



Recommendation System

김재훈 김은하 김정현
박기태 안수빈 이혜연



Table of Contents

1. Content based filtering
2. Collaborative filtering methods
 - 2.1 Memory based collaborative filtering methods
 - 2.2 Model based collaborative filtering methods
3. Deep Learning based

1. Content based filtering

```
In [2]: key = pd.read_csv('place_data.csv')
```

```
In [3]: key.head()
```

```
Out [3]:
```

Unnamed: 0	이름	코드	태그
0	고른햇살	Num=a_nam_0002	#먹어야산다#분식#정대후문#포장#혼밥#가성비
1	영철버거	Num=a_nam_0041	#먹어야산다#이공계#경양식#버거#맥주#치킨#가성비#저렴#포장
2	어흥식당	Num=a_nam_829	#먹어야산다#양식#스테이크#정대후문#혼밥#고기#함박#가성비
3	언니네반점	Num=a_nam_0146	#먹어야산다#중식#정대후문#사천식#혼밥#가성비#저렴#포장#회식
4	특별식당	Num=a_nam_0319	#먹어야산다#일식#돈부리#카레#맥주#고로케#제기동#고대사거리#혼밥#감성

```
In [6]: key.head()
```

```
Out [6]:
```

	name	code	tag
0	고른햇살	Num=a_nam_0002	#먹어야산다#분식#정대후문#포장#혼밥#가성비
1	영철버거	Num=a_nam_0041	#먹어야산다#이공계#경양식#버거#맥주#치킨#가성비#저렴#포장
2	어흥식당	Num=a_nam_829	#먹어야산다#양식#스테이크#정대후문#혼밥#고기#함박#가성비
3	언니네반점	Num=a_nam_0146	#먹어야산다#중식#정대후문#사천식#혼밥#가성비#저렴#포장#회식
4	특별식당	Num=a_nam_0319	#먹어야산다#일식#돈부리#카레#맥주#고로케#제기동#고대사거리#혼밥#감성

‘tag’ 열에 대한 자연어 처리 필요

- 1) Rake() 활용
- 2) Tf-Idf 활용



1. Content based filtering

1) Rake()

- 동시 출현(Co-occurrences) 개념

- 각 단어의 빈도 table을 작성한 후, 단어 t가 다른 단어와 동시에 등장했던 빈도 수 합(degree)를 단어 t의 고유 빈도로 나눈다.

$$ratio(t) = \frac{deg(t)}{freq(t)}$$

```
In [7]: keywords = []
```

```
In [8]: for i in range(465):  
        tag = key['tag'][i]  
  
        r = Rake()  
        r.extract_keywords_from_text(tag)  
  
        # get a list of keywords by rank  
        words = r.get_ranked_phrases()  
        keywords.append(words)
```

```
In [9]: key['key_words'] = keywords
```

1. Content based filtering

Ex. Orange의 빈도수

= ratio(Orange)

= deg(Orange)/freq(Orange)

= (5+5+1+1) / 5 = 12/5 = 2.4

	Apple	Chicken	Pants	Orange	Car	Hat	Pizza	Bike	Train
Apple	7		2	5				1	
Chicken		5			2		4	2	
Pants	2		6			5			
Orange	5			5			1		1
Car		2			6			3	2
Hat			5			5			
Pizza		4		1			4		
Bike	1	2			3			5	2
Train				1	2			2	3

1. Content based filtering

<처리 결과>

In [10]: `key.head()`

Out [10]:

	name	code	tag	key_words
0	고른햇살	Num=a_nam_0002	#먹어야산다#분식#정대후문#포장#혼밥#가성비	[혼밥, 포장, 정대후문, 분식, 먹어야산다, 가성비]
1	영철버거	Num=a_nam_0041	#먹어야산다#이공계#경양식#버거#맥주#치킨#가성비#저렴#포장	[포장, 치킨, 저렴, 이공계, 버거, 먹어야산다, 맥주, 경양식, 가성비]
2	여흥식당	Num=a_nam_829	#먹어야산다#양식#스테이크#정대후문#혼밥#고기#함박#가성비	[혼밥, 함박, 정대후문, 양식, 스테이크, 먹어야산다, 고기, 가성비]
3	언니네반점	Num=a_nam_0146	#먹어야산다#중식#정대후문#사천식#혼밥#가성비#저렴#포장#회식	[회식, 혼밥, 포장, 중식, 정대후문, 저렴, 사천식, 먹어야산다, 가성비]
4	특별식당	Num=a_nam_0319	#먹어야산다#일식#돈부리#카레#맥주#고로케#제기동#고대사거리#혼밥#감성	[혼밥, 카레, 제기동, 일식, 먹어야산다, 맥주, 돈부리, 고로케, 고대사거리, 감성]

