

# Lab: Text Mining and Analytics

Sungkyunkwan University (SKKU)  
Chan Lim (임찬), Bonggeon Cha (차봉건)

PPT : [bit.ly/SDS\\_day5\\_ppt\\_](https://bit.ly/SDS_day5_ppt_)  
실습 코드 : [bit.ly/SDS\\_day5\\_code\\_](https://bit.ly/SDS_day5_code_)

# 실습 자료 다운로드 및 Colab 설정

# 실습 자료 다운로드 및 Colab 설정

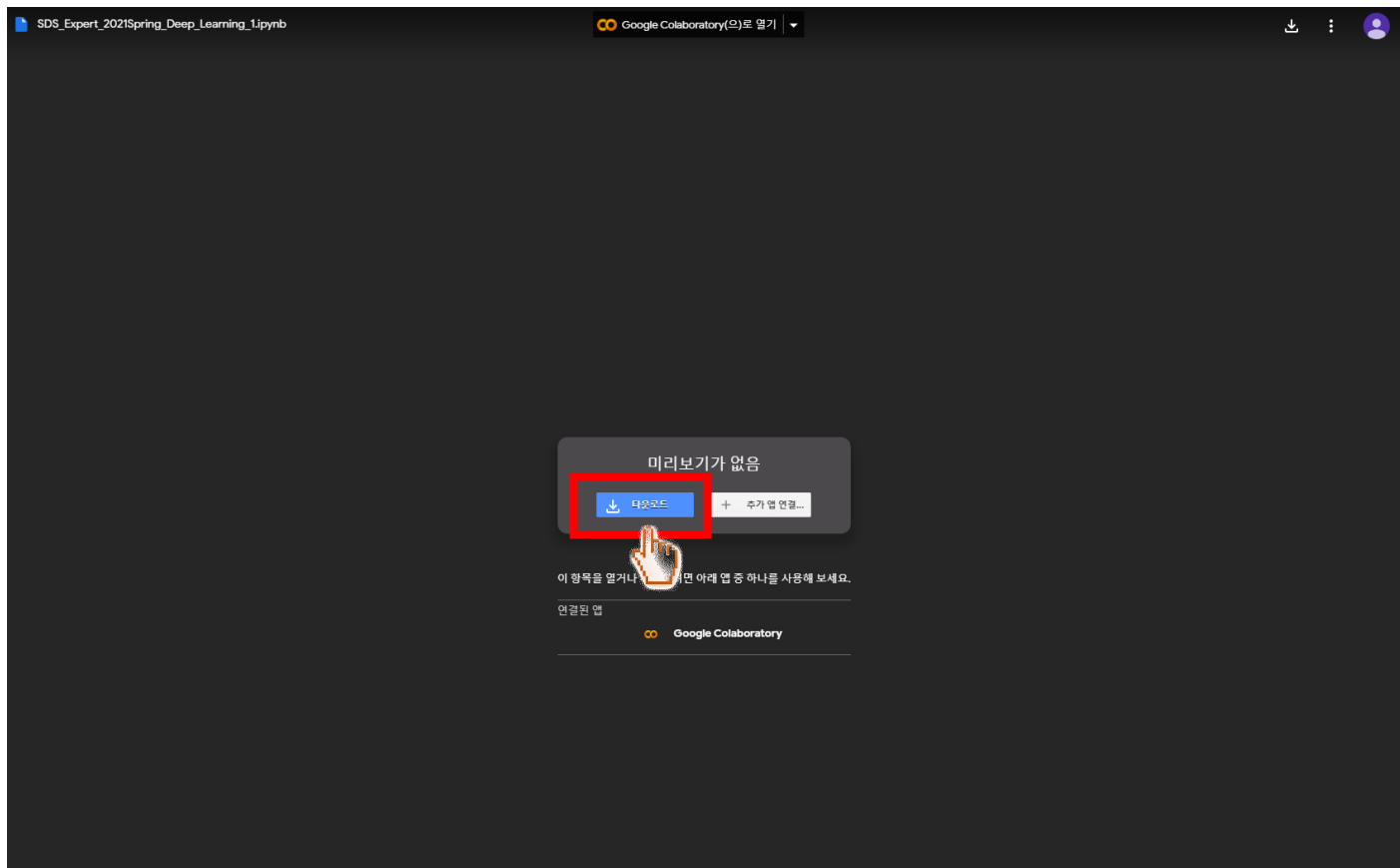
## ➤ 구글 Colab Notebook으로 실습하기

- ◆ NLP 처리 패키지를 사용하기 위해 Colab을 사용해야 합니다.

# 실습 자료 다운로드 및 기본 설정

➤ 실습 코드 URL 접속 후, 다운로드.

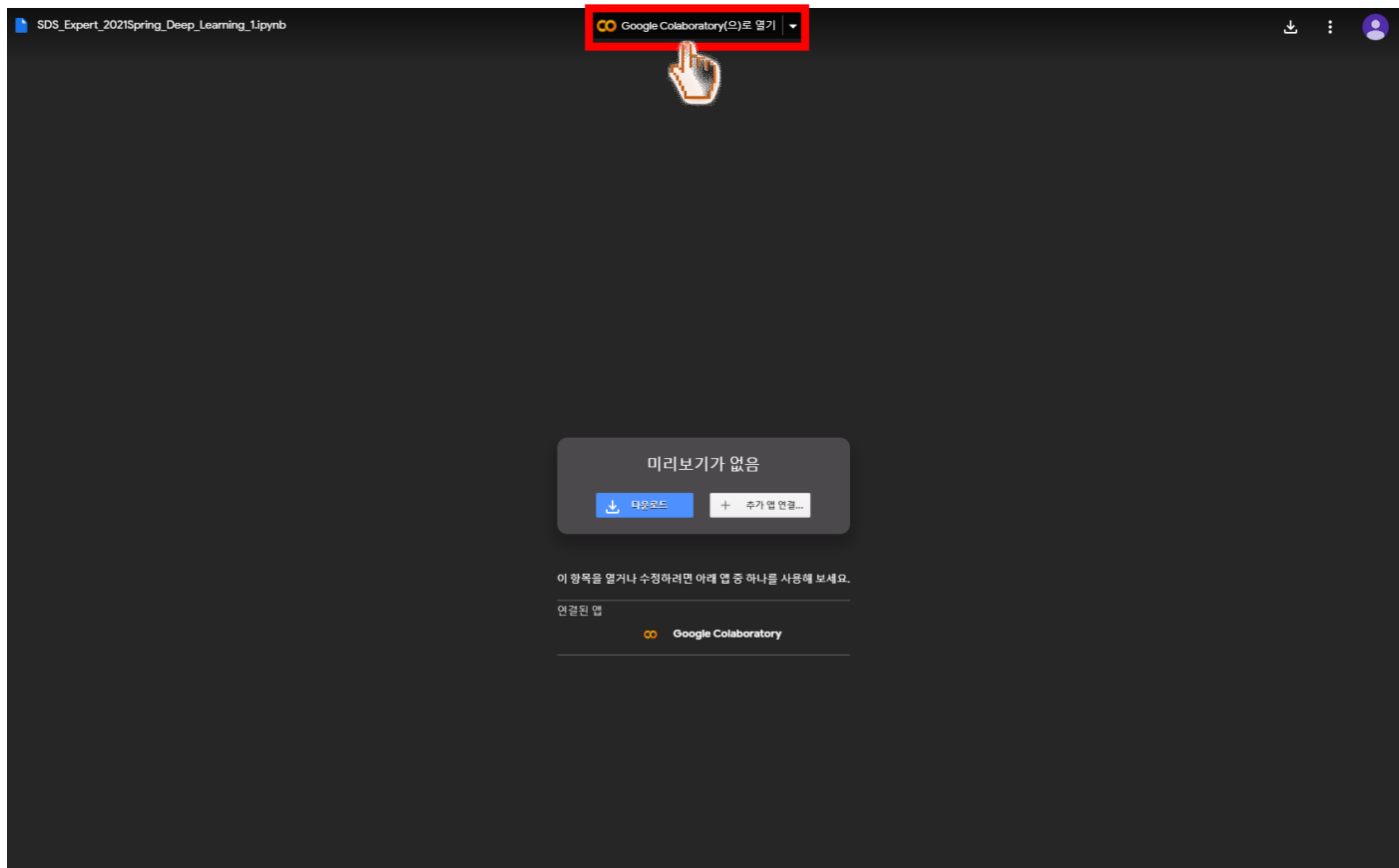
실습 코드 : [http://bit.ly/SDS\\_day5\\_code](http://bit.ly/SDS_day5_code)



# 구글 colab으로 실습하기

- 혹은 Google Colaboratory로 열기 클릭. (구글 로그인 되어야 함)

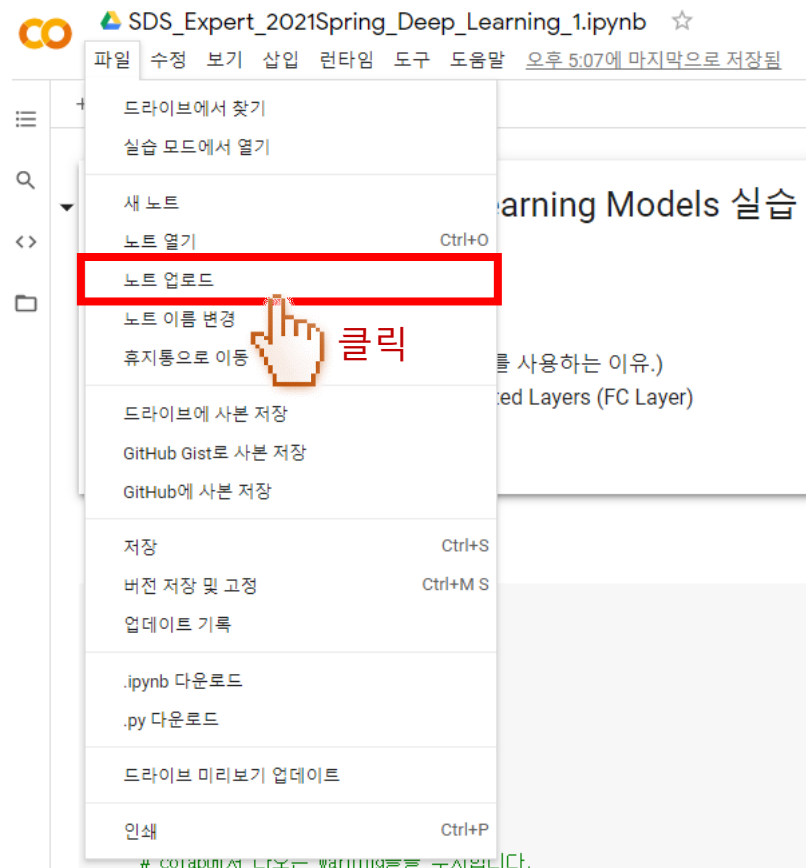
실습 코드 : [http://bit.ly/SDS\\_day5\\_code](http://bit.ly/SDS_day5_code)



# 구글 colab으로 실습하기

➤ 파일 - 노트 업로드 클릭.

실습 코드 : [http://bit.ly/SDS\\_day5\\_code](http://bit.ly/SDS_day5_code)



# 구글 colab으로 실습하기

## ➤ 파일 선택 클릭.

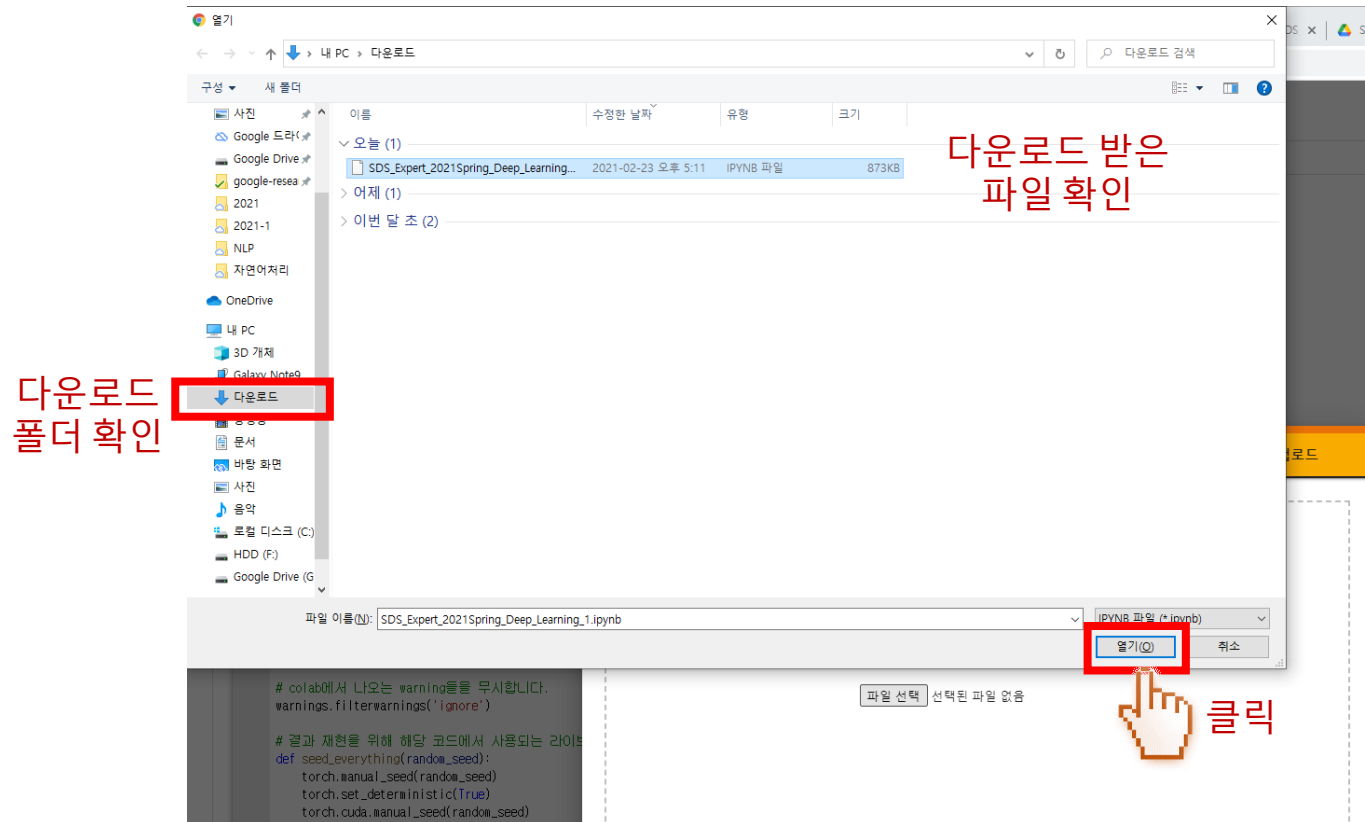
실습 코드 : [http://bit.ly/SDS\\_day5\\_code](http://bit.ly/SDS_day5_code)



# 구글 colab으로 실습하기

➤ 다운로드 받은 파일을 고른 후 열기 클릭.

