SESSION #11

추천시스템

By TEAM1

목차 소개

Chapter 1

Chapter 1.1 추천이란?

추천 시스템이란?

사용자의 과거 데이터, 다른 데이터를 바탕으로
사용자에게 필요한 정보나 제품을 골라서 제시해 주는 시스템

- 협업필터링
- 내용기반필터링
- 지식기반필터링

기존의 추천 시스템

- 수작업을 이용한 추천
- 인기도를 활용한 추천 (데이터가 없는 초기 사용자의 경우 많이 사용)

Chapter 1.1 추천이란?

내용기반 필터링

텍스트 중에서 형태소를 추려내고,
이중에서도 핵심 키워드가 어떤 것인지를 분석하는 기술을 활용
이렇게 추려낸 키워드를 활용해서 소비자의 관심사항에 대한 분석이 가능.

지식기반 필터링

특정 분야에 대한 전문가의 도움을 받아서 그 분야에 대한 전체적인 지식구조를 만들어서 활용
일반적으로 체계도를 만들어 활용한다.

Chapter 1.2 협업필터링의 알고리즘

기본 아이디어

1. 취향이 비슷한 사람들의 집단이 존재한다는 가정
2. 추천의 대상이 되는 사람과 비슷한 집단을 찾는다
3. 이 집단의 사람들이 공통적으로 좋아하는 제품/서비스를 추천

고려할 주요 이슈

1. 사용자가 제품에 대한 추천을 꺼려하는 경우
2. Neighbor를 몇 명으로 할 것인가?
3. 사용자 취향의 유사성(Similarity)을 어떤 기준으로 계산 할 것인가?
4. 추천 대상 사용자가 좋아할 제품을 선정할 때 Neighbor인 모든 사람을 동등하게 평가할 것인가?

Chapter 1.2 협업필터링의 알고리즘

〈유저 별 영화 별점〉

User Movie	M1	M2	M3	M4	M5	Correlation With U1
U1	2	5	3			-
U2	4	4	3	5	1	0.19
U3	1	5	4		5	0.89
U4	3	5	3	2	5	0.94
U5	4	5	3	4		0.65
U3& U4 평균	2	5	3.5	2	5	

①유사성을 계산해 Neighbor 분류 ②U3 & U4가 좋게 평가한 영화 찾기 ③U1이 안본 M5를 추천

Chapter 1.2 협업필터링의 알고리즘

협업 필터링 (Collaborative Filtering)

1. 나와 가장 유사한 성향을 지닌 사람을 기반으로 그 사람이 들은 아이템을 추천 해주는 것 (User Based CF)

2. 내가 선호하는 아이템을 기반으로 가장 유사한 성향의 아이템을 추천 해주는 것 (Item Based CF)

Chapter 2

Chapter 2 추천 알고리즘 적용 사례

넷플릭스

아마존

Chapter 3

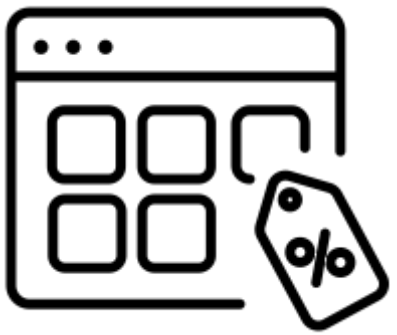
Chapter 3

3.1 UBCF, IBCF
3.2 데이터 전처리
3.3 유사도 계산
3.4 Neighbor 크기 결정
3.5 예상 평가값의 계산과 추천 리스트 작성
3.6 정확도 분석

Chapter 3.1User-based CF vs Item-based CF

추천 시스템을 이루는 가장 중요한 구성 요소들

Users



Items

Chapter 3.1User-based CF vs Item-based CF

유사도를 계산하는 기준이 사용자인가? 아이템인가?

