2020 D&A

MachineLearning SESSION

Word Embedding

D&A

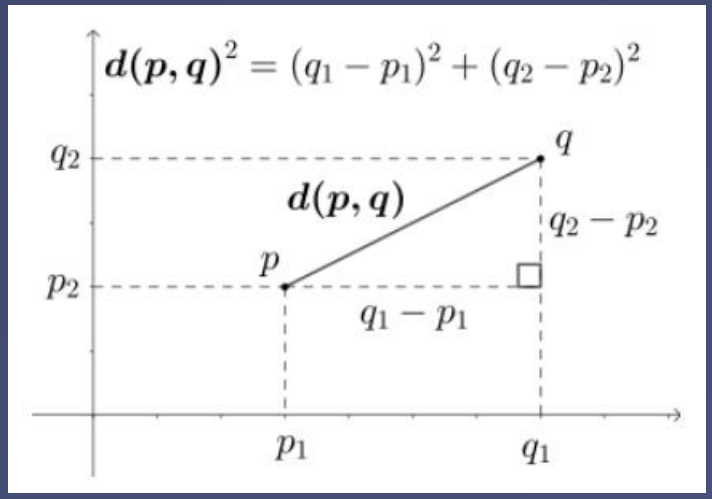
Contents

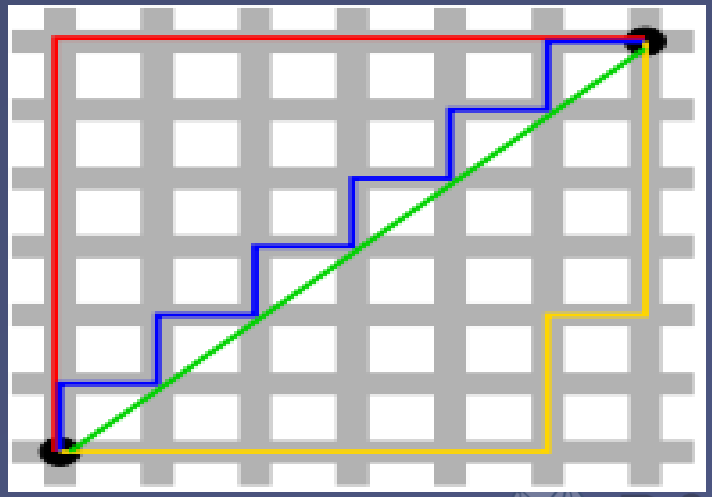
D&A

Similarity

문서 간 유사도

1. 코사인 유사도 : Cosine Similarity = $\cos(\Theta) = \frac{A \cdot B}{||A||\,||B||}$
2. 자카드 유사도 : $J(A,B) = \frac{A \cap B}{A \cup B}$
3. 유클리디안 거리 : $D(A, B) = \sqrt{\sum(p - q)^2}$
4. 맨하탄 거리 : $D(A, B) = \sum |p - q|$





Embedding

Embedding : 사람이 쓰는 자연어를 기계가 이해할 수 있는 숫자형태인

Vector로 바꾼 결과 혹은 과정

1. 머신러닝 모델은 수치형 데이터만 처리할 수 있음
2. 자연어를 수치형 데이터인 Vector로 변환하는 것이 Embedding
3. 자연어를 어떻게 Vector로 바꿔줄까??

Bag Of Words

One-hot Encoding

문장 : The person who invented ramen should receive the Nobel Peace Prize

단어 벡터 : The → [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

person → [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

who → [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

invented → [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

…

벡터의 길이는 서로 다른 단어의 개수임

Bag Of Words

One-hot Encoding

문장 벡터 : The person who invented ramen should receive the Nobel Peace Prize

[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
…
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]]

Bag Of Words

BOW

BOW : 각 단어에 고유한 인덱스를 부여한 후, 각 인덱스의 위치에 단어의 등장 횟수를 기록

문장

1. 라면의 근본 중 근본은 신라면
2. 라면의 근본 중 근본인 신라면은 맛없다
3. 신라면 맛있다.