1. Content based filtering

```
In [12]: count = CountVectorizer()
count_matrix = count.fit_transform(key['key_words'])

count_matrix = count_matrix.toarray()

# 코사인 유사도 행렬 생성
c_sim = cosine_similarity(count_matrix, count_matrix)
```

```
In [14]: # 코사인 유사도 행렬
print(c_sim)

[[1.          0.40824829 0.57735027 ... 0.40824829 0.23570226 0.16666667]
 [0.40824829 1.          0.23570226 ... 0.33333333 0.19245009 0.13608276]
 [0.57735027 0.23570226 1.          ... 0.35355339 0.20412415 0.14433757]
 ...
 [0.40824829 0.33333333 0.35355339 ... 1.          0.57735027 0.40824829]
 [0.23570226 0.19245009 0.20412415 ... 0.57735027 1.          0.23570226]
 [0.16666667 0.13608276 0.14433757 ... 0.40824829 0.23570226 1.          ]]
```

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

'key_words' 문자열 데이터의 벡터 변환
→ CountVectorizer() 이용

코사인 유사도 행렬 생성
→ cosine_similarity() 이용
→ 연관성이 높을 수록 1에 가깝다

1. Content based filtering

```
In [19]: def recommendation(name, top):
indices = c_sim[c_sim.index == name].values[0]
indices = np.sort(indices)[: -1][:top*2]

temp = []
for k in range(len(indices)):
    idx_name = c_sim.index[indices[k] == c_sim[c_sim.index == name].values[0]]
    temp.append(idx_name[0])

# 중복된 값 제거
recommend = []
for t in temp:
    if t in recommend:
        pass
    else:
        recommend.append(t)

# 입력값과 동일한 항목은 제거
recommend = recommend[1:top+1]
return recommend
```

추천 함수 생성

➔ 식당 이름을 입력하면 코사인
유사도가 높은 상위 n개의
유사한 식당 추천

1. Content based filtering

<추천 결과>

```
In [20]: recommendation('고른햇살', 8)
```

```
Out[20]: ['김밥천국 정대후문점', '절대분식', '엄마네', '돈까스하우스', '안암동김밥', '언니네반점', '이세돈까스', '이공김밥']
```

```
In [21]: recommendation('매스플레이트', 4)
```

```
Out[21]: ['정상호프', '어흥식당', '무르무르드구스토', '모이리타']
```

```
In [22]: recommendation('베나레스', 5)
```

```
Out[22]: ['오샬', '비나 레스토랑 정문', '고고인디안쿠진 1호점', '돼야지 고대본점', '고고인디안쿠진 2호점']
```

1. Content based filtering

2) TF-IDF(Term Frequency – Inverse Document Frequency)

In [7]: #tag를 '#'기준으로 split

```
place['tag']=place['tag'].str.split(pat="#")
place.head(2)
```

Out [7]:

Unnamed: 0	이름	코드	태그	tag
0	고른햇살	Num=a_nam_0002	#먹어야산다#분식#정대후문#포장#혼밥#가성비	[, 먹어야산다, 분식, 정대후문, 포장, 혼밥, 가성비]
1	영철버거	Num=a_nam_0041	#먹어야산다#이공계#경양식#버거#맥주#치킨#가성비#저렴#포장	[, 먹어야산다, 이공계, 경양식, 버거, 맥주, 치킨, 가성비, 저렴, 포장]

In [8]: #tag를 띄어쓰기 기준으로 붙이기

```
place['tag']=place['tag'].apply(lambda x: " ".join(x))
place.head(2)
```

Out [8]:

