

# 콘텐츠, 임베딩, 및 연관 규칙 모델 실습

성균관대학교  
박성민, 이재웅

발표자료: [bit.ly/lab\\_lguplus\\_skku](https://bit.ly/lab_lguplus_skku)

코드: [bit.ly/lab2\\_lguplus](https://bit.ly/lab2_lguplus)



# 전체 실습 구성

- 실습 자료 다운로드 및 환경 설정
- 네이버 영화 데이터
- 콘텐츠 기반 추천 모델 실습
- 임베딩 기반 추천 모델 실습
- 연관 규칙 기반 추천 모델 실습



# 실습 자료 다운로드 및 환경 설정

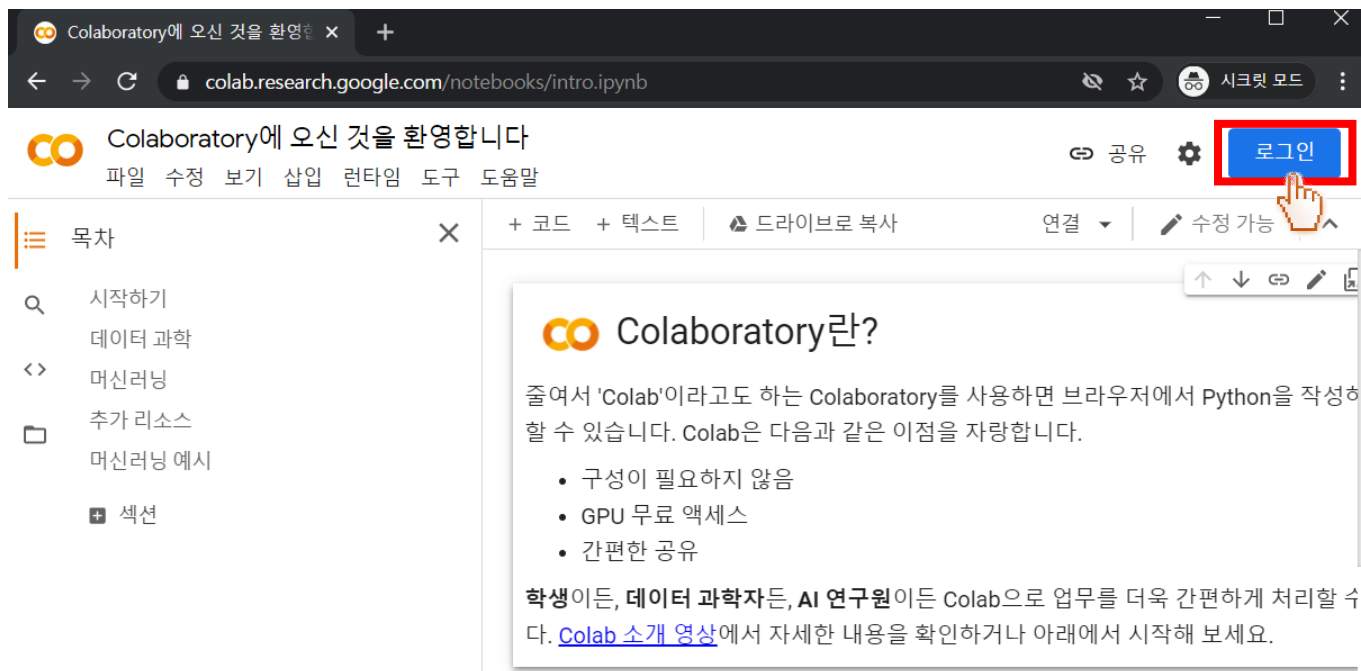
# 실습 자료 다운로드 및 기본 설정



## ➤ Chrome 브라우저를 이용해 Google Colaboratory 접속 후 본인 Google ID로 로그인

Google Colaboratory :

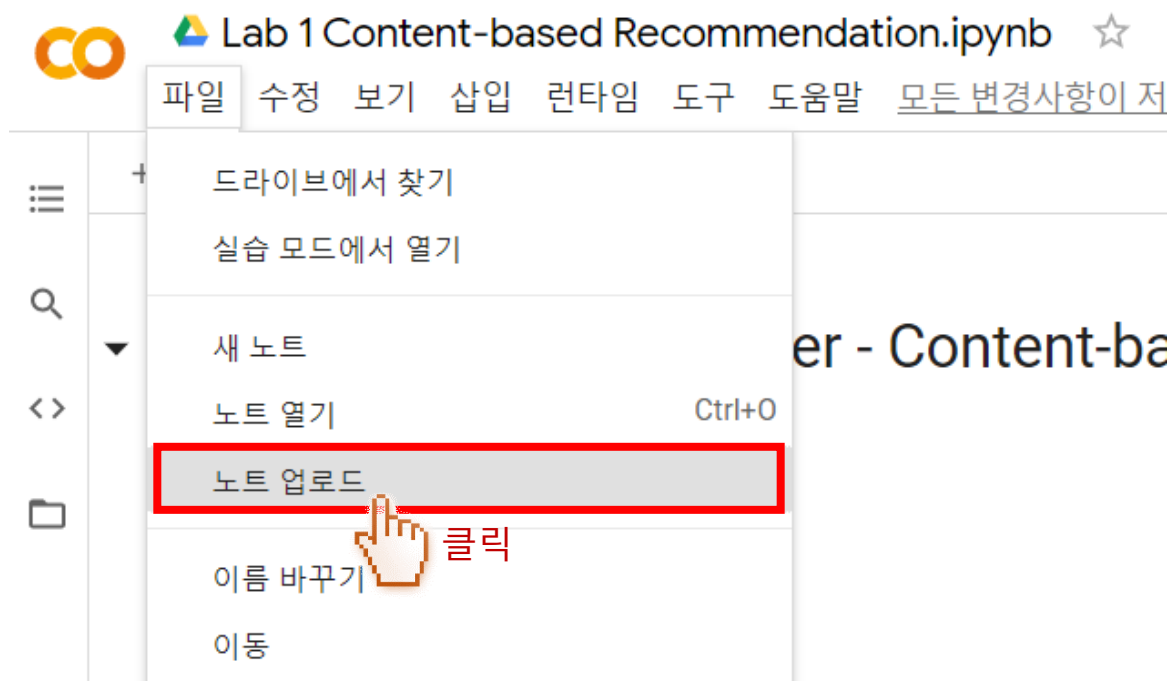
<https://colab.research.google.com>



# 실습 자료 다운로드 및 기본 설정



## ➤ 노트 업로드 클릭



# Google Colab 환경 설정



## ➤ 실행 환경 초기화

- ◆ 런타임 다시 시작

The screenshot shows the Google Colab interface for a notebook titled 'Lab 1 Content-based Recommendation.ipynb'. The 'Runtime' menu is open, displaying various execution options. The option 'Reset runtime' (런타임 초기화) is highlighted with a red rectangle. A hand cursor icon is pointing at this option, with the Korean word '클릭' (click) next to it. The background shows a sidebar with a file explorer and a table of contents for 'Lab 01. LGU+ ...', including sections like 'Text Mining' and 'Recommender System'.

