EE412 Foundation of Big Data Analytics, Fall 2018
HW2

Name: Park Jaeyoung

Student ID: 20170273

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

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

Answer to Problem 1(Clustering)

Solve the following problems, which are on MMDS textbook.

Exercise 7.2.6

For instance, if there is {a, ab, abc, acb, abb, bbb, bbbd, bbbe} as a string space and {a, ab, abc, abb, acb} as a set of string. Then “ab” is minimizing the sum of the distance to the other points – distance of 1 from all other elements in the set, but “abb” is minimizing the maximum distance to the other points – in this case maximum distance is 3.

Exercise 7.2.6

It is obvious to know that three cluster should be {(4, 10), (7, 10), (4, 8), (6, 8)}, {(2, 2), (3, 4), (5, 2)}, {(10, 5), (9, 3), (11, 4), (12, 6), (12, 3)}.
Let these three clusters as A, B, C.
Diameter of A, B, C are 3.60, 3, and 4.24.
Minimum distance between (A, B), (B, C), (C, A) are 4.12, 4.12, 5.
But if (9, 3) is excluded while measuring inter-cluster minimum distance, then (B, C) inter-cluster distance would be 5.83. So this means that in case of point (9, 3) is not selected as three starting point, then three points will appear in different cluster; inter-cluster distance is always greater than diameter of cluster, so once element is selected on any cluster, then element of that cluster would be always nearer than element in other cluster.

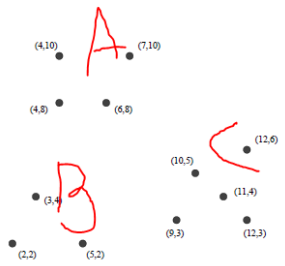


Figure 7.8: Repeat of Fig. 7.2

There would be two cases that (9, 3) is included on starting point.

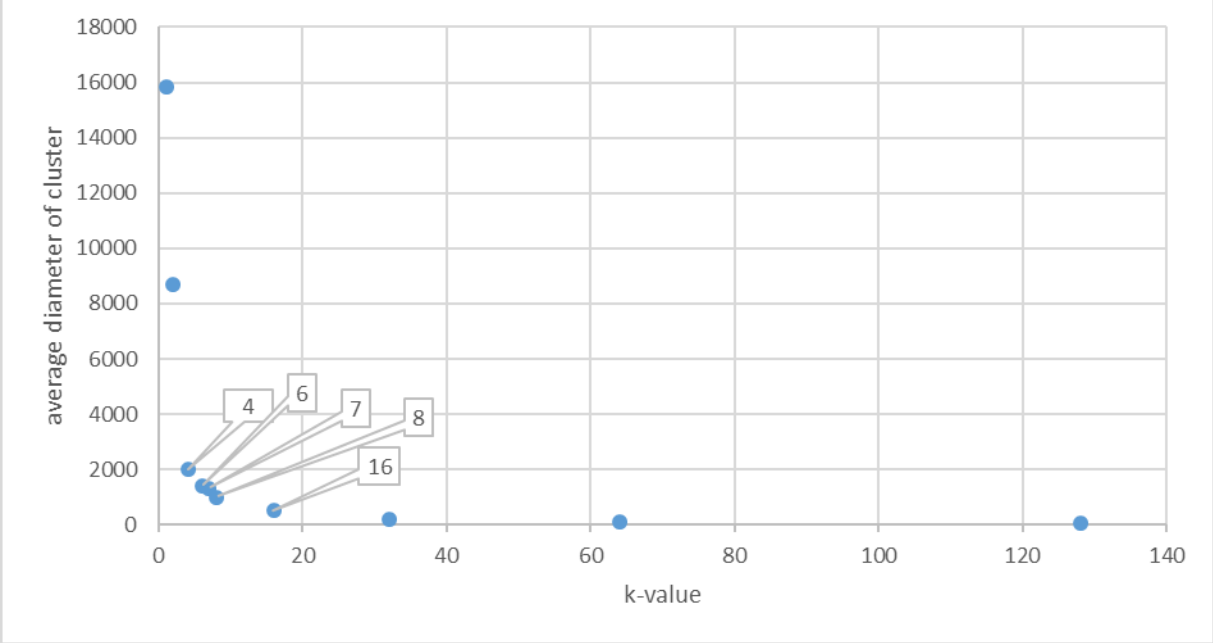
(1) (9, 3) is first point
(2) (9, 3) is second or third point

