Ch.22 Recommender Systems

22. Recommender Systems

- 오 자연이여, 자연이여, 왜 그리도 정직하지 못하여, 이 거짓된 추천서들과 함께 사람들을 세상에 보내려 하는가! - 헨리 필딩

추천 시스템

- 넷플릭스는 당신이 보고 싶어할 만한 영화를 추천합니다.
- 아마존은 당신이 사고 싶어할 만한 제품을 추천합니다.
- 트위터는 당신이 팔로우하고 싶어할 만한 사용자를 추천합니다.

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

- We'll look at the data set of users_interests:

```
 users_interests = [
 ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],  ["NoSQL", "MongoDB", "
 ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],  ["R", "Python", "statis
 ["machine learning", "regression", "decision trees", "libsvm"],
 ["Python", "R", "Java", "C++", "Haskell", "programming languages"],  ["statistics", "probabilit
 ["machine learning", "scikit-learn", "Mahout", "neural networks"],  ["neural networks", "deep l
 ["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"]
]
```

- Librarian suggested books that were relevant to your interests or similar to books you liked.

Recommending What's Popular

```
# 사용자들의 모든 관심사를 리스트로 평탄화한 후, 각 관심사의 빈도를 계산합니다.
popular_interests = Counter(
    interest
    for user_interests in users_interests  # 각 사용자의 관심사 리스트를 순회
    for interest in user_interests         # 각 관심사 리스트 내의 관심사를 순회
).most_common()  # 빈도수가 높은 순서대로 정렬된 리스트를 반환
```

```
popular_interests[:5]
```

```
[('Python', 4), ('R', 4), ('Big Data', 3), ('HBase', 3), ('Java', 3)]
```

```
def most_popular_new_interests(user_interests, max_results=5):
    # 사용자 관심사에 없는 인기 관심사를 추천 목록에 추가합니다.
    suggestions = [
        (interest, frequency)
        for interest, frequency in popular_interests  # 인기 관심사 리스트를 순회
        if interest not in user_interests             # 사용자 관심사에 없는 관심사만 선택
```

```
    ]

    # 추천 목록에서 최대 max_results 개의 항목을 반환합니다.
    return suggestions[:max_results]
```

```
# 두 번째 사용자의 관심사 리스트를 가져와서 가장 인기 있는 새로운 관심사 5개를 추천합니다.
most_popular_new_interests(users_interests[1], 5)
```

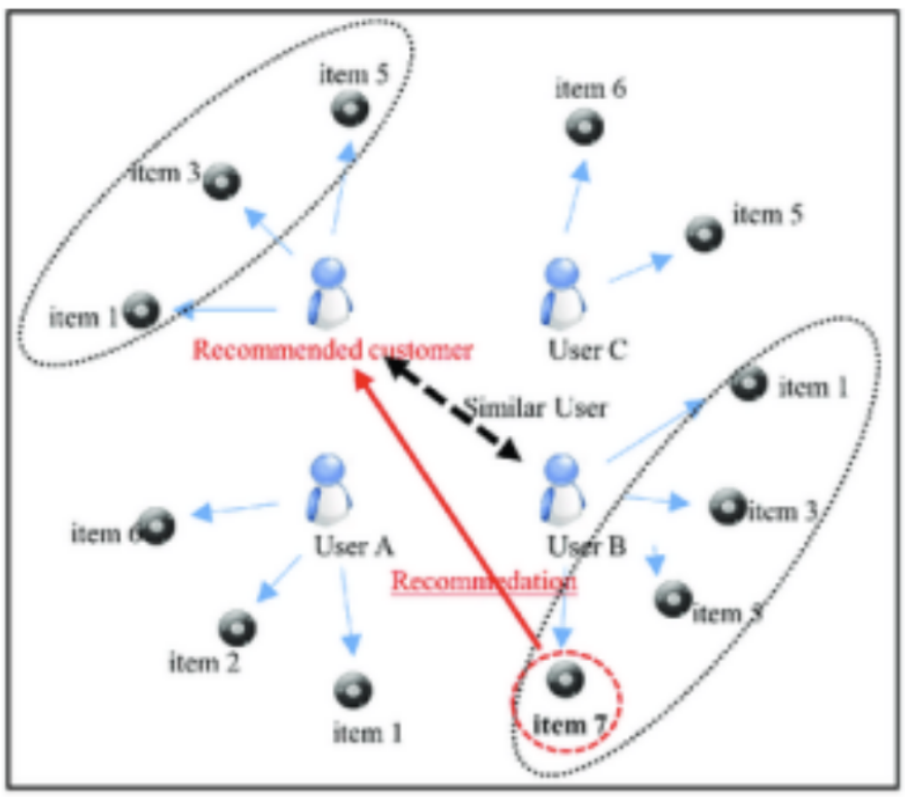
```
[('Python', 4), ('R', 4), ('Big Data', 3), ('Java', 3), ('statistics', 3)]
```

```
# 네 번째 사용자의 관심사 리스트를 가져와서 가장 인기 있는 새로운 관심사 5개를 추천합니다.
most_popular_new_interests(users_interests[3], 5)
```