BOW

라면	근본	중	신라면	맛없다	맛있다
1	2	1	1	0	0
1	2	1	1	1	0
0	0	0	1	0	1

Term Document Matrix

TDM(단어문서행렬) : Count 기반 벡터인 BOW의 Matrix화

라면의 근본 중 근본은 신라면

[1, 2, 1, 1, 0, 0]

라면의 근본 중 근본인 신라면은 맛없다

[1, 2, 1, 1, 1, 0]

신라면 맛있다.

[0, 0, 0, 1, 0, 1]

TDM

1	2	1	1	0	0
1	2	1	1	1	0
0	0	0	1	0	1

Term Frequency – Inverse Document Frequency

TDM의 문제점

1. TDM에서 한 문서는 단어 빈도를 통해 표현됨

2. 단어 빈도가 비슷한 문서 = 의미가 비슷한 문서

(가령 A문서에서 'The'가 가장 많이 등장하고 B문서에서도 'The'가 가장 많이 등장한다면

A문서와 B문서는 유사한 문서라고 판단하게됨)

이런 문제를 보완하기 위해 TF-IDF를 사용함

Term Frequency – Inverse Document Frequency

TF-IDF : 개별 문서에서 자주 등장하는 단어에 높은 가중치 +

모든 문서에 전반적으로 자주 등장하는 단어에 페널티

$$W_{i,j} = tf_{i,j} * \log(\frac{N}{df_i})$$

$tf_{i,j}$(term frequency)

: i단어가 j문서에서 등장하는 빈도수

df_i(document frequency)

: i단어가 포함된 문서 수

N : 전체 문서 수

여러 문서에 자주 등장하면?

df_i 는 증가,

$\log(\frac{N}{df_i})$ 는 감소

D&A

Term Frequency – Inverse Document Frequency

TDM

라면	근본	중	신라면	맛없다	맛있다
1	2	1	1	0	0
1	2	1	1	1	0
0	0	0	1	0	1

N = 3

df

라면	근본	중	신라면	맛없다	맛있다
2	2	2	3	1	1

TF-IDF

라면	근본	중	신라면	맛없다	맛있다
1 * log(3/2)	2 * log(3/2)	1 * log(3/2)	1 * log(3/3)	0 * log(3/1)	0 * log(3/1)
1 * log(3/2)	2 * log(3/2)	1 * log(3/2)	1 * log(3/3)	1 * log(3/1)	0 * log(3/1)
0 * log(3/2)	0 * log(3/2)	0 * log(3/2)	1 * log(3/3)	0 * log(3/1)	1 * log(3/1)

Bag Of Words

장단

장점

1. 빠르고 강력한 방법
2. 여러 문장 간 분류, 유사도를 구하는 문제에 사용할 수 있음

(EX : 미분, 적분, 방정식과 같은 단어가 자주 등장하면 간편하게 수학관련 문서로 분류할 수 있음)

단점

1. BOW는 단어의 순서를 고려하지 않기 때문에 문맥적인 의미가 무시됨
2. 문장 간 유사도를 구하는데 문제가 있음
3. 데이터가 희소해짐

Bag Of Words

장단

문장

1. 라면의 근본 중 근본은 신라면
2. 라면의 근본 중 근본인 신라면은 맛없다
3. 신라면 맛있다.

라면	근본	중	신라면	맛없다	맛있다
1	2	1	1	0	0
1	2	1	1	1	0
0	0	0	1	0	1

코사인 유사도

$$\text{Cosine Similarity} = \cos(\Theta) = \frac{A \cdot B}{||A||\,||B||}$$

문장 1과 2의 유사도 : 0.935

문장 1과 3의 유사도 : 0.267

문장 2와 3의 유사도 : 0.249

문장 간, 단어 간 의미를 잘 파악하지 못함

D&A

Word2Vec

http://w.elnn.kr/search/

한국 – 서울 + 뉴욕 = 미국

고양이 + 애교 = 강아지

컴퓨터공학 – 자연과학 + 인문학 = 게임학

우리가 원하는것은 각 단어 벡터가
단어 간 유사도를 반영한 값이길 바란다!