Unnamed: 0	이름	코드	태그	tag
0	고른햇살	Num=a_nam_0002	#먹어야산다#분식#정대후문#포장#혼밥#가성비	먹어야산다 분식 정대후문 포장 혼밥 가성비
1	영철버거	Num=a_nam_0041	#먹어야산다#이공계#경양식#버거#맥주#치킨#가성비#저렴#포장	먹어야산다 이공계 경양식 버거 맥주 치킨 가성비 저렴 포장

→ #(해시태그)
기호로 구분되어
있는 Keywords를
split

1. Content based filtering

2) Tf-Idf (Term Frequency – Inverse Document Frequency)

- 한 문서 내에서 특정 단어의 상대적 중요도
- 한 문서 내에서 자주 등장하면 가중치를 높게, 그러나 다른 문서에서 자주 등장한다면 가중치를 낮게!

$$tf(d, t) \times \log\left(\frac{n}{1 + df(t)}\right)$$

```
In [10]: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [11]: # tag에 대해서 tf-idf 수행
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(place.tag)
print(tfidf_matrix.shape)

(465, 374)
```

```
In [12]: #코사인유사도
from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
In [13]: cosine_sim.shape
```

```
Out[13]: (465, 465)
```

```
In [14]: cosine_sim
```

```
Out[14]: array([[1.          , 0.1829687 , 0.32195165, ..., 0.17059754, 0.02249117,
        0.01932993],
        [0.1829687 , 1.          , 0.07223991, ..., 0.09000334, 0.01186583,
        0.01019803],
        [0.32195165, 0.07223991, 1.          , ..., 0.09322109, 0.01229005,
        0.01056262],
        ...,
        [0.17059754, 0.09000334, 0.09322109, ..., 1.          , 0.13183762,
        0.11330718],
        [0.02249117, 0.01186583, 0.01229005, ..., 0.13183762, 1.          ,
        0.01493815],
        [0.01932993, 0.01019803, 0.01056262, ..., 0.11330718, 0.01493815,
        1.          ]])
```

1. Content based filtering

In [16]: #추천 함수

```
def place_REC(코드, cosine_sim=cosine_sim):
    #입력한 코드로 부터 인덱스 가져오기
    idx = indices[코드]

    # 모든 장소에 대해서 해당 장소와의 유사도를 구하기
    sim_scores = list(enumerate(cosine_sim[idx]))

    # 유사도에 따라 장소들을 정렬
    sim_scores = sorted(sim_scores, key=lambda x:x[1], reverse = True)

    # 가장 유사한 10개의 장소를 받아옴
    sim_scores = sim_scores[1:11]

    # 가장 유사한 10개 장소의 인덱스 받아옴
    place_indices = [i[0] for i in sim_scores]

    #기존 데이터에서 해당 인덱스의 값들을 가져온다. 그리고 스코어 열을 추가하여 코사인 유사도도 확인할 수 있게 한다.
    result_place = place.iloc[place_indices].copy()
    result_place['score'] = [i[1] for i in sim_scores]

    # 읽어들이는 데이터에서 tag부분만 제거, 코드와 스코어만 보이게 함
    del result_place['tag']

    # 가장 유사한 10개의 장소를 반환
    return result_place
```

추천 함수 생성
➔ 식당 코드를 입력하면
코사인 유사도에 의한
상위 10개의 유사한 식당
목록 제공

1. Content based filtering

<추천 결과>

In [17]: place_REC("Num=a_nam_0002") → 고른햇살 식당 코드

Out[17]:

Unnamed: 0	이름	코드	태그	score
364	364	김밥천국 정대후문점	Num=a_nam_0166 #먹어야산다#분식#정대후문#포장#혼밥	0.882339
438	438	안암동김밥	Num=a_nam_0139 #먹어야산다#분식#정대후문	0.763456
94	94	절대분식	Num=a_nam_0162 #먹어야산다#분식#한식#백반#정대후문#포장#혼밥	0.731105
445	445	안암동김밥	Num=a_nam_0354 #먹어야산다#분식#정대후문#포장#1인분	0.611106
426	426	엄마네	Num=a_nam_0310 #먹어야산다#분식#법대후문#포장#혼밥	0.580730
170	170	두끼떡볶이 고대점	Num=a_nam_0093 #먹어야산다#분식#참살이길#포장	0.551018
410	410	봉구스밥버거 고려대점	Num=a_nam_0127 #먹어야산다#분식#밥버거#정대후문#포장#혼밥#저렴	0.544376
40	40	동네	Num=a_nam_860 #먹어야산다#한식#정대후문#백반#혼밥#저렴#가성비	0.542599
16	16	동우설렁탕	Num=a_nam_0116 #먹어야산다#한식#설렁탕#정대후문#포장#혼밥#회식#가성비	0.522106
55	55	서울쌈냉면 고대점	Num=a_nam_0120 #먹어야산다#한식#냉면#정대후문#포장#혼밥#저렴#가성비	0.496470