In case (1), second point is (4, 10) in cluster A and third point is (2, 2).
Case (2) is impossible.
First, if other points in C is selected, then (9, 3) cannot be selected; all other points in A and B is farther then (9, 3). Then there exist any point in cluster A or B. There is a point always in cluster C that distance between cluster A or B and the point in cluster C is larger than cluster A or B and (9, 3). Distance to (12,

3) is always larger, while x coordinate is same and x is larger than (9, 3), and x coordinate of all points of A, B are less than 9.
Therefore in all condition, three starting points would be in different clusters.

Exercise 7.4.1
Before moving 20% to centroid, distance between two groups are in [i-c, c+o], as all points are in boundaries.
These groups move to centroid, which is same through two cluster, possible distance would be [0.8(i-c), 0.8(c+o)].
Two cluster is merged if there exists any pair of representative points that are close.
By definition of merging, two clusters will be merged if d >= 0.8(c+d) for sure, in extreme case – only one representative point was selected from each cluster and their position is as far as possible.
However, this condition will change if number of representative points change.

Implement the k-Means algorithm using Spark.
hw2_1.py attached.



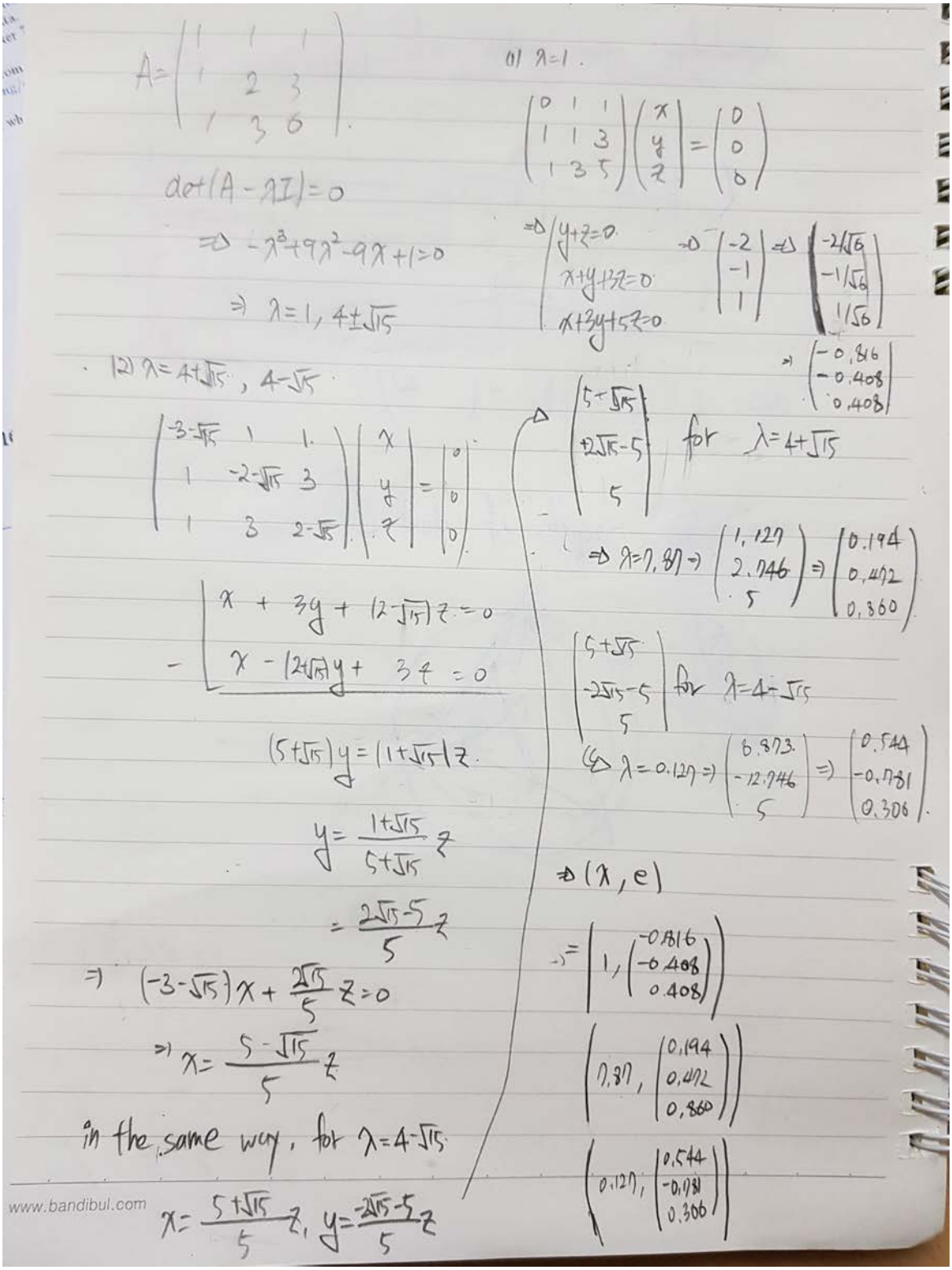
[Figure: Average diameter of group verses number of clusters]

