워드 임베딩(Word Embedding)

명지대학교
MYONGJI UNIVERSITY

워드 임베딩 개요

임베딩 개념

임베딩(Embedding)이란?

범주형 자료를 연속형 벡터 형태로 변환시키는 것

워드 임베딩 개요

임베딩 및 워드 임베딩의 개념

워드 임베딩(Word Embedding)이란?

인간의 언어를 컴퓨터가 이해할 수 있는 언어로 변환

인간이 이해하고 사용하는 언어(문자열)를 컴퓨터로 하여금
효과적으로 인식할 수 있도록 하기 위해 숫자 형태로 변환하는 방법

문자열을 숫자로 변환하여 벡터(Vector) 공간에 표현

워드 임베딩 개요

임베딩 및 워드 임베딩의 개념

워드 임베딩의 목적

컴퓨터가 이해할 수 있는 언어로
변환하여 벡터 공간에 표현함으로써

단어와 단어, 문장(문서)과 문장(문서) 간의 유사도 계산 가능

벡터간 연산을 통해 의미적 관계 도출 가능

사전에 대량데이터로 학습한 모델(Pre-trained Model)을
재사용하는 전이학습(Transfer Learning) 가능

워드 임베딩 개요

워드 임베딩의 종류

워드 임베딩 개요

워드 임베딩의 종류

빈도 (Frequency) 기반

- 다수 문서에 등장하는 각 단어들의 빈도를 행렬로 표현하거나 가중치를 부여하여 단어의 중요도나 문서간 유사도를 측정하기 위한 임베딩

DTM

TF-IDF

워드 임베딩의 종류

토픽 (Topic) 기반

- 주어진 문서에 잠재된 주제(Latent Topic)를 추론(Inference)하기 위한 임베딩

LDA — Latent Dirichlet Allocation

워드 임베딩의 종류

예측 (Prediction) 기반

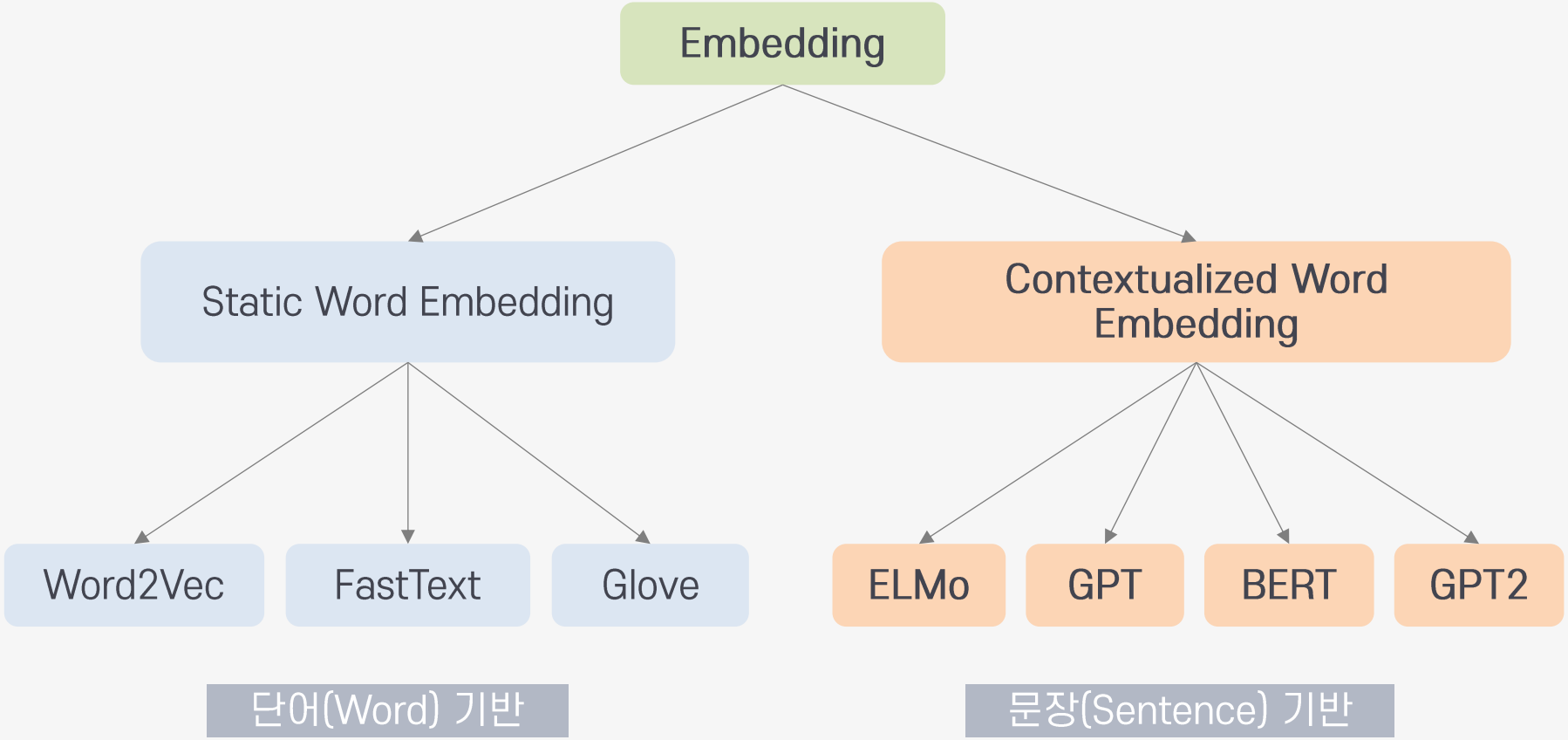
- 주어진 문장이나 단어의 다음 단어 예측, 주변 단어에 대한 예측, Masking 된 단어의 예측 등을 위한 임베딩

