



추천 시스템 CF, CB, hybrid model

대회팀 김희숙 이수연 이지호

목차

#01 Introduction

: CF(Collaborative Filtering), CB(Content-Based filtering) introduction with *Netflix Recommendation Engine*

#02 contents

: Vectorizer

: Similarity Measurement

#03 models

: cf+cb colabo Hybrid model with *Anime-Recommend*

: cf Generalized Matrix Factorization (Keras) model with *Recommendation System (CF) / Anime*

#04 Conclusion















Collaborative Filtering and Content based Filtering















#1. CF & CB introduction with *Netflix Recommendation Engine*

Collaborative Filtering (user based)

Movies that user clicked like button on

user1				
user2				
user3				
user4				













Which movie will you recommend for user 3?

user1				
user2				
user3				
user4				





#1. CF & CB introduction with *Netflix Recommendation Engine*

Collaborative Filtering (user based)

How can we find similar user?

		Antman	Avengers	Spiderman	Titanic	Gatsby
user1						
user2						
user3						
user4						



Let's represent like as 1
and now it is vector representation

		Antman	Avengers	Spiderman	Titanic	Gatsby
user1		1	1	1	0	0
user2		0	0	0	1	1
user3		1	1	0	0	0
user4		0	0	0	0	1

#1. CF & CB introduction with *Netflix Recommendation Engine*

Collaborative Filtering (user based)

Let's get similarity between user1 and user3

		Antman	Avengers	Spiderman	Titanic	Gatsby
user1		1	1	1	0	0
		X	X	X	X	X
user3		1	1	0	0	0
		1	1	0	0	0
		1 + 1 + 0 + 0 + 0 = 2				

Cosine similarity will be 0.81





		Antman	Avengers	Spiderman	Titanic	Gatsby
user1		1	1	1	0	0
		X	X	X	X	X
user3		1	1	0	0	0
		1	1	0	0	0
		Cosine similarity = $\frac{2}{\text{sqrt}(3) * \text{sqrt}(2)} = 0.81$				

$$\text{cosine similarity} = \cos\theta = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \sqrt{\sum_{i=1}^n (B_i)^2}}$$

#1. CF & CB introduction with *Netflix Recommendation Engine*

Collaborative Filtering(user based)

We can check all similarity here

		Antman	Avengers	Spiderman	Titanic	Gatsby	Cosine Similarity with user3
user1		1	1	1	0	0	0.81
user2		0	0	0	1	1	0
user3		1	1	0	0	0	1.00
user4		0	0	0	0	1	0

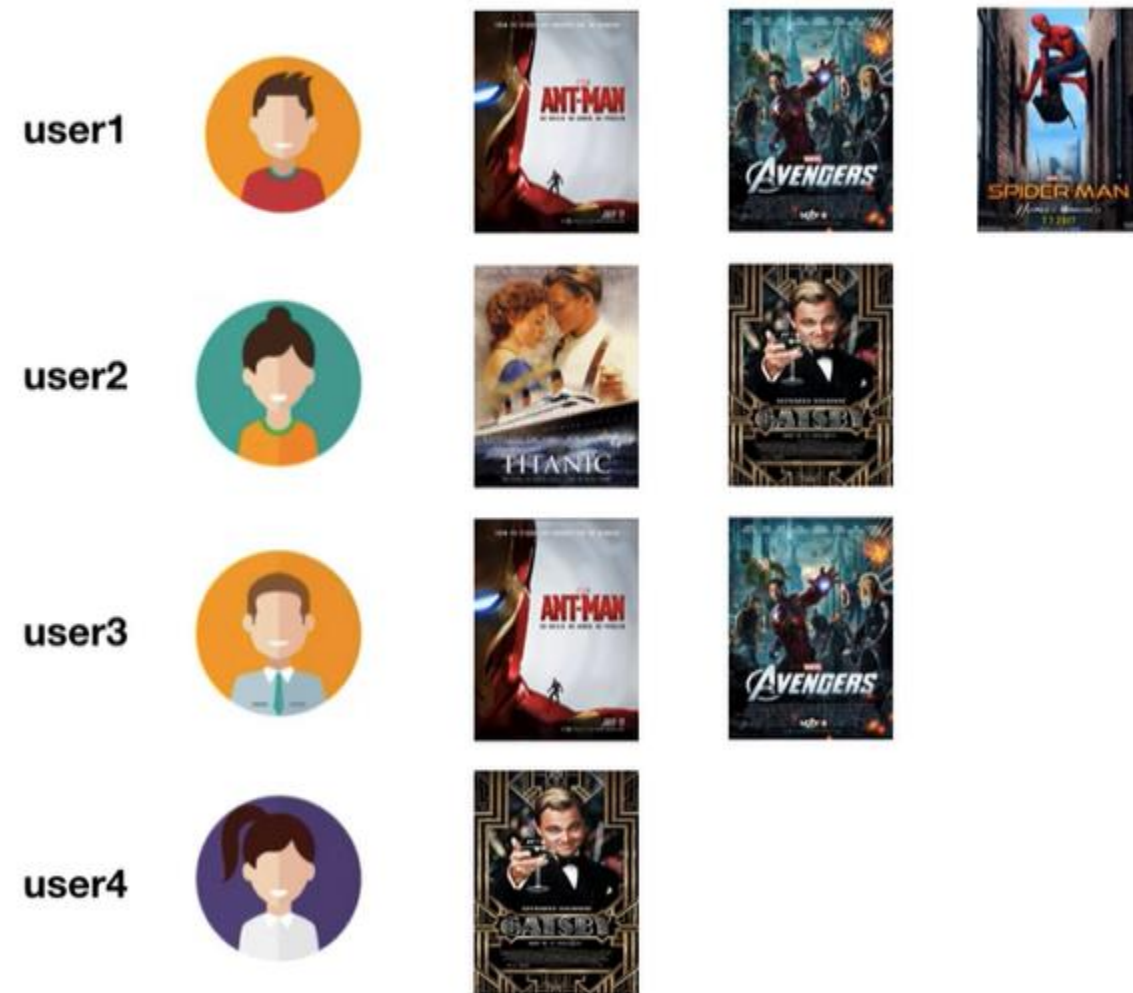
➡ 유사도가 높은 순서대로 추천

➡ 자기 자신의 유사도 = 1 ($\cos 0^\circ = 1$)

#1. CF & CB introduction with *Netflix Recommendation Engine*

Collaborative Filtering (item based)

Which movie will you recommend for user 4?