It seems that k = 8 would be good for clustering.
Correct value of k would be the point in the graph where slope is rapidly change – it means that each (actual) cluster is containing centroid.
There is little decrease of average diameter when number of cluster is changes 8 to 16, compare to data before. This means that appropriate k is in 4 and 8.
So by testing by k=6, there is rapid decrease of 6 to 8.
So by testing by k=7, there is also rapid decrease of 7 to 8.
By binary search of method above, it is assumed that appropriate k = 8.

Answer to Problem 2

Solve the following problems, which are based on the exercises in the textbook.

Exercise 11.1.5



Exercise 11.1.7

For the matrix of Exercise 11.1.5

(a) Starting with a vector of three 1's, use power iteration to find an approximate value of the principal eigenvector.

```
[[0.19382269],
 [0.4722473 ],
 [0.85989258]]
```

(b) Compute an estimate the principal eigenvalue for the matrix.

: 7.87298335

(c) Construct a new matrix by subtracting out the effect of the principal eigenpair, as in Section 11.1.3.

```
[[ 0.7042338 ,  0.2793682 , -0.31216407],
 [ 0.2793682 ,  0.24418683, -0.19707644],
 [-0.31216407, -0.19707644,  0.17859602]]
```

```
import numpy as np
from numpy import linalg as LA
```

```
m = np.array([[1, 1, 1]#
              ,[1, 2, 3]#
              ,[1, 3, 6]])
threshold = 0.000001
d = 1
```

```
x = np.array([[1]#
              ,[1]#
              ,[1]])

while(d > threshold):
    next_x = np.matmul(m, x) / LA.norm(np.matmul(m, x))
    d = LA.norm(np.subtract(next_x, x))
    x = next_x
```

```
x # primary eigenvector
```

```
array([[0.1938227 ],
       [0.47224731],
       [0.85989257]])
```

```
lamb = np.matmul(np.transpose(x), np.matmul(m, x))
```

```
lamb # primary eigenvalue
```

```
array([[7.87298335]])
```

```
eigenvalue=lamb[0][0]
```

```
next_m = np.subtract(m, eigenvalue*np.matmul(x, np.transpose(x)))
```

```
next_m # new matrix which subtract out the effort of principal eigenpair
```

```
array([[ 0.70423374,  0.27936812, -0.31216418],
       [ 0.27936812,  0.24418676, -0.19707647],
       [-0.31216418, -0.19707647,  0.17859615]])
```

(d) From your matrix of (c), find the second eigenpair for the original matrix of Exercise 11.1.5.
Eigenvector: Eigenvalue: 1

```
[[ 0.81649652],
 [ 0.40824804],
 [-0.40824867]]
```