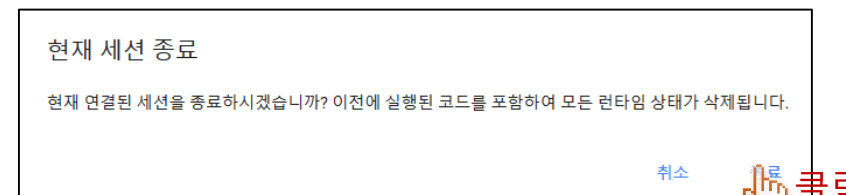
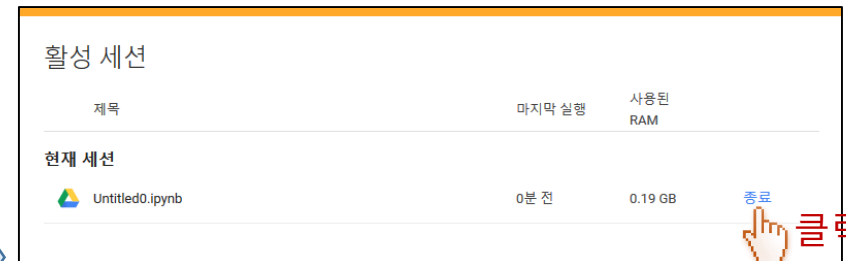
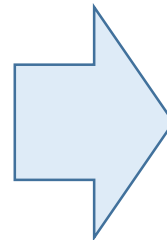
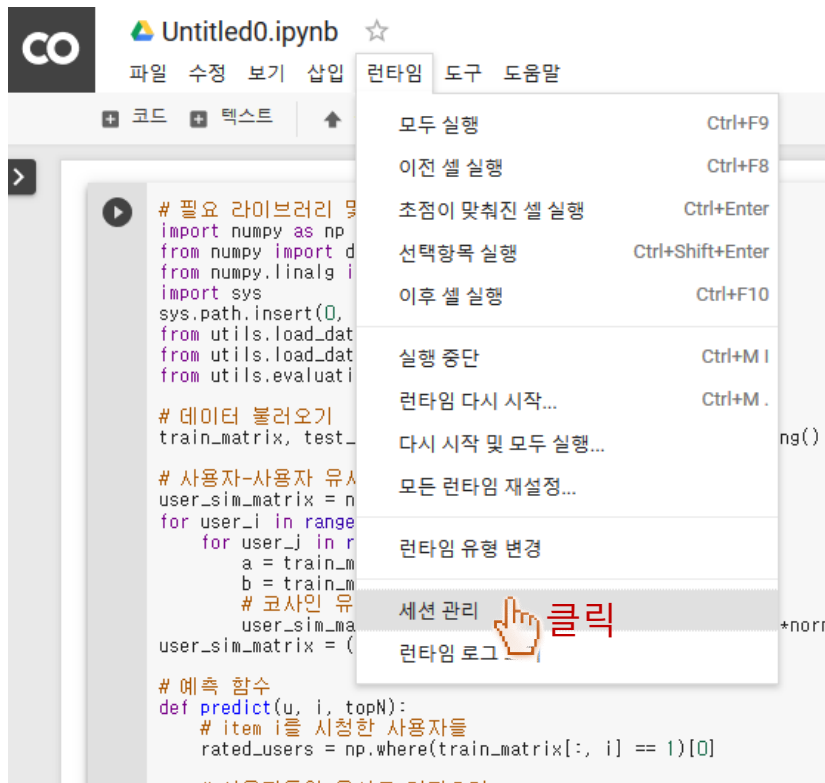
Option	Shortcut
모두 실행	Ctrl+F9
이전 셀 실행	Ctrl+F8
초점이 맞춰진 셀 실행	Ctrl+Enter
선택항목 실행	Ctrl+Shift+Enter
이후 셀 실행	Ctrl+F10
실행 중단	Ctrl+M I
런타임 다시 시작	Ctrl+M .
다시 시작 및 모두 실행	
<b>런타임 초기화</b>	
런타임 유형 변경	

# Google Colab 환경 설정



## ➤ 실행 환경 초기화

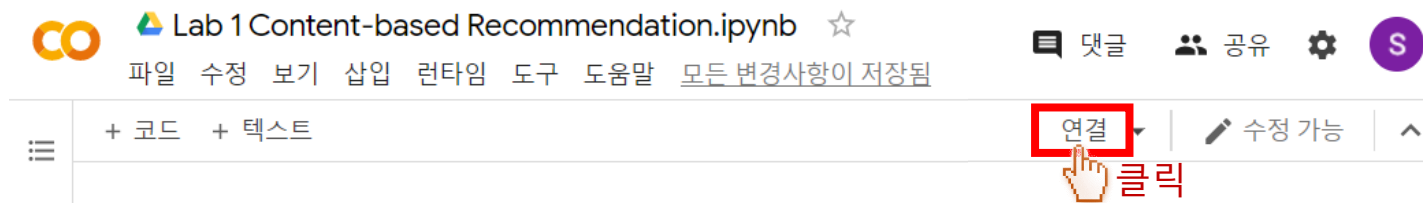
- ◆ 에러로 인하여 처음부터 다시 실행하고 싶을 때 사용



# Google Colab 환경 설정



## ➤ 실행 환경 연결/재연결







# 네이버 영화 데이터

# 네이버 영화 데이터



## ➤ MovieLens 형태의 네이버 영화에서 수집된 가상 데이터셋

- ◆ <https://github.com/lovit/kmrdr>

## ➤ 데이터 구성 정보

- ◆ KMRD-small, KMRD-2m, KMRD-5m  
(본 실습에서는 KMRD-small 데이터셋을 사용)
- ◆ 사용자ID, 영화ID, 평점, timestamp, 제목, 배우, 국가, 장르 포함

## ➤ KMRD-small 데이터셋

- ◆ # of users: 52,028, # of items: 600, # of ratings: 134,331
- ◆ Sparsity: 99.98%

# 네이버 영화 데이터



## ➤ 데이터셋 구성 형태

- ◆ ratings.csv: (사용자ID, 영화ID, 평점, 시간)
- ◆ movies.txt: (영화ID, 영화제목, 출시연도, 관람등급)
- ◆ castings.csv: (영화ID, 배우ID, 주연배우 여부)
  - people.txt: (배우ID, 한국 이름, 영어 이름)
- ◆ countries.csv: (영화ID, 국가)
- ◆ genres.csv: (영화ID, 장르)

## ➤ 모든 데이터셋을 결합(Join)하여 하나의 테이블로 구성

	user_id	item_id	rating	timestamp	title	people	country	genre
0	82	2	7	1494128040	백 투 더 퓨처 2	['마이클 J. 폭스', '크리스토퍼 로이드', '리 톰슨', '토머스 F. 윌슨...]	['미국']	['SF', '코미디']
1	82	3	7	1467529800	백 투 더 퓨처 3	['마이클 J. 폭스', '크리스토퍼 로이드', '메리 스티븐슨', '토머스 F. ...]	['미국']	['서부', 'SF', '판타지', '코미디']
2	82	16	9	1513344120	이티	['헨리 토마스', '디 윌리스', '피터 코요테', '로버트 맥노튼', '드류 베...]	['미국']	['판타지', 'SF', '모험', '가족']
3	82	19	9	1424497980	록키	['실베스터 스탤론', '탈리아 샤이어', '버트 영', '칼 웨더스', '버제스 ...]	['미국']	['드라마', '액션']
4	82	20	7	1427627340	록키 2	['실베스터 스탤론', '탈리아 샤이어', '버트 영', '칼 웨더스', '버제스 ...]	['미국']	['드라마', '액션']

# 데이터 전처리 과정



- 최소 10개 이상의 영화를 평가한 사용자만 남김
- 사용자가 여러 번 평가한 영화는 마지막 리뷰만 남김
- 전처리 후 데이터 통계

	Before pre-processing	After pre-processing
# of users	52,028	2,045
# of items	600	588
# of ratings	134,331	51,003
Sparsity	99.98%	95.76%

# 학습 및 테스트 데이터 구성



- 사용자 별 학습 및 테스트 데이터를 80%, 20%로 구성
- 테스트 항목은 사용자 별 무작위 항목을 20% 선택

항목

사용자

1	1	1			1		1		
	1	1		1		1		1	1
1	1	1	1	1	1	1	1	1	1
1		1		1		1	1		
		1			1		1	1	1

↓

1	1	?			1		1		
	?	1		1		1		1	1
1	1	?	1	1	?	1	1	1	1
1		1		1		1	1		
		1			1		1	1	?

