EE412 Foundation of Big Data Analytics, Fall 2021
HW2

Name: 노현섭

Student ID: 20190220

Discussion Group (People with whom you discussed ideas used in your answers):

On-line or hardcopy documents used as part of your answers:

[TOKEN=1]

Answer to Problem 1

badger
badger
badger
badger
badger
banana
nanana

Answer to Problem 2

Exercise 11.1.7

```python
import numpy as np

M = np.array([[1,1,1],[1,2,3],[1,3,6]])

for i in range(3):
  x = np.ones((3,1))
  tmp = np.zeros((3,1))
  while(np.linalg.norm(x-tmp) > 0.0001):
    tmp = x
    product = np.dot(M, x)
    x = product / np.linalg.norm(product)
  eig_value = np.dot(np.dot(np.transpose(x), M), x)
  M = M - eig_value*np.dot(x, np.transpose(x))
  print ("eig_value: %f") % eig_value
  print (x)
  print (M)
```

(a)

```
[[ 0.19382449]
 [ 0.4722482 ]
 [ 0.85989168]]
```

(b)

```
eigenvalue: 7.872983
```

(c)

```
[[ 0.70422829  0.27936011 -0.31217491]
 [ 0.27936011  0.24418012 -0.19707919]
 [-0.31217491 -0.19707919  0.17860824]]
```

(d)

```
Second eigenpair
eigenvalue: 1.000000
eigenvector:  [[ 0.81649467]
               [ 0.40823775]
               [-0.40826264]]
```

(e)

```
Thread eigenpair
eigenvalue: 0.127017
eigenvector:  [[ 0.54382585]
               [-0.78123612]
               [ 0.30646952]]
```

Exercise 11.3.1

```python
import numpy as np

M = np.array([[1,2,3],[3,4,5],[5,4,3],[0,2,4],[1,3,5]])

# (a) Compute MTM and MMT.
MTM = np.dot(np.transpose(M), M)
MMT = np.dot(M, np.transpose(M))
print (MTM)
print (MMT)

# (b) Compute eigenpairs of MTM and MMT.
print ("eigenpair of MTM")
MTM_pair = np.linalg.eig(MTM)
print (MTM_pair)
print ("eigenpair of MMT")
MMT_pair = np.linalg.eig(MMT)
print (MMT_pair)

# (c) Find SVD (assume all eigenvalues are different)
rank = np.linalg.matrix_rank(M)
print ("This is rank")
print (rank)
eig_values = []
V = np.zeros((M.shape[1], rank))
U = np.zeros((M.shape[0], rank))
values = list(MTM_pair[0])

# print MTM_pair[1]
# Find V, which is matrix of eigenvectors of MTM
for i in range(rank):
    index = values.index(max(values))
    eig_values.append(values[index])
    values[index] = 0
    V[:,i] = MTM_pair[1][:,index]

if V[0,0] < 0:
    V = -V

# Find U, which is matrix of eigenvectors of MMT
values = list(MMT_pair[0])
```

```
39  # Find U, which is matrix of eigenvectors of MMT
40  values = list(MMT_pair[0])
41
42  for i in range(rank):
43    index = values.index(max(values))
44    values[index] = 0
45    U[:,i] = MMT_pair[1][:,index]
46  if U[0,0] < 0:
47    U = -U
48
49  sigma = np.sqrt(np.diag(eig_values))
50  print ("U,V,S of M")
51  print (U)
52  print (V)
53  print (sigma)
54
55  # (d) Set smaller singular value to 0
56  U_1 = U[:,:-1]
57  V_1 = V[:,:-1]
58  sigma_1 = sigma[:-1,:-1]
59  print ("Approximated U,V,S")
60  print (U_1)
61  print (V_1)
62  print (sigma_1)
63
64  # (e) Compare energy of the original and approximation
65  print ("Original energy: %f") % (np.sum(np.square(sigma)))
66  print ("Approximated energy: %f") % (np.sum(np.square(sigma_1)))
```

(a)