(e) Repeat (c) and (d) to find the third eigenpair for the original matrix.
Eigenvector: Eigenvalue: 0.127

```
[[ 0.54384354],
 [-0.78122728],
 [ 0.30646067]]
```

```
m = next_m
```

```
m
```

```
array([[ 0.70423374,  0.27936812, -0.31216418],
       [ 0.27936812,  0.24418676, -0.19707647],
       [-0.31216418, -0.19707647,  0.17859615]])
```

```
x = np.array([[1]₩
              ,[1]₩
              ,[1]])

while(d > threshold):
    next_x = np.matmul(m, x) / LA.norm(np.matmul(m, x))
    d = LA.norm(np.subtract(next_x, x))
    x = next_x
```

```
x # primary eigenvector
```

```
array([[ 0.81649652],
       [ 0.40824804],
       [-0.40824867]])
```

```
lamb = np.matmul(np.transpose(x), np.matmul(m, x))
```

```
lamb # primary eigenvalue
```

```
array([[1.]])
```

```
eigenvalue=lamb[0][0]
```

```
next_m = np.subtract(m, eigenvalue*np.matmul(x, np.transpose(x)))
```

```
next_m # new matrix which subtract out the effort of principal eigenpair
```

```
array([[ 0.03756718, -0.05396498,  0.02116944],
       [-0.05396498,  0.0775203 , -0.03040975],
       [ 0.02116944, -0.03040975,  0.01192917]])
```

```
m = next_m
d = 1
```

```
m
```

```
array([[ 0.03756718, -0.05396498,  0.02116944],
       [-0.05396498,  0.0775203 , -0.03040975],
       [ 0.02116944, -0.03040975,  0.01192917]])
```

```
x = np.array([[1]₩
              ,[1]₩
              ,[1]])

while(d > threshold):
    next_x = np.matmul(m, x) / LA.norm(np.matmul(m, x))
    d = LA.norm(np.subtract(next_x, x))
    x = next_x
```

```
x # primary eigenvector
```

```
array([[ 0.54384354],
       [-0.78122728],
       [ 0.30646067]])
```

```
lamb = np.matmul(np.transpose(x), np.matmul(m, x))
```

```
lamb # primary eigenvalue
```

```
array([[0.12701665]])
```

```
eigenvalue=lamb[0][0]
```

```
next_m = np.subtract(m, eigenvalue*np.matmul(x, np.transpose(x)))
```

```
next_m # new matrix which subtract out the effort of principal eigenpair
```

```
array([[-1.04152797e-14, -9.83241266e-15, -6.50868248e-15],
       [-9.83241266e-15, -1.60982339e-14, -2.36997921e-14],
       [-6.50868248e-15, -2.36997921e-14, -4.88446089e-14]])
```

Exercise 11.3.1

(a) MM^T:

```
m = np.array([[1, 2, 3]#
              ,[3, 4, 5]#
              ,[5, 4, 3]#
              ,[0, 2, 4]#
              ,[1, 3, 5]])
```

```
result = np.matmul(m, (np.transpose(m)))
```

```
result
```

```
array([[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]])
```

M^TM:

```
m = np.array([[1, 2, 3]#
              ,[3, 4, 5]#
              ,[5, 4, 3]#
              ,[0, 2, 4]#
              ,[1, 3, 5]])
```

```
result = np.matmul(np.transpose(m), m)
```

```
result
```

```
array([[36, 37, 38],
       [37, 49, 61],
       [38, 61, 84]])
```

(b) ,

(c)

M^TM:

Eigenvector Eigenvalue: 153.57

```
[[0.40928472],
 [0.56345961],
 [0.71763451]]
```

Eigenvector: Eigenvalue: 15.433

```
[[ 0.81596915],
 [ 0.12587172],
 [-0.56422571]]
```

Eigenvector: Eigenvalue: 0

```
[ [ 0.408 ],
  [ -0.816 ].
  [ 0.408 ] ]
```

*since this vector is eigenvector of M

MM^T:

Eigenvector: Eigenvalue: 153.57

```
[[0.29769591],
 [0.57050851],
 [0.52074188],
 [0.32257923],
 [0.45898553]]
```

Eigenvector: Eigenvalue: 15.433

```
[[ 0.15905953],
 [-0.03320874],
 [-0.73586433],
 [ 0.51038733],
 [ 0.41425319]]
```