# 정량 평가 방법



- 이번 실습에서는 평점을 implicit feedback으로 가정하여 상위 N개 리스트 추천 방식을 사용함
- 상위 N개 영화 리스트 추천
- 평가 지표: Precision, Recall, NDCG@N
  - ◆ 상위 100, 200개 영화 추천



# 콘텐츠 기반 추천 모델 실습

# 콘텐츠 기반 추천 모델 구성



➤ 데이터 표현 → 유사도 계산 → 추천 및 평가



One-hot  
encoding



유사도  
계산



추천 및  
평가

장르나 국가와 같은  
정형 데이터를  
one-hot 벡터로 구성

One-hot vector를  
이용해 영화 간의  
유사도 계산

시청한 영화를  
기준으로 관련 있는  
영화 Top-N 추천

영화 데이터



# 1단계: 데이터 표현



## ➤ 장르, 국가, 배우 등의 정보를 one-hot 벡터로 표현

```
def binary(feature_list, all_feature_list):
```

```
    binary_list = []
```

```
    for feature in all_feature_list:
```

```
        if feature in feature_list:
```

```
            binary_list.append(1)
```

```
        else:
```

```
            binary_list.append(0)
```

```
    return binary_list
```

```
train['genre_bin'] = train['genre'].apply(lambda x: binary(x, all_genre_list))
```

```
train[['user_id', 'item_id', 'title', 'genre', 'genre_bin']].head()
```

	user_id	item_id	title	genre_bin
13978	290	564	베어	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
35370	1008	435	천녀유혼	[0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
34053	952	513	영구와 땡칠이	[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...
5994	79	144	사랑과 영혼	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
35314	1005	31	인디애나 존스 - 최후의 성전	[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...

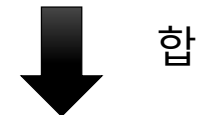
# 1단계: 데이터 표현



## ➤ One-hot 벡터를 하나로 합침

총 19개의 이진수로 구성  
(19개의 영화 장르)

user_id	item_id	title	genre	genre_bin
13978	290	베어	['드라마']	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
35370	1008	천녀유혼	['판타지', '멜로/로맨스', '액션', '공포']	[0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
34053	952	영구와 땡칠이	['가족', '모험', '코미디']	[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...]
5994	79	사랑과 영혼	['멜로/로맨스', '드라마']	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
35314	1005	인디애나 존스 - 최후의 성전	['판타지', '액션', '모험']	[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]



합

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



평균

[0.4, 0.4, 0.4, 0.4, 0.2, 0.2, 0.4, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, ...]

최종 결합된 one-hot 벡터

# 모든 사용자들의 one-hot 벡터 생성



- 사용자가 시청한 영화들의 one-hot 벡터를 평균 내어 사용자별 one-hot 벡터 생성

```
grouped_sum = train['genre_bin'].groupby(by=train['user_id']).sum()

user_bin = {}
for user_idx in range(num_users):
    total_bin = np.zeros(num_genres)
    num_movies = int(len(grouped_sum[user_idx])/num_genres)

    for i in range(num_movies):
        one_movie = np.array(grouped_sum[user_idx][i*num_genres:(i+1)*num_genres])
        zipped_lists = zip(total_bin, one_movie)
        total_bin = [x + y for (x, y) in zipped_lists]

total_bin = np.array(total_bin)
user_bin[user_idx] = (total_bin, num_movies)
```

# 예제: 특정 사용자의 벡터 표현



- 10번 사용자의 시청한 영화 수, one-hot 벡터, 정규화한 one-hot 벡터

```
user_id = 10
total_bin = user_bin[user_id][0]
num_movies = user_bin[user_id][1]
print(f"# of movies watched by user {user_id}: {num_movies}")
print("one-hot vector:", total_bin)
print("normalized one-hot vector:", total_bin / num_movies)
```

## 출력

```
# of movies watched by user 10: 45
one-hot vector: [28. 4. 11. 13. 3. 1. 5. 10. 0. 8. 8. 6. 1. 4. 4. 1. 1. 1. 0.]
normalized one-hot vector: [0.62222222 0.08888889 0.24444444 0.28888889
0.06666667 0.02222222 0.11111111 0.22222222 0. 0.17777778 0.17777778
0.13333333 0.02222222 0.08888889 0.08888889 0.02222222 0.02222222 0.22222222 0.]
```

## 2단계: 유사도 계산



### ➤ 특정 사용자와 모든 영화에 대한 코사인 유사도 계산

```
norm_bin = total_bin / num_movies
# unique item 추리기
Unique_items = train[['item_id', 'title', 'genre', 'genre_bin']].drop_duplicates(['item_id'])
# 특정 user가 본 영화들 제외
train_items_by_user = train.loc[train.user_id==user_id]
unique_items = unique_items[~unique_items['item_id'].isin(train_items_by_user['item_id'])]
unique_items['similarity'] = unique_items['genre_bin'].apply(lambda x: np.array(x).dot(norm_bin) /
(np.array(x).sum() + 1e-10))
unique_items.head()
# cosine similarity를 토대로 top-k item 구하기
sorted_items = unique_items.sort_values(by=['similarity'], axis=0, ascending=False)
sorted_items.head()
```

	item_id	title	genre	genre_bin	country	similarity
13978	564	베어	['드라마']	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	['프랑스']	0.622222
9990	74	챔프	['드라마']	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	['미국']	0.622222
22262	522	칠수와 만수	['드라마']	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	['한국']	0.622222
20830	515	밤 그리고 도시	['드라마']	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	['미국']	0.622222
1532	530	칼리굴라	['드라마']	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	['미국', '이탈리아']	0.622222

# 3단계: 상위 N개 영화 리스트 추천



## ➤ 구한 코사인 유사도를 이용해 유사도 순 top-10 영화 추천

```
top_k = 10
top_k_items = list(sorted_items['item_id'][:top_k])
top_item_df = sorted_items[['title', 'genre', 'item_id']].drop_duplicates(['item_id'])
# 예측한 top-k items
top_item_df[top_item_df['item_id'].isin(top_k_items[:top_k])]
```

	item_id	title	genre
13978	564	베어	['드라마']
9990	74	챔프	['드라마']
22262	522	칠수와 만수	['드라마']
20830	515	밤 그리고 도시	['드라마']
1532	530	칼리굴라	['드라마']
55874	465	집시의 시간	['드라마']
36287	439	제7의 봉인	['드라마']
25068	545	깊은 밤 깊은 곳에	['드라마']
27404	48	라 밤바	['드라마']
24369	500	김의 전쟁	['드라마']

드라마 영화를 제일 많이 봤기 때문에  
드라마 영화 위주로 추천

# 예제: 특정 사용자가 본 영화 확인



## ➤ 10번 사용자가 실제로 본 영화 리스트 출력

```
user_id = 10  
test_items_by_user = test.loc[test.user_id==user_id]  
test_items_by_user[['user_id', 'item_id', 'title', 'genre']]
```

	user_id	item_id	title	genre
486	10	513	영구와 땡칠이	['가족', '모험', '코미디']
464	10	192	늑대와 춤을	['드라마', '서부', '모험']
419	10	66	티파니에서 아침을	['멜로/로맨스', '드라마']
390	10	10	다이 하드	['액션', '스릴러', '범죄']
392	10	14	나 홀로 집에	['코미디', '가족', '모험', '범죄']
477	10	373	아비정전	['멜로/로맨스', '드라마', '범죄']
385	10	1	뽀크 투 더 퓨처	['SF', '코미디']
431	10	87	카사블랑카	['멜로/로맨스', '드라마']
470	10	245	스카페이스	['드라마', '액션', '범죄']
402	10	35	플래툰	['드라마', '전쟁']
399	10	20	툽키 2	['드라마', '액션']
403	10	36	구니스	['코미디', '스릴러', '모험', '가족']

드라마 영화가 12개 중 7개로, 해당 user가 드라마 영화를 실제로 많이 봤음을 알 수 있음.

