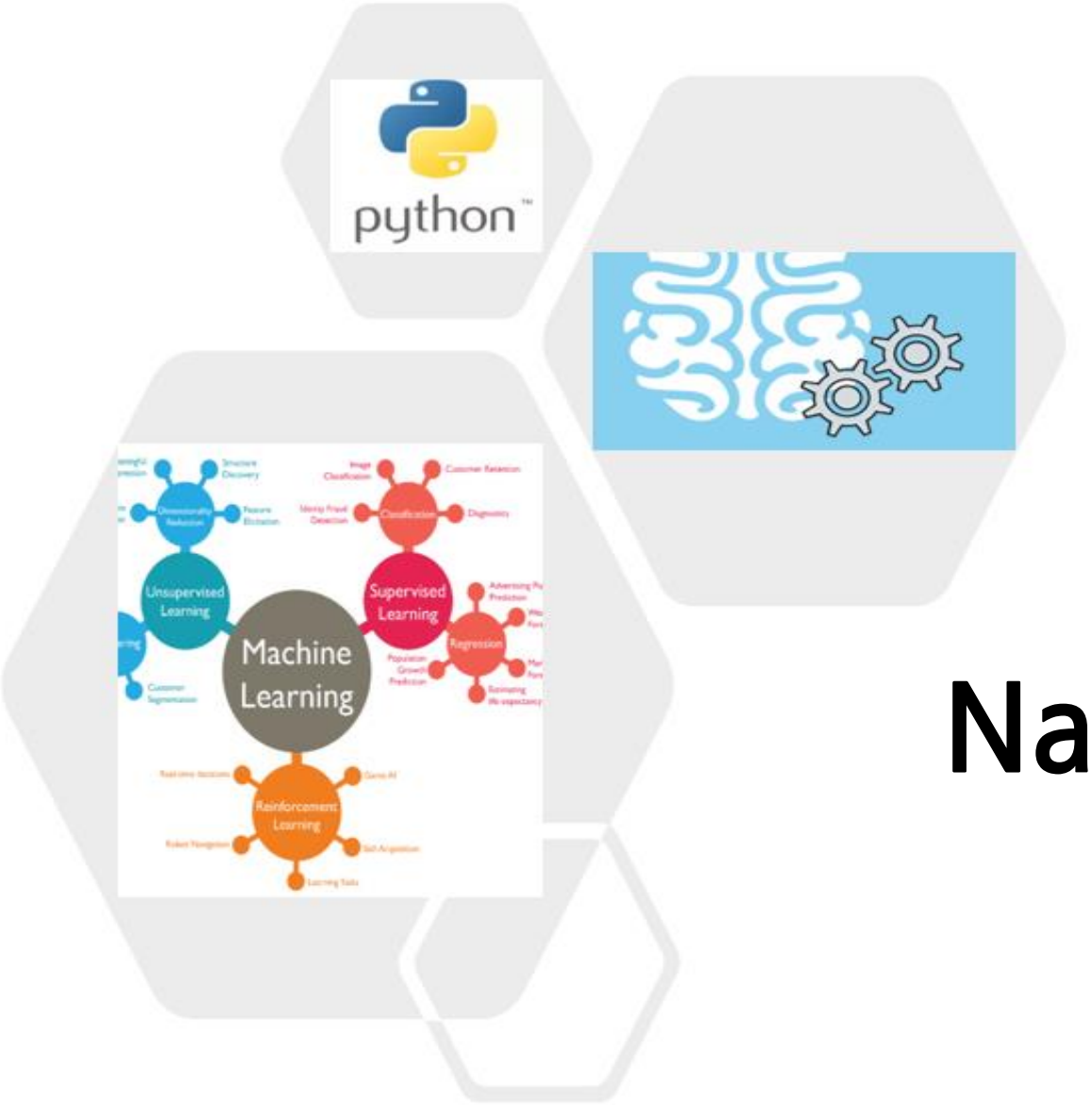


자연어처리 Natural Language Processing

김진숙

KMR 한국경영인증원

자연어 처리 개요

자연어 처리 개요

- 단어 표현
- 자연어 처리 문제
  - 텍스트 분류
  - 텍스트 유사도
  - 텍스트 생성
  - 기계 이해

단어 표현

- 어떻게 자연어를 컴퓨터에게 인식시킬 수 있을까?

- 컴퓨터가 문자를 인식하는 방법
  - 유니코드, 아스키코드
    - 0과 1로만 구성된 값으로 인식
    - 자연어 처리를 위해 만든 모델에는 부적합

- 단어 표현(Word Representation)
  - 텍스트를 자연어 처리를 위한 모델에 적용할 수 있게 언어적인 특성을 반영해서 단어를 수치화하는 방법을 찾는 것
  - 단어를 수치화 할 때는 단어를 주로 벡터로 표현
    → 단어 임베딩(word embedding), 단어 벡터(word vector)
  - 원-핫 인코딩(one-hot encoding), 분포 가설(Distributed Hypothesis)

단어 표현

- **1) 원-핫 인코딩(one-hot encoding)**
  - 단어를 하나의 벡터로 표현하는 방법
  - 벡터 값 중 하나만 1의 값을 가지고 나머지는 0 값을 가지는 방식
  - 각 단어의 벡터에서 그 단어에 해당하는 인덱스의 값을 1로 표현하는 방식

- **BOW(Bag of Word)모델**
  (예) 6개의 단어 (남자, 여자, 아빠, 엄마, 삼촌, 이모)
  남자: [1, 0, 0, 0, 0, 0], 아빠: [0, 0 , 1, 0, 0, 0]