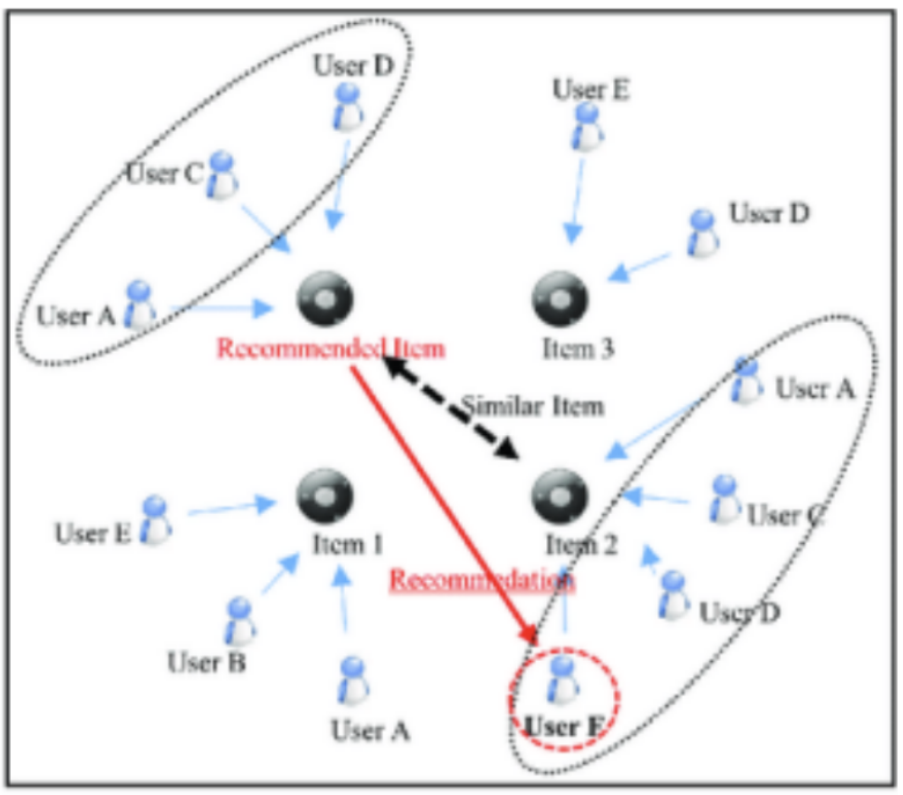
```
[('Big Data', 3), ('HBase', 3), ('Java', 3), ('Hadoop', 2), ('Cassandra', 2)]
```

Collaborative filtering algorithm (협업 필터링 알고리즘)

- (A) 사용자 기반 협업 필터링(User-based collaborative filtering)
- (B) 아이템 기반 협업 필터링(Item-based collaborative filtering)



(A)



(B)

사용자 기반 협업 필터링(User-Based Collaborative Filtering)

- 사용자 기반 협업 필터링은 특정 사용자의 관심사를 고려하기 위해 그 사용자와 비슷한 관심사를 가진 다른 사용자들을 찾아내고, 그 사용자들이 관심을 가지고 있는 항목들을 추천하는 방법입니다.