# 4단계: 성능 평가 확인



## ➤ 평가 지표 Precision, recall, NDCG 계산

```
def compute_metrics(pred_u, target_u, top_k):
    pred_k = pred_u[:top_k]
    num_target_items = len(target_u)
    hits_k = [(i + 1, item) for i, item in enumerate(pred_k) if item in target_u]
    print("실제로 맞춘 items (position, idx):", hits_k)
    num_hits = len(hits_k)
    idcg_k = 0.0
    for i in range(1, min(num_target_items, top_k) + 1):
        idcg_k += 1 / math.log(i + 1, 2)
    dcg_k = 0.0
    for idx, item in hits_k:
        dcg_k += 1 / math.log(idx + 1, 2)
    prec_k = num_hits / top_k
    recall_k = num_hits / min(num_target_items, top_k)
    ndcg_k = dcg_k / idcg_k

    return prec_k, recall_k, ndcg_k
```



# 4단계: 성능 평가 확인



## ➤ 평가 지표 Precision, recall, NDCG 계산

- ◆ 사용자 한 명에 대한 평가 진행

```
top_k = 200
pred_u = list(sorted_items['item_id'])
target_u = list(test_items_by_user['item_id'])

prec, recall, ndcg = compute_metrics(pred_u, target_u, top_k)
print(f"Precision@{top_k}: {prec:.4f}")
print(f"Recall@{top_k}: {recall:.4f}")
print(f"NDCG@{top_k}: {ndcg:.4f}")
```

### 출력

```
실제로 맞춘 items (position, idx): [(106, 20), (127, 66), (130, 87), (199, 35)]
Precision@200: 0.0200
Recall@200: 0.3333
NDCG@200: 0.1108
```

# 추가 실습



- 장르 뿐만 아니라 영화의 국가 및 배우를 활용해 만든 one-hot 벡터를 가지고 평가할 경우 성능 개선 가능!

```
# genre, country, people feature
train['genre_bin'] = train['genre'].apply(lambda x: binary(x, all_genre_list))
train['country_bin'] = train['country'].apply(lambda x: binary(x, all_country_list))
train['people_bin'] = train['people'].apply(lambda x: binary(x, all_people_list))

train[['user_id', 'item_id', 'title', 'genre_bin', 'country_bin', 'people_bin']].head()

# 예시로 genre_bin와 people 두 개를 합치는 방식 사용.
train['genre_people'] = train['genre_bin'] + train['people_bin']
```

# 예제: 다른 메타데이터를 이용한 평가

- 장르, 국가, 배우 정보를 추가적으로 활용하여, 전체 사용자에게 대한 성능 평가 결과

	Prec@200	Recall@200	NDCG@200
장르	0.0098	0.3929	0.1062
장르 + 국가	0.0120	0.4755	0.1285
장르 + 국가 + 배우	0.0128	0.5074	0.1560



# 임베딩 기반 추천 모델 실습

# Prod2Vec 아키텍처 – 영화 데이터



➤ 데이터 전처리 → Prod2Vec → 추천 및 평가



영화 데이터



데이터  
전처리

Train, test 분리  
Rating을 implicit  
feedback으로 여김.



Prod2Vec

각 사용자들의  
시청한 영화 정보를  
활용해 영화 임베딩



추천 및  
평가

시청한 영화를  
기준으로 관련 있는  
**영화 Top-N 추천**  
- 코사인 유사도

# 실습 데이터 : 네이버 영화 데이터셋



- 네이버 영화 리뷰를 크롤링 (crawling)하여 수집
- 사용자 2,000명, 영화 588개, 평점 50,461개
- 메타데이터: 장르, 국가, 배우
  - ◆ 하나의 영화가 여러 개의 장르에 속해 있을 수 있음

user_id	item_id	rating	timestamp	title	people	country	genre
82	2	7	1494128040	백 투 더 퓨처 2	['마이클 J. 폭스', '크리스토퍼 로이드', '리 톨슨', '토머스 F. 윌슨'...	['미국']	['SF', '코미디']
82	3	7	1467529800	백 투 더 퓨처 3	['마이클 J. 폭스', '크리스토퍼 로이드', '메리 스티븐슨', '토머스 F. ...	['미국']	['서부', 'SF', '판타지', '코미디']
82	16	9	1513344120	이티	['헨리 토마스', '디 윌리스', '피터 코요테', '로버트 맥노튼', '드류 베...	['미국']	['판타지', 'SF', '모험', '가족']
82	19	9	1424497980	록키	['실베스터 스탤론', '탈리아 샤이어', '버트 영', '칼 웨더스', '버제스 ...	['미국']	['드라마', '액션']
82	20	7	1427627340	록키 2	['실베스터 스탤론', '탈리아 샤이어', '버트 영', '칼 웨더스', '버제스 ...	['미국']	['드라마', '액션']
...	...	...	...	...	...	...	...

# 1단계: 데이터 전처리



## ➤ 사용자 별 학습 및 테스트 데이터를 80%, 20%로 구성

```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(movie_data, test_size=0.2, \
                                     stratify = movie_data['user_id'], random_state = 1234)

train_users = train_df['user_id'].unique()
train_users = sorted(train_users) # sorting
train = []
for user_id in train_users:
    itemset = train_df[train_df['user_id'] == user_id]['item_id'].tolist()
    train.append(itemset)

test_users = test_df['user_id'].unique()
test_users = sorted(test_users) # sorting
test = []
for user_id in test_users:
    itemset = test_df[test_df['user_id'] == user_id]['item_id'].tolist()
    test.append(itemset)
```

## 2단계: Prod2Vec 학습



- ‘인디애나 존스’라는 영화를 봤을 때 같이 본 영화들이 시청될 확률을 최대화 하도록 학습

사용자가 본 영화들

'65'	'19'	'350'	'188'	'2'	'358'	'82'	'16'	'380'	'483'
'236'	'64'	'83'	'70'	'188'	'26'	'347'	'164'		
'573'	'350'	'35'	'6'	'383'	'81'	'93'	'4'	'94'	'272'
'192'	'382'	'191'	'272'	'58'	'13'	'105'	'564'	'473'	'412'

⋮

	'358'	
--	-------	--



Prod2Vec



'65'	'19'	'350'	'188'	'2'		'82'	'16'	'380'	'483'
------	------	-------	-------	-----	--	------	------	-------	-------



# 2단계: Prod2Vec 학습



## ➤ Gensim 라이브러리



- ◆ NLP 관련 모델, 샘플 데이터셋 등을 제공

```
from gensim.models import Word2Vec
model = Word2Vec(window=260, vector_size=100, sg=1, seed=2021)
# vocabulary 구성
model.build_vocab(train)
# word2vec 학습
model.train(train, total_examples=model.corpus_count, epochs=20)
```

- ◆ window: window 크기 (사용자당 최대 리뷰 수: 257)
- ◆ vector\_size: 은닉층 차원
- ◆ sg: CBoW → sg=0, Skip-gram → sg=1
- ◆ alpha: 학습률
- ◆ seed: 랜덤 시드

# 3단계: 추천 결과 확인



## ➤ 특정 영화와 유사도가 높은 영화 상위 10개 출력

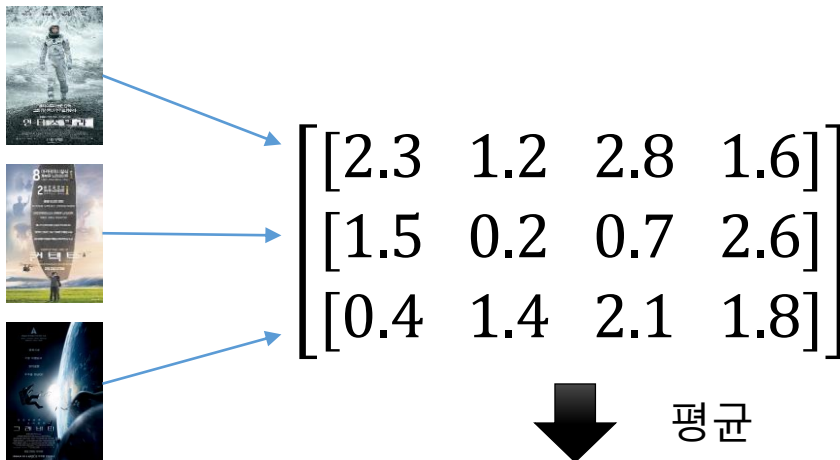
```
movie_title = '이티'  
sim_movies = model.wv.most_similar(title2idx[movie_title], topn=10)  
print('=====')  
print('영화, similarity')  
print('=====')  
for i, (idx, _) in enumerate(sim_movies):  
    print(f'{idx2title[idx]}, {sim_movies[i][1]:.4f}')
```

```
=====
영화, similarity
=====
사운드 오브 뮤직, 0.7146
나 홀로 집에, 0.6906
백 투 더 퓨처, 0.6839
에이리언 2, 0.6808
양들의 침묵, 0.6803
배트맨, 0.6758
영웅본색, 0.6673
죽은 시인의 사회, 0.6658
에이리언, 0.6611
사랑과 영혼, 0.6566
```