사용자를 기준으로 하는 User-based CF

비슷한 사용자 그룹을 알아내고
이 그룹에 속한 사용자들이 공통적으로 좋게 평가한 아이템을 추천하는 방식
계산이 느리지만 정확도가 높은 편 (데이터가 좋지 않으면 정확도 폭망)

아이템을 기준으로 하는 Item-based CF

사용자들의 평가 패턴을 바탕으로
아이템 간의 유사도를 계산해서 사용자가 어떤 아이템을 구매하거나 좋다고 평가하면
그 아이템과 비슷한 아이템을 추천하는 방식
정확도는 떨어지지만 계산이 빠름

Chapter 3.1 User-based CF vs Item-based CF

User-based CF

데이터 크기가 적고 각 사용자에 대한 정확한 정보가 있는 경우

VS

Item-based CF

데이터가 크거나 각 사용자에 대한 정확한 정보가 없는 경우

Chapter 3.2 데이터 전처리

1. 데이터의 종류 살펴보기 (이진값 vs 연속값)

2. 사용자 x 아이템 매트릭스로 표시하기 (계산의 편의성)

3. 웹 로그 파일인 경우 노이즈를 걸러내고 정제된 데이터 사용하기
   (예를 들어 3분 이내 클릭 한 것은 한번으로 한다)

4. 평가한 아이템 수가 일정 기준을 넘는 사용자를 골라 내기
4. 구매한 사용자 수가 일정 수 미만인 아이템 제외하기
   (신뢰도가 떨어지는 것을 막기위하여)

Chapter 3.3 유사도(Similarity) 계산

유사한 성향이라는 것을 어떻게 가늠할까?

사용자 간 혹은 아이템 간 유사도를 계산하는 것이다.

평가 자료가 연속형인 경우
가장 이해하기 쉬운 유사도는 '상관계수' 이다.
하지만 협업필터링에서는 '코사인 유사도'를 더 많이 쓴다.

만일 데이터가 이진형인 경우
'타니모토 계수'를 쓴다.

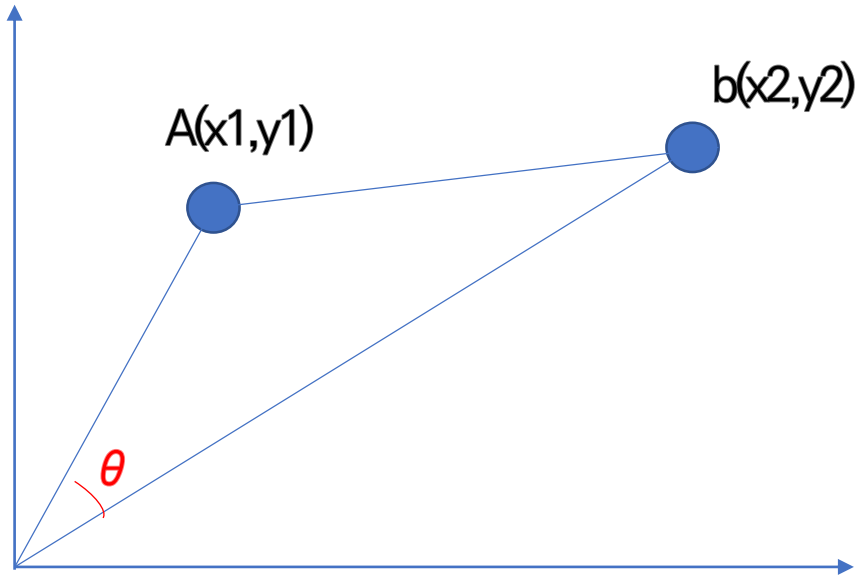
Chapter 3.3 유사도(Similarity) 계산

코사인 유사도

Ai = 사용자 A가 아이템 i 에 대해 평가 한 값
Bi = 사용자 B가 아이템 i 에 대해 평가 한 값

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

Chapter 3.3 유사도(Similarity) 계산



코사인 유사도는 각 아이템을
dimension으로 보고 사용자의 평가값을 좌표로 본다
그렇게 되면 각 사용자의 평가값을 벡터로 해서
두 사용자 간의 벡터의 각도를 구할 수 있다.

