rec09

October 29, 2020

1 CS 1656 – Introduction to Data Science

1.1 Instructor: Alexandros Labrinidis / Teaching Assistant: Evangelos Karageorgos

1.1.1 Additional credits: Xiaoting Li, Phuong Pham, Zuha Agha, Anatoli Shein

1.2 ## Recitation 9: Collaborative Filtering & Similarity Metrics

In this recitation we will be doing a fun exercise to implement collaborative filtering for recommender systems. We will also learn how the choice of similarity metric in collaborative filtering can affect its output of predicted ratings.

Packages you will need for the recitation are,

- pandas
- numpy
- scipy

Recall that numpy package provides nd-arrays and operations for easily manipulating them. Likewise, scipy provides an addtional suite of useful mathematical functions and distributions for numpy arrays, including distance functions which we will use in this recitation to compute the measure of similarity. We will only import the distance funcions we need for today's session as shown below. Note that cityblock is just another name for Manhattan distance metric seen in class.

```
In [1]: import pandas as pd
        import numpy as np
        from scipy.spatial.distance import euclidean, cityblock, cosine
        from scipy.stats import pearsonr
```

1.3 User-Based vs Item-Based Recommendation

There are two type of collaborative filtering method: user-based and item-based.

User-based recommendation assumes that similar users give similar ratings to each item. Whereas item-based recommendation assumes that similar items receive similar ratings from each user. You can think of them as a dual of each other.

In this recitation, we will walk through a toy example for user-based recommendation and you will try out item-based recommendation later in one of your tasks.

1.4 Data Input

```
In [2]: df = pd.read_csv('http://data.cs1656.org/movies_example.csv')
        df
```

Out[2]:

	Name	Alice	Bob	Christine	David	Elaine	Frank
0	The Matrix	2	3.0	4	5.0	5.0	NaN
1	Gone with the Wind	5	NaN	5	NaN	3.0	3.0
2	Jack and Jill	2	1.0	2	2.0	1.0	1.0
3	Planes	4	4.0	5	2.0	NaN	3.0
4	Rocky IV	2	2.0	3	4.0	3.0	NaN

1.4.1 Accessing rows in dataframe

The two ways to access dataframes rows are shown below,

```
In [3]: # Converting value equality test fo a Series of booleans
        df['Name'] == 'The Matrix'
```

```
Out[3]: 0     True
        1    False
        2    False
        3    False
        4    False
        Name: Name, dtype: bool
```

```
In [4]: # First way to access rows
        df[df['Name'] == 'The Matrix']
```

Out[4]:

	Name	Alice	Bob	Christine	David	Elaine	Frank
0	The Matrix	2	3.0	4	5.0	5.0	NaN

```
In [5]: # Second way
        df.iloc[0]
```

```
Out[5]: Name         The Matrix
        Alice                 2
        Bob                   3
        Christine             4
        David                 5
        Elaine                5
        Frank               NaN
        Name: 0, dtype: object
```

1.4.2 Missing values in data frame

To exlude missing values or NaNs in a dataframe, we can use the notnull() function.

```
In [6]: df['Frank'].notnull()
```

```
Out[6]: 0    False
        1     True
        2     True
        3     True
        4    False
        Name: Frank, dtype: bool

In [7]: df['Elaine'].notnull()

Out[7]: 0     True
        1     True
        2     True
        3    False
        4     True
        Name: Elaine, dtype: bool
```

You can also perform logical operations on the boolean Series returned as shown below,

```
In [8]: df['Frank'].notnull() & df['Elaine'].notnull()

Out[8]: 0    False
        1     True
        2     True
        3    False
        4    False
        dtype: bool
```

You can also select subset of rows and columns where the boolean value is True.

```
In [9]: df_notmissing = df[['Frank','Elaine']][df['Frank'].notnull() & df['Elaine'].notnull()]
        df_notmissing

Out[9]:    Frank  Elaine
        1    3.0     3.0
        2    1.0     1.0
```

1.5 Similarity Metrics & Predicted Ratings

Different distance metrics can be used to measure the similarity. In this recitation, we will use Euclidean, Manhattan, Pearson Correlation and Cosine distance metrics to measure the similarity.

1.5.1 Euclidean

```
In [10]: sim_weights = {}
         for user in df.columns[1:-1]:
             df_subset = df[['Frank',user]][df['Frank'].notnull() & df[user].notnull()]
             dist = euclidean(df_subset['Frank'], df_subset[user])
             sim_weights[user] = 1.0 / (1.0 + dist)
         print ("similarity weights: %s" % sim_weights)
```

```
similarity weights: {'Alice': 0.28989794855663564, 'Bob': 0.5, 'Christine': 0.25, 'David': 0.414
```

Now let's find the predicted rating of 'Frank' for 'The Matrix'. We can get all ratings for a movie by accessing a row of the dataframe using iloc learnt earlier. We only slice the columns of ratings we need indicated by the index [1:-1]. In this case we do not need the first column 'Name' and the last column 'Frank'.

```
In [11]: ratings = df.iloc[0][1:-1]
         ratings

Out[11]: Alice        2
         Bob          3
         Christine    4
         David        5
         Elaine       5
         Name: 0, dtype: object
```

