HW7 - Recommendation Systems

Logan Bartels

November 30, 2020

Note

This assignment was completed entirely in a Google Colab notebook. The notebook is a copy of the notebook from week-11. Any additions I made to the notebook (with the exception of test code for the original notebook) are under section "HW7;" however, the entirety of the notebook was used to complete the assignment. Because modifications to existing code were as minimal as commenting out a line of code, I opted to leave the functions where they are in the notebook. Here is the link to the notebook: `https://colab.research.google.com/drive/1Y-fqLOcpgsYkHiqWCwNbPq_1YxYq3f7l?usp=sharing`

Q1

Answer

```
def loadMovieLens():
  # Get movie titles
    movies = {}
    for line in open('u.item', encoding="ISO-8859-1"):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open('u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movies[movieid]] = float(rating)
    return prefs
```

Listing 1: Function to load data from u.item and u.data files.

```
prefs = loadMovieLens()
for line in open("u.user"):
  user = line.split('|')[0:4]
  if user[1] == '23' and user[2] == 'M' and user[3] == 'student':
    print(user)
  else:
    continue
```

Listing 2: Python code used to determine users similar to me.

```
['33', '23', 'M', 'student']
['37', '23', 'M', 'student']
['66', '23', 'M', 'student']
['135', '23', 'M', 'student']
['391', '23', 'M', 'student']
['408', '23', 'M', 'student']
['706', '23', 'M', 'student']
['838', '23', 'M', 'student']
```

Listing 3: Output of users similar to me based on age gender and occupation

```
sorted_movies = sorted(prefs['37'].items(), key=lambda x: x[1], reverse
    =True)

for movie in sorted_movies:
  print(movie[0], movie[1])
```

Listing 4: Python code for printing out a sorted list of a user's rated movies along with their ratings

Note: The movies in the following list are listed in descending order according to the output produced by listing 4. The output produced by listing 4 is omitted from this report for brevity.

Final 3 users with their top and bottom 3 movies:

1. User 37, **My substitute user** (substitute you).
   (a) Top 3 Movies:
      i. Batman (1989)
      ii. Pulp Fiction (1994)
      iii. Top Gun (1986)
   (b) Bottom 3 Movies:
      i. Dragonheart (1996)
      ii. Independence Day (ID4) (1996)
      iii. Jurassic Park (1993)
2. User 66
   (a) Top 3 Movies:
      i. Return of the Jedi (1983)
      ii. Courage Under Fire (1996)
      iii. Ransom (1996)
   (b) Bottom 3 Movies:
      i. Excess Baggage (1997)
      ii. Muppet Treasure Island (1996)

iii. English Patient, The (1996)

3. User 838

   (a) Top 3 Movies:

      i. Princess Bride, The (1987)
      ii. Hunchback of Notre Dame, The (1996)
      iii. Return of the Jedi (1983)

   (b) Bottom 3 Movies:

      i. Air Force One (1997)
      ii. Mars Attacks! (1996)
      iii. Independence Day (ID4) (1996)

Discussion

The function shown in listing 1 is taken directly from the PCI recommendations.py file with modifications made to run in a Google Colab notebook. The function loads in the first two indexes of every item in "u.item," the movie id and its title. It then places them into a dictionary with the id as the key and the title as the value. Then it reads in user ratings from the "u.data" file, storing the user, movie, and rating in a nested dictionary called "prefs." "Prefs" is then returned at the end of the function. One significant modification made is on line 4, and that is the addition of the "encoding=ISO-8859-1" parameter to the "open" function. Without the "encoding=ISO-8859-1" parameter, I was getting a UnicodeDecodeError.

In order to find users similar to me according to age, gender, and occupation, I wrote the code in listing 2. It starts by calling the "loadMovieLens" function, and storing the returned prefs in a variable called "prefs" for later use. Then it opens the "u.user" file and reads the first three indexes of each line. I did this because the first three fields, age, gender, and occupation are what I needed. Then a check is performed to see if the user's age, gender, and occupation match my parameters. In this case, I was looking for 23 year-old male students. Listing 3 shows the output of listing 2. Since I got eight results of users similar to me, I needed to narrow them down to my choice of three.

The code in listing 4 is part of my process of narrowing my selection of three similar users. The code starts by sorting the dictionary of a single user's movies and their ratings by the ratings in descending order. The resulting list of tuples is stored in the "sorted_movies" variable. A for-loop is used to loop through the list and print the movies with their ratings. The result is a list of moves sorted in descending order by their ratings. As you can see on the first line, user's ratings that will be sorted and printed are user 37's. I changed the "37" to another similar user's id number after each iteration according to the ids listed in listing 3; I started with user 33, then 37, etc.