두 아이템 (x,y) 에 대해서 평가한
사용자 A, B를 표시한 것

코사인 유사도는 내적공간의 두 벡터간 각도의 코사인값을 이용하여 측정된 벡터간의 유사한 정도를 의미한다.
각도가 0°일 때의 코사인값은 1이며, 다른 모든 각도의 코사인값은 1보다 작다.
따라서 이 값은 벡터의 크기가 아닌 방향의 유사도를 판단하는 목적으로 사용되며, 두 벡터의 방향이 완전히 같을 경우 1,
90°의 각을 이룰 경우 0, 180°로 완전히 반대 방향인 경우 −1의 값을 갖는다. *-출처: 위키백과-*

Chapter 3.3 유사도(Similarity) 계산

타니모토 계수

$$\text{simil}(x,y) = \frac{c}{a + b - c}$$

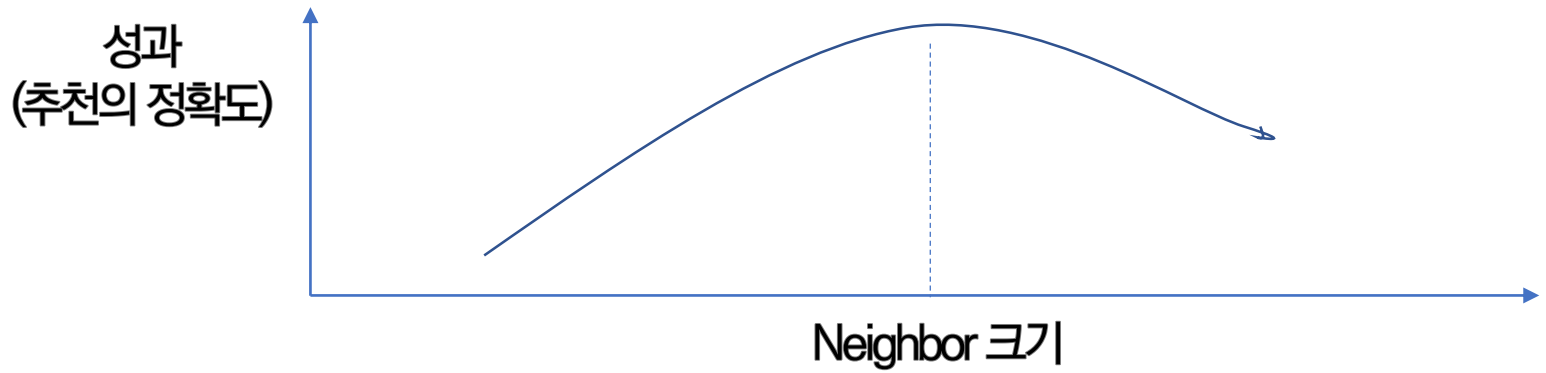
타니모토 계수는 최소 0 (완전 불일치) 최대 1 (완전 일치) 까지의 값을 가진다.
a = 사용자 x 가 1의 값을 갖는 구입한 아이템의 수 / 아이템 x에 대해서 1의 값을 갖는 구입한 사용자의 수
b = 사용자 y 가 1의 값을 갖는 구입한 아이템의 수 / 아이템 y에 대해서 1의 값을 갖는 구입한 사용자의 수
c = 사용자 x,y 가 공통적으로 1의 값을 갖는 아이템의 수

© 2017. SNU Growth Hackers all rights reserved

Chapter 3.4Neighbor 크기 결정

비슷하다고 생각되는 사용자 그룹의 크기를 정하는 것
(Item based CF 에서는 Neighbor개념이 필요 없다)

Neighbor 크기가 얼마일 때 최적이 되는지는 도메인에 따라 다르다
그래서 크기를 바꿔가면서 어느 크기가 정확도를 최고로 하는지
일일이 실험을 통해 찾아야 한다.



Chapter 3.4Neighbor 크기 결정

Neighbor를 정하는 기준

① Top-N-Neighbors – 미리 N을 설정해 놓고 정하기
② Thresholding – 유사도가 어느 수준 이상인 사용자(ex_ 상관계수 0.8이상)

실전에서는 Thresholding이 많이 안 쓰인다. 문턱값을 못 넘는게 많기 때문에

Chapter 3.5 예상 평가값 계산과 추천리스트 작성

UBCF의 경우, 가장 쉬운 방법? 평균

- Neighbor에 속한 사용자(이웃 사용자)의 각 아이템에 대한 평가의 평균을 낸다.
  (하지만 실제로는 이보다 더 정확한 예측을 위해 다양한 방법이 사용된다.)