# 3단계: 추천 결과 확인



- 특정 사용자가 시청한 영화들의 임베딩 벡터의 평균과 다른 영화의 유사도를 바탕으로 상위 N개 리스트 추천



**[1.4 0.8 1.4 1.5]**

↓ 시청하지 않은 영화들과 유사도 계산



# 예제: 특정 사용자의 임베딩 계산



## ➤ 특정 사용자가 시청한 영화 목록의 임베딩 값 평균 계산

```
def aggregate_vectors(movie_list):
    product_vec = []
    for i in movie_list:
        try:
            product_vec.append(model[i])
        except KeyError:
            continue

    return np.mean(product_vec, axis=0)

user_idx = 10
movie_list = train[user_idx]
avg_emb = aggregate_vectors(movie_list)
```

100차원 벡터

```
array([ 0.06656607, -0.06129238,  0.11736292, -0.03738244, -0.10178891,
        -0.2151281 ,  0.04492006, -0.2113235 , -0.00861568,  0.04596372,
         0.05349389, -0.09840404,  0.17692697, -0.17889535,  0.0516682 ,
```

⋮

# 예제: 상위 N개 영화 리스트 추천



## ➤ 평균 임베딩과의 유사도 기준으로 상위 N개 리스트 추천

```
sim_movies_all = model.similar_by_vector(avg_emb, topn=5)
sim_movies = []
for i, _ in enumerate(sim_movies_all):
    if sim_movies_all[i][0] not in movie_list:
        sim_movies.append((sim_movies_all[i][0], sim_movies_all[i][1]))
print('=====')
print('영화, idx, similarity')
print('=====')
pred_u = []
for i, (idx, _) in enumerate(sim_movies):
    pred_u.append(sim_movies[i][0])
print(f'{idx2title[idx]}, {sim_movies[i][0]}, {sim_movies[i][1]:.4f}')
```

```
=====
영화, idx, similarity
=====
나 홀로 집에, 14, 0.7876
이티, 16, 0.7775
백 투 더 퓨처, 1, 0.7763
레인 맨, 188, 0.7753
플래툰, 35, 0.7705
```

# 4단계: 성능 평가 확인



## ➤ 평가 지표 Precision, recall, NDCG 계산

```
def compute_metrics(pred_u, target_u, top_k):
    pred_u = sorted(pred_u)
    target_u = sorted(target_u)
    num_target_items = len(target_u)
    pred_k = pred_u[:top_k]
    hits_k = [(i + 1, item) for i, item in enumerate(pred_k) if item in target_u]
    print("실제로 맞춘 items (position, idx):", hits_k)
    num_hits = len(hits_k)
    idcg_k = 0.0
    for i in range(1, min(num_target_items, top_k) + 1):
        idcg_k += 1 / math.log(i + 1, 2)
    dcg_k = 0.0
    for idx, item in hits_k:
        dcg_k += 1 / math.log(idx + 1, 2)
    prec_k = num_hits / top_k
    recall_k = num_hits / min(num_target_items, top_k)
    ndcg_k = dcg_k / idcg_k

    return prec_k, recall_k, ndcg_k
```

# 4단계: 성능 평가 확인



## ➤ 평가 지표 Precision, recall, NDCG 계산

- ◆ 사용자 한 명에 대한 평가 진행

```
top_k = 200
print(f'모델이 예측한 movies: {pred_u}')
print(f'실제로 본 movies: {target_u}')

prec, recall, ndcg = compute_metrics(pred_u, target_u, top_k)
print(f"Precison@{top_k}: {prec:.3f}")
print(f"Recall@{top_k}: {recall:.3f}")
print(f"NDCG@{top_k}: {ndcg:.3f}")
```

### 출력

```
모델이 예측한 movies: ['188', '16', '1', '70', '192', '33', '14', '380', '42', '34']
실제로 본 movies: ['513', '192', '66', '10', '14', '373', '1', '87', '245', '35', '20', '36']
Precison@200: 0.0150
Recall@200: 0.2500
NDCG@200: 0.2396
```

# 컨텐츠, 임베딩 기반 모델 성능 비교



## ➤ 전체 사용자에게 대한 성능 평가 결과

	Prec@200	Recall@200	NDCG@200
장르	0.0098	0.3929	0.1062
장르 + 국가	0.0120	0.4755	0.1285
장르 + 국가 + 배우	0.0128	0.5074	0.1560
Prod2Vec	0.0222	0.8996	0.3614



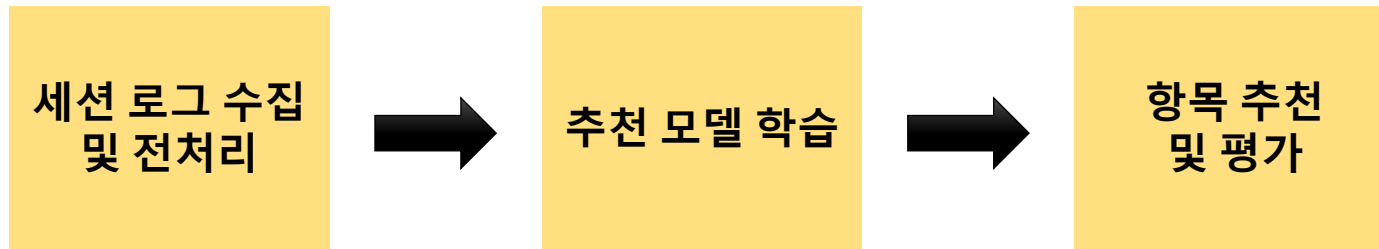


# 연관 규칙 기반 추천 모델 실습

# 연관 규칙 기반 추천 모델 구성



- 세션 로그 수집 및 전처리 → 추천 모델 학습 → 항목 추천 및 평가



➤ 세션 기반 추천의 특징

- ◆ 사용자에게 대한 정보가 없음
- ◆ 시간적 순서가 있는 순차 (sequence) 데이터
- ◆ 세션 당 사용자의 피드백 수가 적음

# 1단계: 데이터 전처리



## ➤ 데이터 형태

- ◆ [세션 ID, 항목 ID] 형태
- ◆ 시간 순서로 정렬되어 있음

## ➤ 학습 / 평가 데이터 나누기

- ◆ 각 사용자 별 평가 (rating)를 시간 순서로 정렬한 뒤, 상위 80%를 학습 데이터로, 하위 20%를 평가 데이터로 나눔

## ➤ 실습을 위해, 한 명의 사용자를 하나의 세션으로 취급

전체 데이터

사용자 ID (세션 ID)	항목 ID
3	1
3	2
3	3
3	5
3	7
3	10
3	12
3	13
3	15
3	20



학습 데이터

사용자 ID (세션 ID)	항목 ID
3	1
3	2
3	3
3	5
3	7
3	10
3	12
3	13

평가 데이터

사용자 ID (세션 ID)	항목 ID
3	15
3	20

## 2-1단계: 연관 규칙 (AR)



- 시간적 순서를 고려하지 않음
- 한 세션에서 항목  $i$ 와  $j$ 가 함께 등장하는 경우를 고려

$$\text{score}_{AR}(i, j) = \frac{1}{\sum_{p \in S_p} \sum_{x=1}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot (|p| - 1)} \sum_{p \in S_p} \sum_{x=1}^{|p|} \sum_{y=1}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot \mathbf{1}_{EQ}(j, p_y)$$