실습 코드 : [http://bit.ly/SDS\\_day5\\_code](http://bit.ly/SDS_day5_code)



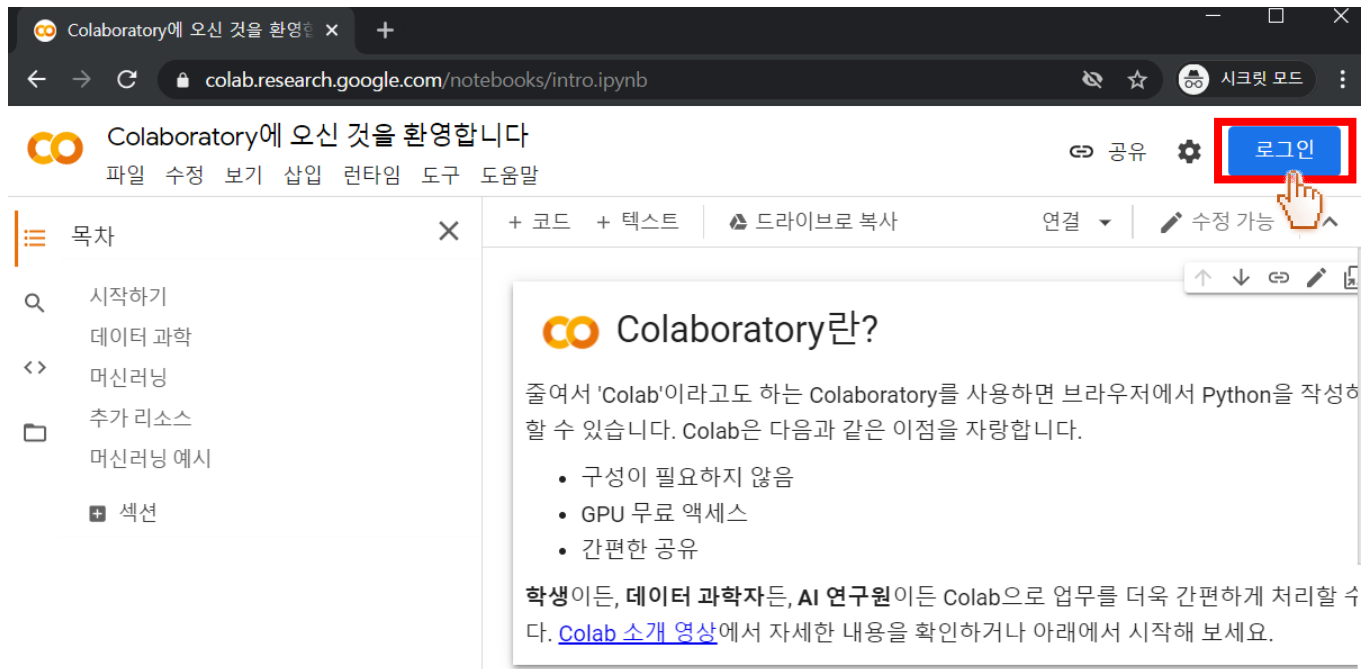


# 실습 자료 다운로드 및 기본 설정

- Chrome 브라우저를 이용해 Google Colaboratory 접속 후 본인 Google ID로 로그인

Google Colaboratory :

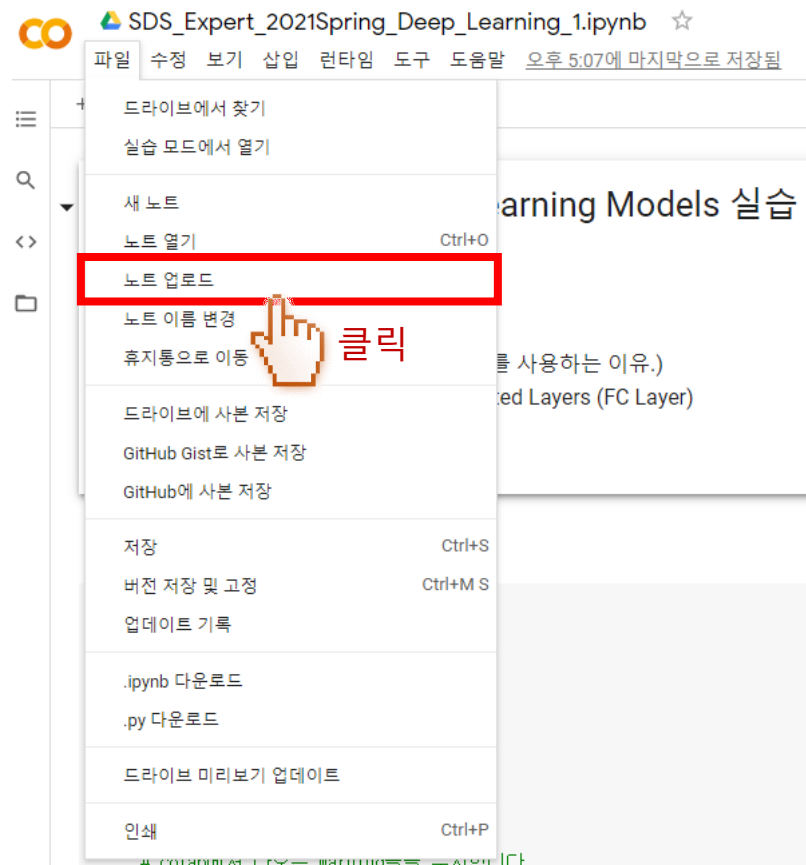
<https://colab.research.google.com>



방법 2.

# 실습 자료 다운로드 및 기본 설정

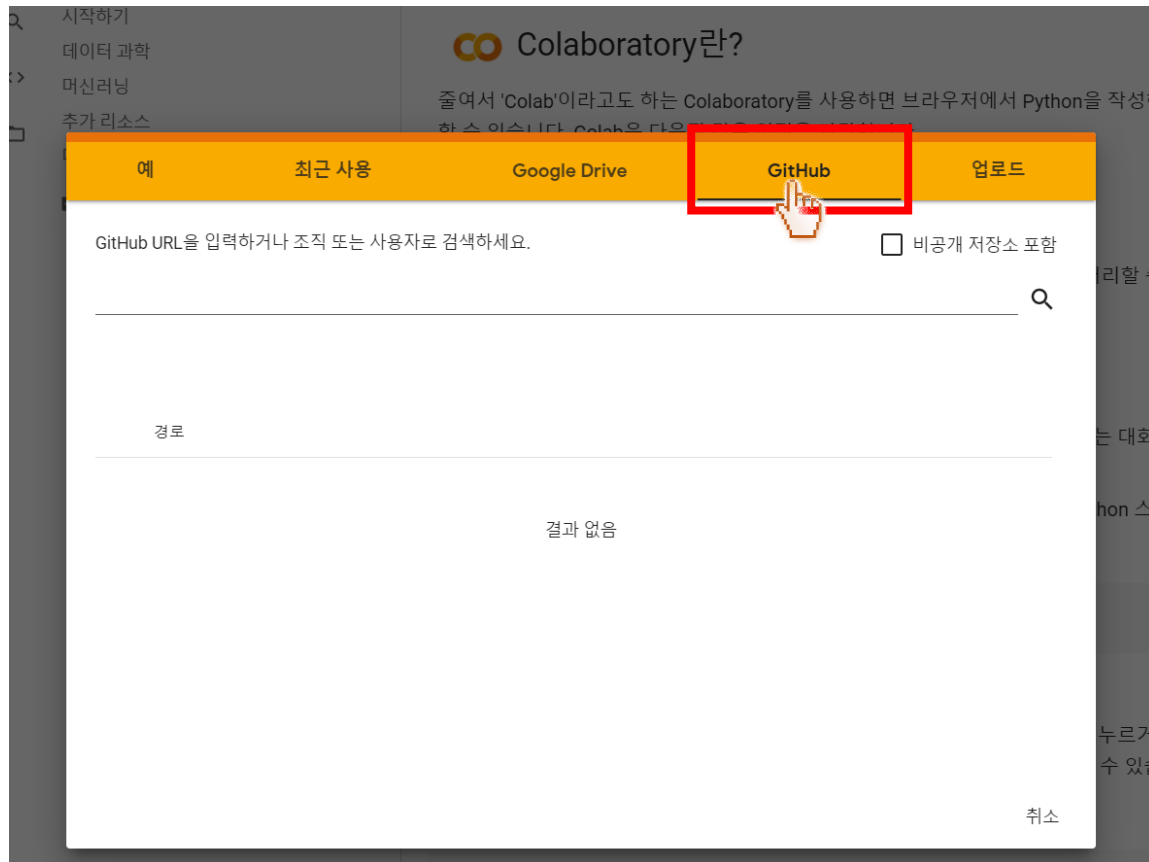
## ➤ 노트 업로드 클릭.



방법 2.

# 실습 자료 다운로드 및 기본 설정

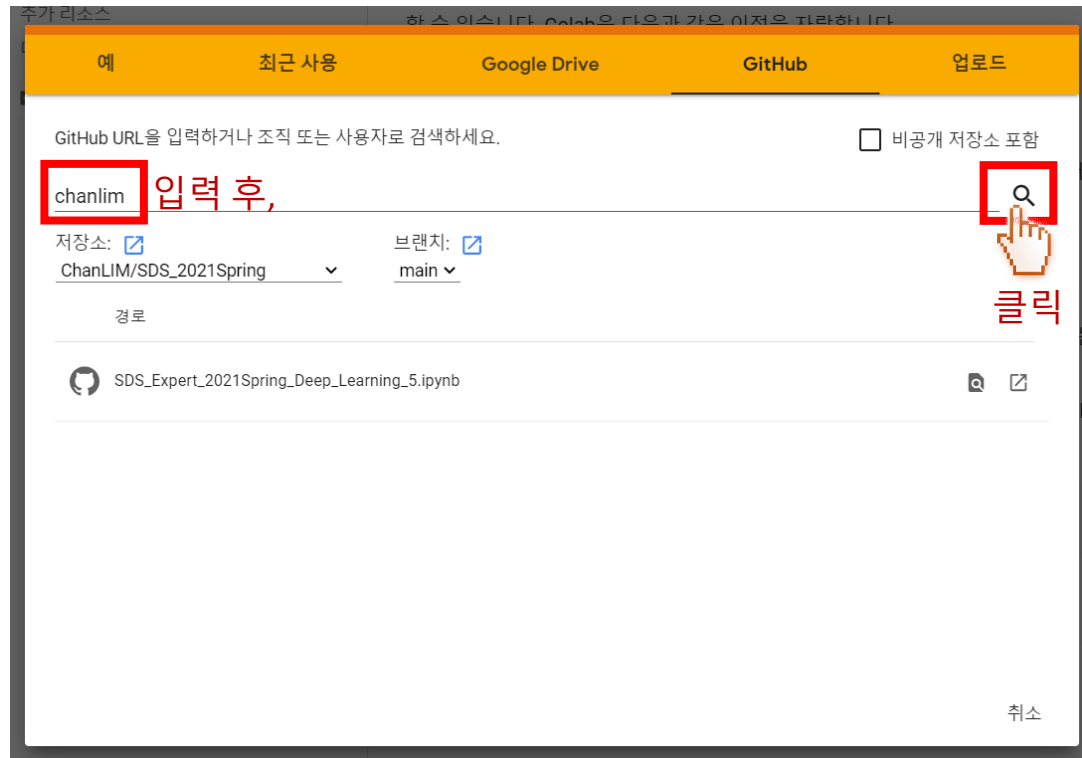
## ➤ GitHub 클릭.



방법 2.