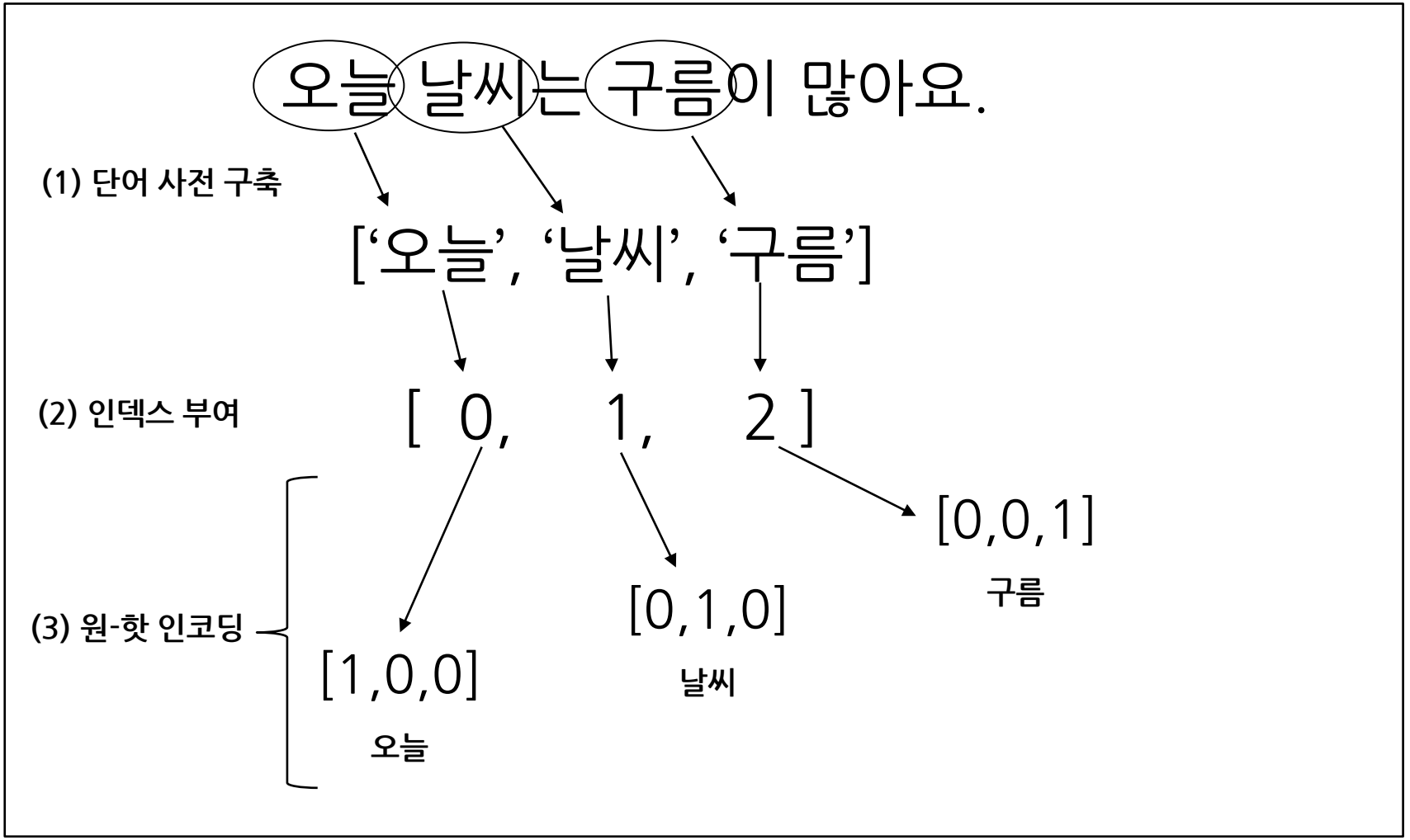
  - **장점**
    - 방법이 매우 간단하고 이해하기 쉽다.

  - **단점**
    - **1) 희소 행렬 문제(희소성, 희소행렬)**
      - 많은 문서에서 단어를 추출하면 매우 많은 단어가 컬럼으로 만들어지게 된다.
      - 단어의 총 개수는 수만~수십만개가 되고, 하나의 문서에 있는 단어는 이 중 극히 일부분이므로 대부분 데이터는 0 값으로 채워지게 된다. → 희소 행렬(sparse matrix)

    - **2) 문맥 의미(Sematic Context) 반영 부족**
      - 단어의 순서를 고려하지 않기 때문에 문장 내에서 단어의 문맥적인 의미가 무시된다.

단어 표현

- 1) 원-핫 인코딩(one-hot encoding)
  - 원-핫 인코딩 과정



단어 표현

- 1) 원-핫 인코딩(one-hot encoding)

  - Python 이용

```
from konlpy.tag import Komoran
import numpy as np

komoran = Komoran()
text = "오늘 날씨는 구름이 많아요."

# 명사만 추출
nouns = komoran.nouns(text)
print(nouns)
```

```
['오늘', '날씨', '구름']
```

```
# 단어 사전 구축 및 단어별 인덱스 부여
dics = {}
for word in nouns:
    if word not in dics.keys():
        dics[word] = len(dics)
print(dics)
```

```
{'오늘': 0, '날씨': 1, '구름': 2}
```