Eigenvector: Eigenvalue: 0

```
 0.94131607,
-0.17481584,
-0.04034212,
-0.18826321,
-0.21515796,
```

Eigenvector: Eigenvalue: 0

```
 0.07520849
-0.07287035
-0.10566284
-0.72571726
 0.67171677
```

*since this vector is eigenvector of M^T

Eigenvector: Eigenvalue: 0

```
 0.12508859,
-0.45318832,
 0.32553276,
 0.72000366,
-0.39318742,
```

```
result
```

```
array([[36, 37, 38],
       [37, 49, 61],
       [38, 61, 84]])
```

```
m = result # or m = next_m
d = 1
threshold = 0.0001
```

```
x = np.array([[1]
             ,[1]
             ,[1]])

while(d > threshold):
    next_x = np.matmul(m, x) / LA.norm(np.matmul(m, x))
    d = LA.norm(np.subtract(next_x, x))
    x = next_x
```

```
x
```

```
array([[0.40928472],
       [0.56345961],
       [0.71763451]])
```

```
lamb = np.matmul(np.transpose(x), np.matmul(m, x))
```

```
lamb
```

```
array([[153.56699646]])
```

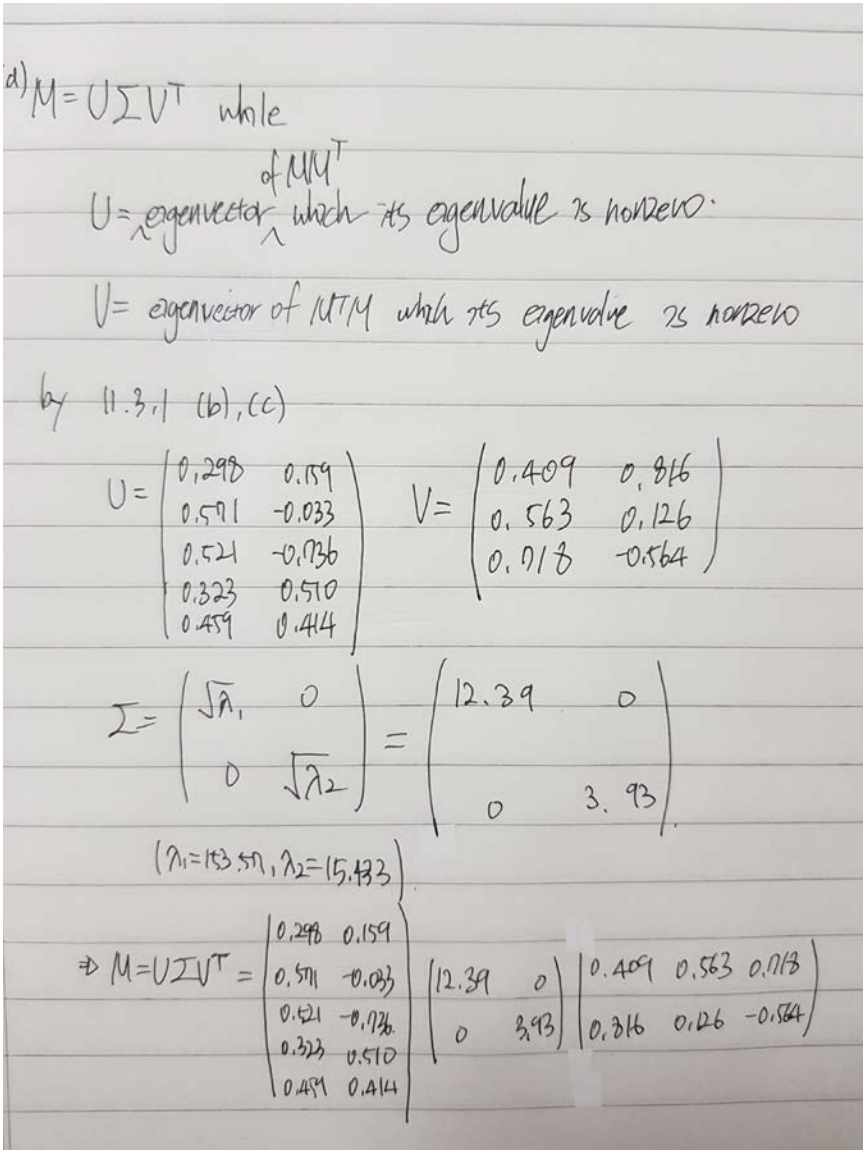
```
eigenvalue=lamb[0][0]
```

```
next_m = np.subtract(m, eigenvalue*np.matmul(x, np.transpose(x)))
```

```
next_m
```

```
array([[10.27538112,  1.58508434, -7.10521243],
       [ 1.58508434,  0.24451573, -1.09605288],
       [-7.10521243, -1.09605288,  4.91310668]])
```