This is item based collaborative filtering



#1. CF & CB introduction with *Netflix Recommendation Engine*

Collaborative Filtering (item based)

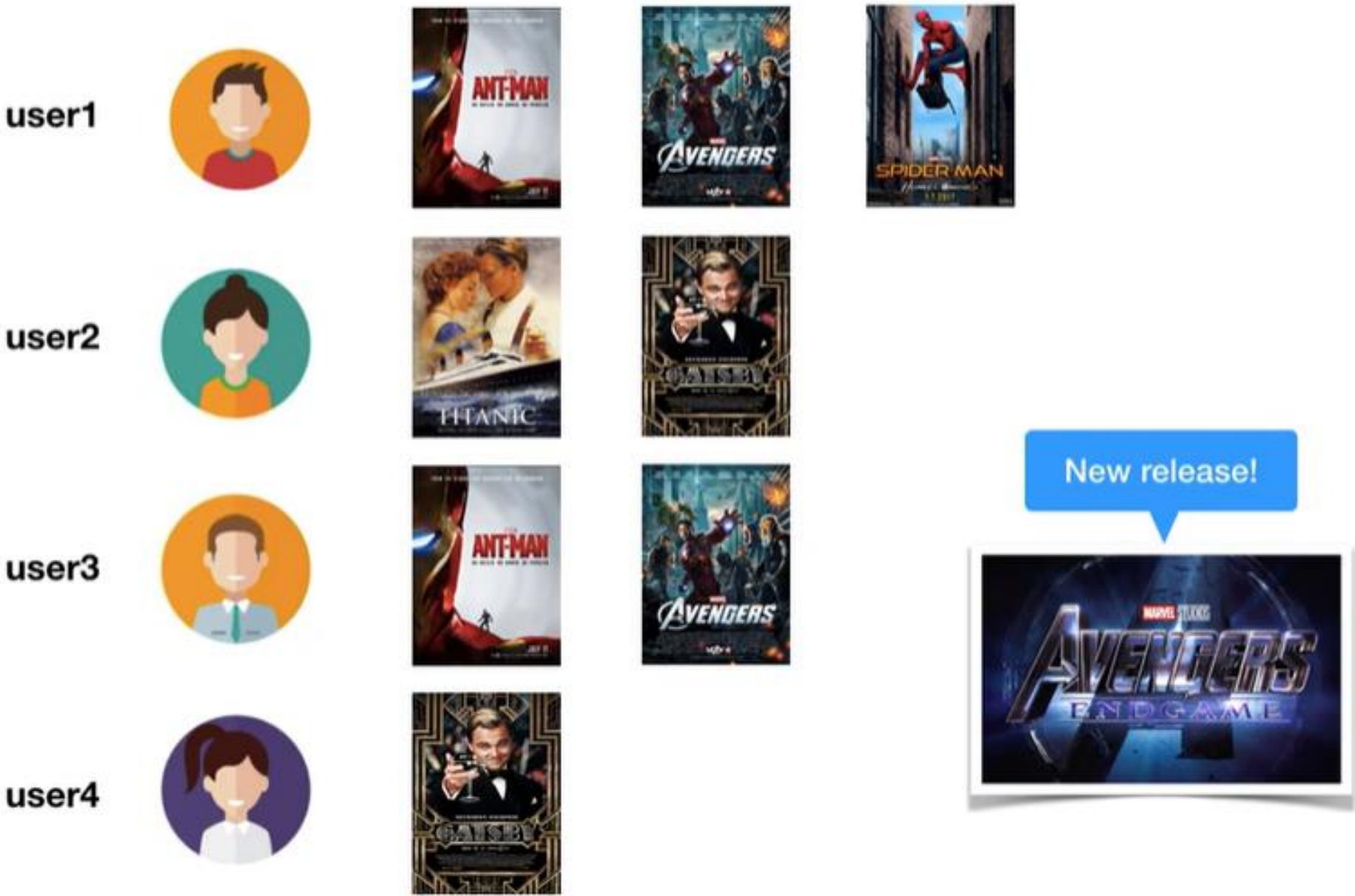
Here is the movie to user matrix

	user1	user2	user3	user4	Cosine Similarity with Gatsby
	1	0	1	0	0
	1	0	1	0	0
	1	0	0	0	0
	0	1	0	0	0.71
	0	1	0	1	1.00