- np.eye() : 단위 행렬 생성

```
# 원-핫 인코딩
nb_classes = len(dics)
targets = list(dics.values())
one_hot_targets = np.eye(nb_classes)[targets]
print(one_hot_targets)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

단어 표현

- **BOW(Bag Of Word) 모델**
  - 문서가 가지는 모든 단어(Word)를 문맥이나 순서를 무시하고 일괄적으로 단어에 대한 빈도값을 부여하여 피처 값을 추출해 내는 모델
  - 문서 내의 모든 단어를 한꺼번에 봉투(Bag) 안에 넣은 뒤에 흔들어 섞는다는 의미로 BOW 모델이라고 함

단어 표현

- BOW(Bag Of Word) 모델
- Bag of Words의 단어 수 기반으로 피처 추출
  - 문장1:
    'My wife likes to watch baseball games and my daughter likes to watch baseball game too'
  - 문장2:
    'My wife likes to play baseball'
- 1) 문장1과 문장2에 있는 모든 단어를 중복을 제거하고 각 단어를 컬럼 형태로 나열하고 각 단어에 고유의 인덱스를 부여
  and:0, baseball:1, daughter:2, games:3, likes:4, my:5, play:6, to:7, too:8, watch:9, wife:10
- 2) 개별 문장에서 해당 단어가 나타나는 횟수를 각 단어에 기재한다.

	Index0	index1	index2	index3	index4	index5	index6	index7	index8	index9	index10
	and	baseball	daughter	games	likes	my	play	to	too	watch	wife
문장1	1	2	1	2	2	2		2	1	2	1
문장2		1			1	1	1	1			1

단어 표현

- BOW(Bag Of Word) 모델의 장단점

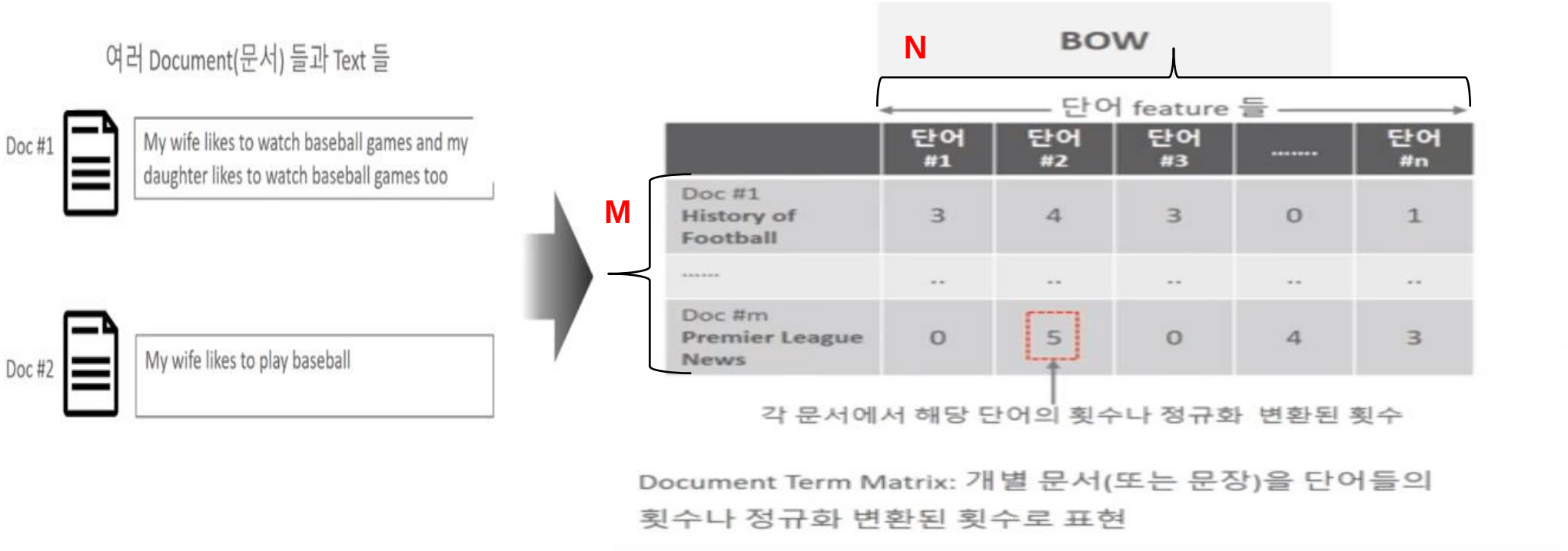
- 장점
  - 쉽고 빠른 구축
  - 단순히 단어의 발생 횟수에 기반하고 있지만, 예상보다 문서의 특징을 잘 나타낼 수 있는 모델

- 단점
  - 1) 문맥 의미(Sematic Context) 반영 부족
    - BOW는 단어의 순서를 고려하지 않기 때문에 문장 내에서 단어의 문맥적인 의미가 무시된다.
    - 이를 보완하기 위해 n-gram 기법을 활용할 수 있지만, 제한적인 부분에 그치므로 언어의 많은 부분을 차지하는 문맥적인 해석을 처리하지 못하는 단점이 있다.

  - 2) 희소 행렬 문제(희소성, 희소행렬)
    - BOW로 피처 벡터화를 수행하면 희소 행렬 형태의 데이터 세트가 만들어짐
    - 많은 문서에서 단어를 추출하면 매우 많은 단어가 컬럼으로 만들어지게 된다.
    - 단어의 총 개수는 수만~수십만개가 되고, 하나의 문서에 있는 단어는 이 중 극히 일부분이므로 대부분 데이터는 0 값으로 채워지게 된다.
    - 이처럼 대규모 컬럼으로 구성된 행렬에서 대부부의 값이 0 인 행렬을 희소 행렬(sparse matrix)라고 한다.
    - 희소 행렬은 일반적으로 ML 알고리즘의 수행시간과 예측 성능을 떨어뜨리게 된다.

단어 표현

- **BOW 피처 벡터화**
  - 텍스트를 특정 의미를 갖는 숫자인 벡터 값으로 변환 → "피처 벡터화"
  - 모든 문서에서 모든 단어를 컬럼 형태로 나열하고 각 문서에서 해당 단어의 횟수나 정규화 된 빈도 값으로 부여하는 데이터 세트 모델로 변경 하는 것
  - M개의 텍스트 문서, N개의 단어 → M X N 피처 벡터화



단어 표현

- BOW의 피처 벡터화 방식
  - 1) 카운트 기반의 벡터화
  - 2) TF-IDF(Term Frequency - Inverse Document Frequency) 기반의 벡터화

- 카운트 기반의 벡터화
  - 단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수(Count)를 부여
  - 카운트가 높을 수록 중요한 단어로 인식
  - 단점) 카운트만 부여할 경우 그 문서의 특징을 나타내기 보다는 자주 사용되는 단어만 찾게 됨

- TF-IDF(Term Frequency - Inverse Document Frequency) 기반의 벡터화
  - 카운트 기반의 벡터화를 보완
  - 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 패널티를 주는 방식으로 값을 부여
  - 문서마다 텍스트가 길고 문서의 개수가 많을 수록 카운트 방식보다는 TF-IDF 방식이 좋은 예측 성능 보장

- TF-IDF(Term Frequency - Inverse Document Frequency)
  - 특정 단어가 다른 문서에는 나타나지 않고 특정 문서에서만 자주 사용된다면 해당 단어는 해당 문서의 특징을 표현하는 중요한 단어일 가능성이 높음
  - 특정 단어가 여러 문서에서 빈번히 나타난다면 해당 단어는 개별 문서를 특징짓는 정보로서 의미 상실

단어 표현

- TF-IDF(Term Frequency - Inverse Document Frequency)
  - TF(Term Frequency)
    - 문서에서 해당 단어가 얼마나 나왔는지를 나타내는 지표
  - DF(Document Frequency)
    - 해당 단어가 몇 개의 문서에서 나타났는지를 나타내는 지표
  - IDF(Inverse Document Frequency)
    - DF의 역수
    - 전체 문서수/DF

$$\text{TFIDF} = \boldsymbol{TF} * \log\frac{N}{DF}$$

Term Frequency

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

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 및 TF-IDF 벡터화 구현 : CounterVectorizer, TfidfVectorizer
- 사이킷런의 CounterVectorizer
  - 카운트 기반의 벡터화를 구현한 클래스
  - 피처 벡터화 및 소문자 일괄변환, 토큰화, 스톱워드 필터링 등의 텍스트 전처리도 함께 수행
  - 텍스트 전처리 및 피처 벡터화를 위한 입력 파라미터를 설정하여 동작
  - fit(), transform()을 통해 피처 벡터화 된 객체를 반환

단어 표현

- 사이킷런의 CounterVectorizer의 입력 파라미터

파라미터 명	파라미터 설명
max_df	전체 문서에 걸쳐서 너무 높은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터 max_df = 100 이면 100 개 이하로 나타나는 단어만 피처를 추출 max_df = 0.95 이면 전체 문서 빈도수 95%이하의 단어만 피처로 추출
min_df	전체 문서에 걸쳐서 너무 낮은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터 min_df = 2 이면 전체 문서에 걸쳐서 2번 이하로 나타나는 단어는 피처로 추출하지 않음 min_df = 0.02 이면 전체 문서에 걸쳐서 하위 2% 이하의 빈도수를 가지는 단어는 추출하지 않음
max_features	추출하는 피처의 개수를 제한하며 정수로 값을 지정 max_features=2000 이면 가장 높은 빈도를 가진 단어순으로 정렬해 2000개까지만 피처로 추출
stop_words	‘english’로 지정하면 영어의 스톱워드로 지정된 단어는 추출에서 제외한다
n_gram_range	Bag of Words 모델의 단어 순서를 보강하기 위한 n_gram 범위를 설정. 튜플 형태로 (범위최소값 범위최대값)을 지정 (1,1)로 지정하면 토큰화된 단어를 1개씩 피처로 추출 (1,2)로 지정하면 토큰화된 단어를 1개씩, 그리고 순서대로 2개씩 묶어서 피처로 추출
analyzer	피처 추출을 수행한 단위를 지정. 디폴트는 ‘word’
token_pattern	토큰화를 수행하는 정규 표현식 패턴을 지정. 디폴트 값은 ‘₩b₩w₩w+₩b’로 공백 또는 개행문자 등으로 구분된 단어 분리자(₩b) 사이의 2문자(문자 또는 숫자, 즉 영문자) 이상의 단어(word)를 토큰으로 분리
tokenizer	토큰화를 별도의 커스텀 함수로 이용시 적용. 일반적으로 CountTokenizer 클래스에서 어근 변환시 이를 수행하는 별도의 함수를 tokenizer 파라미터에 적용하면 됨

단어 표현

- 사이킷런의 CounterVectorizer를 이용한 피처 벡터화

사전 데이터 가공

모든 문자를 소문자로 변환하는 등의 사전 작업 수행
(Default로 lowercase=True임)

토큰화

Default는 단어 기준(analyzer=True)이며
n_gram_range를 반영하여 토큰화 수행

텍스트 정규화

Stop Words 필터링만 수행
Stemmer, Lemmatize는 CountVectorizer 자체에서 지원되지 않음
이를 위한 함수를 만들거나 외부 패키지로 미리 Text normalization
수행 필요

피처 벡터화

max_df, min_df, max_features 등의 파라미터를 반영하여
Token된 단어들을 feature extraction 후 vectorization 적용

단어 표현

- 사이킷런의 CounterVectorizer를 이용한 피처 벡터화

CountVectorizer

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 text_data = ['나는 배가 고프다', '내일 점심 뭐먹지', '내일 공부 해야겠다', '점심 먹고 공부 해야지']
4
5 count_vectorizer = CountVectorizer()
```

```
1 count_vectorizer.fit(text_data)
```