(d) .



(e) .

```
U = np.array([[0.298],#
              [0.571],#
              [0.521],#
              [0.323],#
              [0.459]])
S = np.array([[12.39]])
Vt = np.array([[0.409, 0.563, 0.718]])
```

```
np.matmul(U, np.matmul(S, Vt))
```

```
array([[1.51011798, 2.07871986, 2.65101396],
       [2.89354821, 3.98305047, 5.07962742],
       [2.64017271, 3.63427197, 4.63482642],
       [1.63680573, 2.25310911, 2.87341446],
       [2.32598709, 3.20178663, 4.08327318]])
```

(f) Original energy is 169.003 while one-dimensional approximation's is 153.57. About 90.1% of energy is retained.

Exercise 11.4.2

(a) Jim/John // Matrix/Alien

C, R: by code on right

```
r = 2
C1 = np.array([[1, 3, 4, 5, 0, 0, 0]])
C2 = np.array([[1, 3, 4, 5, 0, 0, 0]])

C1 = C1 / math.sqrt(r * 51/243)
C2 = C2 / math.sqrt(r * 51/243)

C = np.vstack((C1, C2)).transpose()
```

```
C
```

```
array([[1.54348727, 1.54348727],
       [4.6304618 , 4.6304618 ],
       [6.17394907, 6.17394907],
       [7.71743633, 7.71743633],
       [0.        , 0.        ],
       [0.        , 0.        ],
       [0.        , 0.        ]])
```

```
R1 = np.array([[3, 3, 3, 0, 0]])
R2 = np.array([[4, 4, 4, 0, 0]])

R1 = R1 / math.sqrt(r * 27/243)
R2 = R2 / math.sqrt(r * 48/243)

R = np.vstack((R1, R2))
```

```
R
```

```
array([[6.36396103, 6.36396103, 6.36396103, 0.        , 0.        ],
       [0.        , 0.        , 0.        , 7.79422863, 7.79422863]])
```

```
np.matmul(C, np.matmul(U, R))
```

```
array([[0.38895879, 0.38895879, 0.38895879, 0.        , 0.        ],
       [1.16687637, 1.16687637, 1.16687637, 0.        , 0.        ],
       [1.55583516, 1.55583516, 1.55583516, 0.        , 0.        ],
       [1.94479395, 1.94479395, 1.94479395, 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ]])
```

W: X: Y: sigma:

```
[[3, 3]
,[4, 4]]
```

```
[[0.6],
 [0.8]]
```

```
[0.70710678],
[0.70710678]]
```

sqrt (50)

U:

```
[[0.00848528, 0.00848528],
 [0.01131371, 0.01131371]]
```

CUR:

```
[[0.38895879, 0.38895879, 0.38895879, 0., 0.],
 [1.16687637, 1.16687637, 1.16687637, 0., 0.],
 [1.55583516, 1.55583516, 1.55583516, 0., 0.],
 [1.94479395, 1.94479395, 1.94479395, 0., 0.],
 [0.        , 0.        , 0.        , 0., 0.],
 [0.        , 0.        , 0.        , 0., 0.],
 [0.        , 0.        , 0.        , 0., 0.]]
```

(b) Alien/Star Wars // Jack/Jill

C: R: sigma: sqrt(50)

```
[[1.54348727, 1.54348727],
 [4.6304618 , 4.6304618 ],
 [6.17394907, 6.17394907],
 [7.71743633, 7.71743633],
 [0.        , 0.        ],
 [0.        , 0.        ],
 [0.        , 0.        ]]
```

```
[[6.36396103, 6.36396103, 6.36396103, 0.        , 0.        ],
 [0.        , 0.        , 0.        , 7.79422863, 7.79422863]]
```