2. Collaborative filtering methods

User-item interaction matrix

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

1. Memory based (Nearest neighbor based)

- ① User-based
- ② Item-based

2. Model based (Latent factor based)

2. Collaborative filtering methods

User-item interaction matrix

place_id	1	2	3	4	5	6	7	8	9	10	...	401	402	403	404	405	406	407	408	409	410
user_id																					
1	5.0	2.0	4.0	4.0	4.0	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	5.0	NaN	NaN	NaN	NaN	NaN	3.0	5.0	5.0	5.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	3.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	5.0	NaN	4.0	NaN	NaN	NaN	5.0	4.0	5.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...
2514	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2515	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2516	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2517	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2518	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2518 rows × 410 columns

2.1 Memory based collaborative filtering methods

User-based (사용자 기반)

	Item A	Item B	Item C	Item D
User 1	3	4	4	
User2	4	4	4	5

- User 1, 2의 Item A~C에 대한 평가가 비슷하므로 유사한 사용자라고 판단
- User 1에게 Item D를 추천

2.1 Memory based collaborative filtering methods

User-based 시 문제점

	Item A	Item B	Item C	Item D
User 3	3	1	2	4
User 4	5	5	5	4

- User 3은 평가가 짠 편
- User 4는 평가가 후한 편



평준화가 필요 !!

2.1 Memory based collaborative filtering methods

```
rating_data = pd.read_csv('ratings.csv')  
  
Ratings = rating_data[["user_id", "place_id", "평점"]]  
Ratings.head()
```

	user_id	place_id	평점
0	1	1	5.0
1	2	1	5.0
2	3	1	3.0
3	4	1	4.0
4	5	1	4.0

```
Mean = Ratings.groupby(by="user_id", as_index=False)['평점'].mean()  
Mean.head()
```

	user_id	평점
0	1	4.000000
1	2	4.565217
2	3	3.600000
3	4	4.000000
4	5	3.930556

User, place, 평점으로만 df를 만들고

User별 평균 평점을 구함

2.1 Memory based collaborative filtering methods

```
final = pd.pivot_table(Rating_avg, values='adg_평점',  
                        index='user_id', columns='place_id').fillna(0)  
final.head()
```

place_id	1	2	3	4	5	6	7	8	9	10	...
user_id											
1	1.000000	-2.000000	0.0	0.000000	0.0	1.0	0.000000	0.000000	0.000000	0.000000	...
2	0.434783	0.000000	0.0	0.000000	0.0	0.0	-1.565217	0.434783	0.434783	0.434783	...
3	-0.600000	0.400000	0.0	0.000000	0.0	0.0	0.000000	0.000000	-0.600000	0.000000	...
4	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	...
5	0.069444	1.069444	0.0	0.069444	0.0	0.0	0.000000	1.069444	0.069444	1.069444	...

User 간 코사인 유사도

평준화된 평점 테이블 생성
(NaN 은 0으로 채움)

```
b = cosine_similarity(final)  
np.fill_diagonal(b, 0)  
similarity_with_user = pd.DataFrame(b, index=final.index)  
similarity_with_user.columns=final.index  
similarity_with_user.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...
userId											
1	0.000000	0.057133	-0.197203	0.0	-0.099118	0.026868	0.0	-0.020516	0.023338	0.0	...
2	0.057133	0.000000	-0.020280	0.0	-0.013249	0.012174	0.0	-0.095733	-0.007200	0.0	...
3	-0.197203	-0.020280	0.000000	0.0	-0.018441	0.032976	0.0	0.027743	-0.035504	0.0	...
4	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	...
5	-0.099118	-0.013249	-0.018441	0.0	0.000000	0.034037	0.0	0.021939	0.031485	0.0	...

2.1 Memory based collaborative filtering methods