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering






We call it **cold start** problem



#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering

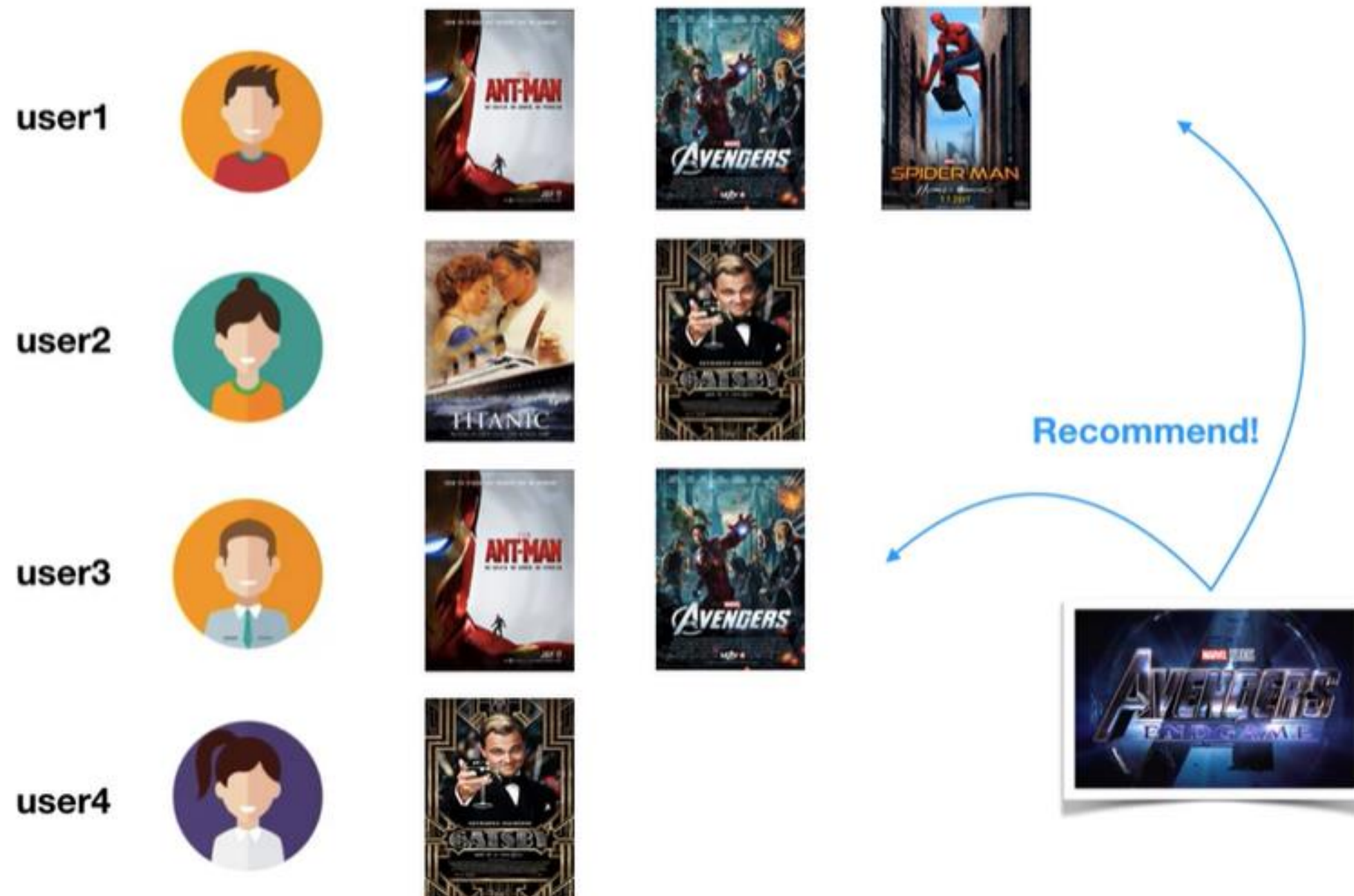
Content based filtering uses meta info as features,
so it can avoid cold start problem!

	Iron_Man	Captain_America	Spider_Man	Leonardo_Dicaprio	based_on_true_story		
	1	1	1	0	0		
	1	1	0	0	0	➡	0
	1	0	1	0	0	➡	0
	0	0	0	1	1	➡	0
	0	0	0	1	0	➡	0

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering

Content based filtering uses meta info as features,
so it can avoid cold start problem!



#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “CB Netflix Recommendation Engine” notebook

[Dataset]

2019년 넷플릭스에서 서비스한 TV쇼와 영화 정보로 구성된 데이터 셋. 넷플릭스는 최근 몇 년간 영화보다는 TV에 점점 더 집중하고 있다.

A	B	C	D	E	F	G	H	I	J	K	L	M
show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description	
s1	TV Show	3%		Joçat o Mig	Brazil	#####	2020	TV-MA	4 Seasons	Internation	In a future where the	
s2	Movie	7:19	Jorge Micl	Demiçat n E	Mexico	#####	2016	TV-MA	93 min	Dramas, In	After a devastating e	
s3	Movie	23:59	Gilbert Cha	Tedd Char	Singapore	#####	2011	R	78 min	Horror Mc	When an army recruit	
s4	Movie	9	Shane Ack	Elijah Woo	United Sta	#####	2009	PG-13	80 min	Action & A	In a postapocalyptic	
s5	Movie	21	Robert Lu	Jim Sturge	United Sta	01-Jan-20	2008	PG-13	123 min	Dramas	A brilliant group of st	

➤ Interesting tasks which can be performed on this dataset

- 1) 다양한 국가에서 이용 가능한 콘텐츠 이해
- 2) 텍스트 기반 기능을 일치시켜 유사한 내용 식별
- 3) 배우/감독 네트워크 분석 및 흥미로운 통찰력 발견

➤ Dataset Feature

제목 및 콘텐츠 타입과 더불어 감독, 출연진, 국가, 방영일, 평점, 콘텐츠 길이, 콘텐츠 소개 글까지 총 11개의 피처를 column으로 총 7781개의 콘텐츠에 대한 정보를 제공하고 있다.

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
▼ 1 # 넷플릭스에 있는 영화와 TV 프로그램의 콘텐츠 기반 추천 시스템을 만드는 프로젝트
2 # 2가지 방법 사용
3 # 1) 캐스트/감독/국가/별점/장르 등의 특징 기반으로 사용한 경우
4 # 2) 해당 영화 혹은 티비 쇼를 묘사하는 단어를 특징 기반으로 사용한 경우
5
6 import numpy as np
7 import pandas as pd
8 import re
9
10 import nltk # 방법2의 description feature를 전처리 하기위해
11 from nltk.corpus import stopwords
12 nltk.download('stopwords')
13 from nltk.tokenize import word_tokenize
```

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
1 print(len(data))
2 data.groupby('type').count()
```

executed in 60ms, finished 10:40:37 2022-03-15

7787

	show_id	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
type											
Movie	5377	5377	5214	4951	5147	5377	5377	5372	5377	5377	5377
TV Show	2410	2410	184	2118	2133	2400	2410	2408	2410	2410	2410

```
1 data = data.dropna(subset=['cast', 'country', 'rating'])
2 print(len(data)) # NaN 값이 포함된 행을 버림(default axis = 0)
```

