

임베딩(embedding)

- 사람이 사용하는 언어를 컴퓨터가 이해할 수 있는 언어(숫자) 형태인 벡터로 변환한 결과 또는 과정을 의미함

임베딩 역할

- 단어 및 문장 간 관련성 계산
- 의미적 혹은 문법적 정보의 함축(예. 왕-여왕, 교사-학생)

임베딩 방법

- 희소 표현 기반 임베딩
- 횃수 기반 임베딩
- 예측 기반 임베딩
- 회수/예측 기반 임베딩

1. 희소 표현 기반 임베딩

- 문장의 단어들을 원-핫 인코딩 형식으로 표현
- 벡터의 길이는 어휘 사전의 크기와 같고, 값은 특정 단어를 나타내는 한 위치만 1이 되고 나머지는 모두 0



원-핫 인코딩 단점

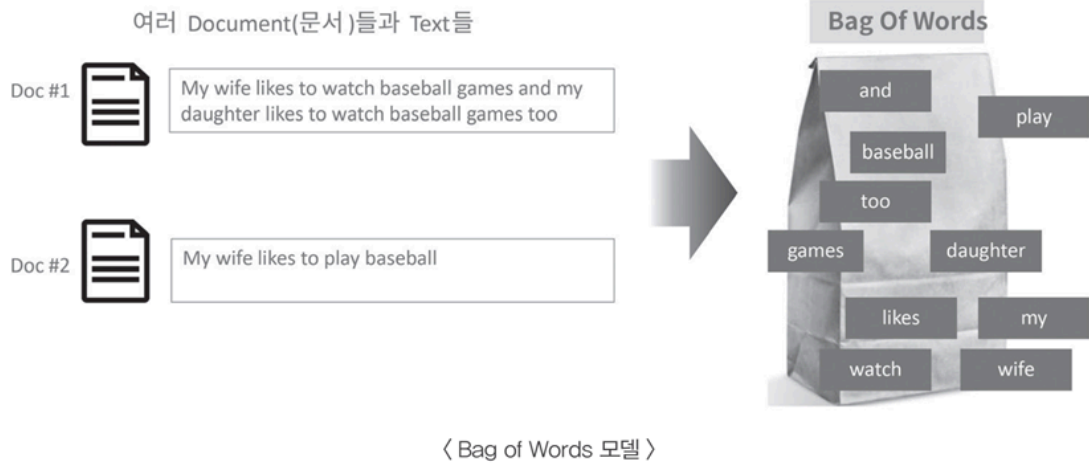
- 단어간의 관계성(유의어, 반의어) 없이 서로 독립적인 관계가 됨
 - 하나의 요소만 1 나머지는 0인 희소벡터
 - 두 단어에 대한 벡터의 내적(inner product)가 0이 됨
- 차원의 저주 문제 발생
 - 하나의 단어를 표현하는데 말뭉치에 있는 수만큼 차원이 존재

2. 횃수 기반 임베딩

Bag of Words 모델

- 문서가 가지는 모든 단어를 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피쳐 값을 추출하는 모델
- 문서 내 모든 단어를 한꺼번에 Bag 안에 넣은 뒤 흔들어서 섞는다는 의미

Bag of Words의 단어 수 기반으로 피쳐 추출 사례



BOW 단계

1. 문장1(Doc #1)과 문장2(Doc #2)에 있는 모든 단어에서 중복을 제거하고, 각 단어(feature 또는 term)를 칼럼 형태로 나열한 뒤, 각 단어에 고유의 인덱스를 부여한다.
2. 개별 문장에서 해당 단어가 나타나는 횟수(Occurrence)를 각 단어(단어 인덱스)에 기재한다.