코사인 유사도(Cosine Similarity)

- 코사인 유사도는 두 사용자가 얼마나 비슷한지를 측정하는 방법 중 하나입니다. 두 사용자 벡터 간의 코사인 각도를 계산하여 유사도를 정의합니다. 이외에도 상관관계(correlation)도 자주 사용됩니다. 코사인 유사도는 다음과 같이 계산됩니다.

```
def cosine_similarity(v, w):
    """코사인 유사도를 계산합니다."""
```

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

```
cosine_similarity([0,0,1,1],[0,0,1,0])
```

```
 0.7071067811865475
```

- first, set comprehension

```
# 고유한 관심사 추출 및 정렬
unique_interests = sorted(list({ interest
                                 for user_interests in users_interests
                                 for interest in user_interests }))
```

Binary vector for user

```
def make_user_interest_vector(user_interests):
    """주어진 관심사 목록에 대해 고유한 관심사 목록 기반의 벡터를 생성합니다."""
    return [1 if interest in user_interests else 0
            for interest in unique_interests]

# 사용자 관심사 벡터 매트릭스 생성
user_interest_matrix = list(map(make_user_interest_vector, users_interests))
```

```
 [(9, 0.5669467095138409),
 (1, 0.3380617018914066),
 (8, 0.1889822365046136),
 (13, 0.1690308509457033),
 (5, 0.1543033499620919)]
```

- user_similaries[i][j]는 사용자 i와 j 사이의 코사인 유사성을 제공합니다.
- user_simililities[i]는 다른 모든 사용자에 대한 user i의 유사성 벡터입니다.

```
# 사용자 유사도 행렬 계산
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]
```

Find most similar users

```
def most_similar_users_to(user_id):
    pairs = [(other_user_id, similarity)  # 유사도 페어를 찾습니다
             for other_user_id, similarity in enumerate(user_similarities[user_id])  # 다른 사용자
             if user_id != other_user_id and similarity > 0]  # 본인 제외, 유사도 0 초과인 사용자만 포

    return sorted(pairs,  # 유사도 페어를 정렬합니다
                  key=lambda pair: pair[1],  # 유사도를 기준으로
                  reverse=True)  # 내림차순으로 정렬하여 가장 유사한 사용자부터 나열
```

- **`enumerate(user_similarities[user_id])`를 사용하여 해당 사용자의 유사도 행렬을 열거합니다.**
- **`user_id != other_user_id` 조건을 통해 본인과의 유사도는 제외하고, `similarity > 0` 조건을 통해 유사도가 0보다 큰 사용자만 포함합니다.**

```
most_similar_users_to(0)
```

Use-based suggestions

```
def user_based_suggestions(user_id, include_current_interests=False):
    # 유사도 합산을 위한 딕셔너리 초기화
    suggestions = defaultdict(float)

    # 가장 유사한 사용자들을 가져옴
    for other_user_id, similarity in most_similar_users_to(user_id):
        # 유사한 사용자의 관심사에 유사도를 누적
        for interest in users_interests[other_user_id]:
            suggestions[interest] += similarity

    # 유사도에 따라 관심사 정렬
    suggestions = sorted(suggestions.items(),
                         key=lambda pair: pair[1],
                         reverse=True)

    # 현재 사용자의 관심사를 포함할지 여부에 따라 반환값 결정
    if include_current_interests:
        return suggestions
    else:
        return [(suggestion, weight)
                for suggestion, weight in suggestions
                if suggestion not in users_interests[user_id]]
```

- **유사도 합산:** `defaultdict(float)` **를 사용하여 관심사별 유사도를 누적합니다.**
- `most_similar_users_to(user_id)` **를 호출하여** `user_id` **와 유사한 사용자들을 찾고, 그 사용자들의 관심사에 유사도를 추가합니다.**
- **관심사 정렬:** `suggestions.items()` **를 유사도 값에 따라 내림차순으로 정렬합니다.**
- **현재 관심사 포함 여부 결정:** `include_current_interests` **가** `True` **인 경우 모든 관심사를 반환합니다.**
- `include_current_interests` **가** `False` **인 경우 현재 사용자의 관심사를 제외한 관심사만 반환합니다.**

```
# user0: ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandr a"],
user_based_suggestions(0)
```

```
[('MapReduce', 0.5669467095138409),
 ('MongoDB', 0.50709255283711),
 ('Postgres', 0.50709255283711),
 ('NoSQL', 0.3380617018914066),
 ('neural networks', 0.1889822365046136),
 ('deep learning', 0.1889822365046136),
 ('artificial intelligence', 0.1889822365046136),
 ('databases', 0.1690308509457033),
 ('MySQL', 0.1690308509457033),
 ('Python', 0.1543033499620919),
 ('R', 0.1543033499620919),
 ('C++', 0.1543033499620919),
 ('Haskell', 0.1543033499620919),
 ('programming languages', 0.1543033499620919)]
```