하지만 추천 대상 사용자와 각 이웃 사용자 간의 유사도가 다른데,
당연히 유사도가 높은 사용자의 평가치에 더 높은 가중치를 주어야 할 것이다.

Chapter 3.5 예상 평가값 계산과 추천리스트 작성

그래서 가중 평균한 것을 추천 대상 사용자의 예상 평점으로 사용할 수 있다.

$$p_{ai} = \frac{\sum_{u=1}^{n} w_{au} \times r_{ui}}{\sum_{u=1}^{n} w_{au}}$$

a : 사용자 u : 이웃 사용자 n : 이웃 사용자 수

p_{ai} : 아이템 i에 대한 사용자 a 의 예상 평점

w_{au} : 사용자 a와 u의 유사도

r_{ui} : 아이템 i에 대한 사용자 u의 평점

Chapter 3.5 예상 평가값 계산과 추천리스트 작성

가중평균도 부족하다. 각 이웃 사용자의 평가 성향을 반영

하지만 가중평균 방법의 문제는
사용자에 따라 평가 경향이 다른데 이것이 반영되지 않는다는 것

예를 들어 일부 사용자의 평가 경향이 매우 박한데
우연히 이들만 특정 아이템에 대한 평가 값이 존재하는 경우에는
이 아이템에 대한 예상 평점이 낮게 나오는 경우가 발생할 수 있다.

그래서 각 이웃 사용자의 평가 경향을 예상 평점 계산에 적용하는 개선된 방법이 고안.

Chapter 3.5 예상 평가값 계산과 추천리스트 작성

이제 이 값을 가지고 높은 순서 대로 대상 사용자에게 추천을 해주면 된다

$$p_{ai} = \overline{r_a} + \frac{\sum_{u=1}^{n} w_{au} \times (r_{ui} - \overline{r_U})}{\sum_{u=1}^{n} w_{au}}$$

a : 사용자 u : 이웃 사용자 n : 이웃 사용자 수

p_{ai} : 아이템 i에 대한 사용자 a 의 예상 평점

w_{au} : 사용자 a와 u의 유사도

r_{ui} : 아이템 i에 대한 사용자 u의 평점

$\overline{r_a}$: 사용자 a의 전체 평점 평균

$\overline{r_U}$: 사용자 u의 전체 평점 평균

Chapter 3.5 예상 평가값 계산과 추천리스트 작성

IBCF의 경우

이 경우에는 각 아이템에 대한 예상 평점을 사용자 별로 계산할 필요가 없다.

어떤 사용자가 특정 아이템을 선택 했다면
해당 아이템과 유사도가 높은 아이템을 추천하면 된다.

서로 다른 사용자라도
같은 아이템을 선택하는 한 모두 같은 아이템을 추천 받게 된다.

Chapter 3.6 정확도 분석

결정 해야될 것이 많은 협업필터링
최적의 결과를 위해서는 어떤 세팅이 좋은 결과를 가져오는지 평가해야 한다.
평가 방법으로 시뮬레이션방법이 가장 널리 쓰인다.

시뮬레이션방법

데이터를 2개의 세트로나누고
Training set을 이용해서 Test set의 추천을 만들어 낸다.

우선 Test set이 없는 상태에서 Training set만을 가지고
Test의 예상 평점을 계산한다.
Test의 실제 평점을 가지고 있기 때문에 이렇게 계산된 예상 평점과 실제 평점과의 차이를 계산할 수 있으며 이것이 정확도라고 할 수 있다.