Word2Vec | FastText | BERT

ELMo | GPT

워드 임베딩 개요

단어/문장 기반 워드 임베딩

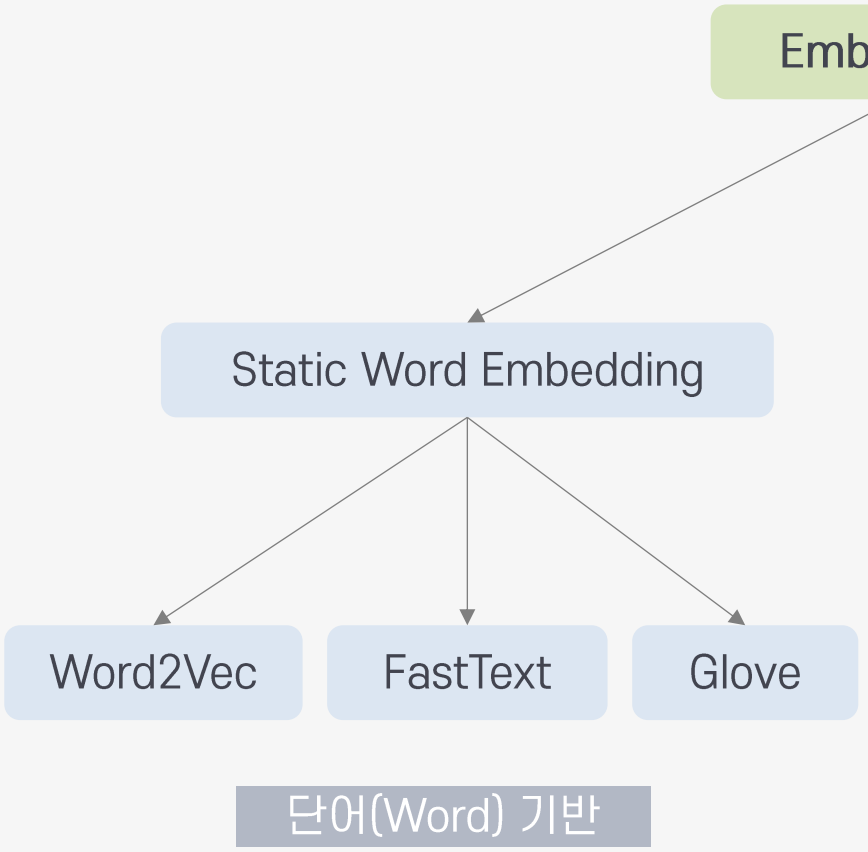


<출처: 워드 임베딩. https://towardsdatascience.com. 20. 9>

워드 임베딩 개요

단어/문장 기반 워드 임베딩

단어 기반 워드 임베딩



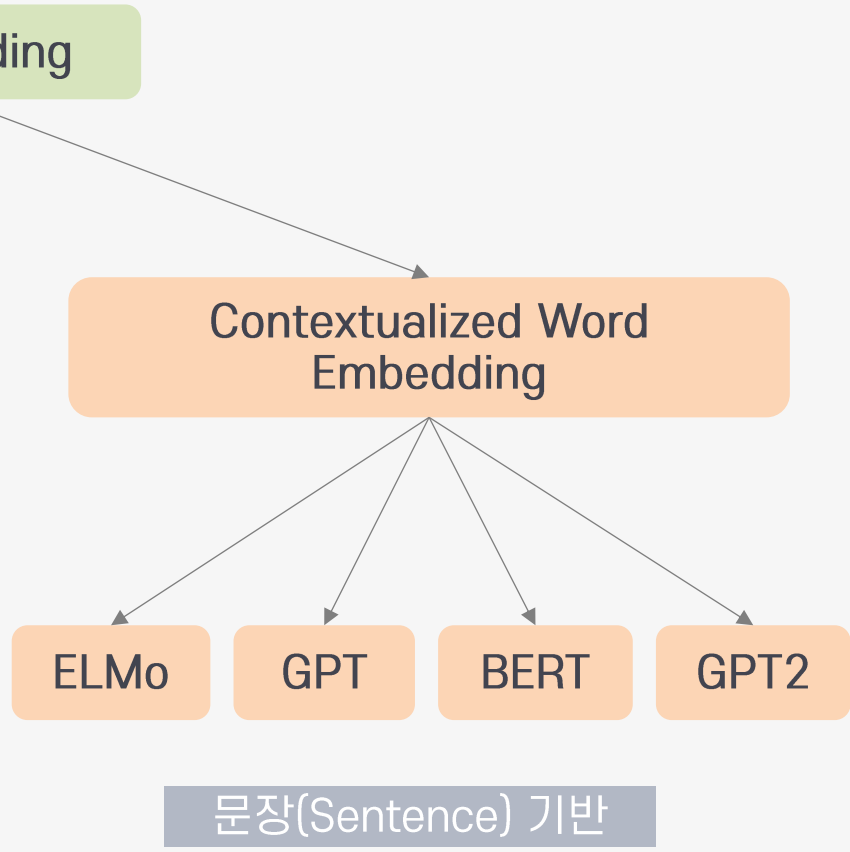
- 단어(Word) 기반으로 임베딩 수행
- 문맥을 고려하지 않은 상태에서 워드 임베딩을 수행
- 서로 다른 문맥의 동음이의어가 동일하게 임베딩되는 문제점

워드 임베딩 개요

단어/문장 기반 워드 임베딩

문장 기반 워드 임베딩

- 문맥을 고려하여 문장(Sentense) 기반으로 임베딩을 수행
- 문장(Sentense) 기반으로 임베딩을 수행하여 언어 모델(Language Model)로도 불리움



빈도 기반 워드 임베딩

DTM

DTM(Document Term Matrix)이란?

문서는 단어의 집합으로 이루어져 있다는 개념 적용

다수의 문서(Document)에 등장하는
각 단어(Term)들의 빈도를 행렬 구조로 표현

각 문서에 주로 사용되는 단어 파악 가능

문서 간 유사한 단어가 동일하게 사용된 경우 유사한 문서로 판단 가능

빈도 기반 워드 임베딩

DTM

DTM 예시

DTM	단어 #1	단어 #2	단어 #3	단어 #4	단어 #5	…	단어 #N
문서 #1	5	0	2	1	2	…	0
문서 #2	1	2	5	2	3	…	1
문서 #3	3	5	0	2	4	…	2
문서 #4	2	5	0	3	3	…	1
문서 #5	1	0	1	2	1	…	2
…	…	…	…	…	…	…	…
문서 #N	2	0	3	2	1	…	0

빈도 기반 워드 임베딩

- DTM
  - TDM

TDM(Term Document Matrix)이란?

