# Chapter 4. 카운트 기반의 단어 표현 (Count based word Representation)

초급 4조 기대현, 조다희, 홍다솔

목차

# 1) 다양한 단어의 표현 방법

## 1. 단어의 표현 방법

**국소 표현(Local Representation) 방법;** 이산 표현 (Discrete Representation)

:해당 단어 그 자체만 보고, 특정값을 맵핑하여 단어를 표현하는 방법 (단어의 뉘앙스를 표현할 수 없음)

**분산 표현(Distributed Representation);** 연속 표현 (Continuous Representation)

: 단어를 표현하기 위해 주변 단어를 참고 (단어의 뉘앙스를 표현할 수 있음)

**puppy(강아지)** → 1

**cute(귀여운)** → 2

**lovely(사랑스러운)** → 3

puppy(강아지), cute(귀여운), lovely(사랑스러운) 상관관계가 존재

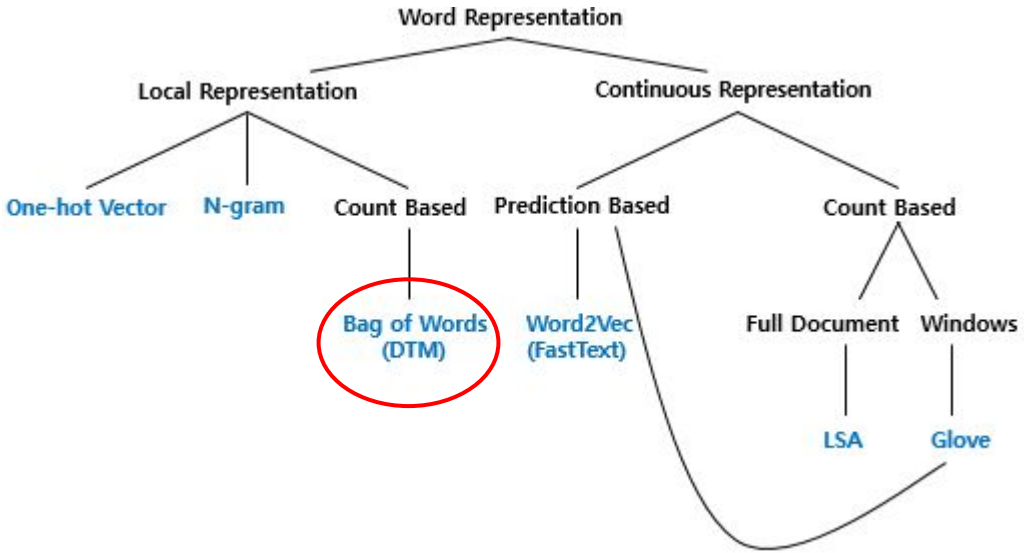
→ puppy(강아지)라는 단어 근처에는 주로 cute(귀여운), lovely(사랑스러운)이라는 단어가 자주 등장,

→ puppy라는 단어는 cute, lovely한 느낌이다로 단어를 정의

https://wikidocs.net/31767

# 1) 다양한 단어의 표현 방법

## 2. 단어 표현의 카테고리화




https://wikidocs.net/31767

# 2) Bag of Words(BoW)

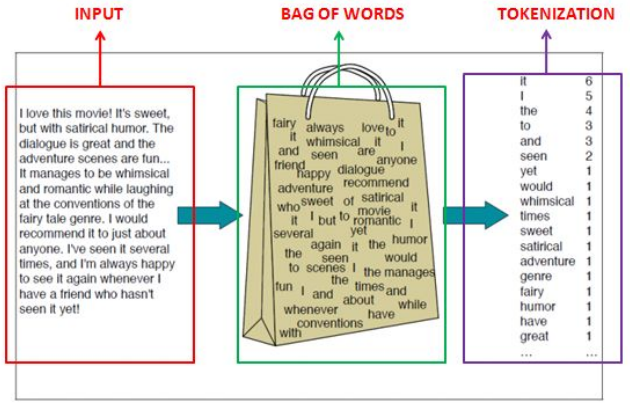
## 1. Bag of Words란?

**Bag of Words**란 단어들의 순서는 전혀 고려하지 않고, 단어들의 **출현 빈도(frequency)에만 집중하는 텍스트 데이터의 수치화 표현 방법**

**BoW를 만드는 과정**

(1) 우선, 각 단어에 고유한 정수 인덱스를 부여합니다.