물품의 수가 매우 많아지면 이 접근 방식은 제대로 작동하지 않습니다.

- 고차원 벡터 공간에서는 대부분의 벡터가 서로 매우 멀리 떨어져 있어서 (따라서 매우 다른 방향을 가리키게 됩니다).
- 즉, 관심사가 많을 경우 "가장 유사한 사용자"가 실제로는 전혀 유사하지 않을 수 있습니다.
- 저와 "가장 유사한" 쇼핑객이 누구든, 그는 아마도 저와 비슷하지 않을 것이며, 그의 구매는 거의 확실히 형편없는 추천을 만들어낼 것입니다.

아이템 기반 협업 필터링

- 대안적인 접근 방식은 관심사 간의 유사성을 직접 계산하는 것입니다.
- 그런 다음 각 사용자의 현재 관심사와 유사한 관심사를 집계하여 제안을 생성할 수 있습니다.

interest_user_matrix: Transpose user_interest_matrix

```
# interest_user_matrix 생성
interest_user_matrix = [[user_interest_vector[j]
                                              for user_interest_vector in user_interest_matrix
```

1. `user_interest_matrix`는 사용자-관심사 행렬을 나타냅니다.
2. `unique_interests`는 고유한 관심사 목록을 나타냅니다.
3. `interest_user_matrix`는 각 관심사에 대한 사용자 벡터를 생성합니다.

```
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]
```

```
def most_similar_interests_to(interest_id):
    # interest_id에 해당하는 관심사와 다른 관심사들 사이의 유사도 벡터를 가져옴
    similarities = interest_similarities[interest_id]

    # interest_id와 다른 모든 관심사에 대한 유사도 쌍을 생성
    pairs = [
        (unique_interests[other_interest_id], similarity)  # (관심사, 유사도) 쌍 생성
        for other_interest_id, similarity in enumerate(similarities)  # 모든 관심사에 대해 반복
        if interest_id != other_interest_id and similarity > 0  # 자기 자신 제외 및 유사도가 0보다 큰
    ]

    # 유사도 순으로 내림차순 정렬하여 반환
    return sorted(pairs, key=lambda pair: pair[1], reverse=True)
```

1. **유사도 벡터 추출:** `interest_similarities[interest_id]`를 통해 주어진 `interest_id`에 대한 유사도 벡터를 가져옵니다.
   a. 이 벡터는 해당 관심사와 다른 모든 관심사들 사이의 유사도를 나타냅니다.
2. **유사도 쌍 생성:**
   a. `enumerate(similarities)`를 사용하여 `similarities` 벡터의 각 요소에 대해 인덱스(`other_interest_id`)와 유사도(`similarity`)를 가져옵니다.
   b. `if interest_id != other_interest_id and similarity > 0` 조건을 통해 자기 자신과의 유사도(항상 1이므로 제외) 및 유사도가 0인 경우를 제외합니다.
   c. `unique_interests[other_interest_id]`를 사용하여 `other_interest_id`에 해당하는 관심사 이름을 가져와서 `(관심사, 유사도)` 쌍을 생성합니다.
3. **유사도 순 정렬:**