```
# top 30 neighbours for each user
sim_user_30_u = find_n_neighbours(similarity_with_user, 30)
sim_user_30_u.head()
```

	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top24	top25	top26	top27	top28	top29	top30
userId																					
1	357	1048	136	1595	306	1193	22	19	88	355	...	139	11	53	230	220	330	997	1210	158	114
2	87	143	1159	1016	2039	1333	394	175	851	885	...	281	113	36	1075	1003	116	981	410	228	2046
3	1099	1195	1287	245	227	360	81	382	142	1075	...	199	466	176	831	527	241	821	543	948	963
4	2518	836	843	842	841	840	839	838	837	835	...	865	856	863	862	861	860	859	858	857	855
5	1012	380	1014	172	145	171	258	135	22	1121	...	404	255	158	1061	1201	20	76	210	129	211

각 User와 유사한 상위 30명 user를 구함

2.1 Memory based collaborative filtering methods

<추천 결과>

```
user = int(input("Enter the user id to whom you want to recommend : "))
predicted_movies = User_item_score1(user)
print(" ")
print("The Recommendations for User Id : 5")
print(" ")
for i in predicted_movies:
    print(i)
```

Enter the user id to whom you want to recommend : 5

The Recommendations for User Id : 5

동우설렁탕
히포크라테스 스프
투고샐러드 고려대점
칠기마라탕
야순네 식당

특정 유저를 넣었을 때 similar user group에서
평점이 높은 식당들을 추천

2.1 Memory based collaborative filtering methods

Item-based (아이템 기반)

	User 1	User 2	User 3	User 4
Item A	5	4	4	
Item B	4	3	4	5

- Item A, B가 비슷한 평점 분포를 보이므로 유사한 아이템으로 판단
- User4에게 Item A를 추천

2.1 Memory based collaborative filtering methods

```
item_based_col.head(5)
```

place_id	1	2	3	4	5	6	7	8	9	10	...
place_id											
1	0.000000	-0.165682	-0.147370	0.019495	0.000000	0.072141	-0.271849	0.113666	-0.043021	-0.008340	...
2	-0.165682	0.000000	0.006377	0.006595	0.000000	-0.209432	0.013500	-0.004141	-0.027503	0.055496	...
3	-0.147370	0.006377	0.000000	0.007449	-0.074787	-0.021286	-0.001958	-0.163389	-0.118702	0.042446	...
4	0.019495	0.006595	0.007449	0.000000	0.000000	0.107699	-0.020434	0.152670	-0.021780	0.145747	...
5	0.000000	0.000000	-0.074787	0.000000	0.000000	-0.099937	0.000000	-0.117541	0.000000	-0.152737	...

5 rows × 410 columns

```
def get_recommendation(place_id):  
    return item_based_col[place_id].sort_values(ascending = False)[:4]
```

