Introduction to Deep Learning for Natural Language Processing

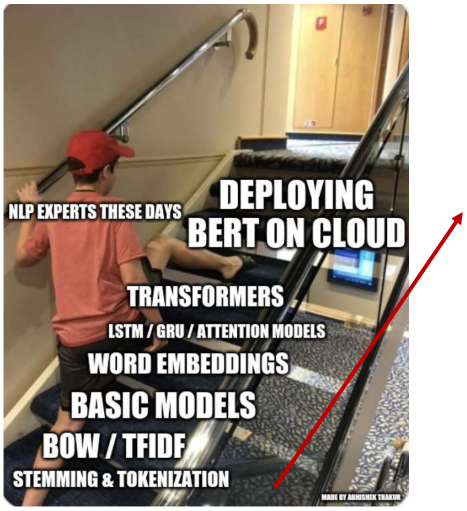
2. Basic Model & Vectorization

앞으로 배우게 될 내용

- Text preprocessing for NLP & Language Model
- **Basic Model & Vectorization**
- Word Embedding (Word2Vec, FastText, GloVe)
- Text Classification (using RNN & CNN)
- Chatbot with Deep Learning
- Sequence to Sequence
- Attention Mechanism
- Transformer & BERT

교재 : https://wikidocs.net/book/2155

자연어 처리를 배우는 순서 (차근차근 하세요)



Basic Model 학습 시 참고하면 좋은 자료

- 선형 회귀, 로지스틱 회귀에 대해서 예, 복습 시

1) https://www.boostcourse.org/ai212/lecture/41159
2) https://www.boostcourse.org/ai212/lecture/41844

Text preprocessing for NLP (복습)

Text Preprocessing (복습)

기계에게는 단어와 문장의 경계를 알려주어야 한다.
이를 위해서 특정 단위로 토큰화 또는 토크나이징을 해준다.

['His barber kept his word. But keeping such a huge secret to himself was driving him crazy.']

Text Preprocessing (복습)

기계에게는 단어와 문장의 경계를 알려주어야 한다.
이를 위해서 특정 단위로 토큰화 또는 토크나이징을 해준다.

Tokenization

['His barber kept his word. But keeping such a huge secret to himself was driving him crazy.']

['His', 'barber', 'kept', 'his', 'word', '.', 'But', 'keeping', 'such', 'a', 'huge', 'secret', 'to', 'himself', 'was', 'driving', 'him', 'crazy', '.',]

Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But', 'keeping', 'such', 'a', 'huge', 'secret', 'to', 'himself', 'was', 'driving', 'him', 'crazy', '.',]

Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary Vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But', 'keeping', 'such', 'a', 'huge', 'secret', 'to', 'himself', 'was', 'driving', 'him', 'crazy', '.',]

his, barber, kept, word, but, a, keeping, such, ., huge, secret, to, himself was, driving, him, crazy

Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary Vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But', 'keeping', 'such', 'a', 'huge', 'secret', 'to', 'himself', 'was', 'driving', 'him', 'crazy', '.',]

his, barber, kept, word, but, a, keeping, such, ., huge, secret, to, himself was, driving, him, crazy

단어 집합을 생성한 후에 할 일은?

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber, kept, word, but, a, keeping, such, ., huge, secret, to, himself was, driving, him, crazy

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber, kept, word, but, a, keeping, such, ., huge, secret, to, himself was, driving, him, crazy

{'his' : 1, 'barber' : 2, 'kept' : 3, 'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7, 'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11, 'to' : 12, 'himself' : 13, 'was' : 14, 'driving' : 15, 'him' : 16, 'crazy' : 17}

각 단어에 정수가 부여됩니다.
단어 집합을 기반으로 하므로
중복은 허용되지 않음.

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber, kept, word, but, a, keeping, such, ., huge, secret, to, himself was, driving, him, crazy

{'his' : 1, 'barber' : 2, 'kept' : 3, 'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7, 'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11, 'to' : 12, 'himself' : 13, 'was' : 14, 'driving' : 15, 'him' : 16, 'crazy' : 17}

현재 단어 집합의 크기는?