a. `sorted(pairs, key=lambda pair: pair[1], reverse=True)`를 사용하여 유사도 순으로 쌍을 내림차순으로 정렬합니다.이로 인해 가장 유사한 관심사가 처음에 오도록 정렬됩니다.

```
# interest 0 is Big Data
most_similar_interests_to(0)
```

```
('Hadoop', 0.8164965809277261),
 ('Java', 0.6666666666666666),
 ('MapReduce', 0.5773502691896258),
 ('Spark', 0.5773502691896258),
 ('Storm', 0.5773502691896258),
 ('Cassandra', 0.4082482904638631),
 ('artificial intelligence', 0.4082482904638631),
 ('deep learning', 0.4082482904638631),
 ('neural networks', 0.4082482904638631),
 ('HBase', 0.3333333333333333)]
```

```
def item_based_suggestions(user_id, include_current_interests=False):
    # 추천 항목을 저장할 사전, 기본값을 0.0으로 설정
    suggestions = defaultdict(float)

    # 사용자의 관심사 벡터를 가져옴
    user_interest_vector = user_interest_matrix[user_id]

    # 사용자의 각 관심사에 대해 반복
    for interest_id, is_interested in enumerate(user_interest_vector):
        if is_interested == 1:  # 사용자가 해당 관심사에 관심이 있는 경우
            # 해당 관심사와 유사한 관심사들을 가져옴
            similar_interests = most_similar_interests_to(interest_id)
            for interest, similarity in similar_interests:
                # 유사도를 기반으로 추천 점수를 누적
                suggestions[interest] += similarity

    # 추천 항목들을 유사도 순으로 내림차순 정렬
    suggestions = sorted(suggestions.items(), key=lambda pair: pair[1], reverse=True)

    if include_current_interests:
        # 현재 관심사를 포함하여 추천 리스트 반환
        return suggestions
    else:
        # 현재 관심사를 제외하고 추천 리스트 반환
        return [
            (suggestion, weight)
            for suggestion, weight in suggestions
            if suggestion not in users_interests[user_id]
        ]
```

1. **추천 항목 초기화:** `suggestions`를 `defaultdict(float)`으로 초기화하여, 추천 항목과 그에 대한 누적 유사도를 저장합니다.
2. **사용자의 관심사 벡터 가져오기:** `user_interest_vector = user_interest_matrix[user_id]`를 통해 주어진 `user_id`에 대한 관심사 벡터를 가져옵니다.
3. **관심사 벡터를 순회:**
    a. `for interest_id, is_interested in enumerate(user_interest_vector)`를 통해 사용자의 각 관심사에 대해 반복합니다.
    b. `if is_interested == 1` 조건을 통해 사용자가 해당 관심사에 관심이 있는 경우에만 처리합니다.
4. **유사한 관심사 찾기:**

a. `similar_interests = most_similar_interests_to(interest_id)`를 통해 해당 관심사와 유사한 관심사들의 리스트를 가져옵니다.
b. `for interest, similarity in similar_interests`를 통해 각 유사한 관심사와 그 유사도를 반복합니다.
c. `suggestions[interest] += similarity`를 통해 유사도를 누적하여 추천 점수를 계산합니다.

5. **추천 항목 정렬:** `suggestions = sorted(suggestions.items(), key=lambda pair: pair[1], reverse=True)`를 통해 추천 항목들을 유사도 순으로 내림차순 정렬합니다.
6. **결과 반환:**
   a. `if include_current_interests:` 조건을 통해 현재 관심사를 포함할지 여부를 결정합니다.
   b. `include_current_interests`가 `True`이면 정렬된 `suggestions` 리스트를 그대로 반환합니다.
   c. `include_current_interests`가 `False`이면 현재 관심사를 제외한 추천 리스트를 반환합니다.
   d. `users_interests[user_id]`를 통해 사용자의 현재 관심사를 확인하고, 이를 제외한 추천 항목들만 반환합니다.

```
# user 0: ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandr a"],
item_based_suggestions(0)
```

```
[('MapReduce', 1.861807319565799),
('MongoDB', 1.3164965809277263),
('Postgres', 1.3164965809277263),
('NoSQL', 1.2844570503761732),
('MySQL', 0.5773502691896258),
('databases', 0.5773502691896258),
('Haskell', 0.5773502691896258),
('programming languages', 0.5773502691896258),
('artificial intelligence', 0.4082482904638631),
('deep learning', 0.4082482904638631),
('neural networks', 0.4082482904638631),
('C++', 0.4082482904638631),
('Python', 0.2886751345948129),
('R', 0.2886751345948129)]
```