Normalization

Counting scheme

세션  $p$ 에서 항목  $i$ 와 항목  $j$ 가 동시에 등장하는 횟수

- ◆  $S_p$ : 현재 세션을 제외한 모든 세션
- ◆  $\mathbf{1}_{EQ}(a, b)$ :  $a$ 와  $b$ 가 같은 항목이면 1, 아니면 0

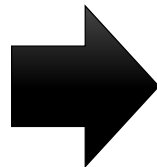
# 연관 규칙의 Counting Scheme 구현



- 항목  $i, j$  의 모든 조합에 대하여, 하나의 세션에서  $i, j$  가 등장하는 횟수로 규칙을 구함
- Counting scheme 식

$$\sum_{p \in S_p} \sum_{x=1}^{|p|} \sum_{y=1}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot \mathbf{1}_{EQ}(j, p_y)$$

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



$i$	$j$	등장 횟수
1	2	2
1	3	1
2	1	2
2	3	1
3	1	1
3	2	1

# 연관 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1, 2]
session_id = 1
item_id = 3
```

```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for item_id2 in last_items:
            rules[item_id][item_id2] += 1
            rules[item_id2][item_id] += 1

last_items.append(item_id)
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	
2	1	1
2	3	
3	1	
3	2	

# 연관 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1, 2]
session_id = 1
item_id = 3
item_id2 = 1
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	1
2	1	1
2	3	
3	1	1
3	2	

```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for item_id2 in last_items:
            rules[item_id][item_id2] += 1
            rules[item_id2][item_id] += 1
        last_items.append(item_id)
```

# 연관 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1, 2]
session_id = 1
item_id = 3
item_id2 = 1
```

```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for item_id2 in last_items:
            rules[item_id][item_id2] += 1
            rules[item_id2][item_id] += 1

        last_items.append(item_id)
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	1
2	1	1
2	3	
3	1	1
3	2	



# 연관 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1, 2]
session_id = 1
item_id = 3
item_id2 = 2
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	1
2	1	1
2	3	1
3	1	1
3	2	1

```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for item_id2 in last_items:
            rules[item_id][item_id2] += 1
            rules[item_id2][item_id] += 1
        last_items.append(item_id)
```

# 연관 규칙의 Normalization 구현



- 항목  $i$ 가 등장하는 각 '세션의 길이-1'의 값으로 정규화
- Counting scheme 식

$$\frac{1}{\sum_{p \in S_p} \sum_{x=1}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot (|p| - 1)}$$

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



$i$	$ p  - 1$ 의 합
1	2+1
2	2+1
3	2



$i$	Normalization 값
1	$\frac{1}{3}$
2	$\frac{1}{3}$
3	$\frac{1}{2}$


# 연관 규칙의 Normalization 코드



```
cur_session = -1
normal = dict()
items = []
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        if len(items) != 0:
            session_len = len(items) - 1 # |p|-1
            for item_in_session in items:
                if not item_in_session in normal:
                    normal[item_in_session] = 0
            normal[item_in_session] += session_len

        cur_session = session_id
        items = []
    else:
        items.append(item_id)

session_len = len(items) - 1 # |p|-1
for item_in_session in items:
    if not item_in_session in normal:
        normal[item_in_session] = 0
    normal[item_in_session] += session_len
```



세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1

```
Items=[1,2,3]
session_len = 2
item_in_session=1
```

**normal**

<i>i</i>	
1	2
2	
3	

# 연관 규칙의 Normalization 코드



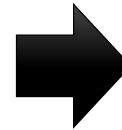
$$\frac{1}{\sum_{p \in S_p} \sum_{x=1}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot (|p| - 1)} \sum_{p \in S_p} \sum_{x=1}^{|p|} \sum_{y=1}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot \mathbf{1}_{EQ}(j, p_y)$$

rules

<i>i</i>	<i>j</i>	
1	2	2
1	3	1
2	1	2
2	3	1
3	1	1
3	2	1

normal

<i>i</i>	
1	3
2	3
3	2



rules

<i>i</i>	<i>j</i>	
1	2	2/3
1	3	1/3
2	1	2/3
2	3	1/3
3	1	1/2
3	2	1/2

```
for item_id1 in rules.keys():
    normal_term = normal[item_id1]

    for item_id2 in rules[item_id1].keys():
        rules[item_id1][item_id2] = rules[item_id1][item_id2] / normal_term
```



## 2-2단계: 순차 규칙 (SR)

- 시간적 순서 및 간격을 고려함
- 한 세션에서 항목  $i$  등장 후  $j$ 가 등장하는 경우를 고려

$score_{SR}(i, j)$

$$= \frac{1}{\sum_{p \in S_p} \sum_{x=2}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot x} \sum_{p \in S_p} \sum_{x=2}^{|p|} \sum_{y=1}^{x-1} \mathbf{1}_{EQ}(i, p_y) \cdot \mathbf{1}_{EQ}(j, p_x) \cdot w_{SR}(x - y)$$

Normalization

Counting scheme

Weighting scheme

세션  $p$ 에서 항목  $i$  이후 항목  $j$ 가 등장하는 횟수

$w_{SR}(x) = 1/x$ , where  $x$  corresponds to the number of steps between the two items

- ◆  $S_p$ : 현재 세션을 제외한 모든 세션
- ◆  $\mathbf{1}_{EQ}(a, b)$ :  $a$ 와  $b$ 가 같은 항목이면 1, 아니면 0

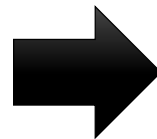
# 순차 규칙의 Counting Scheme 구현



- 항목  $i, j$  의 모든 조합에 대하여, 하나의 세션에서  $i$  이후  $j$  가 등장하는 경우  $i, j$  의 거리의 역을 가중하여 규칙을 구함
- Counting scheme 식

$$\sum_{p \in S_p} \sum_{x=2}^{|p|} \sum_{y=1}^{x-1} \mathbf{1}_{EQ}(i, p_y) \cdot \mathbf{1}_{EQ}(j, p_x) \cdot w_{SR}(x - y)$$

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



$i$	$j$	
1	2	1/(2-1)
1	3	1/(3-1)
2	1	1/(2-1)
2	3	1/(3-2)
3	1	0
3	2	0

# 순차 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1,2]
session_id = 1
item_id = 3
```

```
range(1, len(last_items) + 1) = [1,2]
i = 1
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for i in range(1, len(last_items) + 1):
            prev_item = last_items[-i]

            weight = 1/i
            rules[prev_item][item_id] += weight

last_items.append(item_id)
```

rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	
2	1	
2	3	
3	1	
3	2	

# 순차 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1,2]
session_id = 1
item_id = 3
```

```
range(1, len(last_items) + 1) = [1,2]
i = 1
prev_item = 2
weight = 1
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for i in range(1, len(last_items) + 1):
            prev_item = last_items[-i]

            weight = 1/i
            ➡ rules[prev_item][item_id] += weight

        last_items.append(item_id)
```

rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	
2	1	
2	3	1
3	1	
3	2	



# 순차 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1,2]
session_id = 1
item_id = 3
```

```
range(1, len(last_items) + 1) = [1,2]
i = 2
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for i in range(1, len(last_items) + 1):
            prev_item = last_items[-i]

            weight = 1/i
            rules[prev_item][item_id] += weight

last_items.append(item_id)
```

rules

<i>i</i>	<i>j</i>	
1	2	1
1	3	
2	1	
2	3	1
3	1	
3	2	

# 순차 규칙의 Counting Scheme 코드



```
cur_session = 1
last_items = [1,2]
session_id = 1
item_id = 3
```

```
range(1, len(last_items) + 1) = [1,2]
i = 2
prev_item = 1
weight = 1/2
```

session\_item

세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1