```
get_recommendation(32)
```

```
place_id  
348    0.512191  
206    0.465754  
283    0.417475  
261    0.352269
```

맛집에 대한 코사인 유사도 계산

<추천 결과>

	맛집	place_id
2316	부부 바지락 손칼국수	32
5733	학생회관 2층카페 Orgo	206
6283	돌냄비열우동	261
6480	신원오리	283
6798	선일해장국	348

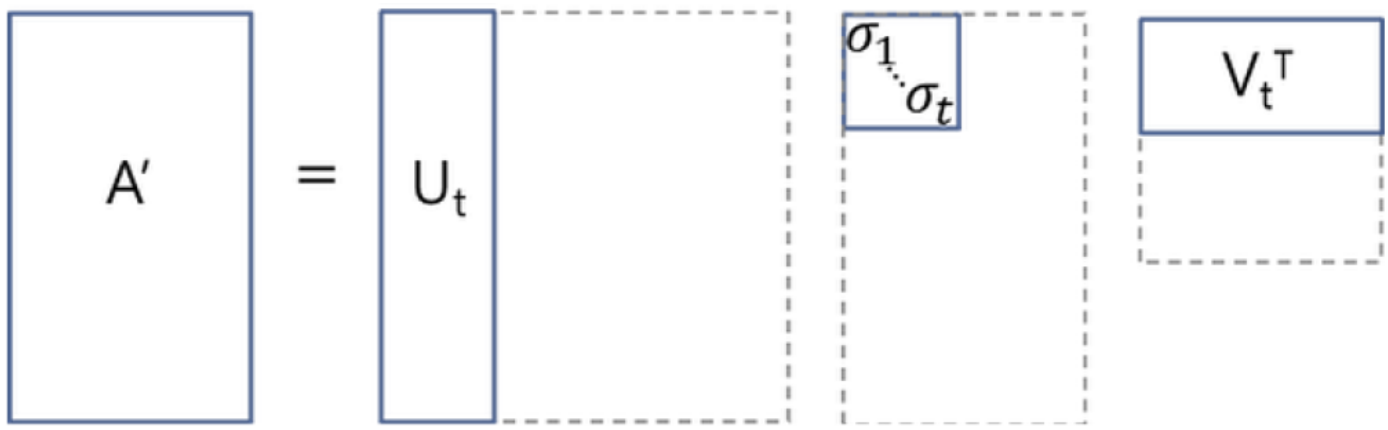
2.2 Model based collaborative filtering methods

SVD (특이값 분해)

The diagram illustrates the SVD decomposition of matrix A into matrices U , Σ , and V^T . Matrix A is shown as a gray rectangle with dimensions $m \times n$. A vertical gray strip on its left side is labeled A_k . Matrix U is a blue rectangle with dimensions $m \times m$. A vertical blue strip on its left side is labeled U_k . Matrix Σ is a green rectangle with dimensions $m \times n$. It contains a diagonal sequence of green squares, with the top-left square labeled Σ_k and several zeros (0) placed along the diagonal. Matrix V^T is an orange rectangle with dimensions $n \times n$. A horizontal orange strip at its top is labeled V_k^T . The decomposition is represented by the equation: $A = U \times \Sigma \times V^T$.

2.2 Model based collaborative filtering methods

Truncated SVD



The diagram illustrates the Truncated SVD equation: $A' = U_t \Sigma V_t^T$. It shows three matrices represented by rectangles. The first rectangle on the left is labeled A' . An equals sign follows. The second rectangle is labeled U_t . To its right is a dashed rectangle containing a smaller solid rectangle labeled $\sigma_1 \dots \sigma_t$, representing the singular values. To the right of this is another dashed rectangle containing a smaller solid rectangle labeled V_t^T . This visualizes the decomposition of matrix A into three components, where only the top t singular values and their corresponding vectors are used to form A' .

→ 데이터 정보를 상당히 압축했는데도 불구하고 행렬 A 에 근사하는 행렬 A' 를 만들 수 있다.

2.2 Model based collaborative filtering methods

```
user_item_rating.head()
```

place_id	1	2	3	4	5	6	7	8	9	10	...
user_id											
1	5.0	2.0	4.0	4.0	4.0	5.0	0.0	0.0	0.0	0.0	...
2	5.0	0.0	0.0	0.0	0.0	0.0	3.0	5.0	5.0	5.0	...
3	3.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	...
4	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	...
5	4.0	5.0	0.0	4.0	0.0	0.0	0.0	5.0	4.0	5.0	...

5 rows × 410 columns

```
item_user_rating = user_item_rating.values.T  
print(item_user_rating)  
item_user_rating.shape
```

```
[[5. 5. 3. ... 0. 0. 0.]  
 [2. 0. 4. ... 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. 0.]]
```

(410, 2518)

User-item interaction matrix

Item에 대한 latent factor를 보기 위해,
행렬 분해 SVD를 위한 준비

2.2 Model based collaborative filtering methods

```
# 사이킷런에서 제공하는 TruncatedSVD 이용
SVD = TruncatedSVD(n_components=8)
matrix = SVD.fit_transform(item_user_rating)
matrix.shape
```

(410, 8)

```
matrix[0:3]
```

```
array([[ 5.01637613, -4.94627798, -0.51166944, -1.9349214 , -0.89581175,
        -0.57087617,  0.70362022,  0.17755513],
       [ 1.89332587, -2.0849931 , -0.75253403, -0.30311568, -0.22752439,
        -1.34854757, -0.2458224 ,  1.11617476],
       [11.03102141, -9.64892943, -3.50323486, 13.66102959, -2.73255907,
        -6.66211511, -2.21219085, 12.53853448]])
```