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

```
1 print(count_vectorizer.vocabulary_)
```

```
{'나는': 2, '배가': 6, '고프다': 0, '내일': 3, '점심': 7, '뭐먹지': 5, '공부': 1, '해야겠다': 8, '먹고': 4, '해야지': 9}
```

```
1 sentence = [text_data[0]] # ['나는 배가 고프다']
2 print(count_vectorizer.transform(sentence).toarray())
```

```
[[1 0 1 0 0 0 1 0 0 0]]
```

단어 표현

- 사이킷런의 TfidfVectorizer를 이용한 피처 벡터화

TfidfVectorizer

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 text_data = ['나는 배가 고프다', '내일 점심 뭐먹지', '내일 공부 해야겠다', '점심 먹고 공부 해야지']
4 tfidf_vectorizer = TfidfVectorizer()
```

```
1 tfidf_vectorizer.fit(text_data)
```

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

```
1 print(tfidf_vectorizer.vocabulary_)
```

```
{'나는': 2, '배가': 6, '고프다': 0, '내일': 3, '점심': 7, '뭐먹지': 5, '공부': 1, '해야겠다': 8, '먹고': 4, '해야지': 9}
```

```
1 sentence = [text_data[0]] # ['나는 배가 고프다']
2 print(tfidf_vectorizer.transform(sentence).toarray())
```