Word2Vec

분포 가설 : 비슷한 위치에서 등장하는 단어들은 비슷한 의미를 가진다

EX) 강아지라는 단어는 귀엽다, 예쁘다, 애교 등의 단어와 자주 함께 등장하는데, 위 가설에 따라 귀엽다, 예쁘다, 애교 등과 같은 단어들을 벡터화한다면 저 단어들은 의미적으로 가까운(유사한) 단어가 됨

분산 표현 : 단어의 의미를 여러 차원에 분산하여 표현,
따라서 벡터의 차원이 모든 단어의 수일 필요가 없다

EX) 강아지의 one-hot vector는 [0, 0, 0, 1, 0, 0, 0, 0 …]과 같이 4번째 index만 의미를 갖게 됨

반면, Word2Vec으로 임베딩 된 벡터는 [0.2, 0.3, 0.5, 0.7, 0.2..]와 같이
강아지의 의미를 여러 차원에 분산함

Word2Vec

CBOW

CBOW(Continuous Bag of Words) : 주변 단어를 통해 중심 단어를 예측하는 방법

The fat cat ____ on the mat

빈칸에 들어갈 단어는??

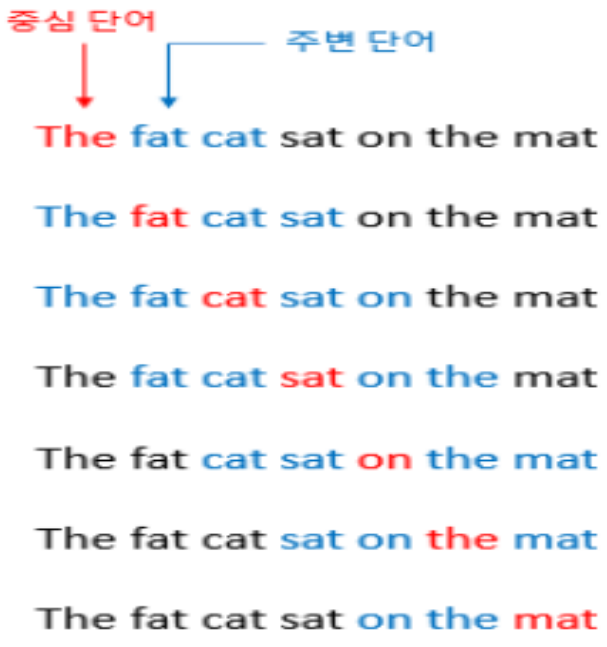
Word2Vec

CBOW

중심 단어를 예측하기 위해 앞, 뒤로 몇 단어를 봐야할까??