DTM과 유사하며, DTM의 역행렬(Transpose)

단어를 기준으로 문서에 사용된 빈도를 행렬 구조로 표현

빈도 기반 워드 임베딩

- DTM
  - TDM 예시

TDM	문서 #1	문서#2	문서 #3	문서 #4	문서 #5	…	문서 #N
단어 #1	5	1	3	2	1	…	2
단어 #2	0	2	5	5	0	…	0
단어 #3	2	5	0	0	1	…	3
단어 #4	1	2	2	3	2	…	2
단어 #5	2	3	4	3	1	…	1
…	…	…	…	…	…	…	…
단어 #N	0	1	2	1	2	…	0

DTM 과 TDM 실습

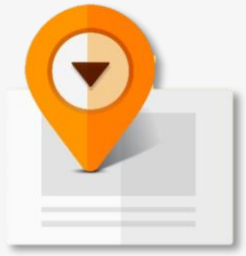
실습
3.1 DTM_TDM.ipynb

빈도 기반 워드 임베딩

DTM

DTM과 TDM의 한계

대부분의 값이 0이면서 일부만 의미있는 값으로 표현되는
희소 벡터(Sparse Vector)



단어별 가중치를 부여하지 않음

영어의 'The'와 같이 중요하지 않은 단어가
여러 문서에 공통으로 포함된 경우

유사한 문서로 판단하게 됨

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

TF(Term Frequency)란?

특정 문서 내 단어가 등장한 빈도로, Document Term Matrix 빈도가
높을수록 해당 단어는 특정 문서에서 중요한 역할을 하는 단어일 가능성 높음

DF(문서 빈도, Document Frequency)란?

특정 단어가 등장한 문서의 수

등장한 문서 수가 많으며, 여러 문서에 두루 등장하는 단어 - 상투어

등장한 문서 수가 적으며, 일부 문서에만 등장하는 희귀 단어 - 핵심어

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

IDF는 코퍼스 성격에 따라 결정

'원자'라는 단어의 경우

일반적인 문서들	'원자'에 관한 논문을 모아놓은 코퍼스
자주 등장하지 않기 때문에 IDF 값이 높아지고 문서의 핵심어가 됨	'원자'라는 단어가 여러 문서에 두루 등장하기 때문에 상투어가 됨

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

IDF(역 문서 빈도, Document Frequency)란?

DF에 반비례 하는 값으로 단어의 희귀한 정도에 대한 표현

IDF 계산 방법

$$IDF = \frac{문서수}{DF}$$

DF가 0인 경우도 있을 수 있으므로

$$IDF = \frac{문서수}{1 + DF}$$

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

IDF(역 문서 빈도, Document Frequency)란?

DF에 반비례 하는 값으로 단어의 희귀한 정도에 대한 표현

IDF 계산 방법

$$IDF = \frac{문서수}{DF}$$

DF가 0인 경우도 있을 수 있으므로

$$IDF = \frac{문서수}{1 + DF}$$

문서수가 많아지면 IDF가 지나치게 커질 수 있으므로

$$IDF = \log\left(\frac{문서수}{1 + DF}\right)$$

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

TF-IDF(단어 빈도-역 문서 빈도, Term Frequency – Inverse Document Frequency)란?

특정 문서 내에서 단어가 등장한 단순 빈도와 희귀 단어 등장 빈도를
동시에 고려한 접근 방법

특정 단어가 특정 문서 내에서
얼마나 중요한 것인지를 나타내는 통계적 가중치

문서 간 유사도 계산에서도 많이 사용

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

TF-IDF(단어 빈도-역 문서 빈도, Term Frequency – Inverse Document Frequency)란?

특정 문서 내에서 단어가 등장한 단순 빈도와 희귀 단어 등장 빈도를
동시에 고려한 접근 방법

TF-IDF 계산 방법

$$TF\text{-}IDF = TF \times IDF$$

$$IDF = \log\left(\frac{\text{문서수}}{1 + DF}\right)$$

빈도 기반 워드 임베딩

단어 빈도-역 문서 빈도(TF-IDF)

Python에서 TF-IDF 적용

- sklearn패키지의 TfidfVectorizer를 사용

```
from sklearn.feature_extraction.text import Tfidfvectiorizer
corpus = ["나는 학교에 간다",
          "나는 친구를 만난다",
          "나는 영화보러 간다",
          "영화가 재밌다"]

tfidfv = Tfidfvectiorizer().fit(corpus)
print(tfidfv.transform(corpus).toarray())
print(tfidfv.vocabulary_)
```

```
[[0.55349232 0.44809973 0.         0.         0.         0.
  0.         0.70203482]
 [0.         0.41137791 0.64450299 0.         0.         0.
  0.64450299 0.        ]
 [0.55349232 0.44809973 0.         0.         0.70203482 0.
  0.         0.        ]
 [0.         0.         0.         0.70710678 0.         0.70710678
  0.         0.        ]]
{'나는':1, '학교에':7, '간다':0, '친구를':6, '만난다':2, '영화보러':4, '영화가':3, '재밌다':5}
```