# 실습 자료 다운로드 및 기본 설정

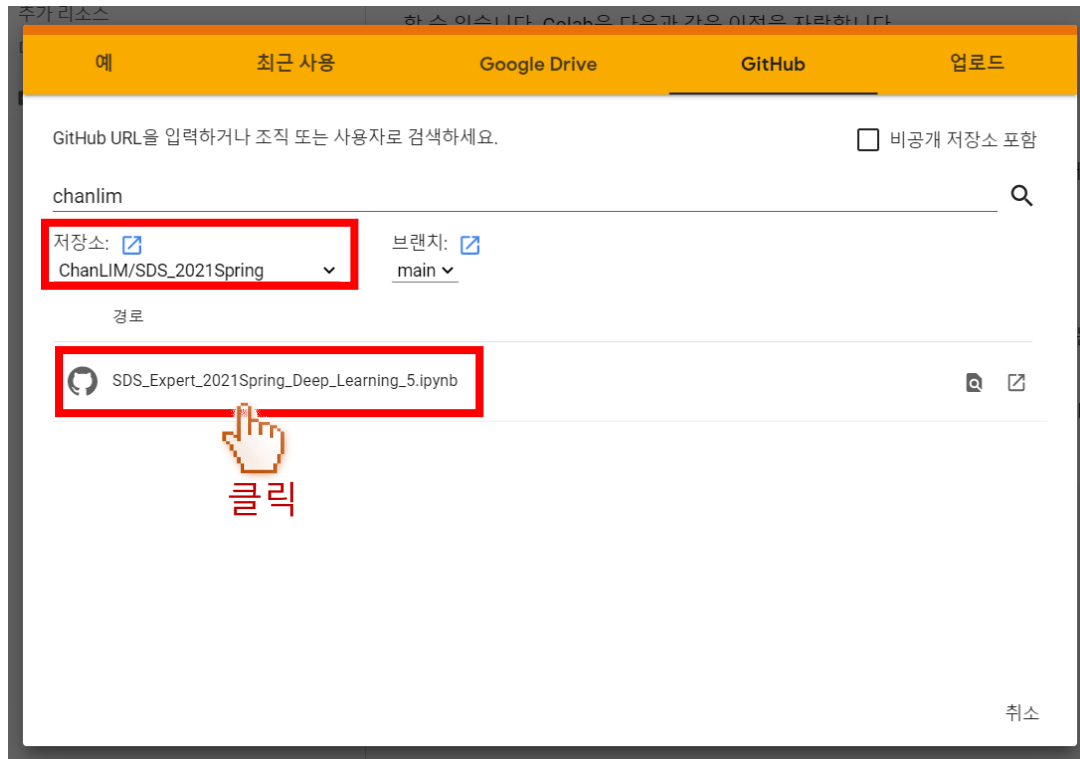
➤ 검색창에 'chanlim' 입력 후 검색 버튼 클릭.



방법 2.

# 실습 자료 다운로드 및 기본 설정

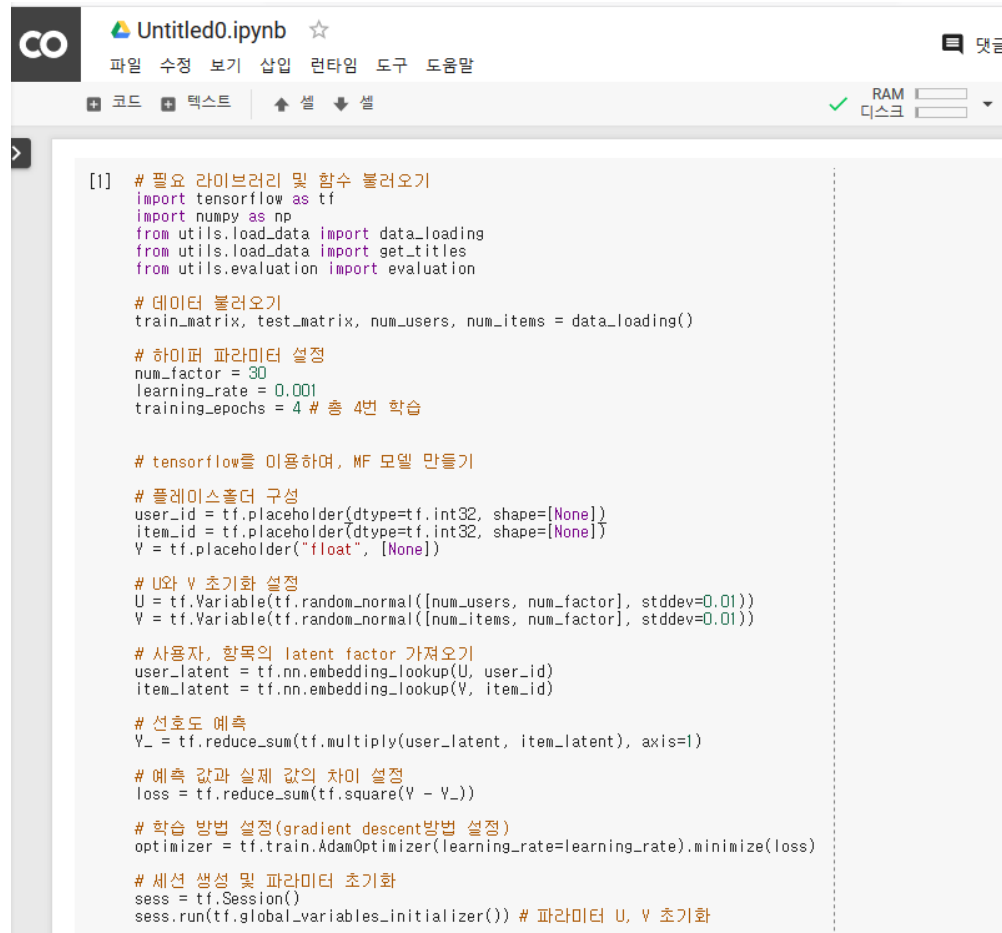
- **ChanLIM/SDS\_2021Spring 저장소에 있는**  
**SDS\_Expert\_2021Spring\_Deep\_Learning\_5\_short.ipynb**  
**클릭하여 실행**



방법 2.

# 구글 colab으로 실습하기

## ➤ Shift + Enter로 코드 실행



```
[1] # 필요 라이브러리 및 함수 불러오기
import tensorflow as tf
import numpy as np
from utils.load_data import data_loading
from utils.load_data import get_titles
from utils.evaluation import evaluation

# 데이터 불러오기
train_matrix, test_matrix, num_users, num_items = data_loading()

# 하이퍼 파라미터 설정
num_factor = 30
learning_rate = 0.001
training_epochs = 4 # 총 4번 학습

# tensorflow를 이용하여, MF 모델 만들기

# 플레이스홀더 구성
user_id = tf.placeholder(dtype=tf.int32, shape=[None])
item_id = tf.placeholder(dtype=tf.int32, shape=[None])
Y = tf.placeholder("float", [None])

# U와 V 초기화 설정
U = tf.Variable(tf.random_normal([num_users, num_factor], stddev=0.01))
V = tf.Variable(tf.random_normal([num_items, num_factor], stddev=0.01))

# 사용자, 항목의 latent factor 가져오기
user_latent = tf.nn.embedding_lookup(U, user_id)
item_latent = tf.nn.embedding_lookup(V, item_id)

# 선호도 예측
Y_ = tf.reduce_sum(tf.multiply(user_latent, item_latent), axis=1)

# 예측 값과 실제 값의 차이 설정
loss = tf.reduce_sum(tf.square(Y - Y_))

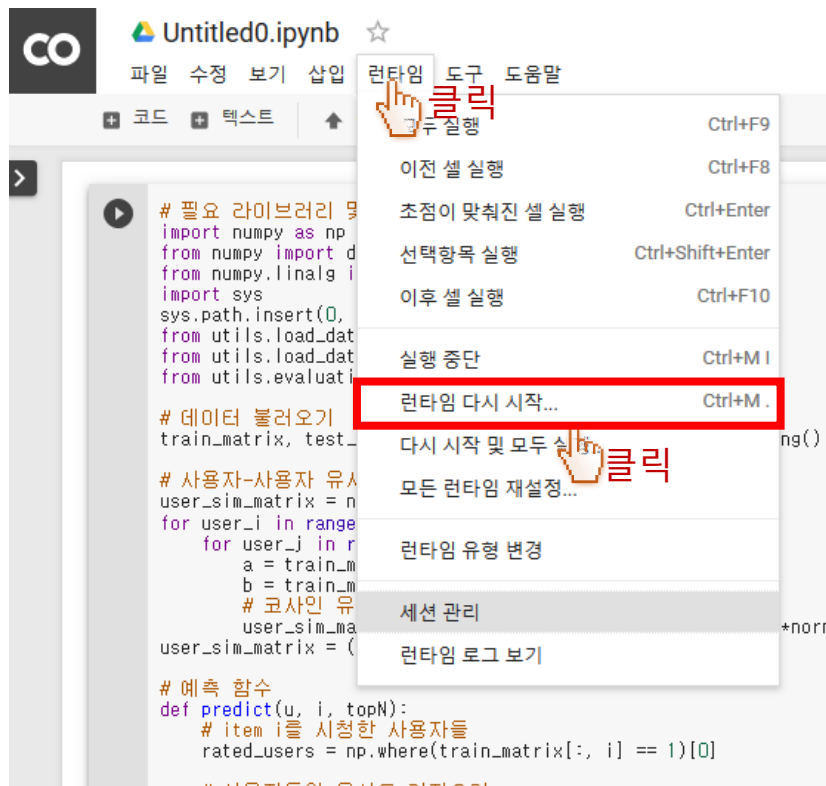
# 학습 방법 설정 (gradient descent 방법 설정)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

# 세션 생성 및 파라미터 초기화
sess = tf.Session()
sess.run(tf.global_variables_initializer()) # 파라미터 U, V 초기화
```

# 구글 colab으로 실습하기

## ➤ 실행 환경 초기화

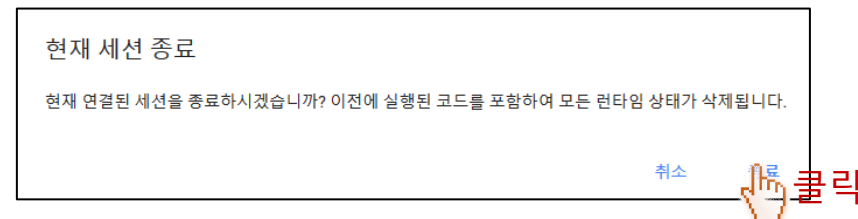
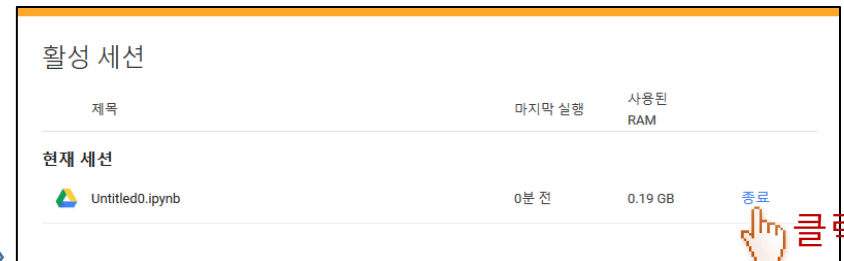
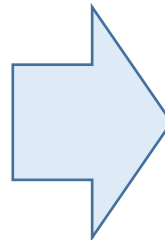
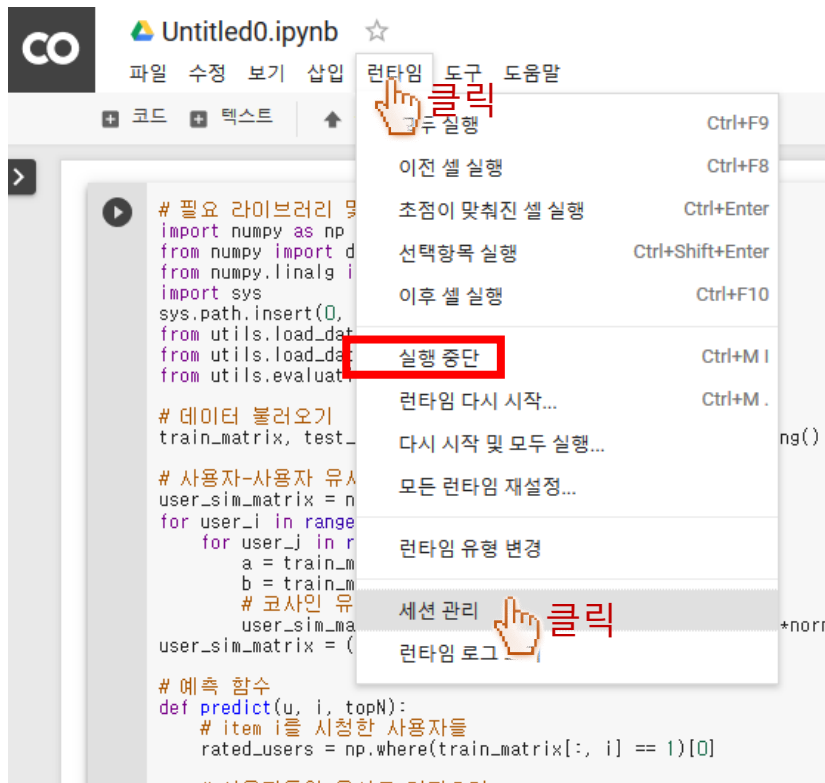
### ◆ 런타임 다시 시작



# 구글 colab으로 실습하기

## ➤ 실행 환경 초기화

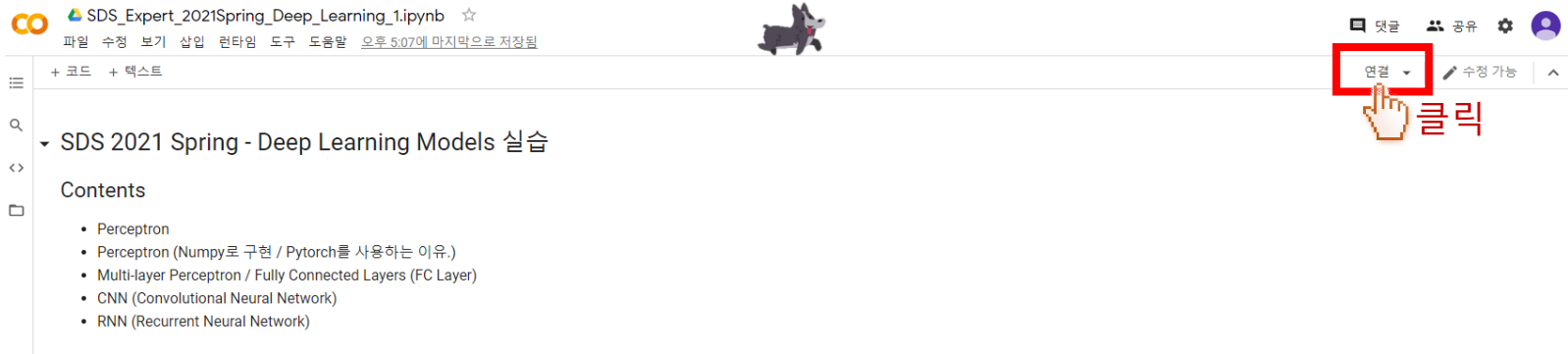
- ◆ 에러로 인하여 처음부터 다시 실행하고 싶을 때 사용





# 구글 colab으로 실습하기

## ➤ 실행 환경 연결/재연결



The screenshot shows the Google Colab interface for a notebook titled "SDS\_Expert\_2021Spring\_Deep\_Learning\_1.ipynb". The top bar includes the Colab logo, the notebook title, and a star icon. Below the title, there are links for "파일" (File), "수정" (Edit), "보기" (View), "삽입" (Insert), "런타임" (Runtime), "도구" (Tools), and "도움말" (Help). A status message indicates the notebook was last saved on May 5, 2021, at 5:07 PM. The left sidebar shows the notebook's contents, including a section titled "SDS 2021 Spring - Deep Learning Models 실습" with a list of topics: Perceptron, Perceptron (Numpy로 구현 / Pytorch를 사용하는 이유.), Multi-layer Perceptron / Fully Connected Layers (FC Layer), CNN (Convolutional Neural Network), and RNN (Recurrent Neural Network). The top right corner features icons for chat, share, settings, and user profile. A red box highlights the "연결" (Connect) button, with a hand icon pointing to it and the text "클릭" (Click) written next to it.