executed in 30ms, finished 10:40:37 2022-03-15

6652

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

#방법1) Movie :캐스트/감독/국가/별점/장르 등의 특징 기반 추천 엔진 개발

-> cast, director, country, genre, rating 을 binary vector로 표현

```
1 movies = data[data['type'] == 'Movie'].reset_index()
2 movies = movies.drop(['index', 'show_id', 'type', 'date_added', 'release_year', 'duration', 'description'],
3                       axis=1)
4 movies.head()
```

executed in 88ms, finished 10:40:38 2022-03-15

	title	director	cast	country	rating	listed_in
0	7:19	Jorge Michel Grau	Demián Bichir, Héctor Bonilla, Oscar Serrano, ...	Mexico	TV-MA	Dramas, International Movies
1	23:59	Gilbert Chan	Tedd Chan, Stella Chung, Henley Hii, Lawrence ...	Singapore	R	Horror Movies, International Movies
2	9	Shane Acker	Elijah Wood, John C. Reilly, Jennifer Connelly...	United States	PG-13	Action & Adventure, Independent Movies, Sci-Fi...
3	21	Robert Luketic	Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar...	United States	PG-13	Dramas
4	122	Yasir Al Yasiri	Amina Khalil, Ahmed Dawood, Tarek Lotfy, Ahmed...	Egypt	TV-MA	Horror Movies, International Movies

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
1 # Feature
2 # Cast
3
4 # actors 리스트에 각 영화에 출연진 목록을 리스트로 담음.
5 actors = []
6 for i in movies['cast']:
7     actor = re.split(r',\s*', i)
8     actors.append(actor)
9
10 # 모든 배우 출연진을 flat_list 리스트에 담음
11 flat_list = []
12 for sublist in actors:
13     for item in sublist:
14         flat_list.append(item)
15
16 # 중복 제거 및 정렬해서 actors_list 생성
17 actors_list = sorted(set(flat_list))
18 binary_actors = [[0]*0 for i in range(len(set(flat_list)))]
19
20 for i in movies['cast']:
21     k = 0
22     for j in actors_list:
23         if j in i:
24             binary_actors[k].append(1.0)
25         else:
26             binary_actors[k].append(0.0)
27     k+=1
28
29 binary_actors = pd.DataFrame(binary_actors).transpose()
```

```
1 # Director
2 directors = []
3
4 for i in movies['director']:
5     if pd.notna(i):
6         director = re.split(r',\s*', i)
7         directors.append(director)
8
9 flat_list2 = []
10 for sublist in directors:
11     for item in sublist:
12         flat_list2.append(item)
13
14 directors_list = sorted(set(flat_list2))
15 binary_directors = [[0]*0 for i in range(len(set(flat_list2)))]
16
17 for i in movies['director']:
18     k=0
19     for j in directors_list:
20         if pd.isna(i):
21             binary_directors[k].append(0.0)
22         elif j in i:
23             binary_directors[k].append(1.0)
24         else:
25             binary_directors[k].append(0.0)
26     k+=1
27
28 binary_directors = pd.DataFrame(binary_directors).transpose()
```

-> Cast, Director, country, genre 동일한 방식으로 binary vector로 표현하는 처리 적용

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
1 # 모든 피쳐들 movies axis로 합치기
2 # (binary_actors, binary_directors, binary_countries, binary_genres, binary_ratings)
3
4 binary = pd.concat([binary_actors, binary_directors, binary_countries, binary_genres],
5                    axis=1,
6                    ignore_index=True) # ratings는 사용 안하네??
7
8 binary
9
```

	0	1	2	3	4	5	6	7	8	9	...	26570	26571	26572	26573	26574	26575	26576	26577	26578	26579
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
1	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
2	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	1.0	0.0	0.0	1.0	0.0	0.0	0.0
3	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	0.0	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	0.0	...	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
...
4756	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	0.0	0.0	0.0	0.0	0.0
4757	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4758	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	0.0	0.0	0.0	0.0	0.0
4759	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
4760	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	1.0	0.0	0.0	0.0	0.0	0.0

4761 rows × 26580 columns

Row = # of movies

Col = collection of binary vector features.

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
1 # search: 찾고자 하는 영화 인풋값.
2
3 def recommender(search):
4     cs_list = []
5     binary_list = []
6     # movies 에서 찾는다
7     if search in movies['title'].values:
8         idx = movies[movies['title'] == search].index.item() # 인풋 search와 타이틀이 일치하는 데이터 인덱스 찾기
9         for i in binary.iloc[idx]:
10             binary_list.append(i) # search 영화 정보(행)를 binary_list 리스트에 추가한다.
11     point1 = np.array(binary_list).reshape(1,-1)
12     point1 = [val for sublist in point1 for val in sublist]
13     ...
14     for sublist in point1:
15         for val in sublist:
16             val
17     ...
18     for j in range(len(movies)): # movies의 모든 행
19         binary_list2 = []
20         for k in binary.iloc[j]: # binary(movie)벡터를 모두 펼쳐서 binary_list2에 담음
21             binary_list2.append(k)
22         point2 = np.array(binary_list2).reshape(1,-1)
23         point2 = [val for sublist in point2 for val in sublist]
24         dot_product = np.dot(point1, point2)
25         norm_1 = np.linalg.norm(point1)
26         norm_2 = np.linalg.norm(point2)
27         cos_sim = dot_product / (norm_1 * norm_2)
28         cs_list.append(cos_sim) # j행 영화(point2)와 search(point1)과의 코사인 유사도를 cs_list 리스트에 담는다.
```

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
30 movies_copy = movies.copy()
31 movies_copy['cos_sim'] = cs_list
32 results = movies_copy.sort_values('cos_sim', ascending=False) # 카피한 movie 데이터를 cos_sim 기준 내림차순 정렬
33 results = results[results['title'] != search]
34 top_results = results.head(5)
35 return(top_results)
36 # 없으면, tv에서 찾는다. 위의 방식과 동일
37 elif search in tv['title'].values:
38     idx = tv[tv['title'] == search].index.item()
39     for i in binary2.iloc[idx]:
40         binary_list.append(i)
41     point1 = np.array(binary_list).reshape(1, -1)
42     point1 = [val for sublist in point1 for val in sublist]
43     for j in range(len(tv)):
44         binary_list2 = []
45         for k in binary2.iloc[j]:
46             binary_list2.append(k)
47         point2 = np.array(binary_list2).reshape(1, -1)
48         point2 = [val for sublist in point2 for val in sublist]
49         dot_product = np.dot(point1, point2)
50         norm_1 = np.linalg.norm(point1)
51         norm_2 = np.linalg.norm(point2)
52         cos_sim = dot_product / (norm_1 * norm_2)
53         cs_list.append(cos_sim)
54     tv_copy = tv.copy()
55     tv_copy['cos_sim'] = cs_list
56     results = tv_copy.sort_values('cos_sim', ascending=False)
57     results = results[results['title'] != search]
58     top_results = results.head(5)
59     return(top_results)
60 else:
61     return("Title not in dataset. Please check spelling.")
```


