Collaborative Filtering AI

December 15, 2022

[2]:
```python
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
```

0.0.1 Importing the data

[4]:
```python
data = pd.read_csv('sheet5.csv').dropna()
# Creates a new dataframe without the user ids.
data_items = data.drop('user', 1).dropna()
```

[11]:
```python
data
```

[11]:

	user	cars	trucks	dogs	cats	coffee	tea	hot	cold	running	swimming
1	2	1	0	1	0	0	1.0	0	1.0	0	1
2	3	0	1	0	0	1	1.0	0	1.0	1	1
3	4	1	1	1	0	0	0.0	0	1.0	0	1
4	5	1	0	1	1	0	0.0	1	0.0	0	0
5	6	1	1	1	1	1	0.0	1	0.0	0	1
..	…	…	…	…	…	…	…	…	…	…	…
94	95	0	0	1	1	0	0.0	1	1.0	1	0
95	96	0	0	0	0	0	0.0	1	1.0	1	0
96	97	1	0	0	1	1	1.0	0	1.0	0	0
97	98	0	1	0	0	1	0.0	0	0.0	1	0
98	99	0	0	1	0	1	1.0	0	1.0	0	1

[98 rows x 11 columns]

The first step is to normalize the data:

1. take the data set, "data_items" that doesn't have the users column, and square every entry: np.square(data_items)
2. take each row and sum all entries in the row together: np.square(data_items).sum(axis=1)
3. take the sqaure root of each value (the sums): np.sqrt(np.square(data_items).sum(axis=1))

[105]:
```python
magnitude = np.sqrt(np.square(data_items).sum(axis=1))
```

```
#step by step:
print(data_items)
print('takes the square of every entry in the dataframe then takes the sum of␣
 ↪the rows:')
print(np.square(data_items).sum(axis=1))
print('then takes the sqaure root of each value:')
print(np.sqrt(np.square(data_items).sum(axis=1)))
```

```
    cars  trucks  dogs  cats  coffee  tea  hot  cold  running  swimming
1      1       0     1     0       0  1.0    0   1.0        0         1
2      0       1     0     0       1  1.0    0   1.0        1         1
3      1       1     1     0       0  0.0    0   1.0        0         1
4      1       0     1     1       0  0.0    1   0.0        0         0
5      1       1     1     1       1  0.0    1   0.0        0         1
..   …     …   …   …     …  …  …   …      …        …
94     0       0     1     1       0  0.0    1   1.0        1         0
95     0       0     0     0       0  0.0    1   1.0        1         0
96     1       0     0     1       1  1.0    0   1.0        0         0
97     0       1     0     0       1  0.0    0   0.0        1         0
98     0       0     1     0       1  1.0    0   1.0        0         1

[98 rows x 10 columns]
takes the square of every entry in the dataframe then takes the sum of the rows:
1     5.0
2     6.0
3     5.0
4     4.0
5     7.0
     …
94    5.0
95    3.0
96    5.0
97    3.0
98    5.0
Length: 98, dtype: float64
then takes the sqaure root of each value:
1     2.236068
2     2.449490
3     2.236068
4     2.000000
5     2.645751
        …
94    2.236068
95    1.732051
96    2.236068
97    1.732051
98    2.236068
```

```
Length: 98, dtype: float64
```

0.0.2 Step 2 is to make these vectors into unit vectors by dividing by the magnitude

1. divide each value in data_items by the magniture calculated earlier, for example if we have the first row of data in data_items which is 1,0,1,0,0,1,0,1,0,1 we will divide all the 1's and 0's by the corresponding magnitude which is 2.236068. Keep in mind each row has its own corresponding magnitude which is stored in that variable, row 1's magnitude is 2.236068, row 2's is 2.449490 and so on and so forth.

