Big Data Analytics

Summer 2022

Number 05, Submission Deadline: June 12, 2023, 11:59 PM
Recommendation Systems

1. **Song Recommendations** (10 P)
   You are given the utility matrix below, it represents ratings of songs as given by five users of a music app. Please use it to answer the following questions:

   - How similar are the music tastes of users `user-1` and `user-2`?
   - How similar are the music tastes of users `user-1` and `user-3`?
   - Will user `user-5` like the song `song-6`?
   - Will user `user-5` like the song `song-1`?

   You are free to implement this using any of the solutions discussed in the lecture (except for UV decomposition, since that is covered by task 2). Note that the `artist` variable contains an association between the band and the songs, should your chosen approach require that data.

```
num_users = 5
num_items = 8

utility = pd.DataFrame(np.nan,
index=[f"user-{i}" for i in range(1, num_users+1)],
columns=[f"song-{j}" for j in range(1, num_items+1)])

artists = {
        "artist-1": ["song-1", "song-2", "song-3"],
        "artist-2": ["song-4"],
        "artist-3": ["song-5", "song-6"],
        "artist-4": ["song-7", "song-8"]
}

user_likes = """
user-1 song-1 5
user-1 song-4 4
user-1 song-5 1
```

```
user-1 song-6 1

user-2 song-2 1
user-2 song-6 5
user-2 song-7 4
user-2 song-8 2

user-3 song-3 2
user-3 song-4 5
user-3 song-6 2

user-4 song-1 2
user-4 song-5 5
user-4 song-2 2

user-5 song-7 1
user-5 song-2 5
user-5 song-3 3
user-5 song-4 5
"""
for user_id, song, rating in [line.split(" ") for line in
filter(lambda line: line.strip() != "",
user_likes.strip().split("\n"))]:
rating = float(rating)
utility.at[user_id, song] = rating

utility_original = utility.copy()

###
# alternatively as a numpy.ndarray:
utility_np = utility.to_numpy()
```

2. **UV Decomposition:** (10 P)

   (a) Perform incremental UV decomposition on the utility matrix given below. Pick a dimensionality d that seems sensible to you.

```
# this time we will use a randomly generated utility matrix
import random

num_users = 100
num_items = 300
# generate ratings for at least 15% of all songs but no more than
minmax_ratings = [int(num_items*0.15), int(num_items*0.75)]
rating_range = [1, 5]
```

```
# generate utility table
users = [f"user-{i}" for i in range(1, num_users+1)]
songs = [f"song-{j}" for j in range(1, num_items+1)]
utility = pd.DataFrame(np.nan,
index=users,
columns=songs)

possible_ratings = [r for r in range(rating_range[0], rating_range
num_possible_ratings = len(possible_ratings)

# human ratings are often skewed to the extreme choices (e.g. 1 st
# let's reflect this by generating rankings that have a similar a
rating_distribution = [np.max([0.1, np.abs(((i+0.5)-(num_possible
for i in range(num_possible_ratings)]
rating_distribution = rating_distribution / np.max(rating_distribu
rating_distribution = rating_distribution / np.sum(rating_distribu
print("possible ratings", possible_ratings)
print("distribution", rating_distribution, np.sum(rating_distribut

def generate_rating():
# unbiased version
# return np.random.randint(rating_range[0], rating_range[1]+1)
return np.random.choice(possible_ratings, 1, p=rating_distribution

# generate random ratings
for user in users:
num_ratings = np.random.randint(minmax_ratings[0], minmax_ratings[
rated_songs = random.sample(songs, num_ratings)
ratings = [generate_rating() for _ in range(len(rated_songs))]

#print(user_id, rating, rated_songs, ratings)
for song, rating in zip(rated_songs, ratings):
utility.at[user, song] = rating

# the following can be used to check the rating distribution:
allratings = np.array(utility.to_numpy().tolist())
allratings = allratings[~np.isnan(allratings)]
for rating, freq in zip(*np.unique(allratings, return_counts=True)
print("rating: %s, freq: %s" % (rating, freq))
```

(b) Explain briefly how the results help you with making recommendations to your users.