Now we will find our predicted rating by multiplying each user weight with its corresponding rating for the movie matrix.

```
In [12]: predicted_rating = 0.0
         weights_sum = 0.0
         for user in df.columns[1:-1]:
             predicted_rating += ratings[user] * sim_weights[user]
             weights_sum += sim_weights[user]

         predicted_rating /= weights_sum
         print ("predicted rating: %f" % predicted_rating)

predicted rating: 4.136268
```

1.5.2 Manhattan (Cityblock)

We repeat our method of finding predicted rating using cityblock distance now.

```
In [13]: sim_weights = {}
         for user in df.columns[1:-1]:
             df_subset = df[['Frank',user]][df['Frank'].notnull() & df[user].notnull()]
             dist = cityblock(df_subset['Frank'], df_subset[user])
             sim_weights[user] = 1.0 / (1.0 + dist)
         print ("similarity weights: %s" % sim_weights)

         predicted_rating = 0
         weights_sum = 0.0
         ratings = df.iloc[0][1:-1]
         for user in df.columns[1:-1]:
             predicted_rating += ratings[user] * sim_weights[user]
```

```
            weights_sum += sim_weights[user]

        predicted_rating /= weights_sum
        print ("predicted rating: %f" % predicted_rating)
```

```
similarity weights: {'Alice': 0.2, 'Bob': 0.5, 'Christine': 0.16666666666666666, 'David': 0.3333
predicted rating: 4.196970
```

1.5.3 Pearson Correlation Coefficient

```
In [14]: sim_weights = {}
         for user in df.columns[1:-1]:
             df_subset = df[['Frank',user]][df['Frank'].notnull() & df[user].notnull()]
             sim_weights[user] = pearsonr(df_subset['Frank'], df_subset[user])[0]
         print ("similarity weights: %s" % sim_weights)

         predicted_rating = 0.0
         weights_sum = 0.0
         ratings = df.iloc[0][1:-1]
         for user in df.columns[1:-1]:
             predicted_rating += ratings[user] * sim_weights[user]
             weights_sum += sim_weights[user]

         predicted_rating /= weights_sum
         print ("predicted rating: %s" % predicted_rating)
```

```
similarity weights: {'Alice': 0.9449111825230679, 'Bob': 1.0, 'Christine': 1.0, 'David': nan, 'E
predicted rating: nan
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:3399: PearsonRConstantInputWarni
  warnings.warn(PearsonRConstantInputWarning())
```

Why nan? Because anything divided by 0 is undefined. Computing it again with this modfication gives the following.

```
In [15]: predicted_rating = 0.0
         weights_sum = 0.0
         ratings = df.iloc[0][1:-1]
         for user in df.columns[1:-1]:
             if (not np.isnan(sim_weights[user])):
                 predicted_rating += ratings[user] * sim_weights[user]
                 weights_sum += sim_weights[user]

         predicted_rating /= weights_sum
         print ("predicted rating: %f" % predicted_rating)
```

```
predicted rating: 3.520947
```

1.6 Tasks

For your tasks, use the movie ratings data we collected from a previous class in movie_class_responses.csv. You will predict missing movie ratings of a student based on other students with similar tastes. The first column, 'Alias' is the name of the movie, while all other columns are user names od students. The ratings are from 1 to 5, while there are a lot of missing values (missing movie ratings).

```
In [16]: df = pd.read_csv('http://data.cs1656.org/movie_class_responses.csv')
         df.head()
```

Out[16]:

	Alias	potatofaminewasaninsidejob	Thug Nugget	Arjen_Robben_Is_Cool	Oh-Long Johnson	BigD	Starlord	captainamerica	BabyKangaroo	Ryan	...	Garbage	CleverAlias	Kirito	ryan	Geek in the Pink	Michelle	Jim Jarmush	rater	RonJohnson	Dory
0	50 Shades of Grey	1.0	NaN	NaN	1.0	NaN	1.0	1.0	NaN	1	...	3	3	1.0	1	1	NaN	1.0	NaN	2	2.0
1	Airplane	NaN	NaN	NaN	5.0	NaN	3.0	NaN	4.0	4	...	5	3	5.0	1	3	NaN	NaN	NaN	4	NaN
2	Au Revoir Les Enfants	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	3	...	3	3	NaN	3	3	NaN	NaN	NaN	4	NaN
3	Blues Brothers	NaN	NaN	NaN	4.0	NaN	NaN	NaN	2.0	3	...	5	3	4.0	3	4	NaN	4.0	NaN	4	NaN
4	Dark Night	NaN	3.0	NaN	5.0	5.0	5.0	NaN	5.0	5	...	4	3	4.0	4	4	NaN	NaN	4.0	5	5.0

[5 rows x 32 columns]

** Task 1: User-based Recommendation with Cosine Metric**

For a specified user, calculate ALL missing movie ratings using user-based recommendation with Cosine Metric.

** Task 2: Item-based Recommendation with Cosine Metric**

Repeat the task above by doing an item-based recommendation instead of a user based recommendation. To calculate a missing movie rating using item-based recommendation, you are supposed to find similarity between movies instead of users. In other words, you measure the similarity of the user's missing rating movie with movies that the user has rated in the past. Then

compute a weighted average using similar movie weights and their ratings to find out the predicted rating. You need to predict ALL missing movie ratings for the user.

** Task 3: User-based Recommendation with Cosine Metric**

Repeat Task 1 while computing the weighted average using just top 10 most similar users instead of all users.