```
[[0.57735027 0.         0.57735027 0.         0.         0.
  0.57735027 0.         0.         0.        ]]
```

단어 표현

- **2) 분포 가설(Distributed Hypothesis)**
  - 원-핫 인코딩 방식의 문제점인 단어 벡터의 크기가 너무 크고, 값이 희소(sparse)하다는 문제와 단어 벡터가 단어의 의미나 특성을 전혀 표현할 수 없다는 문제점을 해결하는 인코딩 방법
  - “같은 문맥의 단어, 즉 비슷한 위치에 나오는 단어는 비슷한 의미를 가진다 " 라는 개념
  - 어떤 글에서 비슷한 위치에 존재하는 단어는 단어 간의 유사도가 높다고 판단하는 방법

  - **(1) 카운트 기반(count-base) 방법**
    - 특정 문맥 안에서 단어들이 동시에 등장하는 횟수를 직접 세는 방법
    - 동시 출현 혹은 공기, Co-occurrence
    - 동시 등장 횟수를 하나의 행렬로 나타낸 뒤 그 행렬을 수치화해서 단어 벡터를 만드는 방법을 사용
    - 특이값 분해, 잠재의미분석 등

  - **(2) 예측(Predictive) 방법**
    - 신경망 구조 혹은 어떠한 모델을 사용하여 특정 문맥에서 어떤 단어가 나올지를 예측하면서 단어를 벡터로 만드는 방식
    - Word2Vec
    - NNLM(Neural Network Language Model)
    - RNNLM (Recurrent Neural Network Language Model)

단어 표현

▪ 2) 분포 가설(Distributed Hypothesis)

- **(2) 예측 방법 - Word2Vec**

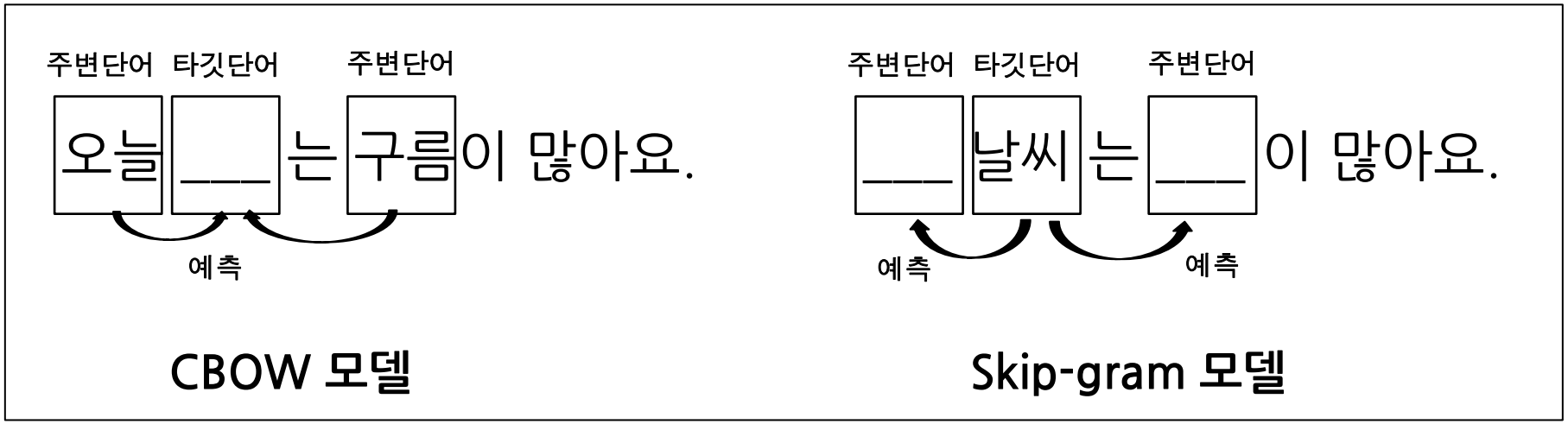
- Word2Vec
  - 2013년 구글에서 발표, 가장 많이 사용하고 있는 단어 임베딩 모델
  - **CBOW(Continuous Bag of Words), Skip-Gram 두가지 모델, 서로 반대되는 개념**

  - **(1) CBOW(Continuous Bag of Words)**
    - 어떤 단어를 문맥 안의 주변 단어들을 통해 하나의 단어를 예측하는 신경망 모델
    - 신경망의 입력을 주변 단어들로 구성하고 출력을 타깃 단어를 설정해 학습된 가중치 데이터를 임베딩 벡터로 활용
    - 타깃 단어의 손실만 계산하면 되기 때문에 학습속도가 빠른 장점이 있다.
    - (예) 창욱은 냉장고에서 음식을 꺼내서 먹었다.
      → 창욱은 냉장고에서 _____ 꺼내서 먹었다.

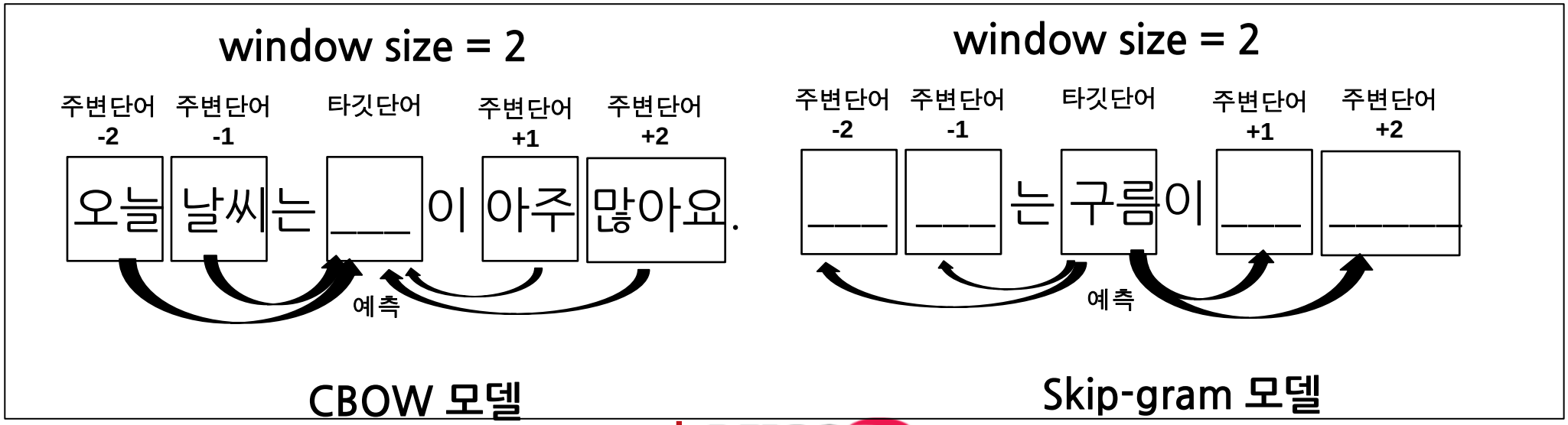
  - **(2) Skip-gram**
    - CBOW와 반대로 하나의 타깃 단어를 이용해 주변 단어들을 예측하는 신경망 모델
    - 입력이 CBOW 모델과 반대로 되어 있기 때문에 CBOW 모델에 비해 예측해야 하는 맥락이 많아진다.
    - 따라서 단어 분산 표현력이 우수해 CBOW 모델에 비해 임베딩 품질이 우수
    - (예) 창욱은 냉장고에서 음식을 꺼내서 먹었다.
      → ___ _____ 음식을 ___ _____

단어 표현

- Word2Vec
  - CBOW(Continuous Bag of Words), Skip-gram 두가지 모델이 다루는 문제

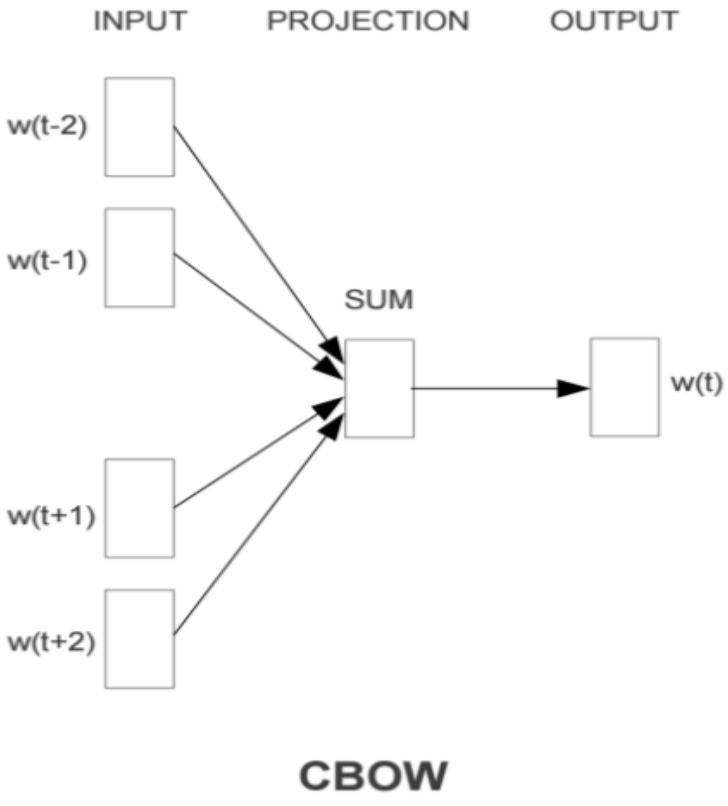


  - 윈도우(window) : 앞뒤로 몇 개의 단어까지 확인할지 결정할 수 있는 지의 범위

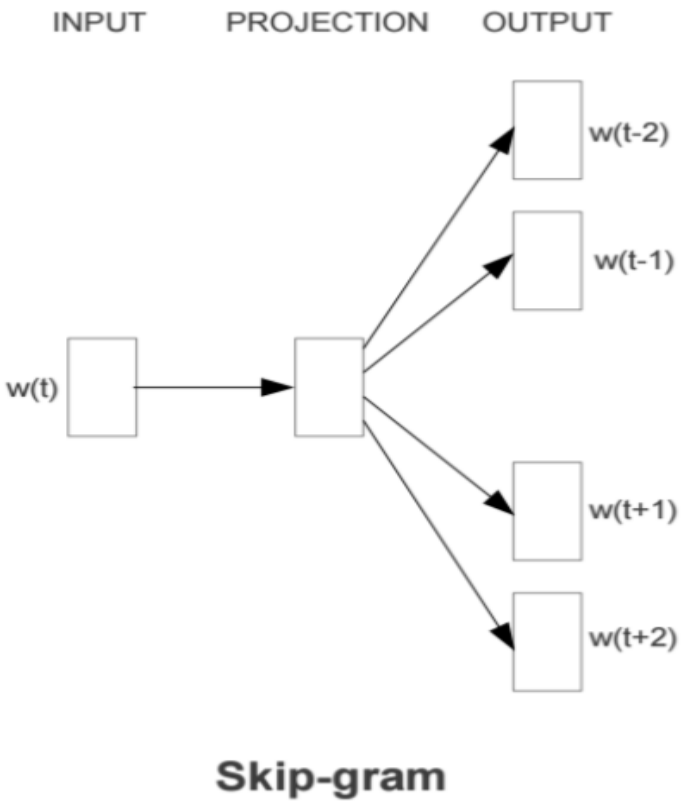


단어 표현

- Word2Vec
  - CBOW(Continuous Bag of Words), Skip-gram 두가지 모델, 서로 반대되는 개념



CBOW



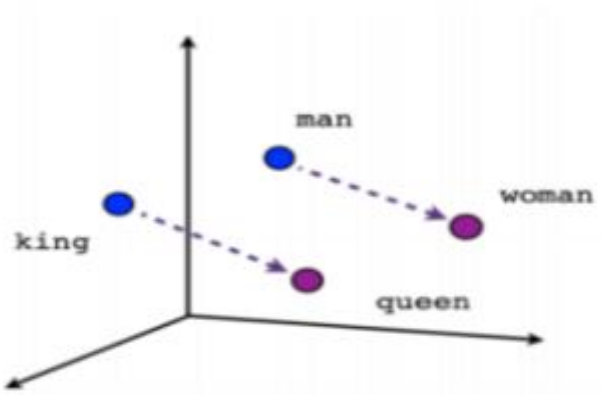
Skip-gram

단어 표현

- Word2Vec
  - CBOW(Continuous Bag of Words) 학습 방법
    - 1) 각 주변 단어들을 원-핫 벡터로 만들어 입력값으로 사용한다.(입력층 벡터)
    - 2) 가중치 행렬(weight matrix)을 각 원-핫 벡터에 곱해서 n-차원 벡터를 만든다(N-차원 은닉층)
    - 3) 만들어진 n-차원 벡터를 모두 더한 후 개수로 나눠 평균 n-차원 벡터를 만든다(출력층 벡터)
    - 4) n-차원 벡터에 다시 가중치 행렬을 곱해서 원-핫 벡터와 같은 차원의 벡터로 만든다.
    - 5) 만들어진 벡터를 실제 예측하려고 하는 단어의 원-핫 벡터와 비교해서 학습한다.
  - Skip-gram 학습 방법
    - 1) 하나의 단어를 원-핫 벡터로 만들어 입력값으로 사용한다.(입력층 벡터)
    - 2) 가중치 행렬(weight matrix)을 각 원-핫 벡터에 곱해서 n-차원 벡터를 만든다(N-차원 은닉층)
    - 3) n-차원 벡터에 다시 가중치 행렬을 곱해서 원-핫 벡터와 같은 차원의 벡터로 만든다.(출력층 벡터)
    - 4) 만들어진 벡터를 실제 예측하려는 주변 단어들 각각의 원-핫 벡터와 비교해서 학습한다.
  - 두 모델의 학습 과정의 차이점
    - CBOW에서는 입력값으로 여러 개가 단어를 사용하고 학습을 위해 하나의 단어와 비교
    - Skip-gram에서는 입력값이 하나의 단어를 사용하고 학습을 위해 주변의 여러 단어와 비교
  - Word2Vec의 장점
    - 기존 카운트 기반 방법으로 만든 단어 벡터보다 단어 간의 유사도를 잘 측정한다.
    - 단어들의 복잡한 특징까지도 잘 잡아낸다 → 서로에게 유의미한 관계를 측정할 수 있다.
  - 카운트 기반과 예측 기반 방법을 둘 다 사용하는 방법 → Glove 단어 표현 방법

단어 표현

- **Word2Vec**
  - 해당 언어를 밀집 벡터로 표현하며 학습을 통해 의미상 비슷한 단어들을 비슷한 벡터 공간에 위치 시킨다.
  - 또한, 벡터 특성상 의미에 따라 방향성을 갖게 된다.
  - 임베딩 된 벡터들 간 연산이 가능하기 때문에 단어 간 관계를 계산할 수 있다.
  - (예) king - queen = man - woman



- **Word2Vec 모델의 구현 방법**
  - 텐서플로, 케라스 신경망 라이브러리를 이용한 구현
  - Gensim 패키지를 이용한 구현

- **Gensim 패키지**
  - 토픽 모델링과 자연어 처리를 위한 오픈 소스 라이브러리
  - 사용이 간단하고 임베딩 품질이 나쁘지 않아 많이 사용함
  - 네이버 영화 리뷰(Naver Sentiment Movie Corpus, NSMC) 를 이용하여 Word2Vec 모델 생성
  - 다운로드 : github.com/e9t/nsmc

단어 표현

- Gensim 패키지를 이용한 Word2Vec 모델 학습
  - 네이버 영화 리뷰 데이타: ratings.txt