SDS\_Expert\_2021Spring\_Deep\_Learning\_1.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 오후 5:07에 마지막으로 저장됨

+ 코드 + 텍스트

SDS 2021 Spring - Deep Learning Models 실습

Contents

- Perceptron
- Perceptron (Numpy로 구현 / Pytorch를 사용하는 이유.)
- Multi-layer Perceptron / Fully Connected Layers (FC Layer)
- CNN (Convolutional Neural Network)
- RNN (Recurrent Neural Network)

연결 ▼ 수정 가능 ^

클릭

# 목차

## ➤ Pandas Tutorial

## ➤ Text Mining

- ◆ Text Preprocessing: Konlpy
- ◆ Word Representation
  - BOW (Bag-of-word)
  - DTM (Document-Term Matrix)
  - TF-IDF (Term Frequency – Inverse Document Frequency)
- ◆ Topic modeling
  - LDA (Latent Dirichlet Allocation) (+ word cloud)

## ➤ Recommendation System

- ◆ MF(Matrix Factorization)

# Text Mining

# 목표 및 데이터 소개

## ➤ 목표: 5가지 분야 뉴스 데이터 분석

- ◆ 분야 별 200개, 총 1,000개의 뉴스 분석
- ◆ 각 뉴스 분야/본문이 추출되어 있음
- ◆ 본문 데이터만 가지고 분석 진행.

	분류	본문
0	경제	빛 내' 주식투자 규모, 최고치 경신...남북경협·바이오株에 몰려wt【서울=뉴시스】...
1	스포츠	신진서 9단, 이세돌 꺾고 GS칼텍스배 첫 우승wt국내 바둑 랭킹 2위인 '젊은 피...
2	경제	삼성바이오로직스 "민감한 정보 무분별하게 노출...유감이다"wt【서울=뉴시스】류난영 기...
3	기술&IT	카톡+멜론→카카오', 카카오, 카카오M 합병wt카카오M 내 지적재산권 따로 사업화할...
4	스포츠	배드민턴 대표팀, 세계남녀단체선수권 출격wt[스포티비뉴스=정형근 기자] 지난해 세계...

# Pandas 튜토리얼

# Pandas Tutorial

## ➤ Pandas: python에서 사용하는 데이터 분석 라이브러리

- ◆ 표(table) 형태의 data를 다루는 데에 있어 편리
- ◆ Series: 1차원 data
- ◆ DataFrame: 2차원 data

```
df = pd.Series([2, -3, 4, -5])  
df
```

```
0    2  
1   -3  
2    4  
3   -5  
dtype: int64
```

Series

```
df = pd.DataFrame([[1, 2, 3, 4], [5, 6, 7, 8]])  
df
```

```
   0  1  2  3  
0  1  2  3  4  
1  5  6  7  8
```

DataFrame (직관적)

# Pandas Tutorial

## ➤ DataFrame 다루기

- ◆ DataFrame 생성
- ◆ DataFrame 내 data 접근/수정

# Pandas Tutorial

## ➤ DataFrame 생성

- ◆ Array / List 형태의 data로 생성
  - Array 형태 그대로 table을 구성
  - Row, column index는 따로 지정

	col0	col1	col2	col3
row0	1	2	3	4
row1	5	6	7	8
row2	9	10	11	12

# Array 형태의 data로 생성.

```
array_data = [[1,2,3,4],  
              [5,6,7,8],  
              [9,10,11,12]]
```

```
row = ['row0','row1','row2']
```

```
column = ['col0', 'col1', 'col2', 'col3']
```

```
data = pd.DataFrame(data=array_data, index=row, columns=column)  
data
```



# Pandas Tutorial

## ➤ DataFrame 내 data 접근/수정

- ◆ Column을 이용하여 바로 접근 가능.
- ◆ `Dataframe.loc[]` 형태로 접근 가능.
- ◆ 특정 열/행에 접근하는 경우 Series형태로 output 출력.
- ◆ 특정 data에 접근하여 value 출력.

	col0	col1	col2	col3
row0	1	2	3	4
row1	5	6	7	8
row2	9	10	11	12

```
# column으로 바로 접근  
data['col0']
```

```
row0    1  
row1    5  
row2    9  
Name: col0, dtype: int64
```

```
# 특정 열에 접근  
data.loc['row0',:]
```

```
col0    1  
col1    2  
col2    3  
col3    4  
Name: row0, dtype: int64
```

```
# 특정 행에 접근  
data.loc[:, 'col0']
```

```
row0    1  
row1    5  
row2    9  
Name: col0, dtype: int64
```

```
# 특정 data에 접근  
data.loc['row0', 'col0']
```

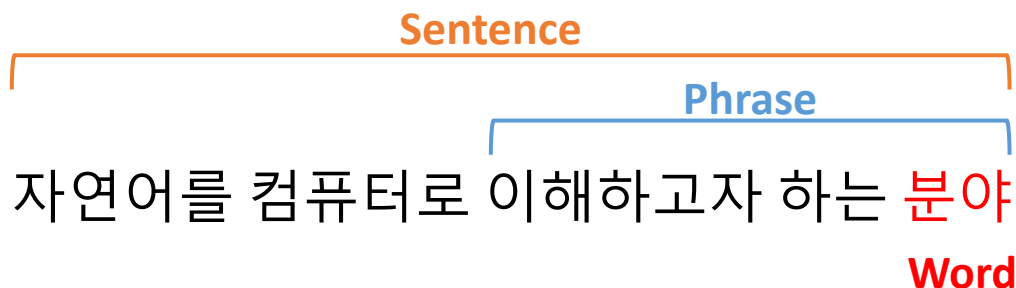
```
1
```

# Text preprocessing 실습

# Representations of Language

## ➤ 자연어를 처리하기 위한 단위

- ◆ Document
- ◆ Sentence
- ◆ Phrase
- ◆ Word



## ➤ 의미의 최소 단위 Word

- ◆ 한국어에서는 형태소
- ◆ Word를 이해하여 Phrase, Sentence, Document를 이해하자
  - Sentence = Word Sequence
  - Document = Sentence Sequence

# Text Preprocessing

## ➤ 한국어, 영어는 서로 문법이 다름.

- ◆ 전처리할때 사용하는 패키지가 다름. (한국어 – KoNLPy, 영어 – NLTK)
- ◆ 한국어 뉴스 데이터 -> KoNLPy 사용,
- ◆ KoNLPy
  - Hannanum: 한나눔, KAIST Semantic Web Research Center 에서 개발.
  - Kkma: 꼬꼬마, 서울대학교 IDS(Intelligent Data Systems) 연구실에서 개발.
  - Komoran: 코모란, Shineware에서 개발
  - Mecab: 메카브, 일본어용 형태소 분석기를 한국어를 사용할 수 있도록 수정.
  - Okt(Open Korean Text): 오픈 소스 한국어 분석기.

# Text Preprocessing

## ➤ KoNLPy

- ◆ KoNLPy에서 공통적으로 제공하는 함수
  - nouns: 명사 추출
  - morphs: 형태소 추출
  - pos: 형태소 추출 + 품사 부착

```
from konlpy.tag import Okt
Okt = Okt() # Okt (Open Korean Text)

example = '맷돌 손잡이를 어이라 그래요 어이.'

print(Okt.nouns(example))      # 명사 추출

print(Okt.morphs(example))     # 형태소 추출

print(Okt.pos(example))        # 품사 부착
```

```
['맷돌', '손잡이', '어이', '어이']
['맷돌', '손잡이', '를', '어이', '라', '그래요', '어이', '.']
[('맷돌', 'Noun'), ('손잡이', 'Noun'), ('를', 'Josa'), ('어이', 'Noun'), ('라', 'Josa'), ('그래요', 'Adjective'), ('어이', 'Noun'), ('.', 'Punctuation')]
```

- Morphs를 사용하여 가장 작은 단위인 형태소를 추출하고, 불용어를 제거하는 형태로 전처리 진행.

# Text Preprocessing

## ➤ KoNLPy

- ◆ Okt를 이용하여 morphs (형태소 추출)을 사용 하였을 때, 영어는 단어 그대로 tokenize 되어 나옴.
- ◆ 영어에 대해서는 따로 더 이상 전처리를 진행하지 않음.
  - “Dream On Air’라는 이름으로~” 라는 text에서 Dream On Air는 의미를 갖는 단어라고 생각할 수 있음.

# Text Preprocessing

## ➤ 전체 preprocessing 과정

```
import re

#불용어 정의
stopwords = ['의','가','이','은','로','및','들','는','좀','잘','강','과','도','을','를','에게','으로','자','에','와','어','하','한','하다','한다','라는','된','에서','하고','할','될','이다','있다','이었다','했다','하는','있는','쥔','입니다','됐다','까지']

tokenized_data = []
for index in range(len(dataset['본문'])):
    element = dataset.loc[index, '본문']

    # 특수문자 제거
    element = re.sub(r"^[^가-힣a-zA-Z0-9]", '', element)

    # 형태소 추출
    element = Okt.morphs(element)

    # 불용어 제거
    element = [word for word in element if not word in stopwords] # 불용어 제거

    # preprocessing을 거친 data로 수정
    tokenized_data.append(element)
    element = ' '.join(elem for elem in element)
    dataset.loc[index, '본문'] = element
```

# Word Representation 실습



# Word Representation

## ➤ Local Representation (국소 표현)

- ◆ 단어 자체만 보고 특정 값을 부여하여 표현.
- ◆ 강아지, 고양이, 민들레 -> [1, 0, 0], [0, 1, 0], [0, 0, 1]
- ◆ **Count based word representation (ex. Bow, DTM, TF-IDF)**

## ➤ Distributed Representation (분산 표현)

- ◆ 주변 단어를 참고하여 단어의 의미 뉘앙스를 담아서 표현.
- ◆ 강아지, 고양이, 민들레 -> [2.1, -1.3], [1.8, -0.9], [-1.5, 1.7]
- ◆ Word embedding (using Neural Network) (ex. Word2Vec)

# Word Representation

## ➤ BoW (Bag of Words)

- ◆ 단어의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도에만 집중한 표현 방법.

- ◆ Document -> Set of words!

- ◆ ex)

- ◆ ‘맷돌 손잡이를 어이라 그래요 어이!’



- ◆ {'그래요', '손잡이를', '맷돌', '어이', '어이라'}

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



# Word Representation

## ➤ BoW (Bag of Words)

- ◆ 단어의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도에만 집중하는 표현 방법.

- ◆ Document -> Set of words!

document = ['맷돌 손잡이를 어이라 그래요 어이']

# sklearn 사용하여 BoW

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vect = CountVectorizer()
```

```
bow = vect.fit_transform(document).toarray()
```

```
word_column = vect.get_feature_names()
```

```
pd.DataFrame(data=bow, columns=word_column)
```

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



그래요 맷돌 손잡이를 어이 어이라

그래요	맷돌	손잡이를	어이	어이라
0	1	1	1	1

# Word Representation

## ➤ DTM (Document Term Matrix)

- ◆ BoW 표현을 다수의 문서에 대해 행렬로 나타낸 것.

- ◆ Ex)

- ◆ ‘맷돌 손잡이를 어이라 그래요 어이’

- ◆ ‘맷돌 손잡이 알아요?’

- ◆ ‘황당하잖아? 아무것도 아닌 손잡이 때문에 해야 할 일을 못하니까’

	그래요	때문에	맷돌	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이라	일을	해야	황당하잖아
0	1	0	1	0	0	1	0	0	0	1	1	0	0	0
1	0	0	1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	1	1	0	0	0	1	1	1

# Word Representation

## ➤ DTM (Document Term Matrix)

- ◆ BoW 표현을 다수의 문서에 대해 행렬로 나타낸 것.