	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6	Index 7	Index 8	Index 9	Index 10
	and	baseball	daughter	games	likes	my	play	to	too	watch	wife
문장 1	1	2	1	2	2	2		2	1	2	1
문장 2		1			1	1	1	1			1

→ 문장 1에서 baseball은 2회 나타남

BOW 모델의 장점

- 쉽고 빠른 구축
- 단어의 발생횟수에 기반

BOW 모델의 단점

- **문맥 의미(Semantic Context) 반영 부족:**
 - 단어의 순서를 고려하지 않기 때문에 문장 내에서 단어의 문맥적인 의미가 무시됨
 - 보완 방법 : n-gram 기법을 활용
- **희소행렬 문제(희소성, 희소행렬):**
 - BOW로 피쳐 벡터화를 수행하면 희소 행렬 형태의 데이터 세트가 만들어지기 쉬움
 - 문서별 단어 수가 수만 ~ 수십만 개가 될 수 있으며, 한 문서에 있는 단어는 이 중 극히 일부분이므로 대부분의 값이 0으로 채워짐
 - 희소행렬은 ML 알고리즘의 수행 시간과 예측 성능을 떨어뜨림

BOW 피쳐 벡터화

- 모든 문서에서 모든 단어를 칼럼 형태로 나열하고 각 문서에서 해당 단어의 횟수나 정규화된 빈도를 값으로 부여하는 데이터 세트 모델로 변경하는 것



BOW 피쳐 벡터화 방식


카운트 기반의 벡터화

- 단어 피쳐에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수(Count)를 부여하는 경우
- 카운트 값이 높을수록 중요한 단어로 인식
- 카운트만 부여할 경우 그 문서의 특징을 나타내기보다는 언어의 특성상 문장에서 자주 사용될 수 밖에 없는 단어까지 높은 값을 부여하게 됨

TF-IDF(Term Frequency - Inverse Document Frequency) 기반의 벡터화

- 문장에서 자주 사용되는 단어들을 모두 높은 값의 카운트를 가지게 되는 단점 보완
- 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 패널티를 주는 방식
- 특정 문서 내에서 단어의 출현 빈도가 높거나 전체 문서에서 특정 단어가 포함된 문서가 적을수록 TF-IDF 값이 높음
- TF-IDF 사용하는 상황들
 - 키워드 검색을 기반으로 하는 검색 엔진
 - 중요 키워드 분석
 - 검색 엔진에서 검색 결과의 순위 결정


TF-IDF 방식에서 단어별 빈도 계산식



한 개의 문서(Document)

Term Frequency

The	Matrix	is	nothing	but	an	advertising	gimmick
40	5	50	12	20	45	3	2



모든 문서들(Corpus)

Document Frequency

The	Matrix	is	nothing	but	an	advertising	gimmick
2000	190	2300	500	1200	3000	52	12