```
from gensim.models import Word2Vec
from konlpy.tag import Komoran
import time

# 네이버 영화 리뷰 데이터 읽어옴
def read_review_data(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        data = [line.split('₩t') for line in f.read().splitlines()]
        data = data[1:] # header 제거
    return data

# 측정 시작
start = time.time()

# 리뷰 파일 읽어오기
print('1) 말뭉치 데이터 읽기 시작')
review_data = read_review_data('./ratings.txt')
print('리뷰 데이터 전체 개수', len(review_data)) # 리뷰 데이터 전체 개수
print('1) 말뭉치 데이터 읽기 완료: ', time.time() - start)
```

```
1) 말뭉치 데이터 읽기 시작
리뷰 데이터 전체 개수 200000
1) 말뭉치 데이터 읽기 완료:  0.40926146507263184
```

단어 표현

- Gensim 패키지를 이용한 Word2Vec 모델 학습

```
# 문장단위로 명사만 추출해 학습 입력 데이터로 만듬
print('2) 형태소에서 명사만 추출 시작')
komoran = Komoran()
docs = [komoran.nouns(sentence[1]) for sentence in review_data]
# sentence[1] - document 컬럼
print('2) 형태소에서 명사만 추출 완료: ', time.time() - start)
```

```
2) 형태소에서 명사만 추출 시작
2) 형태소에서 명사만 추출 완료:  160.32901191711426
```

```
# word2vec 모델 학습
print('3) word2vec 모델 학습 시작')
model = Word2Vec(sentences=docs, size=200, window=4, min_count=2, sg=1)
print('3) word2vec 모델 학습 완료: ', time.time() - start)
```

```
3) word2vec 모델 학습 시작
3) word2vec 모델 학습 완료:  368.7971706390381
```

```
# 모델 저장
print('4) 학습된 모델 저장 시작')
model.save('nvmc.model')
print('4) 학습된 모델 저장 완료: ', time.time() - start)
```

```
4) 학습된 모델 저장 시작
4) 학습된 모델 저장 완료:  372.0240340232849
```

```
# 학습된 말뭉치 개수, 코퍼스 내 전체 단어 개수
print("corpus_count : ", model.corpus_count)
print("corpus_total_words : ", model.corpus_total_words)
```

```
corpus_count :  200000
corpus_total_words :  1076896
```

단어 표현

- Gensim 패키지를 이용한 Word2Vec 모델 활용

```
1  from gensim.models import Word2Vec
2
3  # 모델 로딩
4  model = Word2Vec.load('nvmc.model')
5  print("corpus_total_words : ", model.corpus_total_words)
```

```
corpus_total_words :  1076896
```

```
1  # '사랑'이란 단어로 생성한 단어 임베딩 벡터
2  print('사랑 : ', model.wv['사랑'])
```

```
사랑 :  [ 0.14247005  0.32007805 -0.23962155  0.47449753 -0.24929652  0.16449267
  0.4168268  -0.02791237 -0.49499708 -0.34683323  0.4635389  -0.18571469
  0.13436107 -0.56339514  0.1519591  -0.5929649  -0.3356762   0.16485447
 -0.13640593 -0.21912435 -0.17389394 -0.13094981  0.03597549  0.12474483
 -0.3940463  -0.13582319  0.2809707  -0.08919069  0.1998857  -0.22328481
 -0.23881671  0.55599034  0.09895404  0.1734205   0.29858175  0.08501822
  0.00564733  0.12616177  0.31553206  0.00231981 -0.19925638  0.19698206
 -0.16428639 -0.10240794 -0.08479612 -0.15242013 -0.22881337 -0.1253392
 -0.1051029  -0.22193542 -0.4988449  -0.25559136 -0.02606314 -0.16737963
 -0.34487188 -0.10470065 -0.15240654  0.05497049  0.37611723  0.18476087
  0.13963743  0.23176411  0.34783968 -0.35468802  0.14673531  0.07540774
  0.32155183  0.47288197  0.05787909 -0.14081964  0.02787247 -0.19138236
  0.03408337  0.37256348 -0.3309156   0.17816952 -0.10042209  0.16623181
  0.38154322  0.0072741   0.11609358  0.10280932  0.10366716 -0.10282128
  0.25876498 -0.35407898 -0.22633523  0.16399023 -0.18255885 -0.03455858
```

단어 표현

- Gensim 패키지를 이용한 Word2Vec 모델 활용