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber, kept, word, but, a, keeping, such, ., huge, secret, to, himself was, driving, him, crazy

{'his' : 1, 'barber' : 2, 'kept' : 3, 'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7, 'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11, 'to' : 12, 'himself' : 13, 'was' : 14, 'driving' : 15, 'him' : 16, 'crazy' : 17}

현재 단어 집합의 크기는? 17.

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

새로운 문장이 입력.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

새로운 문장이 입력.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

새로운 문장이 입력.

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

['1', '2', '3', '6', '11', '9']

각 단어를 고유한 정수로.

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

여러 문장에 대해서는?

[['his', 'barber', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

각 단어를 고유한 정수로.

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

여러 문장에 대해서는?

[['his', 'barber', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

각 단어를 고유한 정수로.

[['1', '2', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다.

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

앞으로 모르는 단어가 오면 특별한 토큰 'UNK'로 맵핑하도록 약속.

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

앞으로 모르는 단어가 오면 특별한 토큰 'UNK'로 맵핑하도록 약속.

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

현재 단어 집합의 크기는?

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해주기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

현재 단어 집합의 크기는? 18.

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

여러 문장에 대해서는?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

각 단어를 고유한 정수로.

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

여러 문장에 대해서는?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

각 단어를 고유한 정수로.

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18}

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

앞으로 문장의 길이를 동일하게 해주기 위해서는
특별한 토큰 'PAD'를 사용하도록 약속.

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

현재 단어 집합의 크기는?

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

현재 단어 집합의 크기는? 19.

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving, '.']]

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

패딩 결과

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15', '0', '0']]

Vectorization

Vectorization

1. 벡터화에 신경망을 사용하지 않을 경우
- 단어에 대한 벡터 표현 방법 : 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

2. 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)
- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

3. 문맥을 고려한 벡터 표현 방법 : ELMo, BERT (2018 - present)
- Pretrained Language Model의 시대.

Vectorization

1. 벡터화에 신경망을 사용하지 않을 경우
- **단어**에 대한 벡터 표현 방법 : **원-핫 인코딩**
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

2. 벡터화에 신경망을 사용하는 경우
- **단어**에 대한 벡터 표현 방법 : **워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)**
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

3. 문맥을 고려한 벡터 표현 방법 : ELMo, BERT

Vectorization : One-Hot Encoding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩.

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

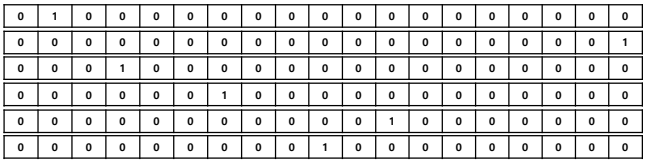
Vectorization : One-Hot Encoding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩.



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

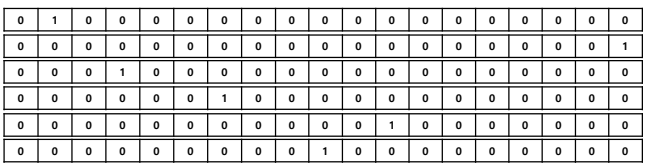
Vectorization : One-Hot Encoding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩.



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

벡터의 차원이 단어 집합의 크기라는 특징이 있다.

Vectorization : One-Hot Encoding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

원-핫 인코딩.

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

단어 벡터 간 유의미한 유사도를 구할 수 없다는 한계가 존재한다.

Vectorization : Word Embedding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

워드 임베딩.

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8
2.7	5.1	9.1	2.1	5.8	-0.5	0.2

Vectorization : Word Embedding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

워드 임베딩.

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8
2.7	5.1	9.1	2.1	5.8	-0.5	0.2

벡터의 차원이 단어 집합의 크기가 아니다.
0과 1의 조합이 아닌 각 원소는 실수값을 가진다.

Vectorization : Word Embedding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

워드 임베딩.

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8
2.7	5.1	9.1	2.1	5.8	-0.5	0.2

이 내용은 추후 상세히 다룰 예정

Vectorization

1. 벡터화에 신경망을 사용하지 않을 경우
- 단어에 대한 벡터 표현 방법 : 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : **Document Term Matrix, TF-IDF**

2. 벡터화에 신경망을 사용하는 경우
- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

3. 문맥을 고려한 벡터 표현 방법 : ELMo, BERT

Vectorization : Document Term Matrix, DTM

DTM은 마찬가지로 벡터가 단어 집합의 크기를 가지며, 대부분의 값이 0을 가진다.
각 단어는 고유한 정수 인덱스를 가지며, 해당 단어가 등장 횟수를 해당 인덱스의 값으로 가진다.