Chapter 3.6 정확도 분석

정확도 (Precision) = **사용자가 실제 선택한 아이템의 수 / 전체 추천된 아이템의 수**

재현율 (Recall) = **맞는 추천 아이템의 수 / 사용자가 선택한 전체 아이템의 수**

F측정치 = 2 x **정확도** x **재현율** / **정확도** + **재현율**

정확도와 재현율은 서로 반대의 관계가 있다.
재현율을 높이기 위해서 추천하는 아이템의 수를 늘리면 정확도가 낮아지고
정확도를 높이기 위해 추천하는 아이템의 수를 줄이면 재현율이 낮아지기 때문이다.
그래서 재현율과 정확도를 동시에 고려하는 F측정치를 많이 사용한다.

Chapter 3.6 정확도 분석

Actual \ Predicted	Negative	Positive
Negative	**A**	**B**
Positive	**C**	**D**

$$\text{Precision} = \frac{a+d}{a+b+c+d}$$

$$\text{True positive recommendation} = \frac{b}{b+d}$$

$$\text{Recall} = \frac{d}{c+d}$$

$$\text{False positive recommendation} = \frac{d}{b+d}$$

Chapter 4

Chapter 4 dataset

```
import math, random
from collections import defaultdict, Counter
from numpy import dot

users_interests = [
    ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
    ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
    ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
    ["R", "Python", "statistics", "regression", "probability"],
    ["machine learning", "regression", "decision trees", "libsvm"],
    ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
    ["statistics", "probability", "mathematics", "theory"],
    ["machine learning", "scikit-learn", "Mahout", "neural networks"],
    ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
    ["Hadoop", "Java", "MapReduce", "Big Data"],
    ["statistics", "R", "statsmodels"],
    ["C++", "deep learning", "artificial intelligence", "probability"],
    ["pandas", "R", "Python"],
    ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
    ["libsvm", "regression", "support vector machines"]
]
```

데이터의 형태:
리스트 안에 리스트

리스트 안에 있는 각각의 리스트는 유저의 Interests를 의미.

User ID: 0 의 Interests = users_interests[0]

Chapter 4.1 인기있는 것 추천

```
popular_interests = Counter(interest
                            for user_interests in users_interests
                            for interest in user_interests).most_common()
```

```
>>> popular_interests
[('Python', 4), ('R', 4), ('Big Data', 3), ('HBase', 3), ('Java', 3), ('stat
istics', 3), ('regression', 3), ('probability', 3), ('Hadoop', 2), ('Cassand
ra', 2), ('MongoDB', 2), ('Postgres', 2), ('scikit-learn', 2), ('statsmodels
', 2), ('pandas', 2), ('machine learning', 2), ('libsvm', 2), ('C++', 2), ('
neural networks', 2), ('deep learning', 2), ('artificial intelligence', 2),
('Spark', 1), ('Storm', 1), ('NoSQL', 1), ('scipy', 1), ('numpy', 1), ('deci
sion trees', 1), ('Haskell', 1), ('programming languages', 1), ('mathematics
', 1), ('theory', 1), ('Mahout', 1), ('MapReduce', 1), ('databases', 1), ('M
ySQL', 1), ('support vector machines', 1)]
```

Chapter 4.1 인기있는 것 추천

```
popular_interests = Counter(interest
                            for user_interests in users_interests
                            for interest in user_interests).most_common()
```

```
def most_popular_new_interests(user_interests, max_results=5):
    suggestions = [(interest, frequency)
                   for interest, frequency in popular_interests
                   if interest not in user_interests]
    return suggestions[:max_results]
```

Chapter 4.2 UBCF

```
def cosine_similarity(v, w):
    return dot(v, w) / math.sqrt(dot(v, v) * dot(w, w))
```

dot 그냥 numpy에서 불러왔음
앞으로 리스트 안의 리스트를 arr로 이해하면 더 말하기 편함.

Chapter 4.2 UBCF

```
unique_interests = sorted(list({ interest
                                 for user_interests in users_interests
                                 for interest in user_interests }))
```

Set 자료형.
users_interests 에서 중복된 것들을 하나로 합쳐서 묶어 두는 것.

sorted() : 대문자, 소문자, 오름차순

Chapter 4.2 UBCF

```
def make_user_interest_vector(user_interests):

    return [1 if interest in user_interests else 0
            for interest in unique_interests]
```

여기서 user_interest는 개별 사용자의 interests 행벡터를 말한다.
즉, users_interest[0] 같은 것을 의미
백터의 열은 unique_interests 의 index와 대응된다.