```
document = [ ' 맷돌 손잡이를 어이라 그래요 어  
이' , ' 맷돌 손잡이 알아요?', '황당하잖아? 아무  
것도 아닌 손잡이 때문에 해야 할 일을 못하니  
까']
```

```
# sklearn 사용하여 DTM (BoW와 방법 동일)  
from sklearn.feature_extraction.text import Count  
Vectorizer  
vect = CountVectorizer()  
bow = vect.fit_transform(document).toarray()  
word_column = vect.get_feature_names()  
  
pd.DataFrame(data=bow, columns=word_column)
```

	그래요	때문에	맷돌	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이라	일을	해야	황당하잖아
0	1	0	1	0	0	1	0	0	0	1	1	0	0	0
1	0	0	1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	1	1	0	0	0	1	1	1

# Word Representation

## ➤ DTM (Document Term Matrix)

- ◆ BoW 표현을 다수의 문서에 대해 행렬로 나타낸 것.
- ◆ Ex)
- ◆ ‘맷돌 손잡이를 어이라 그래요 어이’
- ◆ ‘맷돌 손잡이 알아요?’
- ◆ ‘황당하잖아? 아무것도 아닌 손잡이 때문에 해야 할 일을 못하니까’

	그래요	때문에	맷돌	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이라	일을	해야	황당하잖아
0	1	0	1	0	0	1	0	0	0	1	1	0	0	0
1	0	0	1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	1	1	0	0	0	1	1	1

Text 분석에 중요하지 않은 단어에 높은 가중치를 주게 되는 결과를 가져올 수 있음.

# Word Representation

## ➤ TF-IDF (Term Frequency – Inverse Document Frequency)

- ◆  $TF\text{-}IDF = TF * IDF$
- ◆ TF (Term Frequency)
  - 특정 문서 안에서 특정 단어의 등장 빈도.
  - DTM과 동일.

	그래요	때문에	맏들	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이라	일을	해야	황당하잖아
0	1	0	1	0	0	1	0	0	0	1	1	0	0	0
1	0	0	1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	1	1	0	0	0	1	1	1

# Word Representation

## ➤ TF-IDF (Term Frequency – Inverse Document Frequency)

- ◆  $TF\text{-}IDF = TF * IDF$
- ◆ DF (Document Frequency)
  - 특정 단어가 나타나는 문서의 수
- ◆ IDF (Inverse Document Frequency)
  - DF의 역수.  $\ln\left(\frac{1+n}{1+df}\right) + 1$ . (n = 총 문서의 수, df = DF)

	그래요	때문에	맷돌	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이라	일을	해야	황당하잖아
0	1	0	1	0	0	1	0	0	0	1	1	0	0	0
1	0	0	1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	1	1	0	0	0	1	1	1

‘맷돌’은 0, 1번째 문서에서 등장 -> DF : 2, IDF:  $\ln\left(\frac{1+3}{1+2}\right) + 1$



# Word Representation

## ➤ TF-IDF (Term Frequency – Inverse Document Frequency)

- ◆ Normalize in Sklearn TF-IDF
- ◆ Normalize
  - 각 문서에 대해 normalize.
  - 각 원소의 제곱의 합이 1이 되도록 함. (L2 normalization)

	그래요	때문에	맷돌	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이랴	일을	해야	황당하잖아
0	0.467351	0.000000	0.355432	0.000000	0.000000	0.467351	0.000000	0.000000	0.000000	0.467351	0.467351	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.517856	0.000000	0.517856	0.000000	0.000000	0.000000	0.680919	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.363255	0.000000	0.363255	0.276265	0.000000	0.363255	0.363255	0.000000	0.000000	0.000000	0.363255	0.363255	0.363255

↑  
제곱의 합이 1.

# Word Representation

## ➤ TF-IDF (Term Frequency – Inverse Document Frequency)

◆  $TF\text{-}IDF = TF * IDF$

```
document = [ '맷돌 손잡이를 어이라 그래요 어이', '맷돌 손잡이 알아요?', '황당하잖아? 아무것도 아닌 손잡이 때문에 해야 할 일을 못하니까']
```

```
# sklearn 사용하여 TF-IDF
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
Tfidf_vect = TfidfVectorizer()
```

```
Tfidf = Tfidf_vect.fit_transform(document).toarray()
```

```
word_column = Tfidf_vect.get_feature_names()
```

```
pd.DataFrame(data=bow, columns=word_column)
```

	그래요	때문에	맷돌	못하니까	손잡이	손잡이를	아닌	아무것도	알아요	어이	어이라	일을	해야	황당하잖아
0	0.467351	0.000000	0.355432	0.000000	0.000000	0.467351	0.000000	0.000000	0.000000	0.467351	0.467351	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.517856	0.000000	0.517856	0.000000	0.000000	0.000000	0.680919	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.363255	0.000000	0.363255	0.276265	0.000000	0.363255	0.363255	0.000000	0.000000	0.000000	0.363255	0.363255	0.363255

# Word Representation

## ➤ News Dataset으로 진행했을 때 결과

### ◆ BoW (Bag of Words)

100 억 원	12 일	17 일	18	18 일	1 년	2014 년	2018 년	20 일	22 일	27 일	35	3 년	9 일	gs	lg	가 는	가 를	가 져	감 지	감 치	강 서 구	강 조	갖 고	갈 습 니다	갈 은	갈 을	개 발	개 월	개 장	거 기	거 쳐	검 찰	겁 니 다	게 다 가	격 돌	경 기	경 제	경 제 정 책	계 기	...	탈 루
0	1	1	1	1	1	1	1	1	1	2	1	1	1	1	15	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	2	2	1	1	2	1	1	...	1

1 rows × 365 columns

### ◆ DTM (Document Term Matrix)

0 원	10	100	100 억 원	11	12	12 일	13	136	14	143	148	17	17 일	18	186	18 일	19	19 일	1 년	1 원	1 천	2004 년	200 만	2014 년	2015 년	2016 년	2018	2018 년	20 일	22 일	2725	27 일	2800 만	29	2 년	
0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1	1	0	2	0	0	0	
1	5	2	0	0	1	4	0	1	0	2	0	0	1	0	0	0	0	1	0	2	13	0	0	0	1	1	0	0	0	0	0	0	0	1	1	
2	0	0	1	0	1	2	0	2	1	2	1	1	0	0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	1	0	0
3	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	
4	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	

5 rows × 926 columns

# Word Representation

## ➤ News Dataset으로 진행했을 때 결과

### ◆ TF-IDF(Term Frequency – Inverse Document Frequency)

	0원	10	100	100억 원	11	12	12일	13	136	14	143	148	17	17일	18	186	18일
0	0.000000	0.000000	0.000000	0.03018	0.000000	0.000000	0.03018	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.03018	0.03018	0.000000	0.024349
1	0.157186	0.062874	0.000000	0.000000	0.025363	0.101453	0.000000	0.025363	0.000000	0.050727	0.000000	0.000000	0.031437	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.038008	0.000000	0.030665	0.061330	0.000000	0.061330	0.038008	0.061330	0.038008	0.038008	0.000000	0.000000	0.000000	0.038008	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.025789
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

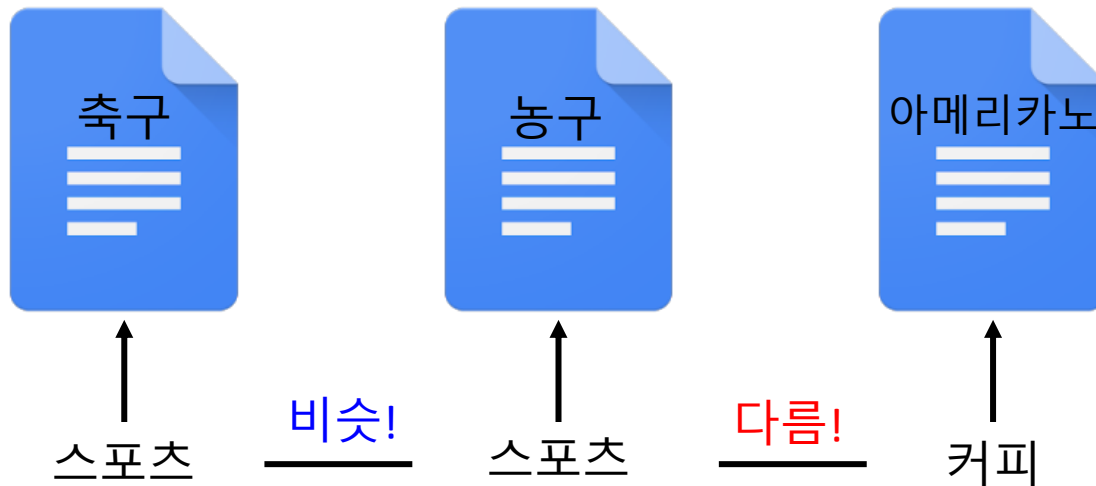
5 rows × 926 columns

# 주제 모델 실습

# 주제 모델이란?

## ➤ Topic Modeling

- ◆ 문서 집합에 대해 topic(주제)을 발견하기 위한 모델.
- ◆ Text의 숨겨진 의미 구조를 발견하기 위해 사용.



이미지 출처:

[https://lh3.googleusercontent.com/proxy/SfWy8CGSOLm4lOTCwLERB28CVw4mpOy6mn91Jvj4x9NWt\\_eCF3qnli5W\\_ouVHAFjFcH-LJ26yvMP2g70MmRIXuX07MUuUBgH81mtHTj9ZnLGwnBNSngF14-CmwpO650Gs\\_vIROBbFMi9Rn\\_KcyHUeGDWCLs4dfG703UV46l49DA96GJr\\_tV3oaxVy2ZjOExJdsjn4tLZ6LJF0J\\_2X6xhL1bazy\\_ifgVOhkt-d\\_JXvwUBpAk](https://lh3.googleusercontent.com/proxy/SfWy8CGSOLm4lOTCwLERB28CVw4mpOy6mn91Jvj4x9NWt_eCF3qnli5W_ouVHAFjFcH-LJ26yvMP2g70MmRIXuX07MUuUBgH81mtHTj9ZnLGwnBNSngF14-CmwpO650Gs_vIROBbFMi9Rn_KcyHUeGDWCLs4dfG703UV46l49DA96GJr_tV3oaxVy2ZjOExJdsjn4tLZ6LJF0J_2X6xhL1bazy_ifgVOhkt-d_JXvwUBpAk)

# Latent Dirichlet Allocation (LDA)

- Motivation: 글을 쓸 때, 어떤 순서로 글을 쓰게 되는가
  - ◆ 사람은 글을 쓰기 전, 글에 대한 **주제**들을 결정한다.
  - ◆ LDA는 **주제**가 정해지면 해당 **주제**에 대해 자주 쓰이는 **단어**가 있다고 생각한다.
- ◆ Ex)
  - ◆ 주제를 커피로 먼저 정한다.
  - ◆ ‘맥도날드 아메리카노가 생각보다 맛있네!’



이미지 출처:

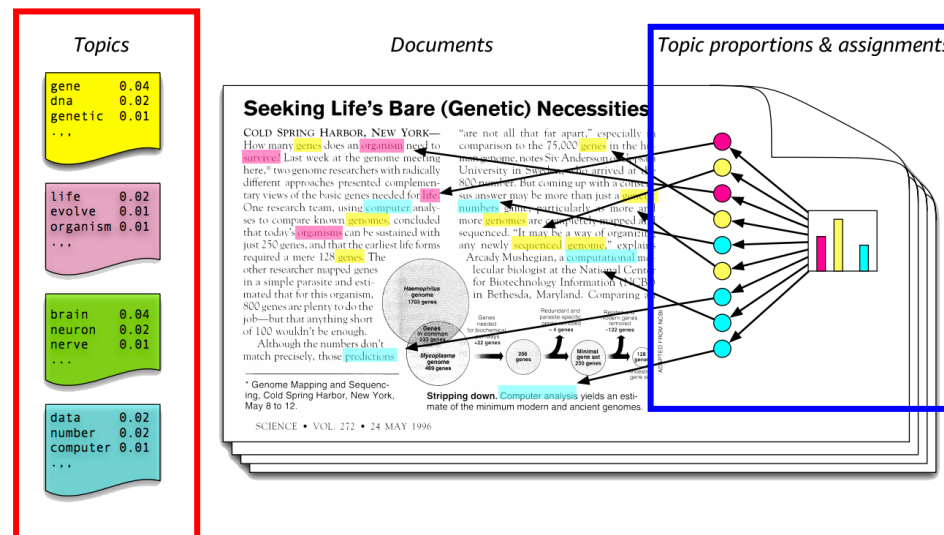
<https://www.google.com/url?sa=i&url=https%3A%2F%2Fblog.daum.net%2Firepublic%2F7888705&psig=AOvVaw0HlrhQpDmeNVli7-QbFguN&ust=1614308152706000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLiHlt2EhO8CFQAAAAdAAAAABAD>

# Latent Dirichlet Allocation (LDA)

## ➤ Topic Modeling

- ◆ 문서 집합에 대해 topic(주제)을 발견하기 위한 모델.
- ◆ Text의 숨겨진 의미 구조를 발견하기 위해 사용.
- ◆ 문서에서 많이 등장하는 단어가 그 문서의 주제가 될 수 있다.

## ➤ 문서 별 토픽 분포 $P(T|D)$ , 토픽 별 단어 분포 $P(W|T)$





# Latent Dirichlet Allocation (LDA)

## ➤ Topic 수는 따로 결정 (Hyperparameter)

- ‘저는 사과랑 바나나를 먹어요’
- ‘우리는 귀여운 강아지가 좋아요’
- ‘저의 깜찍하고 귀여운 강아지가 바나나를 먹어요’
- Topic: 2개라고 가정.

모든 단어를 random하게 하나의 Topic에 할당.

Word	사과		바나나		먹어요	
Topic	1		1		2	
Word	귀여운		강아지		좋아요	
Topic	2		2		1	
Word	깜찍하고	귀여운	강아지	바나나	먹어요	
Topic	1	2	2	1	2	

# Latent Dirichlet Allocation (LDA)

- 가정: 자신의 토픽은 잘못되었고, 다른 단어들은 올바르게 할당되었다.
- 문서의 Topic이 무엇인가
  - ◆ 문서 별 토픽 분포  $P(T|D)$  고려
  - ◆ Topic 1: 50%, Topic2: 50%임을 고려.

Word	사과		바나나		먹어요	
Topic	1		???		2	

Word	귀여운		강아지		좋아요	
Topic	2		2		1	

Word	깜찍하고	귀여운	강아지	바나나	먹어요
Topic	1	2	2	1	2

# Latent Dirichlet Allocation (LDA)

- ‘바나나’가 전체 문서에서 어떤 토픽에 할당되어 있는지.
  - ◆ 토픽 별 단어 분포  $P(W|T)$  고려
  - ◆ 다른 ‘바나나’ 단어가 Topic 1에 해당 함을 고려

Word	사과		바나나		먹어요	
Topic	1		???		2	
Word	귀여운		강아지		좋아요	
Topic	2		2		1	
Word	깜찍하고	귀여운	강아지	바나나	먹어요	
Topic	1	2	2	1	2	

# Latent Dirichlet Allocation (LDA)

- 모든 단어에 대해 차례대로 반복하다 보면 수렴한다!
- ‘저는 사과랑 바나나를 먹어요’
- ‘우리는 귀여운 강아지가 좋아요’
- ‘저의 깜찍하고 귀여운 강아지가 바나나를 먹어요’

	Topic 1	Topic 2
Doc 1	1.0	0
Doc 2	0	1.0
Doc 3	0.4	0.6

	사과	바나나	먹어요	귀여운	강아지	깜찍하고	좋아요
Topic 1	0.2	0.4	0.4	0	0	0	0
Topic 2	0	0	0	0.3	0.3	0.16	0.16

- ◆ 실제 구현: Sklearn 패키지 사용 + TF-IDF(DTM) 행렬을 인풋으로 사용.

LDA 수행 과정에 대한 참고

<https://wikidocs.net/30708>

<https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/06/01/LDA/>

# Latent Dirichlet Allocation (LDA)

- 실제 구현: Sklearn 패키지 사용 + TF-IDF or DTM 행렬을 인풋으로 사용.