[106]:
```python
# unitvector = (x / magnitude, y / magnitude, z / magnitude, ...)
data_items = data_items.divide(magnitude, axis='index')
data_items
```

[106]:
```
        cars    trucks      dogs      cats    coffee       tea       hot  \
1   0.447214  0.000000  0.447214  0.000000  0.000000  0.447214  0.000000
2   0.000000  0.408248  0.000000  0.000000  0.408248  0.408248  0.000000
3   0.447214  0.447214  0.447214  0.000000  0.000000  0.000000  0.000000
4   0.500000  0.000000  0.500000  0.500000  0.000000  0.000000  0.500000
5   0.377964  0.377964  0.377964  0.377964  0.377964  0.000000  0.377964
..       …         …         …         …         …         …         …
94  0.000000  0.000000  0.447214  0.447214  0.000000  0.000000  0.447214
95  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.577350
96  0.447214  0.000000  0.000000  0.447214  0.447214  0.447214  0.000000
97  0.000000  0.577350  0.000000  0.000000  0.577350  0.000000  0.000000
98  0.000000  0.000000  0.447214  0.000000  0.447214  0.447214  0.000000

        cold   running  swimming
1   0.447214  0.000000  0.447214
2   0.408248  0.408248  0.408248
3   0.447214  0.000000  0.447214
4   0.000000  0.000000  0.000000
5   0.000000  0.000000  0.377964
..       …         …         …
94  0.447214  0.447214  0.000000
95  0.577350  0.577350  0.000000
96  0.447214  0.000000  0.000000
97  0.000000  0.577350  0.000000
98  0.447214  0.000000  0.447214

[98 rows x 10 columns]
```

0.0.3 The next step is to calculate the cosine similarity of the table

1. take the new data_items table which should all be divided by magnitude and take the transpose of the table
2. calculate the cosine similarity by running that in a function like this i found for javascript: https://stackoverflow.com/questions/51362252/javascript-cosine-similarity-

function/51362370
3. the inputs in the function for A and B are going to be data_items.T, data_items.T where T means the transpose of the matrix

Below ill show you how I run it in python

[10]:
```
data_items.T
```

[10]:
```
            1    2    3    4    5    6    7    8    9    10  ...   89   90  \
cars      1.0  0.0  1.0  1.0  1.0  0.0  1.0  1.0  1.0  0.0  ...  1.0  0.0
trucks    0.0  1.0  1.0  0.0  1.0  0.0  1.0  1.0  0.0  1.0  ...  0.0  1.0
dogs      1.0  0.0  1.0  1.0  1.0  1.0  0.0  0.0  1.0  0.0  ...  1.0  0.0
cats      0.0  0.0  0.0  1.0  1.0  1.0  0.0  1.0  1.0  0.0  ...  1.0  0.0
coffee    0.0  1.0  0.0  0.0  1.0  1.0  1.0  1.0  0.0  1.0  ...  0.0  0.0
tea       1.0  1.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  1.0  ...  0.0  0.0
hot       0.0  0.0  0.0  1.0  1.0  0.0  0.0  1.0  0.0  1.0  ...  0.0  0.0
cold      1.0  1.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  ...  0.0  1.0
running   0.0  1.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  0.0  ...  0.0  1.0
swimming  1.0  1.0  1.0  0.0  1.0  0.0  1.0  1.0  1.0  1.0  ...  0.0  0.0

            91   92   93   94   95   96   97   98
cars      1.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0
trucks    0.0  1.0  1.0  0.0  0.0  0.0  1.0  0.0
dogs      0.0  1.0  1.0  1.0  0.0  0.0  0.0  1.0
cats      0.0  1.0  1.0  1.0  0.0  1.0  0.0  0.0
coffee    0.0  1.0  0.0  0.0  0.0  1.0  1.0  1.0
tea       1.0  1.0  1.0  0.0  0.0  1.0  0.0  1.0
hot       0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0
cold      1.0  1.0  1.0  1.0  1.0  1.0  0.0  1.0
running   0.0  0.0  1.0  1.0  1.0  0.0  1.0  0.0
swimming  1.0  1.0  1.0  0.0  0.0  0.0  0.0  1.0

[10 rows x 98 columns]
```