Chapter 4.2 UBCF

```
user_interest_matrix = list(map(make_user_interest_vector, users_interests))

user_similarities = [[cosine_similarity(interest_vector_i, interest_vector_j)
                      for interest_vector_j in user_interest_matrix]
                     for interest_vector_i in user_interest_matrix]
```

map함수는 첫번째 인자로 함수를 받고, 두번째 인자로 iterable를 받는다.
그것에 대해서 For문을 돌려주는 것.
Iterable를 반환한다.
즉, list로 싸주지 않으면 실체가 없다.
Raw data 를 백터로 변환하는 모습이다.

Chapter 4.2 UBCF

```
user_interest_matrix = list(map(make_user_interest_vector, users_interests))

user_similarities = [[cosine_similarity(interest_vector_i, interest_vector_j)
                      for interest_vector_j in user_interest_matrix]
                     for interest_vector_i in user_interest_matrix]
```

코사인 유사도 리스트를 구하는 것.
각 행마다 특정 아이디와 다른 아이디 사이의 유사도를 interest 기준으로 구한 것이다.

Chapter 4.2 코딩

```
def most_similar_users_to(user_id):
    pairs = [(other_user_id, similarity)
             for other_user_id, similarity in
                enumerate(user_similarities[user_id])
             if user_id != other_user_id and similarity > 0]

    return sorted(pairs,
                  key=lambda k: k[1],      # most similar
                  reverse=True)
```

User id(index)를 인풋으로 받는다. user_similarities 는 다른 사람과의 유사도가 들어 있는 행벡터이다. 여기서 그 idx와 원소를 받아서 튜플의 형태로 리스트축약해서 넣는다.
단, 이 때 user_id 와 other_user_id 가 다르고, 유사도가 0보다 큰 경우만 넣는다.

Chapter 4.2 UBCF

```
def most_similar_users_to(user_id):
    pairs = [(other_user_id, similarity)
             for other_user_id, similarity in
                enumerate(user_similarities[user_id])
             if user_id != other_user_id and similarity > 0]

    return sorted(pairs,
                  key=lambda k: k[1],      # most similar
                  reverse=True)
```

그리고 이 리스트를 similarity를 기준으로 내림차순으로 sorting한다.

Chapter 4.2 UBCF

```
def user_based_suggestions(user_id, include_current_interests=False):

    suggestions = defaultdict(float)
    for other_user_id, similarity in most_similar_users_to(user_id):
        for interest in users_interests[other_user_id]:
            suggestions[interest] += similarity

    suggestions = sorted(suggestions.items(),
                         key=lambda k: k[1],
                         reverse=True)
```

유저 아이디를 받으면 비슷한 관심도를 가진 유저 기반으로 interest를 추천해주는 함수 관심도 유사도가 높은 차례로 id와 유사도가 나온다. 이 사람의 아이디를 가져와서 이 사람의 interests를 for 문으로 각각 가져온다. 그리고 suggestions에 interests를 key 로해서 유사도를 더해준다. 이를 similiarity를 기준으로 내림차순 sorting한다.

Chapter 4.2 UBCF

```
if include_current_interests:
    return suggestions
else:
    return [(suggestion, weight)
            for suggestion, weight in suggestions
            if suggestion not in users_interests[user_id]]
```

함수인자의 기본값을 기본으로 설정해놓은 것이다.
만약 인자값이 주어지지 않으면 자동으로 기본값으로 세팅된다.
만약 false라면 ,
천하려는 interest가 유저의 interest에 없을 경우에만 리스트에 저장한다.

Chapter 4.3 IBCF

```
interest_user_matrix = [[user_interest_vector[j]
                         for user_interest_vector in user_interest_matrix]
                        for j, _ in enumerate(unique_interests)]

interest_similarities = [[cosine_similarity(user_vector_i, user_vector_j)
                          for user_vector_j in interest_user_matrix]
                         for user_vector_i in interest_user_matrix]
```

user_interest_vector 는 리스트에 각 interest(0,1)상태가 들어가 있는 것이다.
이것은 unique_interests에서 받아온 인덱스와 일치한다.
즉, user_interest_vector[inx]를 통해서 그 유저의 특정 interest의 존재 유무를 알 수 있는 것.
Interest_user_matrix는 행에는 각 interest가 대응되고, 열에는 유저가 interest 를 가지고 있는지 여부가 대응된다.