#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

방법1) Recommender 1 Result.

```
1 recommender('The Conjuring')
executed in 4m 57s, finished 11:06:03 2022-03-15
```

	title	director	cast	country	rating	listed_in	cos_sim
1868	Insidious	James Wan	Patrick Wilson, Rose Byrne, Lin Shaye, Ty Simp...	United States, Canada, United Kingdom	PG-13	Horror Movies, Thrillers	0.388922
968	Creep	Patrick Brice	Mark Duplass, Patrick Brice	United States	R	Horror Movies, Independent Movies, Thrillers	0.377964
1844	In the Tall Grass	Vincenzo Natali	Patrick Wilson, Laysla De Oliveira, Avery Whit...	Canada, United States	TV-MA	Horror Movies, Thrillers	0.370625
969	Creep 2	Patrick Brice	Mark Duplass, Desiree Akhavan, Karan Soni	United States	TV-MA	Horror Movies, Independent Movies, Thrillers	0.356348
1077	Desolation	Sam Patton	Jaimi Paige, Alyshia Ochse, Toby Nichols, Clau...	United States	TV-MA	Horror Movies, Thrillers	0.356348

#1. CF & CB introduction with *Netflix Recommendation Engine*

Content-Based Filtering – “Netflix Recommendation Engine” notebook

```
1 # description 전처리: 토큰화 / 불용어 제거 후 각 단어 토큰들을 binary화 해서 리스트로
2 filtered_movies = []
3 movies_words = []
4
5 for text in movies_des['description']:
6     text_tokens = word_tokenize(text) # description 문장 -> 토큰화한 단어 리스트로 반환
7     # text_tokens 소문자로 변환하고 stopwords 제거
8     tokens_without_sw = [word.lower() for word in text_tokens if not word in stopwords.words()]
9     movies_words.append(tokens_without_sw) # 각 영화에 사용된 단어 토큰 저장
10    filtered = (" ").join(tokens_without_sw)
11    filtered_movies.append(filtered) # 각 영화에 사용된 단어토큰 합쳐서 다시 문장으로 저장(대소문자/불용어 제거 전처리 적용)
12
13 movies_words = [val for sublist in movies_words for val in sublist]
14 movies_words = sorted(set(movies_words))
15
16 # 필터링한 문장을 'description_filtered'로 movies의 열에 새로 추가
17 movies_des['description_filtered'] = filtered_movies
18 movies_des.head()
19
```

```
1 movie_word_binary = [[0]*0 for i in range(len(set(movies_words)))]
2
3 for des in movies_des['description_filtered']:
4     k=0
5     for word in movies_words:
6         if word in des:
7             movie_word_binary[k].append(1.0)
8         else:
9             movie_word_binary[k].append(0.0)
10        k+=1
11
12 movie_word_binary = pd.DataFrame(movie_word_binary).transpose()
13 print(movie_word_binary.shape)
```

(4761, 14577)

	title	description	description_filtered
0	7:19	After a devastating earthquake hits Mexico Cit...	after devastating earthquake hits mexico city ...
1	23:59	When an army recruit is found dead, his fellow...	when army recruit found dead , fellow soldiers...
2	9	In a postapocalyptic world, rag-doll robots hi...	in postapocalyptic world , rag-doll robots hid...
3	21	A brilliant group of students become card-coun...	a brilliant group students become card-countin...

contents



#2.

1. Vectorizer

- CountVecorizer
- TdifVectorizer

2. Similarity Measurement

- Cosine similarity
- Pearson correlation
- Jaccard Similarity
- Adjusted cosine similarity

#2-1. Vectorizer

- CountVectorizer

- Document-Term Matrix

```
doc1 = "She likes python"  
doc2 = "She hates python"
```

	She	likes	hate	pytho n
doc1	1	1	0	1
doc2	1	0	1	1

```
doc1 = "She likes python"  
doc2 = "She hates python"
```

	Potte r	likes	Sing er
doc1	1	1	1
doc2	1	1	1

#2-1. Vectorizer

- CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

doc1 = "I like python"
doc2 = "I hate python"

doc_term_matrix = count_vec.fit_transform([doc1, doc2]).toarray()

print(doc_term_matrix)
```

```
[[0 1 1 1]
 [1 0 1 1]]
```

#2-1. Vectorizer

- TfidfVectorizer

- TF (Term Frequency)

- 특정 단어가 한 문서 내에서 출현한 빈도

- IDF (Inverse Document Frequency)

- 특정한 단어가 출현한 전체 문서의 개수

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

- TF-IDF

- TF * IDF

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

	She	like s	hate	pyth on
doc 1	1	1	0	1
doc 2	1	0	1	1

#2-1. Vectorizer

- TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

doc1 = "She likes python"
doc2 = "She hates python"

tfidf_vec = TfidfVectorizer()
tfidf_vec.fit_transform([doc1, doc2]).toarray()
—
[[0.          0.70490949 0.50154891 0.50154891]
 [0.70490949 0.          0.50154891 0.50154891]]
```

	She	like s	hate	pyth on
doc 1	1	1	0	1
doc 2	1	0	1	1

#2-2. Similarity Measurement

- Cosine similarity
- Adjusted cosine similarity
- Pearson correlation
- Jaccard Similarity

#2-2. Similarity Measurement

- Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

$$\text{Cos_Sim}(u,v) = \frac{\vec{R}_u \bullet \vec{R}_v}{|\vec{R}_u| \cdot |\vec{R}_v|}$$

Where “•” represents dot product of two vectors. \vec{R}_u and \vec{R}_v are rating vectors of user u and v respectively.