```
# 피어슨 상관계수를 구함
corr = np.corrcoef(matrix)
corr.shape
```

(410, 410)

Truncated SVD 패키지를 이용해
Item에 대해 8개의 latent factor를 갖는
Matrix를 만들

2.2 Model based collaborative filtering methods

<추천 결과>

```
item_title = user_item_rating.columns
item_title_list = list(item_title)
useung_sigdang = item_title_list.index(4)
corr_useung_sigdang = corr[useung_sigdang]
```

```
result = list(item_title[(corr_useung_sigdang >= 0.90)][:5])
result
```

```
[4, 13, 26, 29, 54]
```

```
df[df['place_id'].isin(result)]
```

	맛집	place_id
149	우승식당	4
1261	미스터국밥	13
2116	평범식당	26
2179	고래돈까스	29
3009	미스터피자 고대점	54

place_id = 4 인 우승식당과
유사한 latent factor를 가진 식당

2.2 Model based collaborative filtering methods

```
# scipy에서 제공해주는 svd.  
# U 행렬, sigma 행렬, V^t를 반환.
```

```
U, sigma, Vt = svds(user_item_rating, k = 8)
```

```
# U, Sigma, Vt의 내적을 수행하면, 다시 원본 행렬로 복원이 된다.  
# 거기에 + 사용자 평균 rating을 적용한다.
```

```
svd_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + mean.reshape(-1, 1)
```

```
df_svd_preds = pd.DataFrame(svd_user_predicted_ratings, columns = user_item_rating.columns)  
df_svd_preds.head()
```

place_id	1	2	3	4	5	6	7	8	9	10	...
0	4.009325	3.974968	3.965199	4.007389	3.990576	4.099053	3.958572	4.048763	4.046488	4.009766	...
1	4.525400	4.504556	4.815070	4.550356	4.564260	4.662643	4.526465	4.456822	4.929368	4.549477	...
2	3.611219	3.580241	3.647690	3.611115	3.595172	3.648213	3.606360	3.630152	3.133526	3.596155	...
3	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	...
4	4.037926	4.476212	4.324791	4.114203	3.904292	4.236176	4.042873	4.957032	3.592105	4.676570	...

다른 라이브러리를 활용해
SVD한 행렬 각각을 직접 구해
A'를 구함

2.2 Model based collaborative filtering methods

<추천 결과>

```
predictions = recommend_movies(df_svd_preds, 5, df_movies, df_ratings, 10)
```

predictions

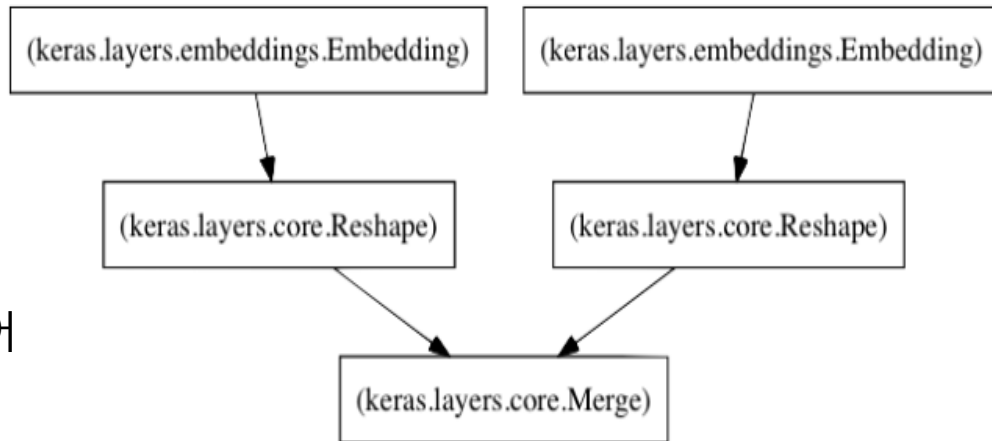
	place_id	맛집	Predictions
	4	12	동우설렁탕 4.880217
235	308	투고샐러드 고려대점	4.543993
193	266	하노이별	4.325065
0	3	언니네반점	4.324791
97	170	베나레스	4.240528
2	6	히포크라테스 스프	4.236176
36	109	제주고깃집	4.219225
272	345	소울키친 & 문순두부 & 콩시콩빠	4.171791
65	138	야순네 식당	4.129789
7	21	유자유	4.129411

A' 에서 5번 User가 가본 곳을 제외하고
평점이 높을 것으로 예상되는 맛집 추천

3. Deep Learning based methods

딥러닝 활용

- 원래 interaction matrix가
2개의 latent factor matrix의 곱으로
표현할 수 있다는 것이 핵심 아이디어



3. Deep Learning based methods

<학습 결과>