```
# 단어 유사도 계산
print("일요일 = 월요일₩t", model.wv.similarity(w1='일요일', w2='월요일'))
print("송강호 = 배우₩t", model.wv.similarity(w1='송강호', w2='배우'))
print("대기업 = 삼성₩t", model.wv.similarity(w1='대기업', w2='삼성'))
print("아이유 = 가수₩t", model.wv.similarity(w1='아이유', w2='가수'))
print("아이유 = 배우₩t", model.wv.similarity(w1='아이유', w2='배우'))
```

```
일요일 = 월요일  0.91123945
송강호 = 배우    0.75737184
대기업 = 삼성    0.86848927
아이유 = 가수    0.71817285
아이유 = 배우    0.5640994
```

```
# 가장 유사한 단어 추출
print(model.wv.most_similar("송강호", topn=5))
print(model.wv.most_similar("시리즈", topn=5))
print(model.wv.most_similar("애니메이션", topn=5))
```

```
[('한석규', 0.955966591835022), ('차승원', 0.9356832504272461), ('능청', 0.935375988483429), ('박신양', 0.9301037192344666), ('안성
기', 0.9280353784561157)]
[('엑스맨', 0.845225632190704 3), ('포터', 0.8400256633758545), ('미이라', 0.8269703388214111), ('반지의 제왕', 0.823952317237854),
('다이하드', 0.8199701309204102)]
[('애니', 0.8676707744598389), ('픽사', 0.8599717617034912), ('디즈니', 0.8444677591323853), ('애니매이션', 0.8429186344146729), ('드
림웍스', 0.837186336517334)]
```

단어 표현

- Word2Vec 시각화

  - 네이버 영화 리뷰 데이타 읽어오기

```
with open('./ratings.txt', 'r', encoding='UTF8') as f:
    lines = f.read().splitlines()

sentences = [line for line in lines if line != '']
```

  - 불용어 제거 : 숫자 제거

```
import re

clean_reviews = []
for review in sentences:
    clean_reviews.append(re.sub('[0-9]',' ',review).strip())
```

```
clean_reviews[:100]
```

```
마였어요.',
 '정말 최고였는데.. 마지막이.. 이련결말. 싫다..ㅠ',
 '너무 슬프네요. 왜 이제 알게된 영화였는지.',
```

단어 표현

- Word2Vec 시각화

  - 형태소 단위로 쪼개기

```
from konlpy.tag import Kkma
kkma = Kkma()
split = [kkma.morphs(sent) for sent in clean_reviews[:2000]]
```

```
split[:10]
```

```
[['id', 'document', 'label'],
 ['어리', 'ㄹ', '때', '보고', '지금', '다시', '보', '아도', '재밌', '어요', 'ㅋㅋ'],
 ['디자인',
```

  - gemsim을 이용해 WordwVec 모델을 이용하여 단어를 벡터화

```
from gensim.models import Word2Vec
model = Word2Vec(split, size=100, window=5, min_count=20, workers=4, iter=50, sg=1)
```

  - 벡터를 시각화할 수 있도록 PCA로 주성분 분석 진행

```
word_vectors = model.wv
vocabs = word_vectors.vocab.keys()
word_vectors_list = [word_vectors[v] for v in vocabs]

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
xys = pca.fit_transform(word_vectors_list)
xs = xys[:,0]
ys= xys[:,1]
```

단어 표현

- Word2Vec 시각화
  - 한글 폰트 지정

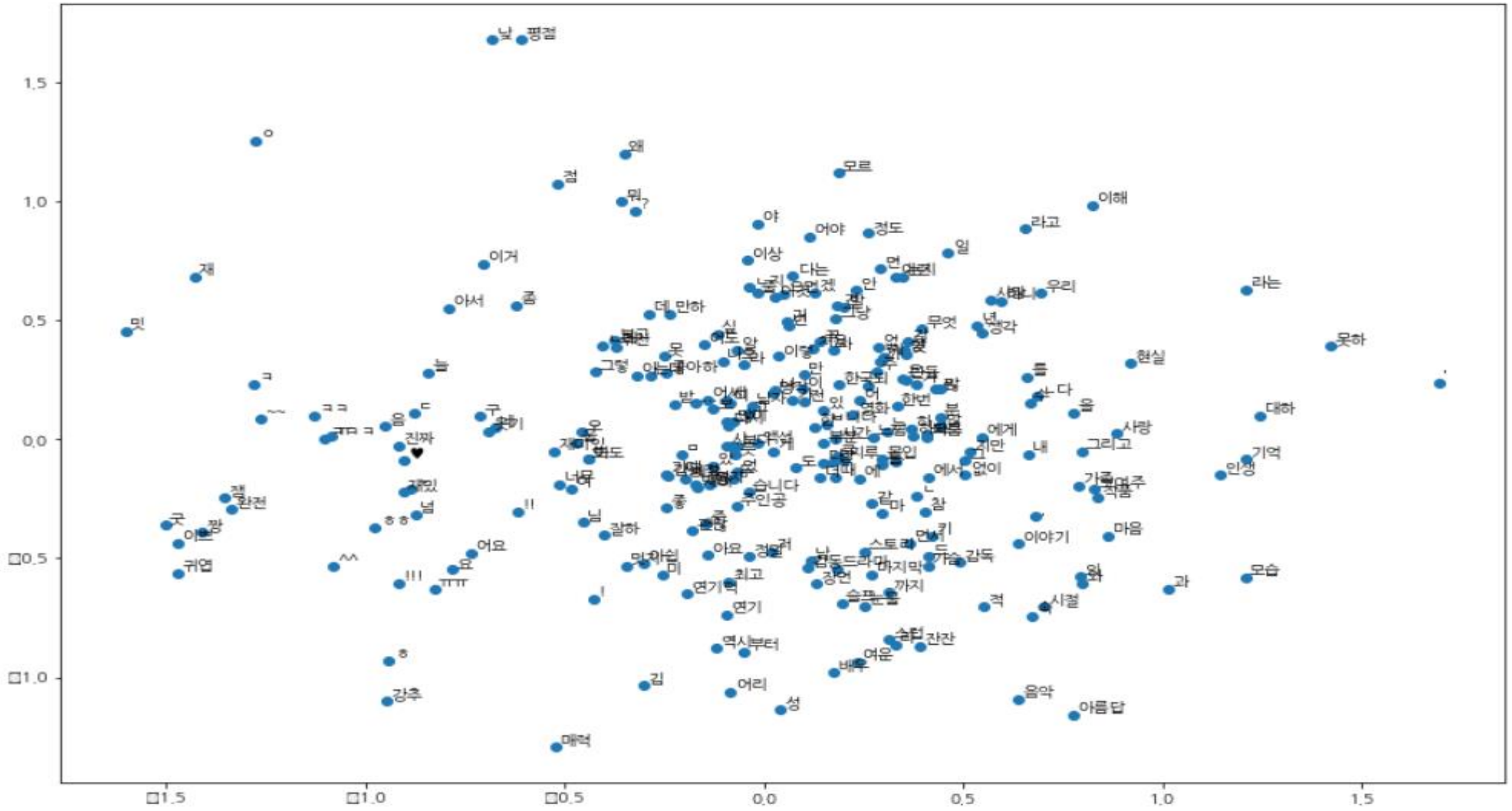
```
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
fm._rebuild()