토픽 기반 워드 임베딩

LDA 개요

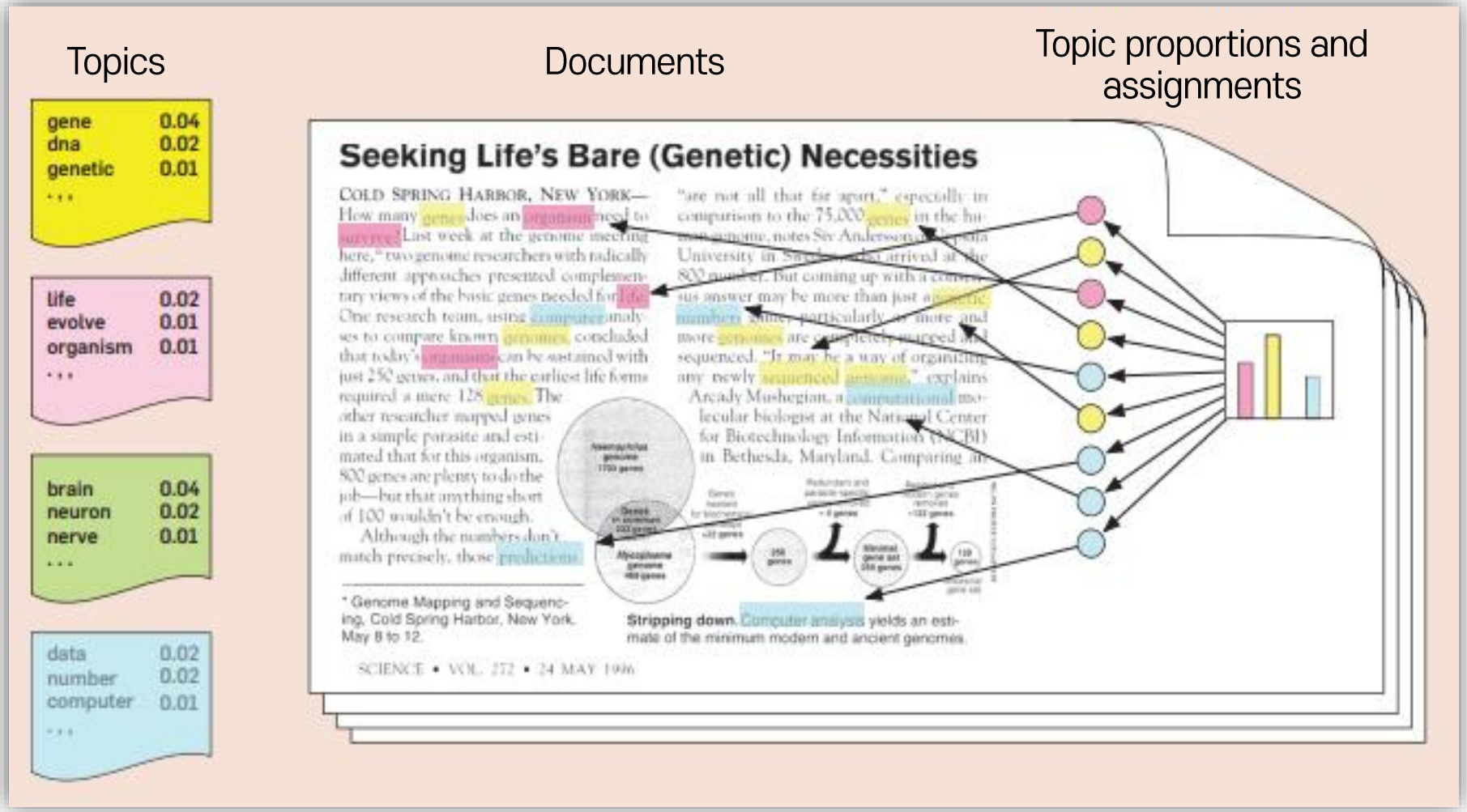
LDA(Latent Dirichlet Allocation)란?