(2) 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터를 




이미지 출처: https://blog.floydhub.com/naive-bayes-for-machine-learning/

https://wikidocs.net/22650

# 2) Bag of Words(BoW)

## 1. Bag of Words란?

문서1 : 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.

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

token=re.sub("(\.)","","정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.")
# 정규 표현식을 통해 온점을 제거하는 정제 작업입니다.
token=okt.morphs(token)
# OKT 형태소 분석기를 통해 토큰화 작업을 수행한 뒤에, token에다가 넣습니다.

word2index={}
bow=[]
for voca in token:
         if voca not in word2index.keys():
             word2index[voca]=len(word2index)
# token을 읽으면서, word2index에 없는 (not in) 단어는 새로 추가하고, 이미 있는 단어는 넘깁니다.
             bow.insert(len(word2index)-1,1)
# BoW 전체에 전부 기본값 1을 넣어줍니다. 단어의 개수는 최소 1개 이상이기 때문입니다.
         else:
            index=word2index.get(voca)
# 재등장하는 단어의 인덱스를 받아옵니다.
             bow[index]=bow[index]+1
# 재등장한 단어는 해당하는 인덱스의 위치에 1을 더해줍니다. (단어의 개수를 세는 것입니다.)
print(word2index)
```

```
('정부': 0, '가': 1, '발표': 2, '하는': 3, '물가상승률': 4, '과': 5, '소비자': 6, '느끼는': 7, '은': 8, '다르다': 9)
```

```
bow
```

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

https://wikidocs.net/22650

# 2) Bag of Words(BoW)

## 2. Bag of Words의 다른 예제들

문서1 : 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.

```
# ('발표': 0, '가': 1, '정부': 2, '하는': 3, '소비자': 4, '과': 5, '물가상승률': 6, '느끼는': 7, '은': 8, '다르다': 9)
[1, 2, 1, 1, 1, 1, 2, 1, 1, 1]
```

문서2 : 소비자는 주로 소비하는 상품을 기준으로 물가상승률을 느낀다.

```
('소비자': 0, '는': 1, '주로': 2, '소비': 3, '하는': 4, '상품': 5, '을': 6, '기준': 7, '으로': 8, '물가상승률': 9, '느낀다': 10)
[1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1]
```

문서3: 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다. 소비자는 주로 소비하는 상품을 기준으로 물가상승률을 느낀다.

```
('정부': 0, '가': 1, '발표': 2, '하는': 3, '물가상승률': 4, '과': 5, '소비자': 6, '느끼는': 7, '은': 8, '다르다': 9, '는': 10, '주로': 11, '소비': 12, '상품': 13, '을': 14, '기준': 15, '으로': 16, '느낀다': 17)
[1, 2, 1, 2, 3, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1]
```

https://wikidocs.net/22650

# 2) Bag of Words(BoW)

## 3. CountVectorizer 클래스로 BoW 만들기

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ['you know I want your love. because I love you.']
vector = CountVectorizer()
print(vector.fit_transform(corpus).toarray()) # 코퍼스로부터 각 단어의 빈도 수를 기록한다.
print(vector.vocabulary_) # 각 단어의 인덱스가 어떻게 부여되었는지를 보여준다.
```

```
[[1 1 2 1 2 1]]
{'you': 4, 'know': 1, 'want': 3, 'your': 5, 'love': 2, 'because': 0}
```

https://wikidocs.net/22650

# 2) Bag of Words(BoW)

## 4. 불용어를 제거한 BoW 만들기

### (1) 사용자가 직접 정의한 불용어 사용

불용어는 자연어 처리에서 별로 의미를 갖지 않는 단어. BoW를 만들때 불용어를 제거하는 일은 자연어 처리의 정확도를 높이기 위해서 선택할 수 있는 전처리 기법

```
from sklearn.feature_extraction.text import CountVectorizer

text=["Family is not an important thing. It's everything."]
vect = CountVectorizer(stop_words=["the", "a", "an", "is", "not"])
print(vect.fit_transform(text).toarray())
print(vect.vocabulary_)
```

```
[[1 1 1 1 1]]
{'family': 1, 'important': 2, 'thing': 4, 'it': 3, 'everything': 0}
```