$$TFIDF_i = TF_i * \log \frac{N}{DF_i}$$

TF_i = 개별 문서에서의 단어 i 빈도
 DF_i = 단어 i를 가지고 있는 문서 개수
 N = 전체 문서 개수

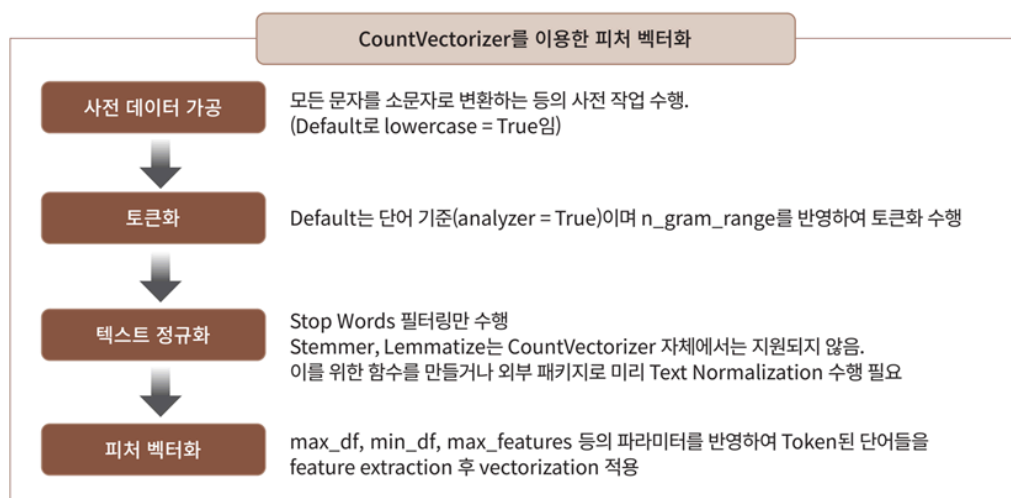
사이킷런의 Count 벡터화 클래스 CountVectorizer

- `sklearn.feature_extraction.text.CountVectorizer`
- 카운트 기반의 벡터화 구현 클래스
- 피처 벡터화만 수행하지 않고, 소문자 일괄 변환, 토큰화, 스톱 워드 필터링 등의 텍스트 전처리도 함께 수행
- 텍스트 전처리 및 피처 벡터화를 위한 입력 파라미터를 설정해 동작함
- `fit()`, `transform()` 통해 피처 벡터화된 객체를 반환함
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

```
CountVectorizer(*, input='content', encoding='utf-8', decode_error='strict',
                strip_accents=None, lowercase=True, preprocessor=None,
                tokenizer=None, stop_words=None, token_pattern='(?u)\b\w\w+\b',
                ngram_range=(1, 1), analyzer='word', max_df=1.0, min_df=1,
                max_features=None, vocabulary=None, binary=False,
                dtype=<class 'numpy.int64'>)
```

CountVertorizer()의 파라미터

파라미터 명	파라미터 설명
max_df	전체 문서에 걸쳐서 너무 높은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 너무 높은 빈도수를 가지는 단어는 스톱 워드와 비슷한 문법적인 특성으로 반복적인 단어일 가능성이 높기에 이를 제거하기 위해 사용됩니다. max_df = 100과 같이 정수 값을 가지면 전체 문서에 걸쳐 100개 이하로 나타나는 단어만 피처로 추출합니다. Max_df = 0.95와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐 빈도수 0~95%까지의 단어만 피처로 추출하고 나머지 상위 5%는 피처로 추출하지 않습니다.
min_df	전체 문서에 걸쳐서 너무 낮은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터입니다. 수백~수천 개의 전체 문서에서 특정 단어가 min_df에 설정된 값보다 적은 빈도수를 가진다면 이 단어는 크게 중요하지 않거나 가비지(garbage)성 단어일 확률이 높습니다. min_df = 2와 같이 정수 값을 가지면 전체 문서에 걸쳐서 2번 이하로 나타나는 단어는 피처로 추출하지 않습니다. min_df = 0.02와 같이 부동소수점 값(0.0 ~ 1.0)을 가지면 전체 문서에 걸쳐서 하위 2% 이하의 빈도수를 가지는 단어는 피처로 추출하지 않습니다.
max_features	추출하는 피처의 개수를 제한하며 정수로 값을 지정합니다. 가령 max_features = 2000으로 지정할 경우 가장 높은 빈도를 가지는 단어 순으로 정렬해 2000개까지만 피처로 추출합니다.
stop_words	'english'로 지정하면 영어의 스톱 워드로 지정된 단어는 추출에서 제외합니다.
n_gram_range	Bag of Words 모델의 단어 순서를 어느 정도 보강하기 위한 n_gram 범위를 설정합니다. 튜플 형태로 (범위 최솟값, 범위 최댓값)을 지정합니다. 예를 들어 (1, 1)로 지정하면 토큰화된 단어를 1개씩 피처로 추출합니다. (1, 2)로 지정하면 토큰화된 단어를 1개씩(minimum 1), 그리고 순서대로 2개씩(maximum 2) 묶어서 피처로 추출합니다.
analyzer	피처 추출을 수행한 단위를 지정합니다. 당연히 디폴트는 'word'입니다. Word가 아니라 character의 특정 범위를 피처로 만드는 특정한 경우 등을 적용할 때 사용됩니다.
token_pattern	토큰화를 수행하는 정규 표현식 패턴을 지정합니다. 디폴트 값은 '\b\w+\b'로, 공백 또는 개행 문자 등으로 구분된 단어 분리자(\b) 사이의 2문자(문자 또는 숫자, 즉 영숫자) 이상의 단어(word)를 토큰으로 분리합니다. analyzer= 'word'로 설정했을 때만 변경 가능하나 디폴트 값을 변경할 경우는 거의 발생하지 않습니다.
tokenizer	토큰화를 별도의 커스텀 함수로 이용시 적용합니다. 일반적으로 CountTokenizer 클래스에서 어근 변환 시 이를 수행하는 별도의 함수를 tokenizer 파라미터에 적용하면 됩니다.



