다',
  '블랙보드 서버 강의 괜찮다'
]
vocab = list(set(w for doc in docs for w in doc.split()))
#set으로 설정해 중복하는 것 하나만 셈. 그 후 리스트로 변환
vocab.sort()
```

```
N = len(docs)
def tf(t, d):
    return d.count(t)
def idf(t):
    df = 0
    for doc in docs:
        df += t in doc
    return log(N/(df + 1))
def tfidf(t, d):
    return tf(t,d)* idf(t)
```

```
result = []
for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tf(t, d))
# 하나씩 각 docs에서 특정 단어의 출현 횟수를 result에 append
tf_ = pd.DataFrame(result, columns = vocab)
tf_
```

3]:

	강의	괜찮다	끊겼다	듣는다	못	블랙보드	서버	유튜브	접속	터졌다
0	0	0	1	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1	1	0	1
2	1	0	0	1	1	0	0	1	0	0
3	1	1	0	0	0	1	1	0	0	0

04 TF-IDF (Term Frequency-Inverse Document Frequency)

```
result = []
for j in range(len(vocab)):
    t = vocab[j]
    result.append(idf(t))
idf_ = pd.DataFrame(result, index = vocab, columns = ["IDF"])
idf_
# log(N/(df + 1))을 반환하는 idf함수를 활용하여, result에 append
```

]:

	IDF
강의	0.287682
괜찮다	0.693147
끊겼다	0.693147
듣는다	0.693147
못	0.693147
블랙보드	0.693147
서버	0.287682
유튜브	0.000000
접속	0.693147
터졌다	0.693147

In [35]:

```
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)
tfidf_
```

Out[35]:

	강의	괜찮다	끊겼다	듣는다	못	블랙보드	서버	유튜브	접속	터졌다
0	0.000000	0.000000	0.693147	0.000000	0.000000	0.000000	0.000000	0.0	0.693147	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.287682	0.0	0.000000	0.693147
2	0.287682	0.000000	0.000000	0.693147	0.693147	0.000000	0.000000	0.0	0.000000	0.000000
3	0.287682	0.693147	0.000000	0.000000	0.000000	0.693147	0.287682	0.0	0.000000	0.000000

04 TF-IDF (Term Frequency-Inverse Document Frequency)

- **sklearn (TfidfVectorizer) 으로 TF-IDF 구현 : log 분자에 1 더해주고, 로그항에 1더해주고, TF-IDF에 L2정규화**

  - L2 : 벡터의 각 원소의 제곱의 합이 1이 되도록 만드는 것 (default, Euclidean Distance)
  - L1 : 벡터의 각 원소의 절댓값의 합이 1이 되도록 크기를 조절 (Manhattan Distance)

$$x_{norm} = \frac{x}{\|x\|_2}$$

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + \ldots + x_m^2}$$

04 TF-IDF (Term Frequency-Inverse Document Frequency)

- **sklearn (TfidfVectorizer) 으로 TF-IDF 구현 : log 분자에 1 더해주고, 로그항에 1더해주고, TF-IDF에 L2정규화**

```
#사이킷런을 통한 DTM, TF-IDF 만들기.
#(log 분자에 1을 더해주고, 로그항에 1을 더해준다, TF-IDF에 L2정규화)

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
    'you know I want your love',
    'I like you',
    'what should I do ',
]
vector = CountVectorizer()
print(vector.fit_transform(corpus).toarray()) # 각 단어의 빈도 수를 기록한다. #DTM
print(vector.vocabulary_) # 각 단어의 인덱스 어떻게 부여되었는가

tfidfv = TfidfVectorizer().fit(corpus) #TI-IDF
print(tfidfv.transform(corpus).toarray())
```

```
[[0 1 0 1 0 1 0 1 1]
 [0 0 1 0 0 0 0 1 0]
 [1 0 0 0 1 0 1 0 0]]
{'you': 7, 'know': 1, 'want': 5, 'your': 8, 'love': 3, 'like': 2, 'what': 6, 'should': 4, 'do': 0}
[[0.         0.46735098 0.         0.46735098 0.         0.46735098
  0.         0.35543247 0.46735098]
 [0.         0.         0.79596054 0.         0.         0.
  0.         0.60534851 0.        ]
 [0.57735027 0.         0.         0.         0.57735027 0.
  0.57735027 0.         0.        ]]
```

감사합니다.®

Count word based Representation