- collaborative filtering 에서 가장 흔히 사용되는 유사도 측정 방법.
- 두 벡터 사이의 코사인 각을 측정하여 유사성 파악.
- null preferences를 음의 값으로 간주.

#2-2. Similarity Measurement

- Cosine Similarity

```
from sklearn.metrics.pairwise import cosine_similarity, cosine_distances

A=np.array([10,3])
B=np.array([8,7])

result=cosine_similarity(A.reshape(1,-1),B.reshape(1,-1))

print(result)
# [[0.91005765]]
```

#2-2. Similarity Measurement

- Adjusted cosine similarity
 - 코사인 유사도는 서로 다른 사용자가 서로 다른 등급 척도를 사용하는 경우를 고려하지 않음.
 - ASS는 유저로부터 계산된 평균 rating을 빼주어 각 사용자가 사용하는 등급 척도의 차이를 고려

$$Adjusted_COS_Sim = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2 + \sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

where \bar{r}_i Average rating of user i for all items rated by user i itself.

#2-2. Similarity Measurement

- Pearson Similarity

```
import numpy as np

x_simple = np.array([-2, -1, 0, 1, 2])
y_simple = np.array([4, 1, 3, 2, 0])
my_rho = np.corrcoef(x_simple, y_simple)

print(my_rho)

#[[ 1.  -0.7]
# [-0.7  1. ]]
```

$$\frac{\sum_{i=1}^n (Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

$$\frac{\sum_{i \in I} (r_{u,i} - \bar{r}_{u,I})(r_{v,i} - \bar{r}_{v,I})}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_{u,I})^2 + \sum_{i \in I} (r_{v,i} - \bar{r}_{v,I})^2}}$$

its Average rating of user u and user v ,
is represented by set I

#2-2. Similarity Measurement

- Pearson Similarity

```
import numpy as np

x_simple = np.array([-2, -1, 0, 1, 2])
y_simple = np.array([4, 1, 3, 2, 0])
my_rho = np.corrcoef(x_simple, y_simple)

print(my_rho)

#[[ 1.  -0.7]
# [-0.7  1. ]]
```

$$\frac{\sum_{i=1}^n (Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

$$\frac{\sum_{i \in I} (r_{u,i} - \bar{r}_{u,I})(r_{v,i} - \bar{r}_{v,I})}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_{u,I})^2 + \sum_{i \in I} (r_{v,i} - \bar{r}_{v,I})^2}}$$

its Average rating of user u and user v ,
is represented by set I

#2-2. Similarity Measurement

- Jaccard Similarity

- 두 유저간의 공통되는 선호도 고려.
- 공통 등급 항목을 더 많이 가지고 있는 경우 사용자의 유사성이 높음.
- 제한된 수의 값을 생성하므로 사용자 식별 작업이 어려움.
- [0, 1] 사이의 값을 가짐

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

$$Jaccard_sim(u, v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|}$$

Where I_u is a set of items rated by user u and I_v is a set of items rated by user v .

#2-2. Similarity Measurement

- Jaccard Similarity

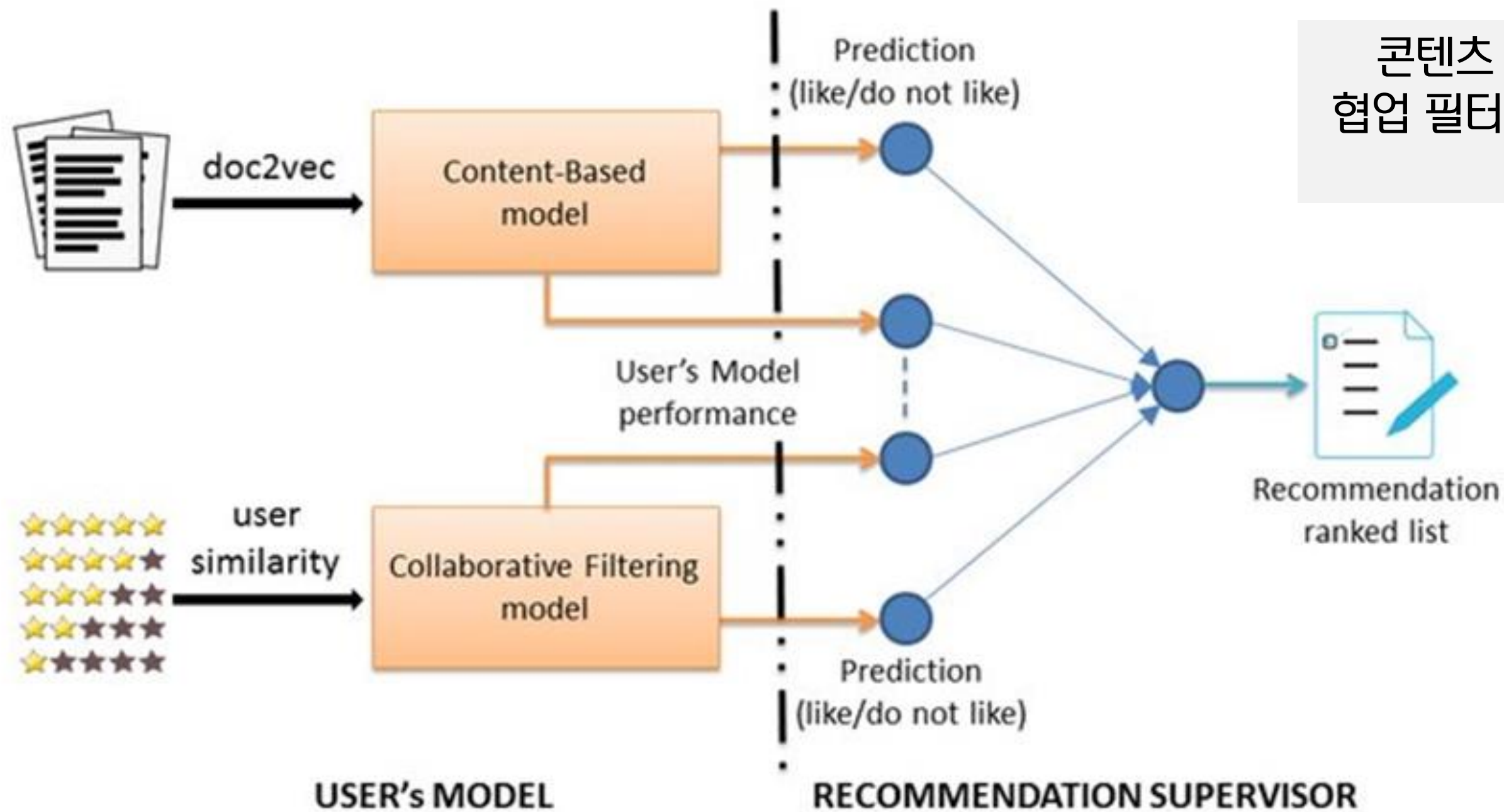
```
def jaccard_similarity(list1, list2):  
    s1 = set(list1)  
    s2 = set(list2)  
    return float(len(s1.intersection(s2)) / len(s1.union(s2)))  
  
list1 = ["삼성전자", "테슬라", "LG전자", "카카오", "풀어비스"]  
list2 = ["삼성전자", "카카오", "넷마블", "현대자동차", "셀트리온"]  
  
print('jaccard_similarity : ', jaccard_similarity(list1, list2))  
# jaccard_similarity : 0.25
```