```
document = dataset['본문']

from sklearn.feature_extraction.text import TfidfVectorizer
Tfidf_vect = TfidfVectorizer()
TF-IDF → Tfidf_vect = Tfidf_vect.fit(document)
Tfidf = Tfidf_vect.fit_transform(document).toarray()
word_index = Tfidf_vect.get_feature_names()

from sklearn.decomposition import LatentDirichletAllocation as LDA

LDA → lda = LDA(n_components=5, random_state=0)
lda = lda.fit(Tfidf)

P(T|D) → topic_word_dist = lda.components_
P(W|T) → doc_topic_dist = lda.transform(Tfidf)
```

# Latent Dirichlet Allocation (LDA)

## ➤ 문서 별 토픽 분포 $P(T|D)$

- ◆ 문서 별 토픽이 어느 비중으로 분포하는지 알 수 있다.
- ◆ 가장 많은 비중을 차지하는 topic -> dominant topic

	Topic_1	Topic_2	Topic_3	Topic_4	Topic_5	Dominant_Topic	Dominant_Topic_Score
Doc_1	0.016320	0.016320	0.016559	0.934483	0.016319	Topic_4	0.934483
Doc_2	0.233529	0.031064	0.672907	0.031436	0.031065	Topic_3	0.672907
Doc_3	0.021026	0.021000	0.021273	0.915701	0.020999	Topic_4	0.915701
Doc_4	0.027147	0.027145	0.027326	0.891234	0.027148	Topic_4	0.891234
Doc_5	0.019229	0.019222	0.922861	0.019465	0.019222	Topic_3	0.922861

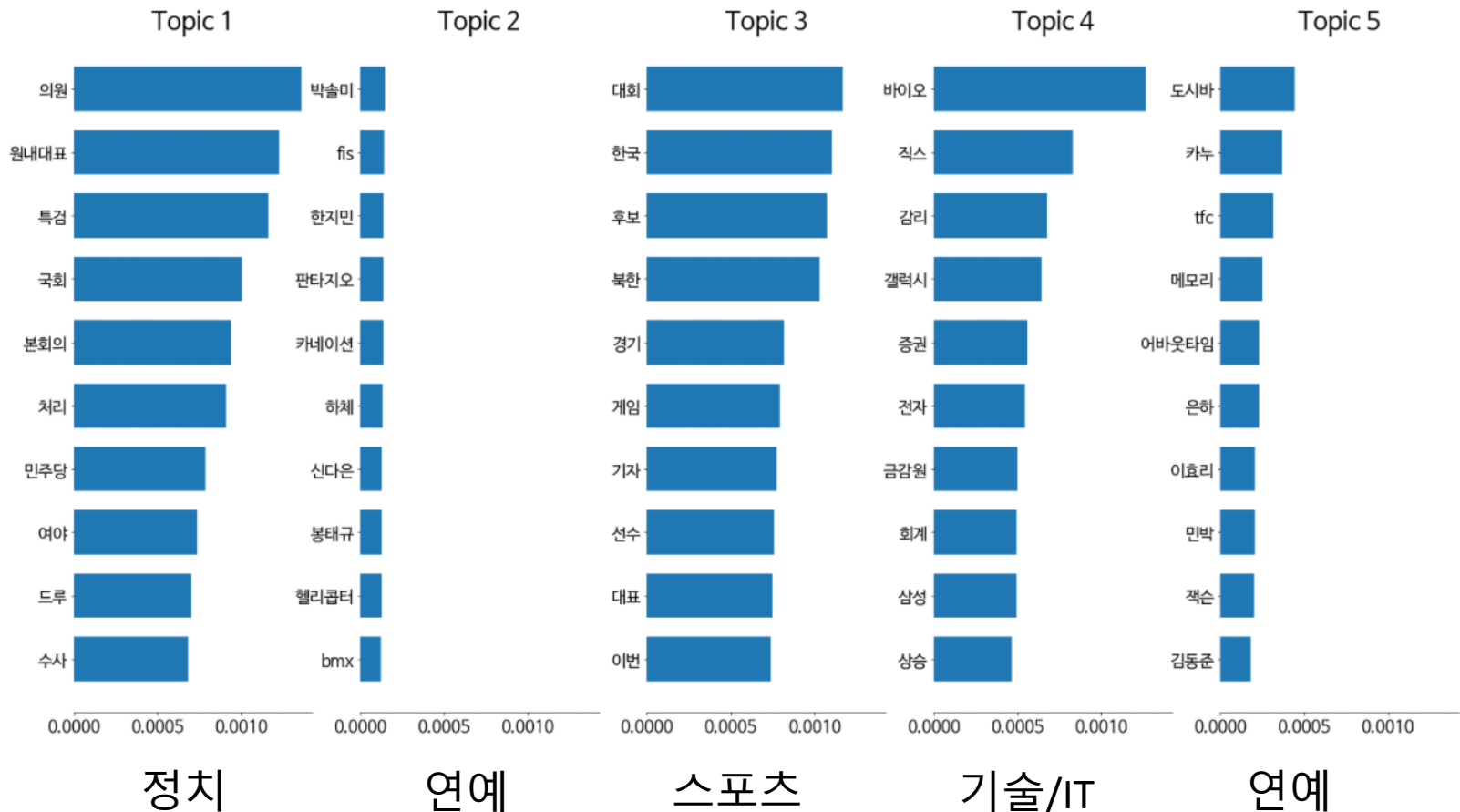
문서 각각에 대해 dominant topic을 찾을 수 있다.  
-> 1,3,4번 문서가 비슷한 주제를 가지고 있구나!  
-> 2,5번 문서가 비슷한 주제를 가지고 있구나!

# Latent Dirichlet Allocation (LDA)

## ➤ 토픽 별 단어 분포 $P(W|T)$

- ◆ 경제/정치의 구분이 모호하고, 연예가 2개로 쪼개진 모습.

Topics in LDA model



# Word Cloud

- Text의 비중에 비례하여 크게 나타냄으로서 중요한 단어를 시각적으로 돋보이게 하는 기법.
- 토픽 별 단어 분포  $P(W|T)$  에서 각 토픽별로 생성.

```
from wordcloud import WordCloud
```

```
wordcloud = WordCloud(font_path= 'NanumBarunGothic',  
                        background_color='white',  
                        width=2500,  
                        height=1800,  
                        max_words=10)
```

```
plot_wordcloud(lda, Tfidf_vect.get_feature_names())
```



# Word Cloud

```
def plot_wordcloud(model, feature_names):  
    fig, axes = plt.subplots(1, 5, figsize=(30, 15), sharex=True, sharey=True) # 1 x 4의 subplots 생성  
    axes = axes.flatten()  
  
    for topic_idx, topic in enumerate(model.components_):  
        topic_word_dict = {}  
        for i, word in enumerate(feature_names):  
            topic_word_dict[word] = topic[i] # 특정 topic에 대한 word들의 비중(중요도)을 저장.  
  