```
cur_session = -1
last_items = []
rules = dict()
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        last_items = []
    else:
        for i in range(1, len(last_items) + 1):
            prev_item = last_items[-i]

            weight = 1/i
            ➡ rules[prev_item][item_id] += weight

        last_items.append(item_id)
```

rules

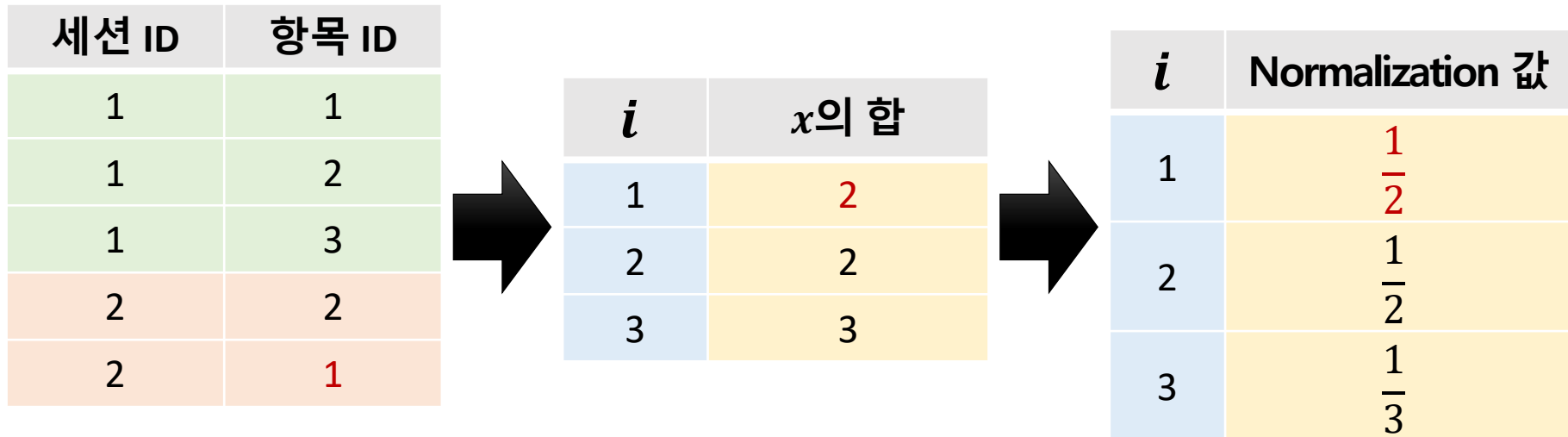
<i>i</i>	<i>j</i>	
1	2	1
1	3	1/2
2	1	
2	3	1
3	1	
3	2	

# 순차 규칙의 Normalization 구현



- 항목  $i$ 가 등장하는 각 세션에서 항목  $i$ 의 순서로 정규화
- Counting scheme 식

$$\frac{1}{\sum_{p \in S_p} \sum_{x=2}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot x}$$



# 순차 규칙의 Normalization 코드



```
normal = dict()
cur_session = -1
x = 1
for row in session_item:
    session_id, item_id = row[0], row[1]
    if session_id != cur_session:
        cur_session = session_id
        x = 1
    else:
        if not item_id in normal:
            normal[item_id] = 0
        normal[item_id] += x
    x += 1
```



세션 ID	항목 ID
1	1
1	2
1	3
2	2
2	1

item\_id = 2  
X = 2

**normal**

$s_{ s }$	
1	
2	2
3	

# 순차 규칙의 Normalization 코드



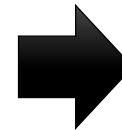
$$\frac{1}{\sum_{p \in S_p} \sum_{x=2}^{|p|} \mathbf{1}_{EQ}(i, p_x) \cdot x} \sum_{p \in S_p} \sum_{x=2}^{|p|} \sum_{y=1}^{x-1} \mathbf{1}_{EQ}(i, p_y) \cdot \mathbf{1}_{EQ}(j, p_x) \cdot w_{SR}(x - y)$$

rules

$s_{ s }$	$i$	
1	2	1
1	3	1/2
2	1	1
2	3	1
3	1	0
3	2	0

normal

$s_{ s }$	
1	2
2	2
3	3



rules

$s_{ s }$	$i$	
1	2	1/2
1	3	1/4
2	1	1/2
2	3	1/2
3	1	0/3
3	2	0/3

```
for item_id1 in rules.keys():
    normal_term = normal[item_id1]

    for item_id2 in rules[item_id1].keys():
        rules[item_id1][item_id2] = rules[item_id1][item_id2] / normal_term
```



# 3단계: 추천 결과 확인

- 항목  $i, j$  의 모든 조합에 대한 규칙을 이용
- 세션의 마지막 항목을 이용하여 추천 함

rules

$s_{ s }$	$i$	
1	2	1
1	3	1/2
2	1	1
2	3	1
3	1	0
3	2	0

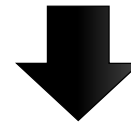
test data

세션 ID	항목 ID
4	2
4	3
4	1



추천 예측 값

항목 ID	2	3
예측 값	1	1/2



Top-1 추천 결과 : 항목 2

# 특정 사용자에게 대한 추천 결과 확인



## ➤ 각 알고리즘 별 특정 사용자에게 대한 추천 결과 비교

```
# 사용자가 이미 본 영화들 제외
```

```
missing_items = list(set(item_list) - set(train_dict[user_id]))
```

```
# 사용자의 마지막 영화를 기준으로 각 알고리즘을 이용한 추천 점수 구함
```

```
predicted_scores = []
```

```
for algo in [AR, SR, MC]:
```

```
    predicted_scores.append(algo.predict(train_dict[user_id][-1], missing_items))
```

```
missing_items = np.asarray(missing_items) # 추천 점수로 top-N 항목 추천
```

```
algo_names = ["AR", "SR", "MC"]
```

```
for i, predicted_score_algo in enumerate(predicted_scores):
```

```
    predicted_score_algo = np.asarray(predicted_score_algo)
```

```
    predicted_score_algo_idx = np.argsort(predicted_score_algo)
```

```
    predicted_score_algo_idx = predicted_score_algo_idx[::-1]
```

```
    ranked_items = missing_items[predicted_score_algo_idx]
```

```
    print("=====")
```

```
    print(algo_names[i] + " results")
```

```
    print("=====")
```

```
    for ranked_item in ranked_items[:10]:
```

```
        print(idx2title[ranked_item])
```

# 특정 사용자에게 대한 추천 결과 확인



출력

User id 1500이 본 마지막 영화: 다이 하드

출력

각 알고리즘 별 추천 영화 Top-3

=====

AR results

=====

터미네이터 2:오리지널  
다이 하드 2  
나 홀로 집에

=====

SR results

=====

다이 하드 2  
터미네이터  
터미네이터 2:오리지널

=====

MC results

=====

다이 하드 2  
빠빠용  
이티



# 4단계: 성능 평가 확인



## ➤ 각 알고리즘 별 평가 지표 Precision, recall, NDCG 비교

```
for user_id in user_list:
    missing_items = list(set(item_list) - set(train_dict[user_id]))

    predicted_scores = []
    for algo in [AR, SR, MC]:
        predicted_scores.append(algo.predict(train_dict[user_id][-1], missing_items))

    missing_items = np.asarray(missing_items)
    sorted_missing_items = []
    for predicted_score_algo in predicted_scores:
        predicted_score_algo = np.asarray(predicted_score_algo)
        predicted_score_algo_idx = np.argsort(predicted_score_algo)
        predicted_score_algo_idx = predicted_score_algo_idx[::-1]
        sorted_missing_items.append(missing_items[predicted_score_algo_idx])

    test_items = test_dict[user_id]
    for i, sorted_missing_algo in enumerate(sorted_missing_items):
        prec, recall, ndcg = compute_metrics(sorted_missing_algo, test_items, top_k)
        prec_list[algo_names[i]].append(prec)
        recall_list[algo_names[i]].append(recall)
        ndcg_list[algo_names[i]].append(ndcg)
```

# 4단계: 성능 평가 확인



```
for algo in algo_names:
    print(algo + " results" )
    print(f"Precision@{top_k}: {np.mean(prec_list[algo]):.3f}")
    print(f"Recall@{top_k}: {np.mean(recall_list[algo]):.3f}")
    print(f"NDCG@{top_k}: {np.mean(ndcg_list[algo]):.3f}")
    print("")
```

## 출력

### AR results

Precision@100: 0.056  
Recall@100: 0.553  
NDCG@100: 0.326

### MC results

Precision@100: 0.040  
Recall@100: 0.421  
NDCG@100: 0.251

### SR results

Precision@100: 0.051  
Recall@100: 0.515  
NDCG@100: 0.297

# Q&A





# Pandas 튜토리얼

# What is Pandas?



## ➤ Pandas: python에서 사용하는 데이터 분석 라이브러리

- ◆ Why?: 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)

# DataFrame 다루기



## ➤ DataFrame 생성

- ◆ Dictionary 형태
- ◆ Array 형태

## ➤ DataFrame 내 data 접근/수정

# DataFrame 생성



## ➤ Dictionary 형태의 data로 생성

- ◆ Key: column index가 됨.
- ◆ Value: 각 행이 됨.
- ◆ Row index: 따로 지정 가능.(default: [0,1,2 ..])

# Dictionary 형태의 data로 생성.