models



#3. hybrid model

하이브리드 추천 시스템 (CB+CF)



콘텐츠 기반 추천 시스템(CB)과
협업 필터링 기반 추천 시스템(CF)을
결합한 모델

#3. hybrid model

#Anime-Recommender(CF, CB, hybrid)

```
from scipy.sparse import csr_matrix
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
```

- 텍스트 특징 추출
- 코사인 유사도 계산
- 협업 필터링
 - 잠재 요인 필터링
 - 행렬 분해 기법(SVD)

- Surprise 추천알고리즘 클래스

클래스명	설명
SVD	행렬 분해를 통한 잠재 요인 협업 필터링을 위한 SVD 알고리즘
KNNBasic	최근접 이웃 협업 필터링을 위한 KNN 알고리즘
BaselineOnly	사용자 Bias와 아이템 Bias를 고려한 SGD 베이스라인 알고리즘

#3. hybrid model

#Anime-Recommender(CF, CB, hybrid)

```
tfidf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')  
tfidf_matrix = tfidf.fit_transform(anime_df['synopsis'])
```

analyzer: 학습 단위
Min-df: 최소 빈도값
ngram_range: 단어 묶음
stop_words: 불용어 목록

```
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

linear_kernel을 통해 앞에서 만들어 놓은
tfidf_matrix를 넣어 코사인 유사도 계산

```
reader = Reader()  
rating_data = Dataset.load_from_df(rating_df, reader)  
trainset = rating_data.build_full_trainset()
```

```
svd = SVD()  
svd.fit(trainset)  
svd.predict(1, 356, 5)
```

predict(): 개별 사용자와 애니에 대한 추천 평점 반환

- 예측 평점(est)
- 실제 평점(r_ui)

#3. hybrid model

#Anime-Recommender(CF, CB, hybrid)

```
def hybrid_recommendations(user_id, title):  
    idx = indices[title]  
    sim_scores = list(enumerate(cosine_sim[idx]))  
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)  
    sim_scores = sim_scores[1:31]  
    anime_indices = [i[0] for i in sim_scores]  
  
    anime_lst = anime_df.iloc[anime_indices][['MAL_ID', 'Name', 'Members', 'Score', 'Genres']]  
    favorite_count = anime_lst[anime_lst['Members'].notnull()]['Members'].astype('int')  
    score_avg = anime_lst[anime_lst['Score'].notnull()]['Score'].astype('float')  
    C = score_avg.mean()  
    m = favorite_count.quantile(0.60)  
    qualified = anime_lst[(anime_lst['Members'] >= m) & (anime_lst['Members'].notnull()) & (anime_lst['Score'].notnull())]  
    qualified['Members'] = qualified['Members'].astype('int')  
    qualified['Score'] = qualified['Score'].astype('float')  
    def weighted_rating(x):  
        v = x['Members']  
        R = x['Score']  
        return (v/(v+m) * R) + (m/(m+v) * C)  
  
    qualified['wr'] = qualified.apply(weighted_rating, axis=1)  
    qualified = qualified.sort_values('wr', ascending=False).head(30)  
  