https://wikidocs.net/22650

# 2) Bag of Words(BoW)

## 4. 불용어를 제거한 BoW 만들기

### (2) CountVectorizer에서 제공하는 자체 불용어 사용

불용어는 자연어 처리에서 별로 의미를 갖지 않는 단어. BoW를 만들때 불용어를 제거하는 일은 자연어 처리의 정확도를 높이기 위해서 선택할 수 있는 전처리 기법

```
from sklearn.feature_extraction.text import CountVectorizer

text=["Family is not an important thing. It's everything."]
vect = CountVectorizer(stop_words="english")
print(vect.fit_transform(text).toarray())
print(vect.vocabulary_)
```

```
[[1 1 1]]
{'family': 0, 'important': 1, 'thing': 2}
```

https://wikidocs.net/22650

# 2) Bag of Words(BoW)

## 4. 불용어를 제거한 BoW 만들기

### (3) NLTK에서 지원하는 불용어 사용

불용어는 자연어 처리에서 별로 의미를 갖지 않는 단어. BoW를 만들때 불용어를 제거하는 일은 자연어 처리의 정확도를 높이기 위해서 선택할 수 있는 전처리 기법

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords

text=["Family is not an important thing. It's everything."]
sw = stopwords.words("english")
vect = CountVectorizer(stop_words =sw)
print(vect.fit_transform(text).toarray())
print(vect.vocabulary_)
```

```
[[1 1 1 1]]
{'family': 1, 'important': 2, 'thing': 3, 'everything': 0}
```

https://wikidocs.net/22650

## 3) 문서 단어 행렬(Document-Term Matrix, DTM)

문서 단어 행렬: 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것

예)

문서1: 먹고 싶은 사과

문서2: 먹고 싶은 바나나

문서3: 길고 노란 바나나 바나나

문서4: 저는 과일이 좋아요

## 3) 문서 단어 행렬(Document-Term Matrix, DTM)

| - | 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 |

## 3) 문서 단어 행렬(Document-Term Matrix, DTM)

BUT 문서 단어 행렬의 **한계**는?

### 1) 희소 표현(Sparse representation)

원-핫 벡터의 경우와 마찬가지로 많은 문서 벡터가 대부분 0의 값을 가질 수 있음 **=>** *공간적 낭비와 계산 리소스 증가*

이와 같이 대부분의 값이 0인 표현을 **희소 벡터(Sparse Vector)** 혹은 **희소 행렬(Sparse Matrix)**라고 지칭

**=>** *많은 양의 저장공간과 계산을 위한 리소스 필요*

## 3) 문서 단어 행렬(Document-Term Matrix, DTM)

BUT 문서 단어 행렬의 **한계**는?

**2) 단순히 빈도 수를 기반으로 접근할 때**

ex) 영어로 DTM을 만들었을 때 불용어인 the는 자주 등장할 수 밖에 없음

=> 문서1, 2, 3에서 the의 빈도 수가 높다고 하여

이 문서들이 유사하다고 볼 수 X

## 4) TF-IDF(Term Frequency-Inverse Document Frequency)

- **TF-IDF란?**

여러 문서들이 있을 때, 어떤 단어가 특정 문서 내에서 얼마나 중요한지를 나타내는 통계적 수치

문서의 핵심어를 추출, 검색 엔진에서 검색 결과의 순위를 결정, 문서들 간 유사도 측정

$$tf\text{-}idf(t, d, D) = tf(t, d) \times idf(t, D)$$

## 4) TF-IDF(Term Frequency-Inverse Document Frequency)

- **TF(d, t)**

특정 문서 d에서의 특정 단어 t의 등장 횟수

ex) TF(‘나는 오늘 점심에 전복집에서 전복죽을 먹었습니다’, ‘전복') = 2

특정 문서 내에 찾고자 하는 키워드가 많이 등장한다면, 그 특정 문서와 내가 찾고자 하는 키워드의 관련성이 높다.

DTM에서 각 단어들이 가진 값이 바로 TF!


TFO

## 4) TF-IDF(Term Frequency-Inverse Document Frequency)

$$df(t,d) = \frac{|\{d \in D : t \in d\}|}{|D|} = \frac{\text{단어 } t\text{가 포함된 문서의 수}}{\text{전체문서의 수}}$$

- DF(t, D)란?

한 단어가 전체 문서 집합 내에서 얼마나 공통적으로 많이 등장하는지를 나타내는 값

## 4) TF-IDF(Term Frequency-Inverse Document Frequency)

$$idf(t, D) = \log\left(\frac{|D|}{1 + |\{d \in D : t \in d\}|}\right) = \log\left(\frac{\text{전체문서의 수}}{1 + \text{단어 } t\text{가 포함된 문서의 수}}\right)$$

IDF는 DF에 역수를 취해준 것

df값이 클 수록(=모든 문서에 흔히 등장하는 단어일 수록) tf-idf의 가중치 값을 낮춰주기 위해, df에 역수를 취하는 것

단순히 역수만 취하면, 전체 문서의 수가 많아질수록 idf의 값이 기하급수적으로 커지게 되므로, idf에 로그를 취해주는 것이 일반적

분모가 0이 되는 것을 방지해주기 위해 1을 더해준다.

# 4) TF-IDF(Term Frequency-Inverse Document Frequency)