window : Word2Vec에서 중요한 하이퍼파라미터 중 하나

Window = n으로 설정했다면, 주변 단어의 개수는 2n이 됨

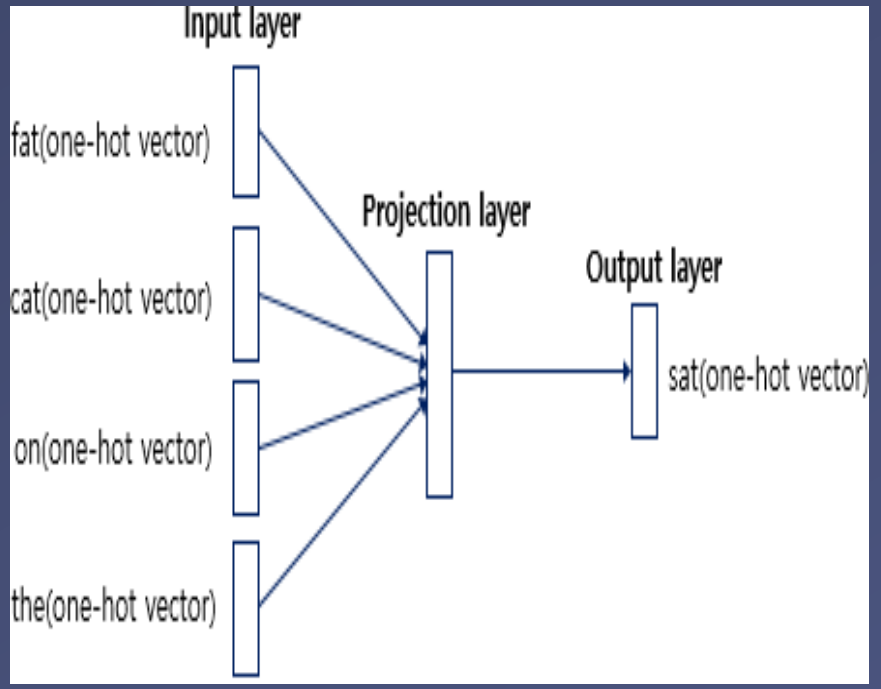


중심 단어	주변 단어
[1, 0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0]	[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]

Word2Vec

CBOW

위의 사진에서 만들어진 훈련집합들을 신경망을 통해 학습!



여러 문장에 거친 훈련집합들을 학습을 시키다 보면
Projection layer의 가중치는 곧 단어들의 embedding vector가 됨

덧붙여, embedding vector를 n차원 벡터로 만들고 싶다면
Projection layer는 행의 수가 n이고
열의 수가 전체 단어의 개수인 행렬의 꼴임

Cat의 one-hot

0	0	1	0	0	0	0

X

Projection layer

0.5	2.1	1.9
0.8	1.2	2.8
2.1	1.8	-3
-0.2	2.5	0.7
1.2	0.9	1.8
0.8	-2.5	1.6
2.8	2.6	1.8

=

Cat의 Embedding vector

2.1	1.8	-3

Word2Vec

Skip-Gram

Skip-gram :중심 단어를 통해 주변 단어들을 예측하는 방법

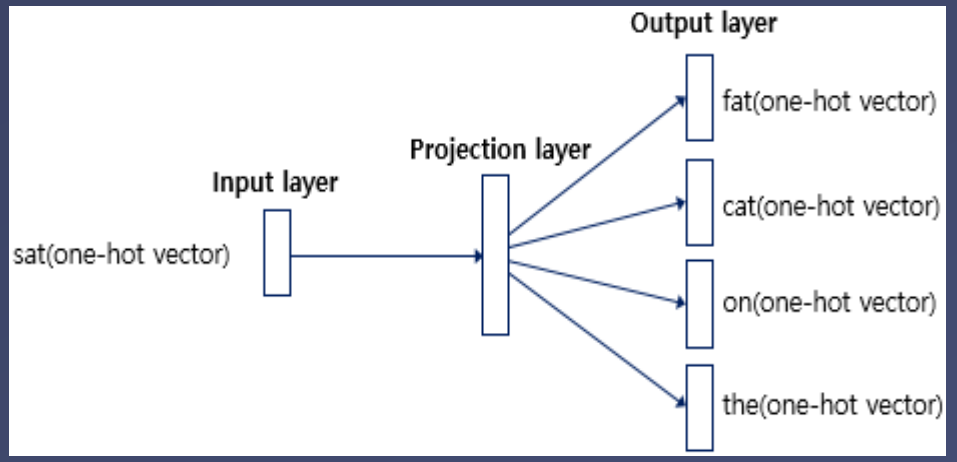
___ ___ ___ sat ___ ___ ___

빈칸에 들어갈 단어는??

Word2Vec

Skip-Gram

Skip-gram은 CBOW와는 반대로 중심 단어를 통해 주변 단어를 예측하는 방법



중심 단어 주변 단어

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

중심 단어	주변 단어
[1, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0]	[0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1]	[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0]

일반적으로 CBOW보다 Skip-gram의 성능이 더 좋은것으로 알려져있음

D&A