After examining each user's ratings for movies my final three choices of similar users are listed in the list above this subsection. Their top and bottom three movies are also listed with their ids.

My selection for the user that will act as my substitute is user 37. I did not pick him because of his top three movies. I picked him because the gave other movies like Star Wars (1977), The Terminator (1984), and The Crow (1994) a 5.0 rating. One outlier I must note for user 37 is they gave Independence Day (1996) a 2.0 rating. I liked that movie much more that he did. I would probably give it a 4.0 rating.

Q2

Answer

```
def topMatches(
    prefs,
    person,
    n=5,
    similarity=sim_pearson,
):
    '''
    Returns the best matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''

    scores = [(similarity(prefs, person, other), other) for other in
   prefs
              if other != person]
    scores.sort()
    scores.reverse()
    return scores[0:n]

topMatches(prefs, "37", n=5)
```

Listing 5: topMatches function copy-pasted from the week-11 Colab notebook with the function call at the bottom.

```
[(1.0000000000000027, '93'),
 (1.0, '937'),
 (1.0, '859'),
 (1.0, '791'),
 (1.0, '754')]
```

Listing 6: Output of top 5 users after call to topMatches function using the prefs dictionary and user 37.

```
[(-1.000000000000004, '491'),
 (-1.0, '185'),
 (-1.0, '228'),
 (-1.0, '469'),
```

```
5  (-1.0, '578')]
```

Listing 7: Output of bottom 5 users after call to topMatches function using the prefs dictionary and user 37.

Discussion

The top five most correlated users (shown in listing 6) to my substitute user, user 37, were determined using the "topMatches" function shown in listing 5. The function call was made in a lone cell in the Colab notebook, but was placed at the bottom of listing 5 for brevity. The function call passes in the "prefs" dictionary that was defined on line 1 in listing 2, the id of my substitute user (37) and "n=5" (for the five most correlated users).

In order to determine the five least correlated users, I simply commented out line 15 of listing 5, which sorts the returned list in descending order, and ran the function call again. This lead to the output shown in listing 7.

Q3

Answer

```
1  def getRecommendations(prefs, person, similarity=sim_pearson):
2      '''
3      Gets recommendations for a person by using a weighted average
4      of every other user's rankings
5      '''
6
7      totals = {}
8      simSums = {}
9      for other in prefs:
10     # Don't compare me to myself
11         if other == person:
12             continue
13         sim = similarity(prefs, person, other)
14         # Ignore scores of zero or lower
15         if sim <= 0:
16             continue
17         for item in prefs[other]:
18             # Only score movies I haven't seen yet
19             if item not in prefs[person] or prefs[person][item] == 0:
20                 # Similarity * Score
21                 totals.setdefault(item, 0)
22                 # The final score is calculated by multiplying each
   item by the
```

```
23                  #   similarity and adding these products together
24                  totals[item] += prefs[other][item] * sim
25                  # Sum of similarities
26                  simSums.setdefault(item, 0)
27                  simSums[item] += sim
28      # Create the normalized list
29      rankings = [(total / simSums[item], item) for (item, total) in
30                  totals.items()]
31      # Return the sorted list
32      rankings.sort()
33      rankings.reverse()
34      return rankings
```

Listing 8: getRecommendations function copy-pasted from the week-11 Colab notebook.

```
1 rankings = getRecommendations(prefs, "37")
2 for movies in rankings:
3   print(movies)
```

Listing 9: Python code to compute and print ratings for films that my substitute user has not seen.

Top and bottom five movie recommendations for user 37, my substitute user:

1. Top 5 recommendations:
   (a) They Made Me a Criminal (1939)
   (b) Someone Else's America (1995)
   (c) Santa With Muscles (1996)
   (d) Prefontaine (1997)
   (e) Marlene Dietrich: Shadow and Light (1996)
2. Bottom 5 recommendations:
   (a) Amityville Curse, The (1990)
   (b) Amityville 3-D (1983)
   (c) Amityville 1992: It's About Time (1992)
   (d) American Strays (1996)
   (e) 3 Ninjas: High Noon at Mega Mountain (1998)

Discussion

Something I must note is the output of listing 9 is omitted from this report for brevity. Determining the top and bottom five movies my user should see was simple. I used the "getRecommendations" function shown in listing 8. That function computes ratings for movies the specified user

has not seen, then puts them into a list. It then sorts that list in descending order, with the movies at the top of the list (the ones with less than or equal to a 5.0 rating) being the ones the user would like the most. The ones at the bottom of the list (the ones with less than or equal to a 1.0 rating) being the ones the user would like the least.