```
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()))
vocab.sort()
```

```
N = len(docs) # 총 문서의 수

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)
```

# 4) TF-IDF(Term Frequency-Inverse Document Frequency)

```
result = []
for i in range(N): # 각 문서에 대해서 아래 명령을 수행
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tf(t, d))

tf_ = pd.DataFrame(result, columns = vocab)
tf_
```

| | 과일이 | 길고 | 노란 | 먹고 | 바나나 | 사과 | 싶은 | 저는 | 좋아요 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| **2** | 0 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 |
| **3** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

# 4) 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_
```

| | IDF |
|---|---|
| 과일이 | 0.693147 |
| 길고 | 0.693147 |
| 노란 | 0.693147 |
| 먹고 | 0.287682 |
| 바나나 | 0.287682 |
| 사과 | 0.693147 |
| 싶은 | 0.287682 |
| 저는 | 0.693147 |
| 좋아요 | 0.693147 |

# 4) TF-IDF(Term Frequency-Inverse Document Frequency)

```
for word in tf_.columns:
    tf_[word] *= idf_.loc[word, 'IDF']
tf_idf = tf_
```

| | 과일이 | 길고 | 노란 | 먹고 | 바나나 | 사과 | 싶은 | 저는 | 좋아요 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.000000 | 0.000000 | 0.000000 | 0.287682 | 0.000000 | 0.693147 | 0.287682 | 0.000000 | 0.000000 |
| **1** | 0.000000 | 0.000000 | 0.000000 | 0.287682 | 0.287682 | 0.000000 | 0.287682 | 0.000000 | 0.000000 |
| **2** | 0.000000 | 0.693147 | 0.693147 | 0.000000 | 0.575364 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **3** | 0.693147 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.693147 | 0.693147 |

# 4) TF-IDF(Term Frequency-Inverse Document Frequency)

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
  '먹고 싶은 사과',
  '먹고 싶은 바나나',
  '길고 노란 바나나 바나나',
  '저는 과일이 좋아요'
]
tfidfv = TfidfVectorizer().fit(corpus)
pd.DataFrame(tfidfv.transform(corpus).toarray())
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00000 | 0.00000 | 0.00000 | 0.526405 | 0.00000 | 0.667679 | 0.526405 | 0.00000 | 0.00000 |
| 1 | 0.00000 | 0.00000 | 0.00000 | 0.577350 | 0.57735 | 0.000000 | 0.577350 | 0.00000 | 0.00000 |
| 2 | 0.00000 | 0.47212 | 0.47212 | 0.000000 | 0.74445 | 0.000000 | 0.000000 | 0.00000 | 0.00000 |
| 3 | 0.57735 | 0.00000 | 0.00000 | 0.000000 | 0.00000 | 0.000000 | 0.000000 | 0.57735 | 0.57735 |

## 4) TF-IDF(Term Frequency-Inverse Document Frequency)

- **보충**

그냥 빈도수만 계산하더라도, tf-idf는 나름 쓸만하지만, tf값을 정규화 시키면 더 나은 성능을 기대할 수 있다!

1. Boolean Frequency
2. Logarithmically Scaled Frequency

$$tf(t,d) = \log(\, f(t,d) + 1 \,)$$

3. Augmented Frequency

$$tf(t,d) \; = 0.5 + 0.5 \cdot \frac{freq(t,d)}{(\text{문서 내 단어들의 } freq(t,d) \text{값 중 최대 값})}$$