M'M

```
[[36 37 38]
 [37 49 61]
 [38 61 84]]
```

MM'

```
[[14 26 22 16 22]
 [26 50 46 28 40]
 [22 46 50 20 32]
 [16 28 20 20 26]
 [22 40 32 26 35]]
```

(b)

- M'M

eigenvalues: [1.53566996e+02, 1.54330035e+01, 4.80589926e-15]

eigenvector: [[-0.40928285, -0.81597848, 0.40824829],
[-0.56345932, -0.12588456, -0.81649658],
[-0.7176358 , 0.56420935, 0.40824829]]

- MM'
  eigenvalues: [1.53566996e+02, -1.03322028e-14, 1.54330035e+01,
    2.54653026e-15, -3.61063094e-15]
  eigenvector: [[0.29769568, 0.94131607, -0.15906393, 0.12508859, 0.07520849],
    [0.57050856, -0.17481584, 0.0332003 , -0.45318832, -0.07287035],
    [0.52074297, -0.04034212, 0.73585663, 0.32553276, -0.10566284],
    [0.32257847, -0.18826321, -0.5103921 , 0.72000366, -0.72571726],
    [0.45898491, -0.21515796, -0.41425998, -0.39318742, 0.67171677]]

(c)

SVD of M

- U
  [[0.29769568 -0.15906393]
  [0.57050856 0.0332003]
  [0.52074297 0.73585663]
  [0.32257847 -0.5103921]
  [0.45898491 -0.41425998]]
- V
  [[0.40928285 0.81597848]
  [0.56345932 0.12588456]
  [0.7176358 -0.56420935]]
- Sigma
  [[12.39221516 0.]
  [0. 3.92848616]]

(d)

One-dimensional approximation

- U
  [[0.29769568]
  [0.57050856]
  [0.52074297]
  [0.32257847]
  [0.45898491]]
- V
  [[0.40928285]
  [0.56345932]
  [0.7176358]]
- Sigma: 12.3922151555

(e)

Compare evergy

- Original evergy: 169
- Retained evergy: 153.566996
- Retained 90.868%

Answer to Problem 3

(a) Solve the following problems.

- Exercise 9.3.1

a. Compute the Jaccard distance between each pair of users.

New matrix

	a	b	c	d	e	f	g	h
A	1	1	0	1	1	0	1	1
B	0	1	1	1	1	1	1	0
C	1	0	1	1	0	1	1	1

distance (A, B) = 4/8 = 1/2
distance (B, C) = 4/8 = 1/2
distance (A, C) = 4/8 = 1/2

b. Compute cosine distance.

distance (A, B) = $\frac{4}{\sqrt{6}\times\sqrt{6}} = \frac{4}{6} = \frac{2}{3}$
distance (B, C) = $\frac{4}{\sqrt{6}\times\sqrt{6}} = \frac{4}{6} = \frac{2}{3}$
distance (A, C) = $\frac{4}{\sqrt{6}\times\sqrt{6}} = \frac{4}{6} = \frac{2}{3}$

c. Repeat (a), treating 3, 4, and 5 as 1 and 1, 2, and blank as 0.

New matrix

	a	b	c	d	e	f	g	h
A	1	1	0	1	0	0	1	0
B	0	1	1	1	0	0	0	0
C	0	0	0	1	0	1	1	1

distance (A, B) = 1 – 2/5 = 3/5
distance (B, C) = 1 – 1/6 = 5/6
distance (A, C) = 1 – 2/6 = 2/3

d. Repeat (b) with new utility matrix from c.

distance (A, B) = $\frac{2}{2\times\sqrt{3}} = 1/\sqrt{3}$
distance (B, C) = $\frac{1}{\sqrt{3}\times 2} = \frac{1}{2\sqrt{3}}$
distance (A, C) = $\frac{2}{2\times 2} = 1/2$

e. Normalize the matrix.