U:

```
[[0.01414214, 0.01414214],
 [0.        , 0.        ]]
```

CUR:

```
[[0.27782771, 0.27782771, 0.27782771, 0.        , 0.        ],
 [0.83348312, 0.83348312, 0.83348312, 0.        , 0.        ],
 [1.11131083, 1.11131083, 1.11131083, 0.        , 0.        ],
 [1.38913854, 1.38913854, 1.38913854, 0.        , 0.        ],
 [0.        , 0.        , 0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        , 0.        , 0.        ]]
```

(c) Matrix/Titanic // Joe/Jane

C: R: sigma: 2, 1

```
[[1.54348727, 0.        ], [[6.36396103, 6.36396103, 6.36396103, 0.        , 0.        ],
 [4.6304618 , 0.        ],  [0.        , 0.        , 0.        , 7.79422863, 7.79422863]]
 [6.17394907, 0.        ],
 [7.71743633, 0.        ],
 [0.        , 6.57267069],
 [0.        , 8.21583836],
 [0.        , 3.28633535]]
```

U:

[[1, 0]

,[0, 0.25]]

CUR:

```
[[ 9.82269281,  9.82269281,  9.82269281,  0.        ,  0.        ],
 [29.46807844, 29.46807844, 29.46807844,  0.        ,  0.        ],
 [39.29077126, 39.29077126, 39.29077126,  0.        ,  0.        ],
 [49.11346407, 49.11346407, 49.11346407,  0.        ,  0.        ],
 [ 0.        ,  0.        ,  0.        , 12.80722452, 12.80722452],
 [ 0.        ,  0.        ,  0.        , 16.00903065, 16.00903065],
 [ 0.        ,  0.        ,  0.        ,  6.40361226,  6.40361226]]
```

Answer to Problem 3

Solve the following problems, which are based on the exercises in the textbook.

Exercise 9.3.1

(a) Jaccard distance between each pair of users.

Jaccard distance = 1 – Jaccard similarity

While Jaccard distance between user A and user B is written as J(A, B),

J(A, B) = 1/2, J(A, C) = 1/2, J(B, C) = 1/2

(b) Cosine distance between each pair

While Cosine distance between user a and user b is written as C(a, b),

C(A, B) = 0.60, C(A, C) = 0.62, C(B, C) = 0.51

(c) Jaccard distance between each pair of users.