        ax = axes[topic_idx]  
        ax.set_title('Topic ' + str(topic_idx + 1), fontdict={'fontsize': 20})  
        wordcloud.generate_from_frequencies(topic_word_dict)  
        ax.imshow(wordcloud)  
        ax.axis('off')
```

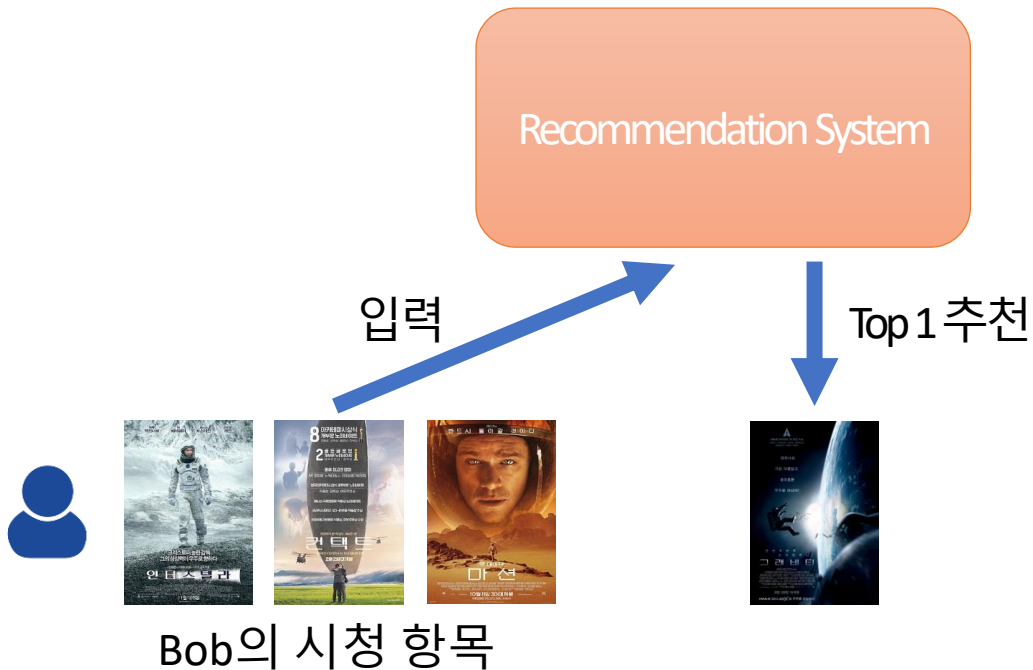


# 추천 모델 실습 - 행렬 분해 모델

# 추천 모델 목표

- 사용자가 시청하지 않은 항목 중 상위 N개의 선호할 항목을 추천 함

전체 항목



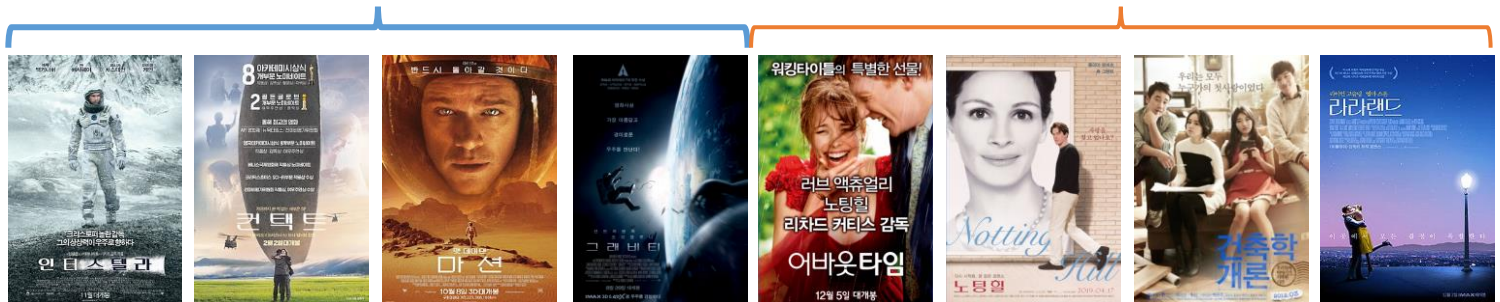
# 추천 모델 목표

## ➤ Bob이 시청하지 않은 Item4 ~ Item8 중 하나를 추천하기

- ◆ 1은 시청 0(공란)은 시청하지 않음을 의미함
- ◆ 공란의 숫자를 예측하여 수치가 가장 높은 영화를 추천

S.F. 영화들

로맨스 영화들



	Item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8
<b>Bob</b>	1	1	1	???	???	???	???	???
User1	1	1	1	1				
User2		1	1	1	1	1		
User3			1	1	1	1	1	
User4					1	1	1	1

# Matrix Factorization (MF)

➤ 각 user와 item을  $k$  차원의 vector로 표현.

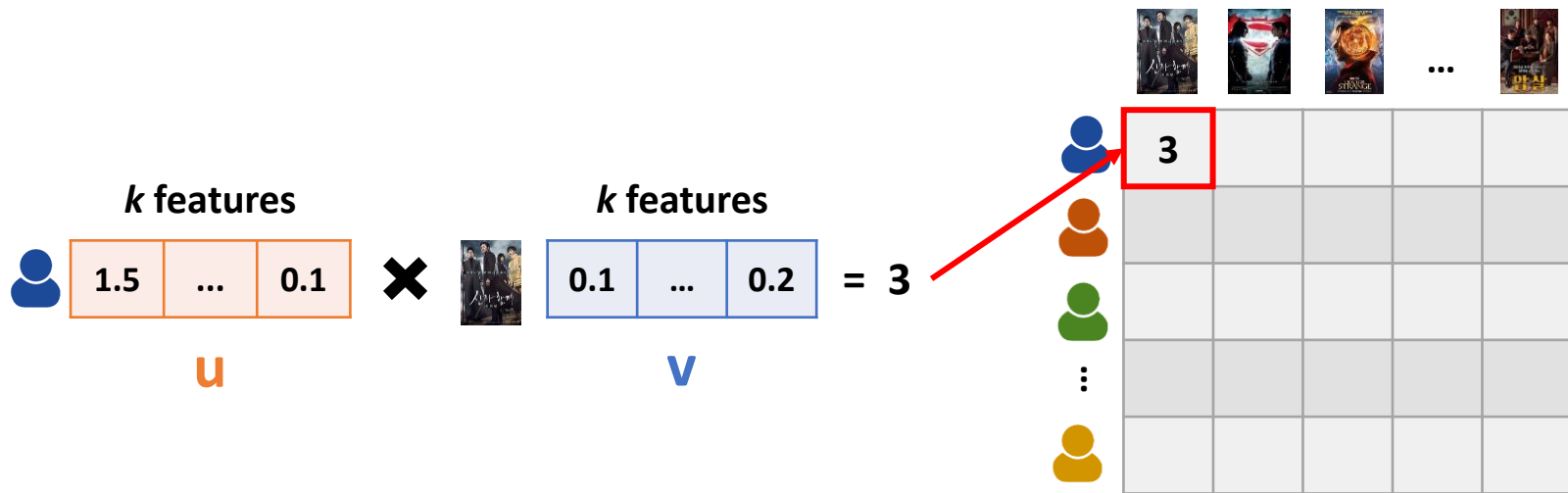
- ◆  $u$ : latent 사용자 벡터 ( $k$  dim vector)
- ◆  $v$ : latent 항목 벡터 ( $k$  dim vector)
  - $k$ : # of latent features



# Matrix Factorization (MF)

➤ 각 user와 item을  $k$  차원의 vector로 표현.

- ◆  $u$ : latent 사용자 벡터 ( $k$  dim vector)
- ◆  $v$ : latent 항목 벡터 ( $k$  dim vector)
  - $k$ : # of latent features



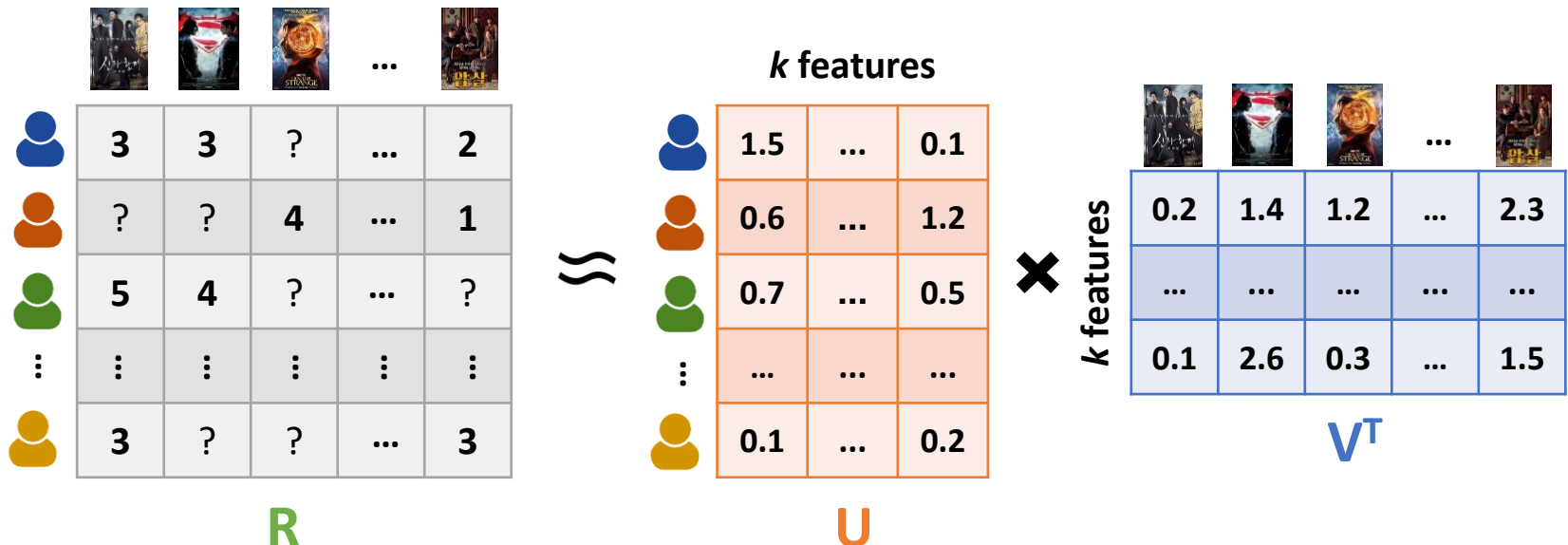
목표 : user와 item latent vector를 이용해  
User가 실제로 보지 않은 영화에 대해서도 예측

# Matrix Factorization (MF)

➤ 사용자-항목 행렬 **R**은 두 latent 행렬 **U**와 **V**의 선형 결합으로 근사할 수 있음

- ◆ **R**: 사용자-항목 행렬 ( $m \times n$  matrix)
- ◆ **U**: latent 사용자 행렬 ( $m \times k$  matrix)
- ◆ **V**: latent 항목 행렬 ( $n \times k$  matrix)
  - $k$ : # of latent features

**Gradient descent method**로 학습 됨



# 단계1: U와 v 초기화

➤ U와 v를 임의의 값으로 초기화 함

<i>Bob</i>	-0.008	0.021	0.029
<i>u1</i>	-0.001	0.039	0.022
<i>u2</i>	-0.008	0.029	-0.009
<i>u3</i>	-0.01	0.016	-0.012
<i>u4</i>	-0.035	0.012	-0.049

U

<i>i1</i>	<i>i2</i>	<i>i3</i>	<i>i4</i>	<i>i5</i>	<i>i6</i>	<i>i7</i>	<i>i8</i>
-0.02	-0.005	-0.003	0.005	-0.013	-0.016	-0.023	-0.017
0.022	0.028	0.028	0.018	0.005	-0.001	0.028	0.005
0.026	0.021	0.006	-0.005	-0.027	-0.023	-0.018	-0.026

$V^T$



# 단계2: MF를 이용한 예측

➤ **U** 와 **V**를 이용하여 **R**을 예측함

➤  $R \approx UV^T$

	<i>i1</i>	<i>i2</i>	<i>i3</i>	<i>i4</i>	<i>i5</i>	<i>i6</i>	<i>i7</i>	<i>i8</i>
<i>Bob</i>	0.001	0.001	0.0	0.0	0.0	0.0	0.0	0.0
<i>u1</i>	0.001	0.001	0.001	0.0	0.0	0.0	0.0	0.0
<i>u2</i>	0.0	0.0	0.0	0.0	0.0	0.0	0.001	0.0
<i>u3</i>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
<i>u4</i>	0.0	0.0	0.0	0.0	0.001	0.001	0.002	0.001

**UV<sup>T</sup>**

<i>Bob</i>	<i>i1</i>	<i>i2</i>	<i>i3</i>	<i>i4</i>	<i>i5</i>	<i>i6</i>	<i>i7</i>	<i>i8</i>
<i>u1</i>	-0.008	0.021	0.029					
<i>u2</i>	-0.001	0.039	0.022					
<i>u3</i>	-0.008	0.029	-0.009					
<i>u4</i>	-0.01	0.016	-0.012					
	-0.035	0.012	-0.049					

**U**

**×**

	<i>i1</i>	<i>i2</i>	<i>i3</i>	<i>i4</i>	<i>i5</i>	<i>i6</i>	<i>i7</i>	<i>i8</i>
	-0.02	-0.005	-0.003	0.005	-0.013	-0.016	-0.023	-0.017
	0.022	0.028	0.028	0.018	0.005	-0.001	0.028	0.005
	0.026	0.021	0.006	-0.005	-0.027	-0.023	-0.018	-0.026

**V<sup>T</sup>**

# 단계3: 예측 값과 정답의 차이를 구함

➤ 실제 정답과 예측 값의 차이:  $(r_{ui} - U_u V_i^T)^2$

$U_u$ :  $u$  - th row of  $U$

$V_i$ :  $i$  - th column of  $V$

$$(r_{u2\ i6} - U_{u2} V_{i6}^T)^2 = (1 - 0.0)^2 = 1$$

$$(r_{u2\ i8} - U_{u2} V_{i8}^T)^2 = (0 - 0.0)^2 = 0$$

	i1	i2	i3	i4	i5	i6	i7	i8
Bob	0001	0001	00	00	00	00	00	00
u1	0001	0001	0001	00	00	00	00	00
u2	00	00	00	00	00	00	0001	00
u3	00	00	00	00	00	00	00	00
u4	00	00	00	00	0001	0001	0002	0001

$UV^T$

예측 값

	i1	i2	i3	i4	i5	i6	i7	i8
Bob	1	1	1					
u1	1	1	1	1				
u2		1	1	1	1	1		
u3			1	1	1	1	1	
u4					1	1	1	1

$R$

실제 정답

# 단계4: 전체 차이를 줄이도록 u와 v를 업데이트

➤ Gradient descent 방법을 이용하여 **u** 와 **v**를 업데이트 함


$$\operatorname{argmin}_{U,V} \sum_{u=1}^m \sum_{i=1}^n (r_{ui} - U_u V_i^T)^2$$

$U_u$ :  $u$  - th row of  $U$

$V_i$ :  $i$  - th column of  $V$

$$\left[ \begin{array}{c|cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & i7 & i8 \\ \hline Bob & 0.001 & 0.001 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ u1 & 0.001 & 0.001 & 0.001 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ u2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.001 & 0.0 \\ u3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ u4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.001 & 0.001 & 0.002 & 0.001 \end{array} \right] - \left[ \begin{array}{c|cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & i7 & i8 \\ \hline Bob & 1 & 1 & 1 & & & & & \\ u1 & 1 & 1 & 1 & 1 & & & & \\ u2 & & 1 & 1 & 1 & 1 & 1 & & \\ u3 & & & 1 & 1 & 1 & 1 & 1 & \\ u4 & & & & & 1 & 1 & 1 & 1 \end{array} \right]^2$$

$UV^T$                        $R$



**업데이트**

**U**

Bob	-0.008	0.021	0.029
u1	-0.001	0.039	0.022
u2	-0.008	0.029	-0.009
u3	-0.01	0.016	-0.012
u4	-0.035	0.012	-0.049

**V<sup>T</sup>**

i1	i2	i3	i4	i5	i6	i7	i8
-0.02	-0.005	-0.003	0.005	-0.013	-0.016	-0.023	-0.017
0.022	0.028	0.028	0.018	0.005	-0.001	0.028	0.005
0.026	0.021	0.006	-0.005	-0.027	-0.023	-0.018	-0.026

# 단계2~4를 반복하여 u와 v를 업데이트

➤ Gradient descent 방법을 이용하여 **u** 와 **v**를 업데이트 함


$$\operatorname{argmin}_{U,V} \sum_{u=1}^m \sum_{i=1}^n (r_{ui} - U_u V_i^T)^2$$

$U_u$ :  $u$  - th row of  $U$

$V_i$ :  $i$  - th column of  $V$

$$\left[ \begin{array}{c|cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & i7 & i8 \\ \hline Bob & 0.197 & 0.289 & 0.366 & 0.284 & 0.192 & 0.19 & 0.109 & 0.019 \\ u1 & 0.273 & 0.41 & 0.53 & 0.416 & 0.301 & 0.298 & 0.178 & 0.041 \\ u2 & 0.259 & 0.437 & 0.626 & 0.522 & 0.506 & 0.503 & 0.343 & 0.14 \\ u3 & 0.205 & 0.382 & 0.588 & 0.509 & 0.568 & 0.564 & 0.404 & 0.189 \\ u4 & 0.044 & 0.149 & 0.302 & 0.29 & 0.445 & 0.442 & 0.344 & 0.192 \end{array} \right] - \left[ \begin{array}{c|cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & i7 & i8 \\ \hline Bob & 1 & 1 & 1 & & & & & \\ u1 & 1 & 1 & 1 & 1 & & & & \\ u2 & & 1 & 1 & 1 & 1 & 1 & & \\ u3 & & & 1 & 1 & 1 & 1 & 1 & \\ u4 & & & & & 1 & 1 & 1 & 1 \end{array} \right]^2$$

$UV^T$   $R$



**업데이트**

**U**

Bob	-0.132	0.446	0.233
u1	-0.174	0.662	0.267
u2	-0.321	0.806	-0.044
u3	-0.388	0.766	-0.257
u4	-0.38	0.392	-0.472

**V<sup>T</sup>**

i1	i2	i3	i4	i5	i6	i7	i8
-0.082	-0.135	-0.228	-0.186	-0.308	-0.311	-0.248	-0.148
0.3	0.5	0.692	0.574	0.489	0.483	0.312	0.104
0.222	0.207	0.119	0.013	-0.287	-0.283	-0.269	-0.201

# 단계2~4를 반복하여 u와 v를 업데이트

➤ Gradient descent 방법을 이용하여 **u** 와 **v**를 업데이트 함

$$\operatorname{argmin}_{U,V} \sum_{u=1}^m \sum_{i=1}^n (r_{ui} - U_u V_i^T)^2$$

$U_u$ :  $u$  - th row of  $U$

$V_i$ :  $i$  - th column of  $V$

업데이트된 **u** 와 **v**

**U**

<b>Bob</b>	-0.559	0.432	0.82
<b>u1</b>	-0.196	0.814	0.694
<b>u2</b>	-0.033	1.102	-0.212
<b>u3</b>	-0.239	0.884	-0.653
<b>u4</b>	-1.085	0.046	-0.672

**V<sup>T</sup>**

	i1	i2	i3	i4	i5	i6	i7	i8
<b>Bob</b>	-0.487	-0.311	-0.209	0.162	-0.451	-0.451	-0.625	-0.727
<b>u1</b>	0.301	0.71	1.072	1.028	0.611	0.61	0.202	-0.159
<b>u2</b>	0.777	0.654	0.294	-0.128	-0.689	-0.688	-0.567	-0.207

**UV<sup>T</sup>**

	i1	i2	i3	i4	i5	i6	i7	i8
<b>Bob</b>	1.041	1.018	0.822	0.248	-0.04	-0.04	-0.02	0.167
<b>u1</b>	0.881	1.094	1.119	0.717	0.107	0.107	-0.1	-0.13
<b>u2</b>	0.183	0.654	1.126	1.155	0.835	0.835	0.365	-0.1
<b>u3</b>	-0.12	0.275	0.807	0.955	1.099	1.098	0.699	0.169
<b>u4</b>	0.02	-0.06	0.07	-0.04	0.981	0.982	1.069	0.921

**R**

	i1	i2	i3	i4	i5	i6	i7	i8
<b>Bob</b>	1	1	1					
<b>u1</b>	1	1	1	1				
<b>u2</b>		1	1	1	1	1		
<b>u3</b>			1	1	1	1	1	
<b>u4</b>					1	1	1	1

**UV<sup>T</sup>**

**R**

# 추천

- ▶ 예측된 값을 이용하여 Item4 ~ Item8 중 하나를 추천
  - ◆ Item4의 예측 값이 가장 높기 때문에 Bob에게 Item4를 추천

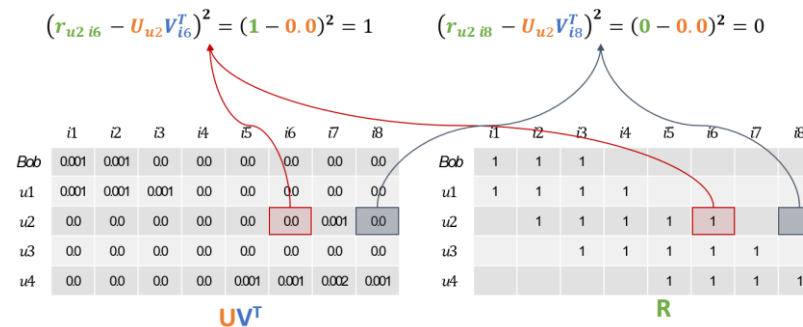


	Item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8
<b>Bob</b>	1	1	1	0.248	-0.04	-0.04	-0.02	0.167
User1	1	1	1	1				
User2		1	1	1	1	1		
User3			1	1	1	1	1	
User4					1	1	1	1

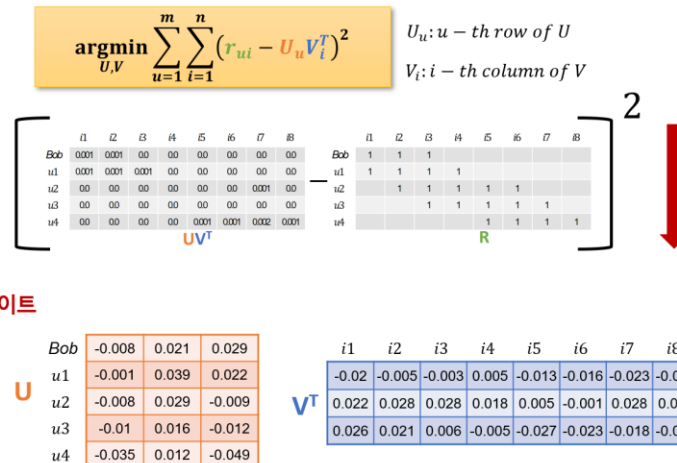


# 정리

## ➤ 단계3: 예측 값과 정답의 차이를 구함



## ➤ 단계4: 전체 차이를 줄이도록 u와 v를 업데이트



## ➤ 단계2~4를 반복하여 u와 v를 업데이트



# 라이브러리 및 데이터 불러오기

## ➤ 필요한 python 라이브러리 및 ML 100K 데이터를 불러옴