New Matrix

	a	b	c	d	e	f	g	h
A	0.666667	1.666667		1.666667	-2.3333		-0.3333	-1.3333
B		0.666667	1.666667	0.666667	-1.3333	-0.3333	-1.3333	
C	-1		-2	0		1	2	0

f. Compute cosine distance with (e).

distance (A, B) = $5.777778/3.651484 \times 2.708013 = 0.584307$

distance (B, C) = $-6.33333/2.708013 \times 3.162278 = -0.73957$

distance (A, C) = $-1.33333/3.651484 \times 3.162278 = -0.11547$

- Exercise 9.3.2

a. Cluster the eight items hierarchically into four clusters.

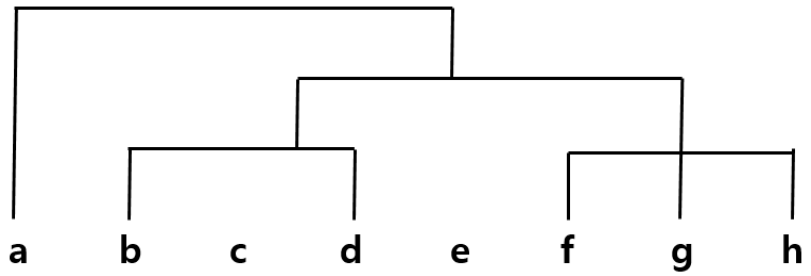
New matrix

	a	b	c	d	e	f	g	h
A	1	1	0	1	0	0	1	0
B	0	1	1	1	0	0	0	0
C	0	0	0	1	0	1	1	1

Jaccard distance

	b	c	d	e	f	g	h
a	1/2	1	2/3	1	1	1/2	1
h	1	1	2/3	1	0	1/2	
g	2/3	1	1/3	1	1/2		
f	1	1	2/3	1			
e	1	1	1				
d	1/3	2/3					
c	1/2						

Clustering (Assume we break ties lexicographically)



Clusters

{a, b, d, g}, {c}, {e}, {f, h}

b. Construct new matrix

New matrix

	{a, b, d, g}	{c}	{e}	{f, h}
A	17/4		1	2
B	7/3	4	1	2
C	10/3	1		3.5

c. Compute cosine distance from (b)

$$\text{distance (A, B)} = {}^{14.91667}/_{4.802343 \times 5.142416} = 0.60402$$

$$\text{distance (B, C)} = {}^{18.7778}/_{5.142416 \times 4.935698} = 0.739824$$

$$\text{distance (A, C)} = {}^{21.16667}/_{4.802343 \times 4.935698} = 0.892999$$

3-c

Algorithm:

우선 초기 데이터베이스는 딕셔너리 형태로, 일반 데이터와 테스트 데이터를 분리하여 정리하였다. 이때 각각의 유저가 첫번째 key 로 들어가며, 두번째 key 들은 영화 리스트이고 각각 score 가 value 가 된다. 테스트 데이터셋은 세개의 세개의 user, item, time 이 튜플을 이루며 key 값이 되며, value 값은 최종적으로 score 가 된다.

이후 일반 데이터를 normalize 하여 cosine distance 를 구해 각각의 유저(첫 번째 key) 안에 새로운 key 인, 자신과 비슷한 user 를 총 300 명 내림차순으로 정리한다. 각각의 value 는 cosine distance 가 된다. 이 similar user set 을 통해 테스트 데이터 셋에서 각각의 user(튜플의 첫번째 값)와 비슷한 user 목록을 iterate 하게 되고, similarity 및 각각의 유저(300)가 item(튜플의 두번째값)을 평가한 목록을 score 로 낸다. (이때 cosine distance 값을 이용해 적절한 scale 에 차이를 준다.) 만약에 300 명의 유저 중에서 평가한 사람이 없다면, 초기 유저로 돌아가 자신이 내린 전체 영화 평점을 평균화 하여 대체한다.

Item 목록에 대해서도 동일한 방식으로 해주면 전체 score list 가 총 두개가 나온다. (코드에서는 movRate, movRate2 라 명시되어 있다. 이 값들을 적절한 scaler 로 (여기서는 2,5 의 비중을 주었다.) 분배해 최종 계산을 하면 (user, item, time)에 대한 예상 rate 가 나온다.