[15]:
```
s = cosine_similarity(data_items.T,data_items.T)
s
```

[15]:
```
array([[1.        , 0.49920191, 0.52448805, 0.52728743, 0.4882291 ,
         0.41932961, 0.49362406, 0.5715197 , 0.54385965, 0.67961448],
        [0.49920191, 1.        , 0.4477215 , 0.48900965, 0.42232651,
         0.54545455, 0.40451992, 0.54885305, 0.61901037, 0.56446336],
        [0.52448805, 0.4477215 , 1.        , 0.50043459, 0.52128604,
         0.49036165, 0.42163702, 0.5148698 , 0.46829291, 0.56873679],
        [0.52728743, 0.48900965, 0.50043459, 1.        , 0.43478261,
         0.53346507, 0.41760763, 0.43738352, 0.44917078, 0.42937693],
        [0.4882291 , 0.42232651, 0.52128604, 0.43478261, 1.        ,
         0.48900965, 0.48354568, 0.41750246, 0.52728743, 0.49071649],
        [0.41932961, 0.54545455, 0.49036165, 0.53346507, 0.48900965,
```

```
        1.        , 0.33709993, 0.52852516, 0.49920191, 0.54355731],
       [0.49362406, 0.40451992, 0.42163702, 0.41760763, 0.48354568,
        0.33709993, 1.        , 0.48241815, 0.57260392, 0.43412157],
       [0.5715197 , 0.54885305, 0.5148698 , 0.43738352, 0.41750246,
        0.52852516, 0.48241815, 1.        , 0.53579972, 0.57966713],
       [0.54385965, 0.61901037, 0.46829291, 0.44917078, 0.52728743,
        0.49920191, 0.57260392, 0.53579972, 1.        , 0.49593489],
       [0.67961448, 0.56446336, 0.56873679, 0.42937693, 0.49071649,
        0.54355731, 0.43412157, 0.57966713, 0.49593489, 1.        ]])
```

0.0.4 Next im adding the matrix back into a dataframe so i can work with it in pandas

[16]:
```
data_matrix = pd.DataFrame(data=s, index= data_items.columns, columns=␣
 ↪data_items.columns)
data_matrix
```

[16]:
```
              cars    trucks      dogs      cats    coffee       tea  \
cars      1.000000  0.499202  0.524488  0.527287  0.488229  0.419330
trucks    0.499202  1.000000  0.447722  0.489010  0.422327  0.545455
dogs      0.524488  0.447722  1.000000  0.500435  0.521286  0.490362
cats      0.527287  0.489010  0.500435  1.000000  0.434783  0.533465
coffee    0.488229  0.422327  0.521286  0.434783  1.000000  0.489010
tea       0.419330  0.545455  0.490362  0.533465  0.489010  1.000000
hot       0.493624  0.404520  0.421637  0.417608  0.483546  0.337100
cold      0.571520  0.548853  0.514870  0.437384  0.417502  0.528525
running   0.543860  0.619010  0.468293  0.449171  0.527287  0.499202
swimming  0.679614  0.564463  0.568737  0.429377  0.490716  0.543557

               hot      cold   running  swimming
cars      0.493624  0.571520  0.543860  0.679614
trucks    0.404520  0.548853  0.619010  0.564463
dogs      0.421637  0.514870  0.468293  0.568737
cats      0.417608  0.437384  0.449171  0.429377
coffee    0.483546  0.417502  0.527287  0.490716
tea       0.337100  0.528525  0.499202  0.543557
hot       1.000000  0.482418  0.572604  0.434122
cold      0.482418  1.000000  0.535800  0.579667
running   0.572604  0.535800  1.000000  0.495935
swimming  0.434122  0.579667  0.495935  1.000000
```