    qualified[['id']] = list(range(1, qualified.shape[0]+1, 1))  
    qualified['est'] = qualified['id'].apply(lambda x: svd.predict(user_id, indices_map.loc[x]['MAL_ID']).est)  
    qualified = qualified.sort_values('est', ascending=False)  
    result = qualified[['MAL_ID', 'Name', 'Genres', 'Score']]  
    return result.head(10)
```

CB

CB를 통해 사용자가 높게 평가한 애니의 members, genres 등을 감안하여 가중치를 계산하고 가중치가 큰 순으로 30개를 뽑는다.

CF

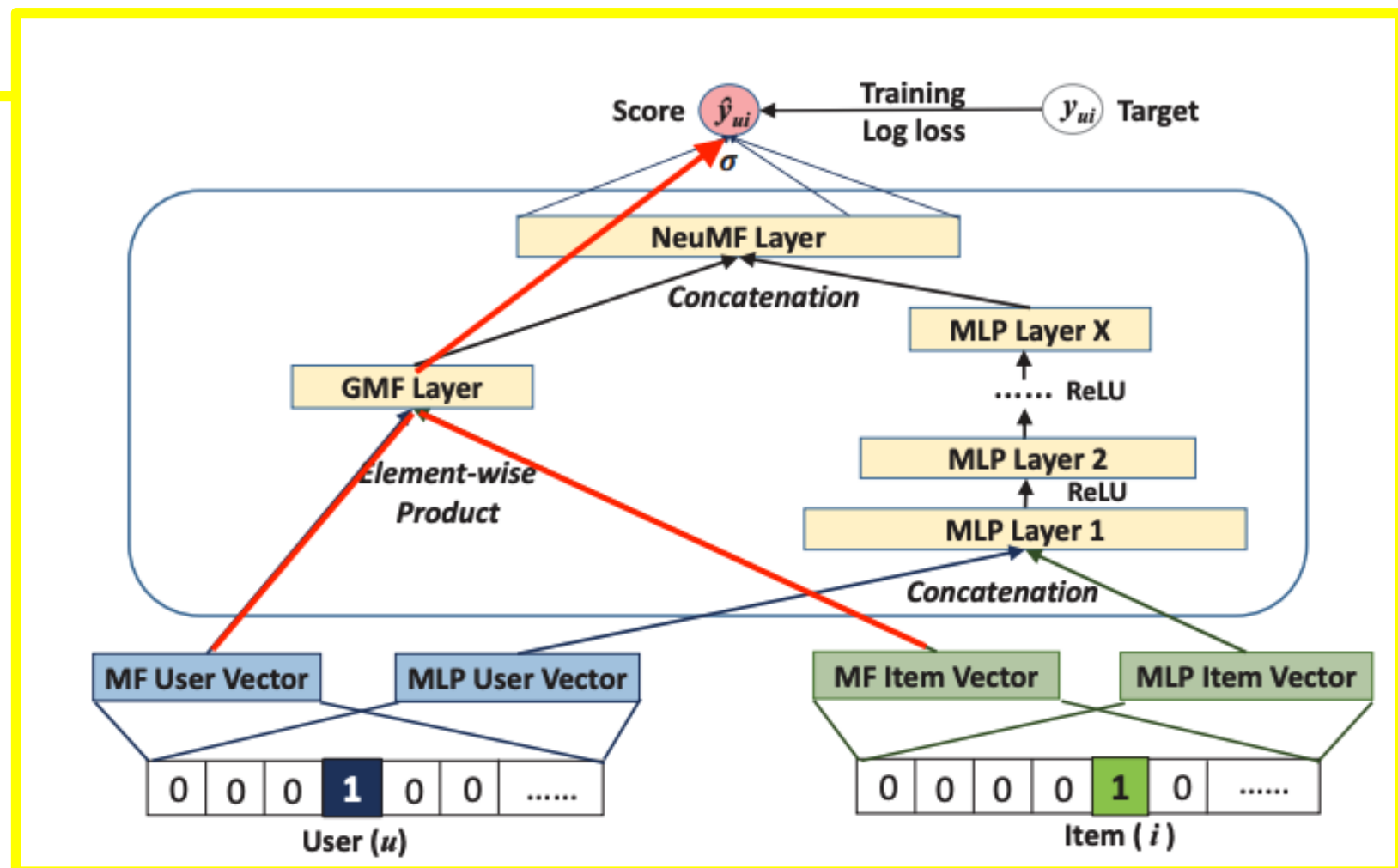
CF에서는 CB의 결과로 추출된 30개 중 사용자의 예상 점수가 높은 상위 10개의 애니 리스트를 최종적으로 출력한다.

#3. Deep learning MF based Keras

Collaborative filtering model_based

- Matrix Factorization (MF) based
 1. TruncatedSVD (Sklearn)
 2. Funk MF (Surprise) ex. SVD
 3. Non negative MF (Surprise)
- Deep learning MF based
 1. Generalized MF (Keras)
 2. Neural Collaborative filtering (Recommender)

Generalized Matrix Factorization (Keras)



#3. Deep learning MF based Keras

#Anime-Recommendation System(CF)

```
# Embedding layers
from keras.layers import Add, Activation, Lambda, BatchNormalization, Concatenate, Dropout, Input, Embedding, Dot, Reshape, Dense, Flatten

def RecommenderNet():
    embedding_size = 128

    user = Input(name = 'user', shape = [1])
    user_embedding = Embedding(name = 'user_embedding',
                               input_dim = n_users,
                               output_dim = embedding_size)(user)

    anime = Input(name = 'anime', shape = [1])
    anime_embedding = Embedding(name = 'anime_embedding',
                                input_dim = n_animes,
                                output_dim = embedding_size)(anime)

    #x = Concatenate()([user_embedding, anime_embedding])
    x = Dot(name = 'dot_product', normalize = True, axes = 2)([user_embedding, anime_embedding])
    x = Flatten()(x)

    x = Dense(1, kernel_initializer='he_normal')(x)
    x = BatchNormalization()(x)
    x = Activation("sigmoid")(x)

    model = Model(inputs=[user, anime], outputs=x)
    model.compile(loss='binary_crossentropy', metrics=["mae", "mse"], optimizer='Adam')

    return model
```

협업 필터링

- 최근접 이웃
- 사용자 기반 / 아이템 기반 둘 다 구현

Conclusion



#차별점

#1 Anime CF

- 모델 기반 추천 알고리즘으로 콜백 함수(l model_checkpoints, lr_callback, early_stopping)를 통해 파라미터를 튜닝해 성능을 높이하고자 함

#2 Anime CF, CB, hybrid

- CB, CF 추천 시스템을 정의 후 이 둘을 결합한 hybrid 추천 시스템 구현(surprise 라이브러리 사용)

#3 Netflix CB

- 콘텐츠의 feature를 binary vector로 표현. 서술 식 description의 단어 벡터를 이용한 엔진.

THANK YOU

