recsys_intro

February 22, 2024

1 CS 1656 – Introduction to Data Science

1.1 Instructor: Alexandros Labrinidis

1.1.1 Teaching Assistants: Evangelos Karageorgos, Xiaoting Li, Zi Han Ding

1.1.2 Additional credits: Phuong Pham, Zuha Agha, Anatoli Shein

1.2 ## Recitation 6: 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.

[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

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

```
[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,

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

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

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

```
[4]:          Name  Alice  Bob  Christine  David  Elaine  Frank
     0  The Matrix      2  3.0          4    5.0     5.0    NaN
```

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

```
[5]: Name         The Matrix
     Alice                 2
     Bob                 3.0
     Christine             4
     David               5.0
     Elaine              5.0
     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.

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

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

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

[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,

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

[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.

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

[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

[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': 0.4142135623730951, 'Elaine': 1.0}
```

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'.

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

[11]:
```
Alice          2
Bob          3.0
Christine      4
David        5.0
Elaine       5.0
Name: 0, dtype: object
```

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

[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.

[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': 0.3333333333333333, 'Elaine': 1.0}
predicted rating: 4.196970
```

1.5.3 Pearson Correlation Coefficient

[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, 'Elaine': 1.0}
predicted rating: nan
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\_stats_py.py:4781:
ConstantInputWarning: An input array is constant; the correlation coefficient is
not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
```

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

[15]:
```
predicted_rating = 0.0
weights_sum = 0.0
ratings = df.iloc[0][1:-1]
```

```python
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
```