I then made a call to the "getRecommendations" function in listing 9, passing in the "prefs" dictionary and user 37. I stored the returned list into a variable called "rankings." Then the entire list is printed out. I simply took the five movies at the top and bottom of the list and put them in the list above this subsection.

Q4

Answer

```
1  def transformPrefs(prefs):
2      '''
3      Transform the recommendations into a mapping where persons are
       described
4      with interest scores for a given title e.g. {title: person} instead
       of
5      {person: title}.
6      '''
7
8      result = {}
9      for person in prefs:
10         for item in prefs[person]:
11             result.setdefault(item, {})
12             # Flip item and person
13             result[item][person] = prefs[person][item]
14     return result
15
16 similarMovies = transformPrefs(prefs)
17
18 for x, y in similarMovies.items():
19   print(x ,y)
20
21 topMatches(similarMovies, "Ace Ventura: Pet Detective (1994)")
```

Listing 10: transformPrefs function copy-pasted from the week-11 Colab notebook along with cells of code used to determine similar movies to my favorite and least favorite movies.

```
1 [(1.0, 'Wonderful, Horrible Life of Leni Riefenstahl, The (1993)'),
2  (1.0, 'When the Cats Away (Chacun cherche son chat) (1996)'),
3  (1.0, 'Welcome To Sarajevo (1997)'),
4  (1.0, 'U.S. Marshalls (1998)'),
```

```
5  (1.0, 'Shiloh (1997)')]
```

Listing 11: Output of calling the topMatches function to get the 5 most correlated movies to Ace Ventura: Pet Detective.

```
1 [(-1.0000000000000007, 'Rent-a-Kid (1995)'),
2  (-1.0000000000000007, 'True Crime (1995)'),
3  (-1.0, '1-900 (1994)'),
4  (-1.0, 'Across the Sea of Time (1995)'),
5  (-1.0, 'Angel and the Badman (1947)')]
```

Listing 12: Output of calling the topMatches function to get the 5 least correlated movies to Ace Ventura: Pet Detective.

```
1 [(1.0000000000000004, 'Loch Ness (1995)'),
2  (1.0, 'Wedding Gift, The (1994)'),
3  (1.0, 'Vermin (1998)'),
4  (1.0, 'Two Much (1996)'),
5  (1.0, 'Traveller (1997)')]
```

Listing 13: Output of calling the topMatches function to get the 5 most correlated movies to Jurassic Park.

```
1 [(-1.0000000000000018, 'Kaspar Hauser (1993)'),
2  (-1.0, '1-900 (1994)'),
3  (-1.0, 'American Dream (1990)'),
4  (-1.0, 'Anna (1996)'),
5  (-1.0, 'Aparajito (1956)')]
```

Listing 14: Output of calling the topMatches function to get the 5 least correlated moves to Jurassic Park.

Discussion

This question was completed using the "transformPrefs" function shown in listing 10 from the week-11 notebook. The function switches the users with their items (movies) in the prefs dictionary passed in. Something to note is lines 16, 18-19, and 21 all have their own cells in the notebook, but were included in listing 10 for brevity. I made a call to "transformPrefs" on line 16, and stored the result in the "similarMovies" variable. Then I printed out the "similarMovies" list to make sure it held the correct data. Then I made a call to the "topMatches" function, passing in the "similarMovies" list, Ace Ventura: Pet Detective, and Jurassic Park (not shown).

I should note that Ace Ventura: Pet Detective and Jurassic Park represent my favorite movie and my least favorite movie in the data set, respectively. To get the top five similar movies to Ace Ventura: Pet Detective (shown in listing 11), I made a call to the unmodified "topMatches" function shown in listing 5. To get the five least similar movies to Ace Ventura: Pet Detective (shown in listing 12), I commented out line 15 of listing 5 to sort the list in ascending order, and called "topMatches" again. To get similar and non-similar movies to Jurassic Park, I modified the call to "topMatches" replacing Ace Ventura: Pet Detective with Jurassic Park. I also commented

out/uncommented out line 15 of listing 5 as needed to produce the output of listings 13 and 14. Those are the five most similar and least similar movies to Jurassic Park, respectively.

Note: the only film in the outputs of correlated movies to Ace Ventura: Pet Detective and Jurassic Park is Shiloh. I had to watch trailers for the rest of them.

Based on my knowledge of the resulting films, I only like two out of the five most correlated films to Ace Ventura: Pet Detective. So I would not call the results accurate for that part. I would call the results accurate for the five least correlated films. I have no interest in any of them.