```
random_user['예상평점'] = random_user.apply(lambda x: predict_rating(5, x['place_id']), axis=1)
random_user.sort_values(by='평점', ascending=False).head(60)
```

	맛집	코드	place_id	닉네임	user_id	평점	예상평점
2317	부부 바지락 손칼국수	Num=a_nam_0314	32	검은여울	5	5.0	5.028696
3352	이공김밥	Num=a_nam_605	71	검은여울	5	5.0	4.957998
3023	밀플랜비 (Meal Plan B)	Num=a_nam_948	55	검은여울	5	5.0	5.008432
3151	서브웨이 고려대점	Num=a_nam_0045	60	검은여울	5	5.0	4.764935
2784	더멜팅	Num=a_nam_801	46	검은여울	5	5.0	5.004982
2762	나정순할매꾸꾸미	Num=a_nam_845	44	검은여울	5	5.0	5.004800
3202	안동반점	Num=a_nam_0370	64	검은여울	5	5.0	5.015846
3223	야마토텐동	Num=a_nam_817	67	검은여울	5	5.0	5.119627
2564	고고인디안쿠진 1호점	Num=a_nam_711	38	검은여울	5	5.0	5.031090
3328	오월키친	Num=a_nam_996	69	검은여울	5	5.0	5.088535

5번 user가 이미 가본 맛집에 대해
매긴 평점과 모델이 학습한 예상 평점이
상당히 유사함

3. Deep Learning based methods

User-based (사용자 기반)

The Recommendations for User Id : 5

동우설령탕
히포크라테스 스프
투고샐러드 고려대점
칠기마라탕
야순네 식당
자스민
절대분식
차이니웁
하노이별
등촌샤브칼국수 안암점

SVD (특이값 분해)

	맛집	Predictions
	동우설령탕	4.880217
	투고샐러드 고려대점	4.543993
	하노이별	4.325065
	언니네반점	4.324791
	베나레스	4.240528
	히포크라테스 스프	4.236176
	제주고깃집	4.219225
	소울키친 몸순두부 & 콩시콩뼈	4.171791
	야순네 식당	4.129789
	유자유	4.129411

딥러닝 활용

	맛집	예상평점
	동네	5.787228
	가츠시돈가스	5.025410
	토담	4.903670
	준호네부대찌개 고대형제점	4.895035
	제기돈	4.887592
	골목국수	4.886300
	서대문 곰장어	4.877253
	박가네 뼈다귀 해장국	4.815744
	시그니처 키친 (뚝닭)	4.745144
	특별식당	4.726214
	등촌샤브칼국수 안암점	4.700752

Next week

- 단순한 딥러닝 모델에 layer들 더 추가해서 깊게 모델 설계
- 발전시킨 딥러닝 모델 포함 각 방법의 장점을 종합한 **Hybrid methods** 개발
- 만든 모델의 **평가 방법**에 대한 고민
- (가능하다면) 웹 개발까지

Reference

<https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>

<https://lsjsj92.tistory.com/563?category=853217>

<https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/06/pcasvdlsa/>

<https://yamalab.tistory.com/92?category=747907>

<https://datascienceschool.net/view-notebook/3e7aadb88ed4f0d87a76f9ddc925d69/>

<https://nesoy.github.io/articles/2017-11/tf-idf>

https://medium.com/@Aaron_Kim/%EB%8B%A8%EC%96%B4-word-%EC%9D%98-%EC%A4%91%EC%9A%94%EB%8F%84%EB%A5%BC-%EC%B8%A1%EC%A0%95%ED%95%98%EB%8A%94-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-text-mining-tf-idf-rake-n-gram-86d9ef10873e

Thank you