0.0.5 Next I define the user whom I want to make predictions for

[13]:
```
data
```

[13]:
```
   user  cars  trucks  dogs  cats  coffee  tea  hot  cold  running  swimming
1     2     1       0     1     0       0  1.0    0   1.0        0         1
```

```
2     3   0     1   0   0     1  1.0   0  1.0      1      1
3     4   1     1   1   0     0  0.0   0  1.0      0      1
4     5   1     0   1   1     0  0.0   1  0.0      0      0
5     6   1     1   1   1     1  0.0   1  0.0      0      1
..   …   …     …   …   …    …  …  …  …     …      …
94   95   0     0   1   1     0  0.0   1  1.0      1      0
95   96   0     0   0   0     0  0.0   1  1.0      1      0
96   97   1     0   0   1     1  1.0   0  1.0      0      0
97   98   0     1   0   0     1  0.0   0  0.0      1      0
98   99   0     0   1   0     1  1.0   0  1.0      0      1

[98 rows x 11 columns]
```

[20]:
```python
user = 3 # The id of the user for whom we want to generate recommendations
user_index = data[data.user == user].index.tolist()[0] # Get the frame index
user_index
```

[20]: 2

1. known_user_likes = data_items.iloc[user_index] this line gets the preferences of the user
2. known_user_likes = known_user_likes[known_user_likes >0].index.values this line gets the preferences of the user that is above zero, so basically the recorded likes of the user and puts it into a list

[21]:
```python
# Get the artists the user has likd.
known_user_likes = data_items.iloc[user_index]
print(known_user_likes)
known_user_likes = known_user_likes[known_user_likes >0].index.values
print(known_user_likes)
```

```
cars        1.0
trucks      1.0
dogs        1.0
cats        0.0
coffee      0.0
tea         0.0
hot         0.0
cold        1.0
running     0.0
swimming    1.0
Name: 3, dtype: float64
['cars' 'trucks' 'dogs' 'cold' 'swimming']
```

0.0.6 Storing the user preferences in a variable again

[22]:
```
# Users likes for all items as a sparse vector.
user_rating_vector = data_items.iloc[user_index]
user_rating_vector
```

[22]:
```
cars        1.0
trucks      1.0
dogs        1.0
cats        0.0
coffee      0.0
tea         0.0
hot         0.0
cold        1.0
running     0.0
swimming    1.0
Name: 3, dtype: float64
```

0.0.7 This line here calculates the dot product between the data_matrix and user_rating_vector and divides it by the sum of data_matrix rows

Dot product in JS:

https://stackoverflow.com/questions/64816766/dot-product-of-two-arrays-in-javascript

[23]:
```
data_matrix.dot(user_rating_vector)
```

[23]:
```
cars        3.274824
trucks      3.060240
dogs        3.055816
cats        2.383492
coffee      2.340061
tea         2.527228
hot         2.236321
cold        3.214910
running     2.662898
swimming    3.392482
dtype: float64
```

[18]:
```
data_matrix.sum(axis=1)
```

[18]:
```
cars        5.747154
trucks      5.540561
dogs        5.457828
cats        5.218518
coffee      5.274686
tea         5.386005
hot         5.047178
```

```
cold        5.616539
running     5.711162
swimming    5.786189
dtype: float64
```

[24]:
```
# Calculate the score.
score = data_matrix.dot(user_rating_vector).div(data_matrix.sum(axis=1))
score
```

[24]:
```
cars        0.569817
trucks      0.552334
dogs        0.559896
cats        0.456737
coffee      0.443640
tea         0.469221
hot         0.443083
cold        0.572401
running     0.466262
swimming    0.586307
dtype: float64
```

[25]:
```
# Remove the known likes from the recommendation.
score = score.drop(known_user_likes)
```

[26]:
```
# Print the known likes and the top recommendations.
print(known_user_likes)
print(score.nlargest(20))
```

```
['cars' 'trucks' 'dogs' 'cold' 'swimming']
tea        0.469221
running    0.466262
cats       0.456737
coffee     0.443640
hot        0.443083
dtype: float64
```

[ ]:
```
```