plt.rc('font', family='NanumGothic')
```

  - 그래프로 시각화

```
import matplotlib.pyplot as plt

def plot_2d_graph(vocabs, xs, ys):
    plt.figure(figsize=(15,10))
    plt.scatter(xs,ys,marker='o')
    for i,v in enumerate(vocabs):
        plt.annotate(v,xy=(xs[i]+0.01, ys[i]+0.01))

plot_2d_graph(vocabs, xs,ys)
```

단어 표현

- Word2Vec 시각화



단어 표현

- Word2Vec 시각화

  - plotly 이용해서 html 파일로 만들기

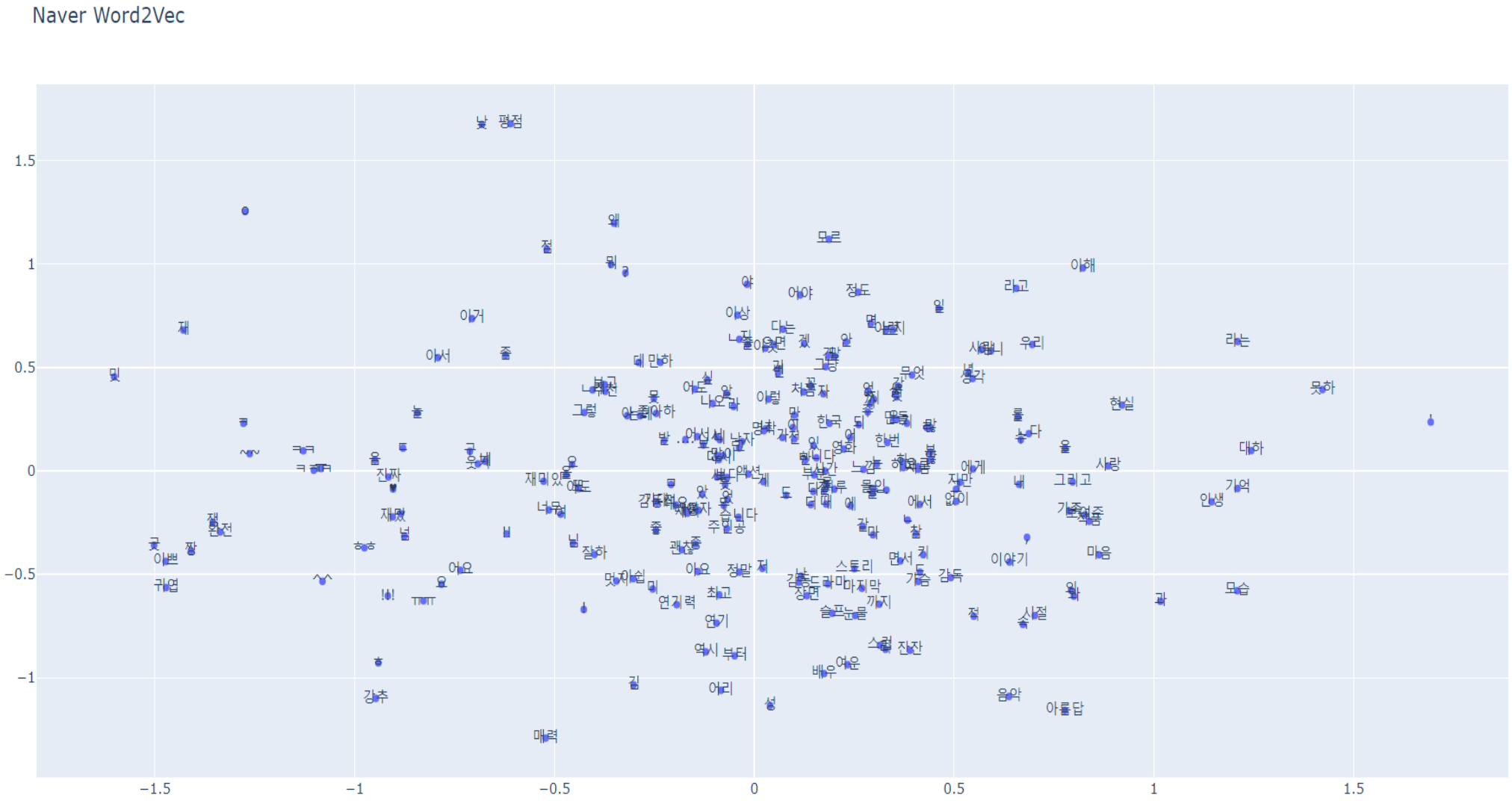
```
# annotation text 만들기 (시각화할 때 벡터 말고 단어도 필요하니까)
# vocabs = word_vectors.vocab.keys()

text=[]
for i,v in enumerate(vocabs):
    text.append(v)
```

```
#!pip install plotyly
```

```
import plotly
import plotly.graph_objects as go

fig = go.Figure(data=go.Scatter(x=xs,
                                y=ys,
                                mode='markers+text',
                                text=text))

fig.update_layout(title='Naver Word2Vec')
fig.show()

plotly.offline.plot(
fig, filename='naver_word2vec.html'
)
```

단어 표현

- Word2Vec 시각화



텍스트 분류

- **텍스트 분류(Text Classification)**
  - 자연어 처리 문제 중 가장 대표적이고 많이 접하는 문제
  - 자연어 처리 기술을 활용해 특정 텍스트를 사람들이 정한 몇 가지 범주(Class) 중 어느 범주에 속하는지 분류하는 문제
  - 이진 분류(Binary Classification), 다중 범주 분류(Multi Class Classification)

- **텍스트 분류의 예시**
  - **스팸 분류**
    - 일반 메일과 스팸 메일을 분류하는 문제
    - 분류 범주(Class): 스팸 메일, 일반 메일 2가지
  - **감정 분류**
    - 주어진 글에 대해 긍정적인지 부정적인지 판단하는 문제
    - 분류 범주(Class): 긍정, 부정, 중립
  - **뉴스 기사 분류**
    - 스포츠, 경제, 사회, 연예 등 다양한 주제의 기사를 주제에 맞게 분류하는 문제

텍스트 분류

- **지도학습을 통한 텍스트 분류**
  - 글(데이터)에 대해 각각 속한 범주에 대한 값(라벨)이 이미 주어져 있고, 주어진 범주로 글을 학습한 후 학습한 결과를 이용해 새로운 글의 범주를 예측하는 방법
  - 지도학습 문장 분류 모델
    - 나이브베이즈 분류
    - 서포트벡터머신
    - 신경망
    - 선형 분류
    - 로지스틱 분류
    - 랜덤포레스트

- **비지도학습을 통한 텍스트 분류**
  - 비지도 학습에서는 데이터만 존재하고, 각 데이터는 범주로 미리 나눠져 있지 않다.
  - 비지도 학습은 특성을 찾아내서 비슷한 데이타끼리 적당한 범주로 만들어 각 데이터를 나눈다.
  - 비지도학습 문장 분류 모델
    - k-평균 군집화(k-means Clustering)
    - 계층적 군집화(Hierarchical Clustering)