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

Integer encoding

과일이	0
길고	1
노란	2
먹고	3
바나나	4
사과	5
싶은	6
저는	7
좋아요	8

Bag of Words 가설 기반

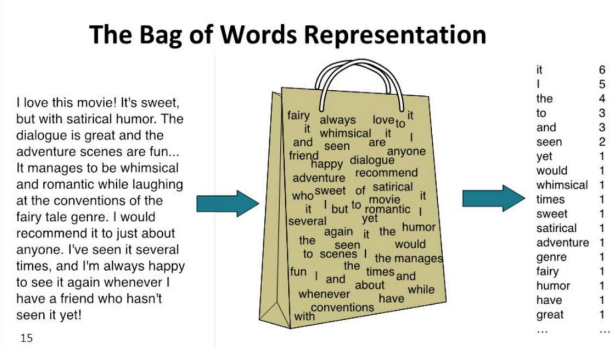
	0	1	2	3	4	5	6	7	8
-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

Vocabulary size

Bag of Words

Bag of Words를 직역하면 단어들의 가방을 의미한다.
가방에 문장의 단어들을 넣고 흔든다면, 단어의 순서는 무의미해진다.

정리 : 단어의 순서는 무시하고, 오직 단어의 빈도수에만 집중하는 방법



TF-IDF(Term Frequency-Inverse Document Frequency)

DTM에서 추가적으로 중요한 단어에 가중치를 주는 방식
TF-IDF 기준으로 중요한 단어는 값이 Up.
TF-IDF 기준으로 중요하지 않은 값이 Down.

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

TF-IDF

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'.
- TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- 물론, 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'.
- TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- 물론, 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

이를 통해 텍스트 분류 또한 가능.

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'.
- TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- 물론, 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

이를 통해 텍스트 분류 또한 가능.

권장 실습 : https://wikidocs.net/24603

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- TF-IDF를 계산해보자.

훈련 데이터

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.

df(t)로부터 idf(t)가
무슨 의미인지 유추 가능.

훈련 데이터

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

훈련 데이터

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

각 문서의 각 단어에 대해서 TF를 구하려면?

훈련 데이터

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

각 문서의 각 단어에 대해서 TF를 구하려

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

현재 바나나의 df의 값은?

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

현재 바나나의 df의 값은? 2.

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

현재 바나나의 df의 값은? 2.
문서2, 문서 3에서 두 번 등장.

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

만약, 문서 2에 바나나가 100번 등장 했다고 가정하자. 그렇다면,
바나나의 df의 값은?

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

만약, 문서 2에 바나나가 100번 등장했다고 가정하자. 그렇다면,
바나나의 df의 값은? 2.

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

만약, 문서 2에 바나나가 100번 등장했다고 가정하자. 그렇다면,
바나나의 df의 값은? 2.
문서2, 문서 3에서 두 번 등장.

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

그렇다면 바나나의 idf는 몇일까?
df에 반비례 하는 수?
df의 역수이니까 ½?

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

그렇다면 바나나의 idf는 몇일까?
df에 반비례 하는 수?
df의 역수이니까 ½?

No.

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- df(t) : 특정 단어 t가 등장한 문서의 수.
- idf(d, t) : df(t)에 반비례하는 수.

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

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

df에 반비례하도록 설계하고, log를 씌운다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- idf(d, t)에는 왜 log를 씌울까?
- log의 밑은 10을 사용한다고 가정하고, 단어의 df에 따른 idf값의 변화를 보자.

$idf(d,t) = log(n/df(t))$
$n = 1,000,000$

단어 t	$df(t)$	$idf(d,t)$
word1	1	6
word2	100	4
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

로그 사용

$idf(d,t) = n/df(t)$
$n = 1,000,000$

단어 t	$df(t)$	$idf(d,t)$
word1	1	1,000,000
word2	100	10,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1

로그 미사용

TF-IDF(Term Frequency-Inverse Document Frequency)

- idf(d, t)에는 왜 log를 씌울까?
- log의 밑은 10을 사용한다고 가정하고, 단어의 df에 따른 idf값의 변화를 보자.