Chapter 4.3 IBCF

```
interest_user_matrix = [[user_interest_vector[j]
                         for user_interest_vector in user_interest_matrix]
                        for j, _ in enumerate(unique_interests)]

interest_similarities = [[cosine_similarity(user_vector_i, user_vector_j)
                          for user_vector_j in interest_user_matrix]
                         for user_vector_i in interest_user_matrix]
```

유사도를 구하는 것.
즉, 이번에는 각 interest에 대해서 각 유저별 관심 여부를 통해서 다른 interest와의 유사도를 구한 것이다.

Chapter 4.3 IBCF

```
def most_similar_interests_to(interest_id):
    similarities = interest_similarities[interest_id]
    pairs = [(unique_interests[other_interest_id], similarity)
             for other_interest_id, similarity in enumerate(similarities)
             if interest_id != other_interest_id and similarity > 0]
    return sorted(pairs,
                  key=lambda k: k[1],
                  reverse=True)
```

Interest_id(index)를 인풋으로 받는다. 앞이랑 구조가 사실 같은 것이다.
similarities는 인풋으로 받은 interest가 다른 interest와 얼마나 비슷한지에 대한 정보를 담고 있다. Enumerate에서 interest의 idx가 나오고 이 indx를 unique_interests에 넣으면 interest가 나온다. 그리고 유사도를 같이 튜플로 묶는다.

Chapter 4.3 IBCF

```
def most_similar_interests_to(interest_id):
    similarities = interest_similarities[interest_id]
    pairs = [(unique_interests[other_interest_id], similarity)
             for other_interest_id, similarity in enumerate(similarities)
             if interest_id != other_interest_id and similarity > 0]
    return sorted(pairs,
                  key=lambda k: k[1],
                  reverse=True)
```

같은 interest 제외, similarity〉0.
그리고 이 리스트를 similarity를 기준으로 내림차순으로 sorting한다.
(Interest , similarity)의 리스트

Chapter 4.3 IBCF

```
def item_based_suggestions(user_id, include_current_interests=False):
    suggestions = defaultdict(float)
    user_interest_vector = user_interest_matrix[user_id]
    for interest_id, is_interested in enumerate(user_interest_vector):
        if is_interested == 1:
            similar_interests = most_similar_interests_to(interest_id)
            for interest, similarity in similar_interests:
                suggestions[interest] += similarity
```

User id 를 인풋으로 받아, 이 사람의 interest여부와 그 indx를 for 문으로 각각 가져온다.

Chapter 4.3 IBCF

```
def item_based_suggestions(user_id, include_current_interests=False):
    suggestions = defaultdict(float)
    user_interest_vector = user_interest_matrix[user_id]
    for interest_id, is_interested in enumerate(user_interest_vector):
        if is_interested == 1:
            similar_interests = most_similar_interests_to(interest_id)
            for interest, similarity in similar_interests:
                suggestions[interest] += similarity
```

Most_similar_interests_to(): 유사도가 높은 차례로 interest와 유사도가 나온다. 만약, interest가 존재하면, 이 interest와의 유사도로 소팅되어있는 interest의 리스트를 most_similar_interest_to()를 통해 구한다. 그리고 suggestions에 interests를 key로해서 유사도를 더해준다. 이를 similiarity를 기준으로 내림차순 sorting한다.

Chapter 4.3 IBCF

```
if include_current_interests:
    return suggestions
else:
    return [(suggestion, weight)
            for suggestion, weight in suggestions
            if suggestion not in users_interests[user_id]]
```

함수인자의 기본값을 기본으로 설정해놓은 것이다.
만약 인자값이 주어지지 않으면 자동으로 기본값으로 세팅된다.
만약 false라면 ,
천하려는 interest가 유저의 interest에 없을 경우에만 리스트에 저장한다.

Thank you