코퍼스로부터 토픽(주제)을 추출하는
토픽 모델링(Topic Modeling) 기법 중 하나

문서에 대하여 각 문서에 어떤 주제들이 존재하는지에 대한 확률 모형

토픽 기반 워드 임베딩

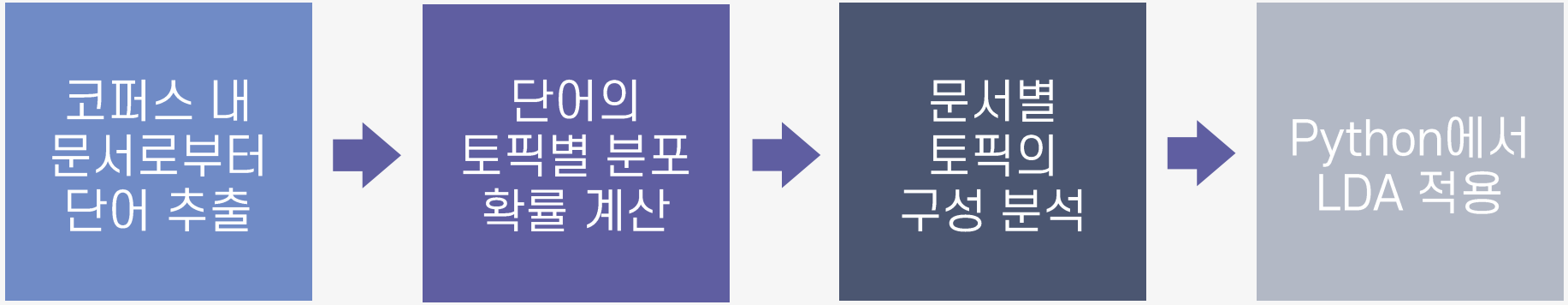
LDA 개요



<출처: David M. Blei. Probabilistic Topic Models. communications of the ACM. 2012>

토픽 기반 워드 임베딩

LDA 수행 과정



토픽 기반 워드 임베딩

LDA 수행 과정

코퍼스 내 문서로부터 단어 추출

- 빈도 수 상위 단어 기준으로 추출

코퍼스(Corpus)

brain
computer
data
dna
evolve
gene
genetic
life
...

LDA 수행 과정

단어의 토픽별 분포 확률 계산

- 각 토픽별 계산된 값의 합은 1

Topic	Topic #1	Topic #2	Topic #3	Topic #4
brain	0.000	0.000	0.040	0.001
computer	0.000	0.000	0.000	0.010
data	0.000	0.000	0.000	0.020
dna	0.020	0.007	0.002	0.000
evolve	0.009	0.010	0.001	0.000
gene	0.040	0.007	0.000	0.001
genetic	0.010	0.008	0.000	0.002
life	0.005	0.020	0.000	0.000
…	…	…	…	…
합계	1.000	1.000	1.000	1.000

토픽 기반 워드 임베딩

LDA 수행 과정

단어의 토픽별 분포 확률 계산

- 각 토픽별 계산된 값의 합은 1

Topic	Topic #1	Topic #2	Topic #3	Topic #4
brain	0.000			
computer	0.000			
data	0.000			
dna	0.020			
evolve	0.009			
gene	0.040			
genetic	0.010			
life	0.005			
…	…	…	…	…
합계	1.000	1.000	1.000	1.000

- Topic #1은 gene이라는 단어의 등장 확률이 0.04, dna는 0.02, genetic은 0.01
- Topic #1은 '유전자' 관련 주제로 판단 가능

LDA 수행 과정

문서별 토픽의 구성 분석

- 문서별 토픽이 차지하는 비중을 분석

Docs	Topic #1	Topic #2	Topic #3	Topic #4	합계
문서 #1	0.501	0.289	0.001	0.209	1.000
문서 #2	0.051	0.399	0.479	0.071	1.000
문서 #3	0.374	0.001	0.604	0.021	1.000

- 문서 #1을 보면 Topic#10이 차지하는 비중이 높으므로 문서 #1의 주제는 Topic #1일 가능성이 높음(50.1%)
- Topic #1은 '유전자' 관련 주제이므로 문서 #1의 주제는 '유전자'로 판단 가능

LDA 수행 과정

Python에서 LDA 적용

- lda 패키지의 LDA 클래스를 사용
- lda 패키지 설치

```
pip install lda
```

- lda 패키지를 이용한 LDA 적용

```
import lda
import lda.datasets
X = lda.datasets.load_reuters()
vocab = lda.datasets.load_reuters_vocab()
titles = lda.datasets.load_reuters_titles()
model = lda.LDA(n_topics=20, n_iter=1500, random_state=1)
model.fit(X)
topic_word = model.topic_word_
```