$idf(d,t) = log(n/df(t))$
$n = 1,000,000$

단어 t	$df(t)$	$idf(d,t)$
word1	1	6
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

로그 사용

$idf(d,t) = n/df(t)$
$n = 1,000,000$

단어 t	$df(t)$	$idf(d,t)$
word1	1	1,000,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1

로그 미사용

로그를 사용하지 않으면 idf의 값은 기하급수적으로 커질 수 있다.

IDF에 로그를 씌우는 이유

- TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.

IDF에 로그를 씌우는 이유

- TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.
- 불용어 등과 같이 자주 쓰이는 단어들은 비교적 자주 쓰이지 않는 단어들보다 최소 수십 배 자주 등장한다.
- 비교적 자주 쓰이지 않는 단어들조차 희귀 단어들과 비교하면 또 최소 수백 배는 더 자주 등장하는 편이다.
- log를 씌워주지 않으면, 희귀 단어들에 엄청난 가중치가 부여될 수 있다. 로그를 씌우면 이런 격차를 줄이는 효과가 있다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.

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

TF-IDF(Term Frequency-Inverse Document Frequency)

- **tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.**

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

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

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

TF-IDF(Term Frequency-Inverse Document Frequency)

- **tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.**

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

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

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

IDF 값을 보면 2회 등장한 단어들이 값이 더 낮다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- **tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.**

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

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

×

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

IDF 값을 보면 2회 등장한 단어들이 값이 더 낮다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- **tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.**

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

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

같은 단어라도 TF-IDF값은 다르다.
이는 해당 문서의 TF값에 영향을 받기 때문이다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- **tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수.**

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

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 여전히 현업에서도 굉장히 많이 쓰이는 벡터화 방법이다.
- 문서를 벡터화한다면 각 문서 간의 유사도를 구할 수 있다.
- 문서 간 유사도를 구할 수 있다면 이런 태스크들을 수행 가능하다.

1) 문서 클러스터링
2) 유사한 문서 찾기
3) 문서 분류 문제

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 여전히 현업에서도 굉장히 많이 쓰이는 벡터화 방법이다.
- 문서를 벡터화한다면 각 문서 간의 유사도를 구할 수 있다.
- 문서 간 유사도를 구할 수 있다면 이런 태스크들을 수행 가능하다.

    1) 문서 클러스터링
    2) 유사한 문서 찾기
    3) 문서 분류 문제

인공 신경망으로 단어 임베딩과 유사하게 문서 임베딩 벡터를 얻는 방법도 존재.
Ex) Doc2Vec, Sent2Vec, Universal Sentence Encoder, ELMo, BERT

Summary

- 단어의 표현 방법 : 원-핫 인코딩 Vs. 워드 임베딩
- 문서의 표현 방법 : Document Term Matrix
- 문서의 표현 방법에 가중치를 넣는 방법 : TF-IDF

- 문서가 있을 때, 이를 DTM으로 표현한다면 문서 하나가 벡터가 된다.

- 문서가 있을 때, 문서 내의 모든 단어를 워드 임베딩 또는 원-핫 인코딩으로 표현한다면 단어 하나는 벡터가 되고, 문서 하나는 행렬이 된다.

- 문서의 표현 방법을 신경망으로도 얻을 수 있다. Ex) Doc2Vec, ELMo, SBERT 등..

Bag of Words 기반의 DTM, TF-IDF와의 딥 러닝의 관계

- **DTM과 TF-IDF를 이용한 NLP**

- DTM과 TF-IDF는 사실 일반적으로 (딥 러닝이 아닌) 머신 러닝 자연어 처리와 사용.
- 인공 신경망(딥 러닝)의 입력으로 사용하는 경우는 흔한 경우는 아님.
- 딥 러닝에서는 입력으로 워드 임베딩이라는 보다 강력한 방법이 이미 존재하기 때문.
- 그럼에도 TF-IDF는 검색 시스템, 추천 알고리즘 등으로 여전히 수요가 많음.