사이킷런의 TF-IDF 벡터화 클래스 `TfidfVectorizer`

- 파라미터와 사용 방법은 CountVectorizer와 동일

- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
TfidfVectorizer(*, input='content', encoding='utf-8', decode_error='strict',
strip_accents=None, lowercase=True, preprocessor=None,
tokenizer=None, analyzer='word', stop_words=None,
token_pattern='(?u)\b\w+\b', ngram_range=(1, 1), max_df=1.0,
min_df=1,
max_features=None, vocabulary=None, binary=False,
type=<class 'numpy.float64'>,
norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False)
```

BOW 피처벡터의 희소 행렬 표현 방식

- 희소 행렬을 물리적으로 적은 메모리 공간을 차지하도록 변환
 - CSR 형식 : 더 많이 사용
 - COO 형식
- CountVectorizer/TfidfVectorizer는 피처벡터들을 CSR 형태의 희소 행렬로 반환

←

수십만 개의 칼럼

→

	단어 1	단어 2	단어 3	단어 1000	단어 2000	단어 10000	단어 20000	단어 100000
수천 ~ 수만개 레코드	문서1	1	2	2	0	0	0	0	0	0	0	0	0
	문서2	0	0	1	0	0	1	0	0	1	0	0	1
	문서
	문서 10000	0	1	3	0	0	0	0	0	0	0	0	0

BOW의 Vectorization 모델은 너무 많은 0값이 메모리 공간에 할당되어 많은 메모리 공간이 필요하며
연산 시에도 데이터 액세스를 위한 많은 시간이 소모됩니다.

COO(Coordinate) 형식의 희소행렬

- 0이 아닌 데이터만 별도의 데이터 배열에 저장하고,
- 데이터가 가리키는 행과 열의 위치를 별도의 배열로 저장하는 방식
- 예. [[3, 0, 1], [0, 2, 0]]
 - 0이 아닌 데이터 : [3, 1, 2]
 - 0이 아닌 데이터의 위치(row, col) : (0,0), (0, 2), (1, 1)

Dense 형식의 원본 데이터

```
[ [0,0,1,0,0,5],
  [1,4,0,3,2,5],
  [0,6,0,3,0,0],
  [2,0,0,0,0,0],
  [0,0,0,7,0,8],
  [1,0,0,0,0,0]]
```

0이 아닌 데이터 값 배열

```
[1, 5, 1, 4, 3, 2, 5, 6, 3, 2, 7, 8, 1]
```

0이 아닌 데이터 값의 행과 열 위치

```
(0, 2), (0,5)
(1, 0), (1, 1), (1, 3), (1, 4), (1, 5)
(2, 1), (2, 3)
(3, 0)
(4, 3), (4, 5)
(5, 0)
```

행 위치 배열

```
[0, 0, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5]
```

열 위치 배열

```
[2, 5, 0, 1, 3, 4, 5, 1, 3, 0, 3, 5, 0]
```

CSR(Compressed Sparse Row) 형식의 희소 행렬

- 0이 아닌 위치를 나타내는 행 위치 배열에서 고유한 값의 시작 위치만 별도의 위치 배열로 가지도록 변환
- 이때 총 항목수도 함께 저장