While Jaccard distance between user A and user B is written as J(A, B),

J(A, B) = 3/5, J(A, C) = 2/3, J(B, C) = 5/6

(d) Cosine distance between each pair

While Cosine distance between user a and user b is written as C(a, b),

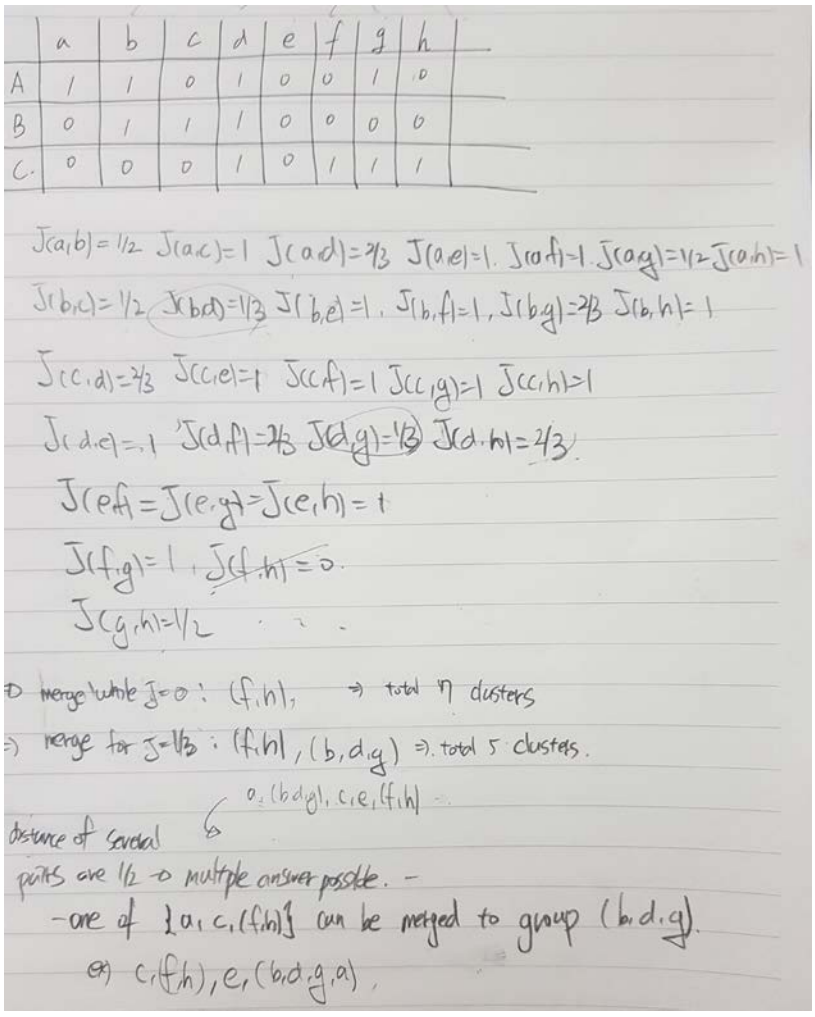
C(A, B) = arccos(0.58) = 54.55°, C(A, C) = arccos(0.5) = 60 °, C(B, C) = arccos(0.29) = 73.14 °

(e) Normalized matrix

	a	b	c	d	e	f	g	h
A	2/3	5/3		5/3	-7/3		-1/3	-4/3
B		2/3	5/3	2/3	-4/3	-1/3	-4/3	
C	-1		-2	0		1	2	0

(f) Cosine distance from Normalized matrix
While Cosine distance between user a and user b is written as C(a, b),
C(A, B) = arccos(0.584) = 54.27°, C(A, C) = arccos(-0.116) = 96.66°, C(B, C) = arccos(-0.74) = 137.73°

Exercise 9.3.2
(a)



(b) For instance, four clusters are {c, (f, h), e, (b, d, g, a)}

	c	(f, h)	e	(b, d, g, a)
A		2	1	4.25
B	4	2	1	2.33
C	1	3.5		3.33

(c) Let name of cluster as P, Q, R, S for each c, (f, h), e, (b, d, g, a)
While Cosine distance between user a and user b is written as C(a, b),
C(A, B) = arccos(0.60) = 53.1°
C(A, C) = arccos(0.89) = 27.1°
C(B, C) = arccos(0.74) = 42.27°

Exercise 9.4.3
(a)
Assume u_{11} is x.
d { $(1.617x + 1 - 5)^2 + (x + 1 – 2)^2 + (x + 1 – 4)^2 + (x + 1 – 4)^2 + (x + 1 – 3)^2$ } / dx = 0
x = 2.314

(b)
Assume u_{52} is x.
d { $(1.617 + x - 4)^2 + (1 + x - 4)^2 + (1 + x - 5)^2 + (1 + x - 4)^2$ } / dx = 0
x = 3.096

(c)
Assume v_{22} is x.
d { $(2.314 + x - 2)^2 + (1 + x - 1)^2 + (1 + x - 5)^2 + (1 + 3.096 * x - 4)^2$ } / dx = 0
x = 1.097

Implement collaborative filtering

hw2_3b.py attached.

User-based method
175 5.0
261 5.0
440 5.0
480 5.0
527 5.0
**additional movies, e.g. 175, 261, 440, 480, 527, 832, 899 have their rating 5.0

Movie-based method
21 5.0
59 5.0
60 5.0
85 5.0
132 5.0
** also additional movies have their rating 5.0.

Movie Recommendation Challenge

hw2_3c.py attached.

Similar with hw2_3b.
The difference is that empty slot was recognized as np.nan. So there was some process needed to verify those.
Also it was possible to assign number of element in one cluster.
In this example, n = 7 was used.