```
dict_data = {'col0': [1,2,3,4],  
             'col1': [5,6,7,8],  
             'col2': [9,10,11,12],}  
row = ['row0','row1','row2', 'row3']
```

```
data = pd.DataFrame(data=dict_data, index=row)  
data
```

	col0	col1	col2
row0	1	5	9
row1	2	6	10
row2	3	7	11
row3	4	8	12

# DataFrame 생성



## ➤ Array 형태의 data로 생성

- ◆ Array 형태 그대로 table을 구성
- ◆ Row, column index는 따로 지정

	col0	col1	col2
row0	1	5	9
row1	2	6	10
row2	3	7	11
row3	4	8	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
```



# DataFrame내 데이터 접근/수정



- Column을 이용하여 바로 접근 가능
- Dataframe.loc[] 형태로 접근 가능
- 특정 열/행에 접근하는 경우 Series형태로 output 출력
- 특정 data내에 entry에 접근

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

```
row0    0  
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
```



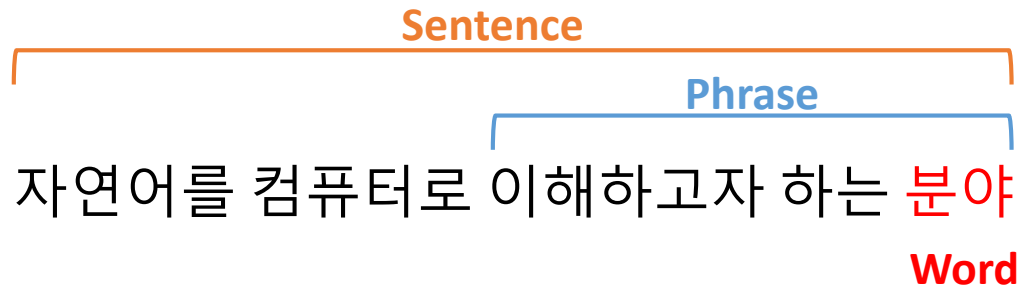
# 데이터 전처리 실습

# 자연어 표현 방법



## ➤ 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

# 텍스트 전처리



## ➤ 한국어와 영어는 문법이 다름

- ◆ 전처리 할 때 사용하는 패키지가 다름
  - 한국어 – KoNLPy, 영어 – NLTK
- ◆ 한국어 뉴스 데이터 → KoNLPy 사용

## ➤ KoNLPy

- ◆ **Okt**(Open Korean Text): 오픈 소스 한국어 분석기
- ◆ 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

# 텍스트 전처리



## ➤ 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.

# 텍스트 전처리 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','하  
다','한다','라는','된','에서','하고','할','될','이다','있다','이었다','했다','하는','있는','죠','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
```



# 단어 표현 실습

# 단어 표현 방법



## ➤ 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: Word2Vec



# 단어 표현 방법



## ➤ Bag of Words (BoW)

- ◆ 문서에서 단어의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도만을 고려한 표현 방법

- ◆ Document → A Set of words!

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!



## ➤ 예제

- ◆ ‘사람이 온다는 건 실은 어마어마한 일이다’
- ◆ {'사람이', '실은', '어마어마한', '온다는', '일이다'}

# 예제: BOW 표현



```
document = ['사람이 온다는 건 실은 어마어마한 일이다']  
  
# 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)
```

# 문서-단어 행렬



## ➤ Document term matrix (DTM)

- ◆ BoW 표현을 다수의 문서에 대해 행렬로 나타낸 것

## ➤ 예제 문서

- ◆ ‘사람이 온다는 건 실은 어마어마한 일이다.’
- ◆ ‘그는 그의 과거와 현재와 그리고 그의 미래와 함께 오기 때문이다.’
- ◆ ‘한 사람의 일생이 오기 때문이다.’

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	온다는	일생이	일이다	함께	현재와
0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0
1	1	1	1	2	1	1	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0

# 예제: 문서-단어 행렬



```
document = ['사람이 온다는 건 실은 어마어마한 일이다.', '
그는 그의 과거와 현재와 그리고 그의 미래와 함께 오기 때
문이다.', '한 사람의 일생이 오기 때문이다.']
```

```
# sklearn 사용하여 DTM (BoW와 방법 동일)
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)
```

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	온다는	일생이	일이다	함께	현재와
0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0
1	1	1	1	2	1	1	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0

# 문서-단어 행렬



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

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	온다는	일생이	일이다	함께	현재와
0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0
1	1	1	1	2	1	1	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0

# TF-IDF 기반 표현 방법



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

◆  $TF\text{-}IDF = \text{TF} * IDF$

## ➤ 단어 빈도수 (Term Frequency, TF)

- ◆ 특정 문서 안에서 특정 단어의 등장 빈도
- ◆ DTM과 동일

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	온다는	일생이	일이다	함께	현재와
0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0
1	1	1	1	2	1	1	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0

# TF-IDF 기반 표현 방법



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

- ◆  $TF-IDF = TF * IDF$
- ◆ 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)

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	문다는	일생이	일이다	함께	현재와
0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0
1	1	1	1	2	1	1	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0

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

# TF-IDF 기반 표현 방법



## ➤ TF-IDF = TF \* IDF

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

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	온다는	일생이	일이다	함께	현재와
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.447214	0.447214	0.447214	0.000000	0.447214	0.000000	0.447214	0.000000	0.000000
1	0.299385	0.299385	0.299385	0.59877	0.227690	0.299385	0.000000	0.000000	0.000000	0.000000	0.227690	0.000000	0.000000	0.000000	0.299385	0.299385
2	0.000000	0.000000	0.000000	0.000000	0.428046	0.000000	0.562829	0.000000	0.000000	0.000000	0.428046	0.000000	0.562829	0.000000	0.000000	0.000000

↑  
제곱의 합이 1.



# 예제: TF-IDF 기반 표현 방법



```
document = ['사람이 온다는 건 실은 어마어마한 일이다.',  
'그는 그의 과거와 현재와 그리고 그의 미래와 함께 오기  
때문이다.', '한 사람의 일생이 오기 때문이다.']
```

```
# sklearn 사용하여 TF-IDF
```

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

```
Tfidf_vect = TfidfVectorizer()
```

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

```
word_index = Tfidf_vect.get_feature_names()
```

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

	과거와	그는	그리고	그의	때문이다	미래와	사람의	사람이	실은	어마어마한	오기	온다는	일생이	일이다	함께	현재와
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.447214	0.447214	0.447214	0.000000	0.447214	0.000000	0.447214	0.000000	0.000000
1	0.299385	0.299385	0.299385	0.59877	0.227690	0.299385	0.000000	0.000000	0.000000	0.000000	0.227690	0.000000	0.000000	0.000000	0.299385	0.299385
2	0.000000	0.000000	0.000000	0.000000	0.428046	0.000000	0.562829	0.000000	0.000000	0.000000	0.428046	0.000000	0.562829	0.000000	0.000000	0.000000