```
import numpy as np
import torch
from utils.load_data import data_loading
from utils.load_data import get_titles
from utils.evaluation import evaluation

# 데이터 불러오기
train_matrix, test_matrix, num_users, num_items = data_loading()
```

- ◆ Numpy: 행렬이나 배열을 쉽게 처리할 수 있도록 지원하는 라이브러리
- ◆ data\_loading: 직접 구현한 데이터 불러오기 함수, 데이터를 행렬 형태로 저장
- ◆ get\_titles: 직접 구현한 영화 목록 불러오기 함수
- ◆ evaluation: 직접 구현한 추천 평가 함수



	Item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8
Bob	1	1	1					
User1	1	1	1	1				
User2		1	1	1	1	1		
User3			1	1	1	1	1	
User4					1	1	1	1

# 하이퍼 파라미터 설정

## ➤ 학습에 필요한 latent 행렬 형태, 학습 비율, 학습 횟수 설정

# 하이퍼파라미터 설정

num\_epochs = 550

learning\_rate = 1e-2

num\_factors = 64



# Pytorch를 이용하여 MF 모델 생성

## ➤ Model 구성 및 U와 V Embedding 설정

```
class MatrixFactorization(torch.nn.Module):  
    def __init__(self, n_users, n_items, n_factors=20):  
        super().__init__() # 행의 차원 열의 차원  
        self.user_factors = torch.nn.Embedding(n_users, n_factors, sparse=False)  
        self.item_factors = torch.nn.Embedding(n_items, n_factors, sparse=False)  
        self.sigmoid = torch.nn.Sigmoid()  
  
    def forward(self):  
        return self.sigmoid(torch.matmul(self.user_factors.weight, self.item_factors.weight.T))
```

## ➤ 사용자 id와 아이템 id에 대한 Embedding 구성

- ◆ Embedding은 학습한 user와 item의 vector.

## ➤ U와 V는 랜덤하게 초기화



# MF를 이용한 예측

## ➤ 사용자와 항목의 latent 벡터를 이용하여 선호도 예측

```
def forward(self):  
    return self.sigmoid(torch.matmul(self.user_factors.weight, self.item_factors.weight.T))
```

- ◆ `self.user_factors.weight`
  - Embedding의 행렬에서 weight를 가져옴
- ◆ `torch.matmul(A, B)`
  - 행렬 A와 B를 곱함



# 예측 값과 정답의 차이 및 학습방법 설정

## ➤ GD를 이용하여 차이를 줄여나가도록 설정

```
model = MatrixFactorization(num_users, num_items, n_factors=num_factors).cuda()  
loss_func = torch.nn.MSELoss()  
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate) # learning rate
```

## ➤ 학습에 필요한 Model, Loss, Optimizer 선언.

## ➤ Loss – MSELoss()

## ➤ Optimizer - torch.optim.Adam()

$$\operatorname{argmin}_{U,V} \sum_{u=1}^m \sum_{i=1}^n (r_{ui} - U_u V_i^T)^2$$

# 모델 학습

## ➤ U, V 학습

```
ratings = torch.FloatTensor(train_matrix).cuda() # training matrix를 복사

# 사용자 시청 정보를 이용하여 u와 v를 업데이트 함.
for epoch in range(num_epochs):
    total_loss = 0
    optimizer.zero_grad()

    # 예측
    prediction = model.forward()
    loss = loss_func(prediction, ratings)

    # Backpropagate
    loss.backward()

    # Update the parameters
    optimizer.step()
```

## ➤ 위에서 설정한 'optimizer'를 실행하여 GD방법으로 학습

# MF를 이용한 예측 및 평가

## ➤ 학습된 모델을 이용하여 전체 항목을 예측하고 평가

```
predicted_matrix = np.zeros((train_matrix.shape))  
rows, cols = predicted_matrix.nonzero()
```

```
# MF를 이용하여 전체 항목의 값을 예측함.
```

```
with torch.no_grad():
```

```
    predicted_matrix = model.forward().cpu().numpy()
```

```
# 평가
```

```
hit50, hit100 = evaluation(predicted_matrix, train_matrix, test_matrix)
```

```
print("Hit@50: ", hit50)
```

```
print("Hit@100: ", hit100)
```

# 시청 항목과 추천 항목 출력

## ➤ 사용자의 실제 시청 항목과 예측한 Top 20 추천 항목을 출력

```
# 시청 항목과 추천 항목 출력
print("-----")
u_id = 201
titles = get_titles()

print("사용자 시청 영화")
rated_items = np.where(train_matrix[u_id] == 1)[0]
rated_items = rated_items[0:20]
for i in rated_items:
    print(titles[i])

print("-----")

print("추천 영화 Top 20")
predicted_matrix[u_id, rated_items] = 0
topN_items = np.argsort(predicted_matrix[u_id])
topN_items = topN_items[::-1]
topN_items = topN_items[0:20]
for i in topN_items:
    print(titles[i])
```



# 시청 항목과 추천 항목 출력

## ➤ 사용자의 실제 시청 항목과 예측한 Top 20 추천 항목을 출력

Hit@50: 0.34146341463414637  
Hit@100: 0.4782608695652174

---

사용자 시청 영화  
Toy Story (1995)  
Terminator 2: Judgment Day (1991)  
empire Strikes Back, The (1980)  
Princess Bride, The (1987)  
Amadeus (1984)  
Terminator, The (1984)  
Back to the Future (1985)  
Kolya (1996)  
Contact (1997)  
Full Monty, The (1997)  
emma (1996)  
english Patient, The (1996)  
Schindler's List (1993)  
e.T. the extra-Terrestrial (1982)  
Apartment, The (1960)  
Maltese Falcon, The (1941)  
Boot, Das (1981)  
Local Hero (1983)  
It Happened One Night (1934)

---

추천 영화 Top 20  
Much Ado About Nothing (1993)  
Star Wars (1977)  
Raiders of the Lost Ark (1981)  
Remains of the Day, The (1993)  
Titanic (1997)  
Fargo (1996)  
To Kill a Mockingbird (1962)  
Scream (1996)  
Sense and Sensibility (1995)  
Indiana Jones and the Last Crusade (1989)  
Liar Liar (1997)  
Dead Poets Society (1989)  
Silence of the Lambs, The (1991)  
Hamlet (1996)  
Dances with Wolves (1990)  
Braveheart (1995)  
Graduate, The (1967)  
Apollo 13 (1995)  
2001: A Space Odyssey (1968)  
Stand by Me (1986)

**1회 최고 시청률 누구 없소**

9:44

**JTBC Voyage**

조회수 1729만회 • 2개월 전

**[4K] 릴보이(III BOI)의 킬링벌스를 라이브로! | Good Time,...**

15:57

**dingo freestyle**

조회수 133만회 • 1주 전

**JTBC 봐야지**

0:01 / 9:43

#싱어게인 #누구없소 #진무명

🔥 핫클립 🔥 "난 노란 신호등 같은 존재." 특이한 음색으로 최고의 1분 기록 / 63호 가수가 부르는 '누구 없소' | 싱어게인

조회수 17,294,799회 • 2020. 11. 28.

👍 23만 🗣️ 3.9천 ➡ 공유 📌 저장 ...

**유희열이 절투한 Honey**

10:43

**JTBC Voyage**

조회수 1119만회 • 2개월 전

**🔥 핫클립 🔥 "와 정말 귀한 가수다.."**

8:26

**JTBC Voyage**

조회수 576만회 • 2개월 전

**복면가왕 스페셜 ★국카스텐 하현우 무대 모음집★ (음악대장)**

85:07

**MBC Kpop**

조회수 2367만회 • 1년 전

**서울예대 그 유명한 복도 나는 아무전이다**

1:10

**SBS Entertainment**

조회수 521만회 • 7개월 전

**양희은(Yang HeeEun) X 악동뮤지션(AKMU) - 엄마가 말에게...**

6:02

**SBS Entertainment**

조회수 2700만회 • 4년 전

**63호 전곡 1시간**

1:07:46

**JTBC Voyage**

조회수 9.2만회 • 1개월 전

**[다시보는\_고등레퍼2] 김하은**

1:07:46

**JTBC Voyage**

조회수 9.2만회 • 1개월 전

# Q&A



# Latent Dirichlet Allocation (LDA)

- **과제:** 3개의 Topic으로 LDA를 적용하여 문서 별 토픽 분포  $P(T|D)$ , 토픽 별 단어 분포  $P(W|T)$ 를 54, 55페이지와 같이 결과를 생성하여 제출하세요.