Moving on to Jurassic Park, I'm not sure how to gauge the accuracy of the five most correlated films, considering I don't like the movie. I have no interest in any of the films in the five most correlated movies. Though I could see some people who like Jurassic Park liking Loch Ness. As far as the least correlated movies to Jurassic Park, I think those results are accurate. Even if I liked Jurassic Park, I have no interest in any of five least correlated movies.

Trailer Search Queries and Links

1. The trailer for The Wonderful, Horrible Life of Leni Riefenstahl was found by searching "the Wonderful, Horrible Life of Leni Riefenstahl trailer" on YouTube.
    - Link: `https://youtu.be/91607Aqn-mo`
2. The trailer for When the Cats Away (CHacun cherche son chat) was found by searching "When the Cats Away (Chacun cherche son chat) trailer" on YouTube.
    - Link: `https://youtu.be/8oSunSnkuhU`
3. The trailer for Welcome to Sarajevo was found by searching "Welcome To Sarajevo trailer" on YouTube.
    - Link: `https://youtu.be/IOmm1OD2ArQ`
4. The trailer for U.S. Marshalls was found by searching "U.S. Marshalls trailer" on YouTube
    - Link: `https://youtu.be/_shXM-868Q8`
5. The trailer for Rent-a-Kid was found by searching "rent a kid trailer" on YouTube.
    - Link: `https://youtu.be/00-WPchnLTE`
6. The trailer for True Crime was found by searching "true crime trailer 1995" on YouTube.
    - Link: `https://youtu.be/5zX2WAnifeA`
7. The trailer for 1-900 was found by searching "1-900 1994 trailer" on YouTube.
    - Link: `https://youtu.be/oK4etva_lUU`

8. The trailer for Across the Sea of Time was found by searching “across the sea of time 1995 trailer” on YouTube.
    - Link: `https://youtu.be/CYWtLEJg_OY`
9. The trailer for Angel and the Badman was found by searching “angel and the badman trailer” on YouTube.
    - Link: `https://youtu.be/kw9Hi8le_Cg`
10. The trailer for Loch Ness was found by searching “loch ness 1995 trailer” on YouTube.
    - Link: `https://youtu.be/L90001PVzec`
11. The trailer for The Wedding Gift was found by searching ”the wedding gift 1994 movie” on Bing, and clikcing on a Best Buy ad for the DVD. The trailer is accessed by a play button on the product page.
    - Link: `https://www.bestbuy.com/site/the-wedding-gift-dvd-1994/19271781.p?skuId=19271781&ref=212&loc=1&ref=212&loc=1&ref=06&loc=01&msclkid=9a6f99b3a43e128c0099c67043ae8418&ds_rl=1273104&gclid=CODOgaWwqO0CFUiVswodSKsFZg&gclsrc=ds`
12. I could not locate a trailer for Vermin. The closest thing I could find was a one-sentence plot description on IMDB after searching “vermin 1998 movie” on Bing.
    - Link: `https://www.imdb.com/title/tt0120881/`
13. The trailer for Two Much was found by searching “two much 1996 trailer” on YouTube.
    - Link: `https://youtu.be/ajGpyRSKgKw`
14. The trailer for Traveller was found by searching “traveller 1997 trailer” on YouTube.
    - Link: `https://youtu.be/9dOZyrzR7R0`
15. I could not locate a trailer for Kaspar Hauser, but I found a synopsis on IMDB by searching “kaspar hauser 1993” on Bing.
    - Link: `https://www.imdb.com/title/tt0110246/`
16. I could not find a trailer for American Dream, but I found a synopsis on IMDB by searching “american dream 1990 movie” on Bing.
    - Link: `https://www.imdb.com/title/tt0099028/`
17. I could not find a trailer for Anna, but I found a synopsis on IMDB by searching “anna 1996 movie” on Bing.
    - Link: `https://www.imdb.com/title/tt2661954/`
18. The trailer for Aparajito was found by searching “aparajito 1956” on Bing.
    - Link: `https://www.imdb.com/title/tt0048956/`

References

- How to Sort a Dictionary by Value in Python, https://careerkarma.com/blog/python-sort-a-dictionary-by-value/
- "for line in..." results in UnicodeDecodeError: 'utf-8' codec can't decode byte, https://stackoverflow.com/questions/19699367/for-line-in-results-in-unicodedeco utf-8-codec-cant-decode-byte
- Programming-Collective-Intelligence chapter 2 github, https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter2/recommendati py
- Week-11 Colab notebook, https://colab.research.google.com/github/cs432-websci-fall20/assignments/blob/master/432_PCI_Ch02.ipynb